Multi Agent E-Negotiation System

COMPUTER SCIENCE HONORS PROJECT

FINAL REPORT

By: FARSHAD MUHAMMAD

ID:100792339

Abstract

This paper seeks to propose a new type of Multi-Agent Electronic Negotiation (hereby referred to as E-Negotiations) system as an alternative and solution to the growing demands and rarities of such systems in the world of Electronic Commerce (hereby referred to as E-Commerce). It will first explore existing systems already on the market and then, using a newly invented and unique Negotiation Protocol, seek to design and implement a prototype for a system as an alternative solution using software engineering. Comparisons will be made between some existing systems to try and define what makes the prototype unique and how the added features and capabilities expand the process and function of Electronic Negotiations.

Table of Contents

1) Introduction

1.1) Problem Statement

In today's world everything is done through the internet, from shopping, to travel, to managing businesses, there isn't a part of our daily lives that has not been consumed by the World Wide Web. The growing use of the internet by businesses has led to a massive increase in E-Commerce activity all over the globe, an industry that is $170 Billion large in 2006 and seems to be growing by the day [12]. But even for an industry that has experienced such dramatic growth; the electronic infrastructure for E-Commerce still leaves for much to be desired. For example, there are very scarce amount of easy to use tools that enable E-Negotiations, which is a fundamental activity in a business transaction. Buyers and suppliers alike often engage in negotiations on their business transactions. Buyers and suppliers may also want further statistical information on the product during the negotiation itself, such as the average agreed price of the product in previous similar transactions. Multiple parties may engage in such negotiations at both ends at the same time to further complicate the issue. Thus the problem statement becomes clear, the need for an easy to use multi-agent E-Negotiation system.

1.2) Motivation

As the problem statement suggests, powerful platforms supporting E-Negotiations in the world of E-Commerce are sparse, despite the importance and involvement of negotiations and their associated processes in almost all aspects of E-Commerce. Not only that but the lines between auction systems and negotiations are always blurred and thus making distinctions between these two aspects becomes extremely hard at times. Thus it is definitely a worthwhile endeavor to tackle this problem, providing alternative solutions and a brand new platform for conducting E-Negotiations, expanding the uses and power of E-Commerce systems. Not only that, developing a protocol for the Negotiation process strives to differentiate and provide distinctions between simple auction systems and E-Negotiation platforms.

1.3) <u>Goal & Objectives</u>

The goal of this paper is to provide information on existing systems in the universe of E-Negotiations and then provide a powerful alternative to such systems in the form a base prototype for a new type of E-Negotiation platform (namely the Multi-Agent E-Negotiation Platform). Software engineering practices will be heavily used in the implementation of the prototype and a full design document will need to be created before implementation can take place. The design process will involve performing a thorough requirements analysis, producing a GUI, making a functional relational database model and producing subsystem design according to software engineering design patterns. A full report of tests conducted and system results will also be provided upon completion of the implementation.

A Negotiation protocol will also need to be designed which will provide set rules for the negotiation process. The Negotiation protocol will seek to control information provided to users from the system, maintaining fairness and privacy while still inducing an active competitive aspect to the negotiation process.

Comparisons will also be needed between existing systems outlined to the newly designed and implemented protocol, results of said comparisons will then be used to provide evidence as to the superiority of the proposed solution.

1.4) Outline

This paper will be divided into 4 main chapters, which are Background (Chapter 2), Scope of Work (Chapter 3), System Design (Chapter 4), System Functionality (Chapter 6). All these chapters will also be followed by a small conclusion section. References will be provided at the end of the paper. Below are brief descriptions of each chapter;

Background : This chapter will introduce existing E-Negotiation and provide some information on select systems and how they could be improved upon. A solution to the problem statement is also provided here. This chapter seeks to solidify the goals and provides material to run comparisons with for the proposed solution.

Scope of Work: This chapter will provide brief information on the system to be implemented and its underlying negotiation protocol.

System Design: This chapter will provide the entire software design process for the implementation of the system. Almost all aspects of software engineering practices will be covered in this section; this section can also be seen as the “Design Document” for the implementation.

System Functionality: This chapter will seek to explain what was actually accomplished in the allotted time and the results provided by the system itself. It will also seek to run comparisons between the resulting system and existing systems introduced in the “Background” chapter.

Conclusion: Closing Remarks and Future Work

2) Background

2.1) Existing Systems

Existing systems addressing the aforementioned problems above are sparse and hard to find. Searching for “e-negotiation platforms” or “e-negotiation systems” on major search engines on the web such as google, bing and yahoo mostly provide abstracts from research papers written to model certain types of e-negotiation systems.

One of the only substantial e-negotiation systems present in the online market at the moment is Smartsettle [1-2]. Smartsettle currently provides two solutions to the e-negotiation problem in the form of SmartSettle One and SmartSettle Infinity.

SmartSettle One is a case based negotiation system, in which two parties can conduct negotiations on a small single numerical case (a small debt resolution between two individuals is a good example) [1]. SmartSettle one initiates the negotiation through an e-mail notification upon which the users can accept the case in an online web browser based API. They can then manage the negotiation and SmartSettle One allows them to negotiate and check the overlap between their negotiations until both parties agree to a resolution [1]. SmartSettle One is however limited to only two parties per case and works for only a single numerical issue at a time [1].

SmartSettle Infinity is the multi-agent counterpart to SmartSettle One and allows for multiple parties to negotiate on multiple numerical or non-numerical issues [2]. It is however a business to business tool and is only available after a quote has been received by the company [2]. There is also very limited information provided on the usage and limitations of the product.

Other e-negotiation systems come in the form of research systems built by university departments, such as the “Invite” group of systems from Concordia. Invite is a platform from which research departments create many other negotiation platforms and tools [3]. Four platforms have been created to date using the Invite system which are SimpleNS, Inspire2, Imbins and Imaras [3]. All four of these platforms are used for teaching, research and developmental purposes only [3].

The first system, SimpleNS, is an e-negotiation platform where two parties are paired up randomly and are required to conduct negotiations by a certain negotiation deadline [4]. Negotiations are made in the form of passing messages between parties in an e-mail like system until an agreement is reached [4]. Questionnaires may also be required to be filled out during the negotiation process [4]. SimpleNS is not a multi-agent negotiation tool and can only handle negotiations between two parties at a given time [4].

Inspire2 is another negotiation platform developed using the Invite system. It provides many functions to help facilitate negotiations between two parties such as management of communication, a graphical display of the negotiation progress, post agreement analysis and many more [5]. It is mainly used to simulate negotiations and negotiation like conditions and can even be used as a game to train individuals in the art of negotiating [5]. Like SimpleNS, Inspire2 is also not a multi-agent platform and facilitates negotiations between two parties only [5].

Imbins, unlike the previous two platforms is a multi-agent negotiation platform and facilitates negotiations between many buyers and sellers [6]. The sellers however have no accesses to each other's information, giving the buyer full power over the negotiation process [6]. All negotiations in Imbins must come to an agreement in a preset amount of time [6].

Imaras, is similar to Imbins but instead of having a single negotiation phase, it creates several rounds in which the negotiations take place [7]. Imaras also gives full power to the buyers, as it does not allow any seller to gain access to another seller's information and thus doesn't allow for open competition amongst sellers [7]. All negotiation rounds in Imaras are also timed [7].

As seen above most available platforms for e-negotiations are either limited to one to one relationships between buyers and suppliers, or if they are multi-agent, are not accessible to the general public, but either used in a business to business setting or for academic and research purposes. The multi-agent platforms mentioned above like Imaras and Imbins hinder competition between sellers as well, by only giving buyers access to all the information and thus giving buyers all the power in the negotiation model [6 – 7]. Also, none of these systems seem to provide data mining tools from previous transactions or a way for consumers to compare their current ongoing negotiations with the result of similar previous negotiations held by the system.

2.2) The Proposed Solution

The proposed solution comes in the form of the development from the ground up of a Multi Agent E-Negotiation System. The system will be modeled as an e-commerce platform which will allow buyers and suppliers to communicate and negotiate on a product or set of products. This system will allow a single buyer or a group of buyers to communicate with multiple supplier(s) about the transaction of a single or multiple products. Supplier(s) will then be able to present the buyer(s) with their proposed prices, delivery time, and terms and allow them to compete with other supplier(s) by "sweetening the deal" so to speak and provide buyer(s) with information on how their services would benefit them more than other supplier(s). The buyer(s) will be able to easily manage and concurrently view prices and information supplied by the supplier(s) in a single integrated application. Buyer(s) will also be able to communicate their demands and negotiate a better price for the services or products they need and decide which supplier is offering them the best deal through multiple channels. Buyer(s) will then be able to commit to the deal with one of the supplier(s) which would close the negotiation channels. Buyer(s) and supplier(s) will also be able to navigate through their previous transactions through the same system. The system will also allow supplier(s) to look at each deal offered to the buyer(s) by other supplier(s) to see if they need to make a better offer to keep up with the competition.

Furthermore, both buyers and suppliers will be able to access statistical information about previous transactions of a specific product in the system's database (such as average price agreed upon a product in the last 100 transactions) to compare to their current ongoing negotiation with a simple press of a button.

The Negotiation process itself will be enforced using a Negotiation Protocol which will make the system unique and provide an element of competition between users by trying to reduce the chances that cheating and shirking may occur between participants. This will be achieved by being selective in the type of information each participant has access to in the Negotiation process.

3) Scope of Work

3.1) The Prototype

The system to be implemented detailed in the “Overview” section above will be a prototype for a unique E-Negotiation platform. It will boast all the functionality to provide capabilities required which were stated in the “Proposed Solution” section. Although functionality will be present for all the promised features, they will be basic and simplistic in nature; the main goals for the prototype will be to:

1) Provide basic features to tackle the problem by providing a multi-agent E-Negotiation interface for buyers and suppliers over products and services.
2) The prototype should be designed in such a manner as to be completely extensible and for features to be easily abstracted, changed, removed and built upon. This will be achieved using software engineering practices.
3) The main features the prototype will focus on, is the Negotiation process itself and the protocol that enforces said process.

3.2) The Protocol

The protocol enforcing the Negotiation Process within the prototype will need to reduce the chances that cheating and shirking may occur between participants, while still promoting some form of healthy competition between them. This will be done by implementing certain functionalities to limit the availability of data to some participants; details are provided below:

1) Users who initiate a Negotiation become the creators of said Negotiation.
2) Users who join created/initiated Negotiations become into participants of that Negotiation.
3) Creators are able to view all offers provided by every participant at any given time and are able to communicate with all participants.
4) Participants are not allowed to communicate with other participants; they can only communicate with the creator of the Negotiation.
5) Participants are only able to view their own provided offers and the values requested by the creator, they are not allowed to directly view any offers posted by other participants.
6) At any point in the Negotiation process all participants and the creator will have access to the lowest/highest provided offer on each aspect of a product or service being negotiated upon, though they are not provided information on the provider of the offer.

4) System Design

4.1) Requirements

A list of requirements were produced after much discussion and thought as to the main functionality of the E-negotiation system prototype. Here is the finalized list of requirements for the system.

- Users should have a unique username for logging in.
- Users should have an e-mail address attached to their usernames for providing notifications for when they are offline.
- Users should be able to create/initiate Negotiation Instances on any products and services.
- Once users create a Negotiation Instance, they become the creator/owner of that instance.
- The creator of the Negotiation Instance has to declare whether the instance is of a “buying” or “selling” type.
- Product Negotiation instances should have a name, a set initial price, a set initial amount for shipping, a set amount of time required for shipping and an initial quantity of the product they are negotiating on. These initial values are set by the creator.
- Service Negotiation instances should have a name, a set initial price and a set amount of time required for the completion of the service they are negotiating on. These initial values are set by the creator.

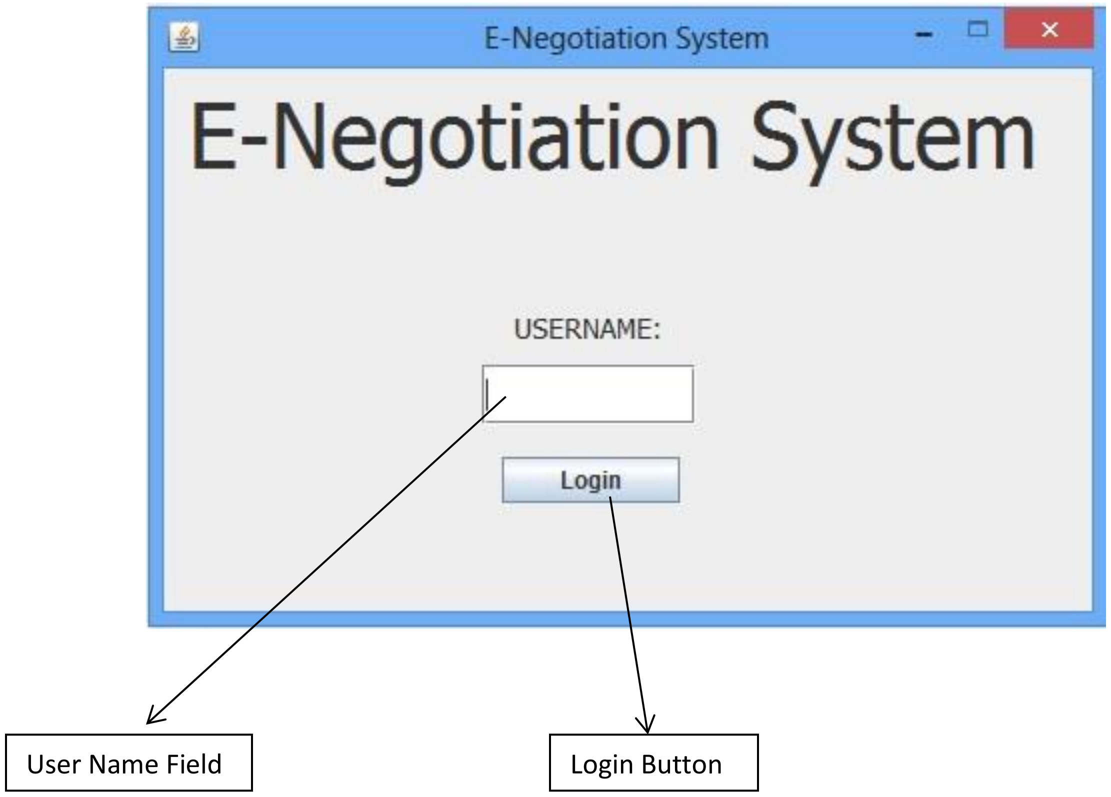
- Once a Negotiation instance has been created it has to be persisted in the database, providing it with a unique ID which the creator is notified of.
- A single Negotiation Instance should allow for up to 4 other users to join it and conduct their Negotiations with the creator of the instance.
- Once a Negotiation Instance has 5 participants (including the creator) it should not allow other users to access the instance.
- Each user in a product Negotiation instance should be able to set values for price, shipping time, shipping cost and quantity. All users are then notified of these set values.
- Each user in a service Negotiation instance should be able to set values for price and time required for completion. All users are then notified of these set values.
- The creator of the Negotiation Instance can see the values set by all users.
- Other users in the Negotiation Instance can only see their own set values and that of the creator.
- The creator of the Negotiation Instance should be able to commit to an offer provided by a user, thus concluding the Negotiation Instance. All users which are online and participants of said instance are notified of the commit.
- E-mail notifications are sent out to all participating users following a commit so they are notified even if they are not online.
- Users can choose to withdraw from a Negotiation Instance at any time, this frees up slots for other users to be able to join and participate in the Negotiation Instance.

- Users should also be able to rejoin a Negotiation Instance they have chosen to withdraw from.
- If a creator withdraws from a Negotiation Instance, that instance is then cancelled and permanently deleted. Participants who are online are notified of this event.
- E-mail notifications of deleted Negotiation Instances are sent to their respective participants.
- Users should be able to easily view and find their created and joined Negotiation Instances.
- Users should be able to view and retrieve their commits and their associated values.
- Users should be able to search for active Negotiation Instances for both products and services.
- Users should be able to easily retrieve a Negotiation Instance using its ID.
- Users should be able to easily communicate with the creator of the Negotiation Instance.
- Users should be able to data mine during a negotiation process, which provides them with averages of information from past transactions.
- Users should be provided with information regarding the lowest/highest input values for their respected fields by each user to promote a certain sense of competition.

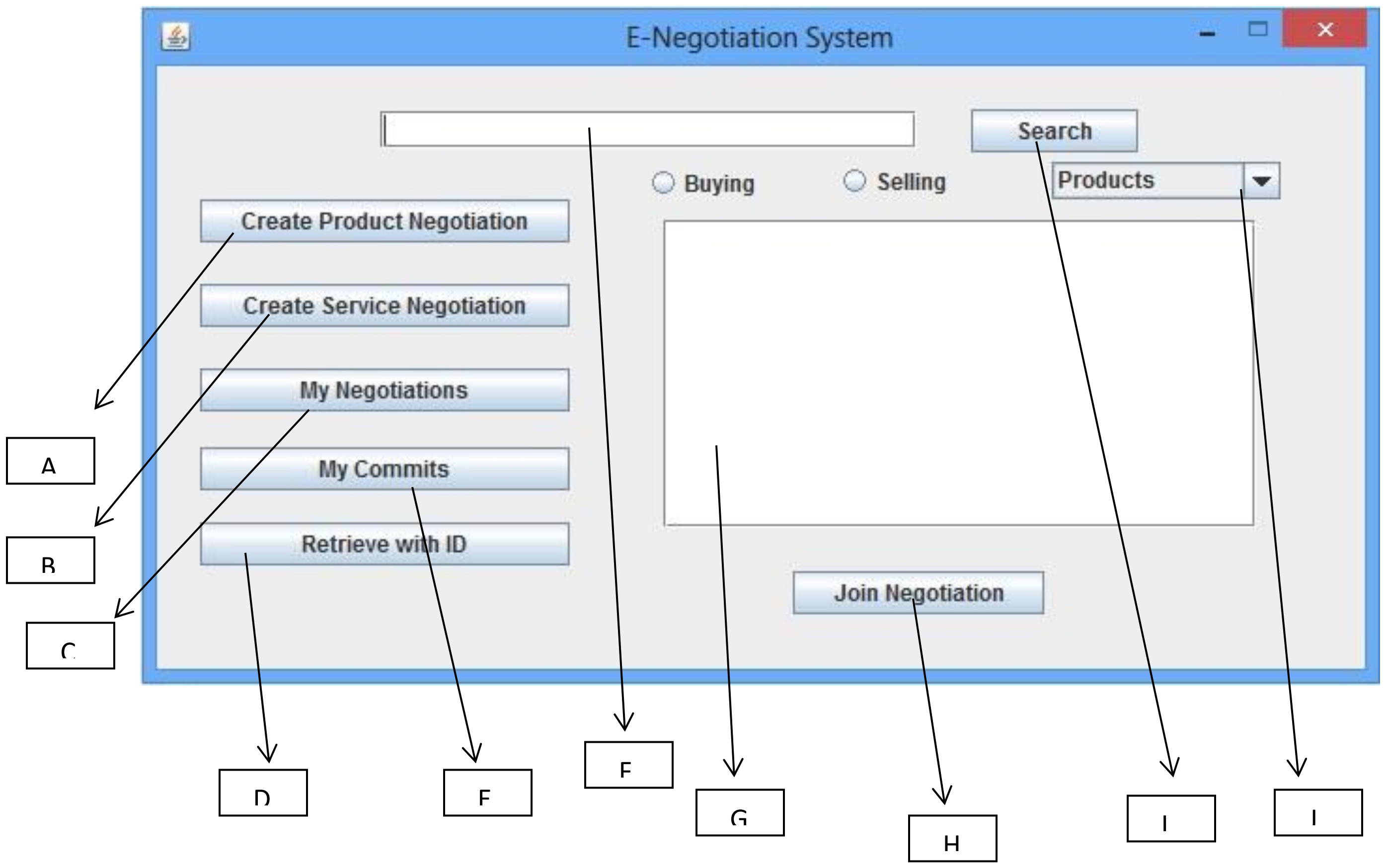
4.2) GUI

Following the finalized requirements analysis, a semi functional GUI was produced for the system using the Eclipse Window Builder. A Card Layout was decided to be used in producing the GUI, using a single JFrame with changing JPanels as needed. Screen shots and details of the GUI are provided below:

1) Login Page – This is the starting page of the application, it simply allows users to enter their login information or gives them an error message if it is invalid.



2) Choice Page – This is the main Navigation Page, details of all the buttons and their uses are provided below.



A) A button that leads to the “Create Product Negotiation Instance” page.

B) A button that leads to the “Create Service Negotiation Instance” page.

C) A button that leads to the “My Negotiations” page.

D) A button that leads to the “Retrieve” page.

E) A button that leads to the “My Commits” page.

F) Text Field for user search input, this is used to search for Negotiation Instances

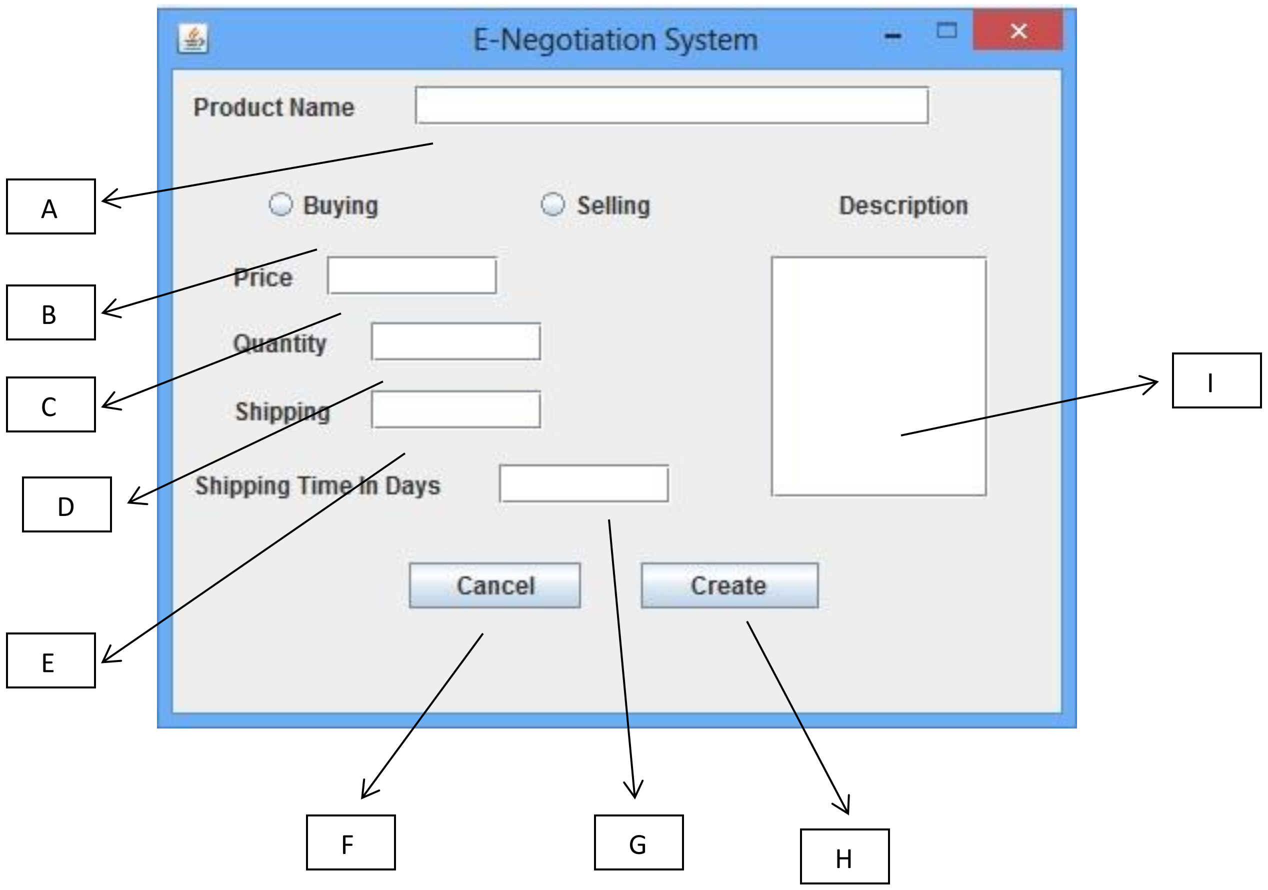
G) List area where search results are displayed

H) Join Negotiation button, allows user to enter selected negotiation instance from the search result list.

I) Search button, populates the list according to the input in the text field.

J) A combobox allowing for users to switch between searching for Product or Service based instances.

3) Create Product Negotiation Instance Page – This page allows a user to create a new Product Negotiation Instance given a product name and values for price, shipping, shipping time, and quantity. The user is then prompted with a notification of the ID of the created instance.



A) Field to enter name of product

B) Radio buttons to specify instance type.

C) Field to enter initial value of Price.

D) Field to enter initial value of Quantity.

E) Field to enter initial value of Shipping

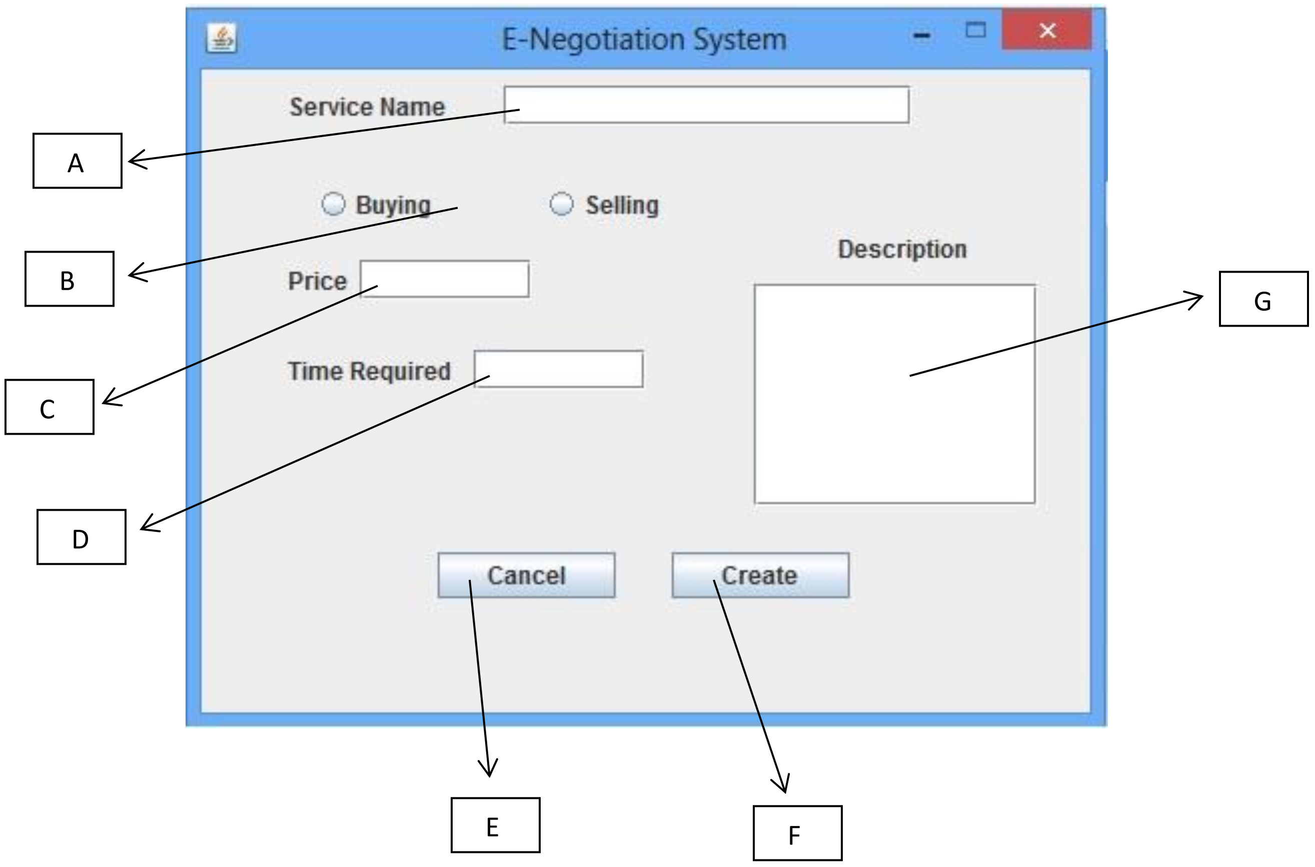
F) Back to Choice Page

G) Field to enter initial value of Shipping Time.

H) Button to submit changes and create instance

I) Field to enter further written description of product

4) Create Service Negotiation Instance Page



A) Field to enter the service name for the Negotiation Instance

B) Radio Buttons to set the type of the Negotiation Instance

C) Field to enter initial price of service

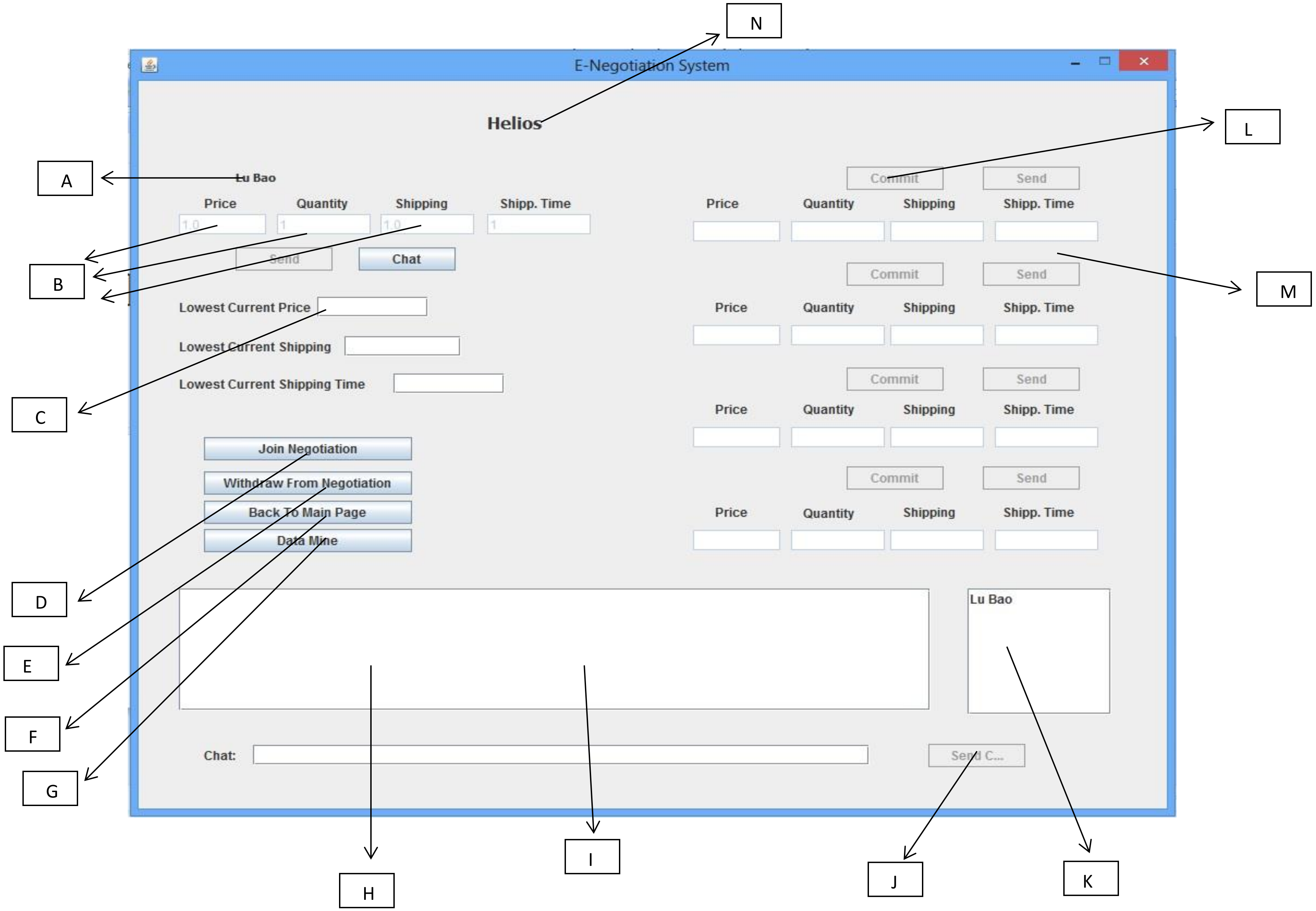
D) Field to enter initial time required for completion of service

E) Back to Choice Page

F) Creates the Negotiation Instance in accordance with supplied values

G) Field to enter further description of service

5) Main Product Negotiation Page – This page is the heart of the program, this is the main Negotiation Instance, where users negotiate on a given product, users can also data mine on the given product provided there are existing transactions in the database with the same product name.


E-Negotiation System
Helios
Lu Bao
Price
Quantity
Shipping
Shipp. Time
1.0
1
1.0
1
Send
Chat
Lowest Current Price
Lowest Current Shipping
Lowest Current Shipping Time
Join Negotiation
Withdraw From Negotiation
Back To Main Page
Data Mine
Commit
Send
Price
Quantity
Shipping
Shipp. Time
Commit
Send
Price
Quantity
Shipping
Shipp. Time
Commit
Send
Price
Quantity
Shipping
Shipp. Time
Commit
Send
Price
Quantity
Shipping
Shipp. Time
Lu Bao
Chat:
Send C...
A
B
C
D
E
F
G
H
I
J
K
L
M
N

A) User Name

B) User Value Fields

C) Lowest Values Fields – at each step the lowest values offered by each user are displayed here

D) This button is used to join the negotiation and take up a user slot

E) This button is used to withdraw from the negotiation, if the creator withdraws from the negotiation, the negotiation instance is cancelled and deleted

F) Back to Choice Page

G) This button is used to Data Mine- Averages of previous transactions of the product are presented to the user

H) This is the main chat area, it changes according to the user selected from the user list

I) Chat messages are inputted here

J) This button is used to send chat messages to selected user

K) This is the user list In the current negotiation, used for chats

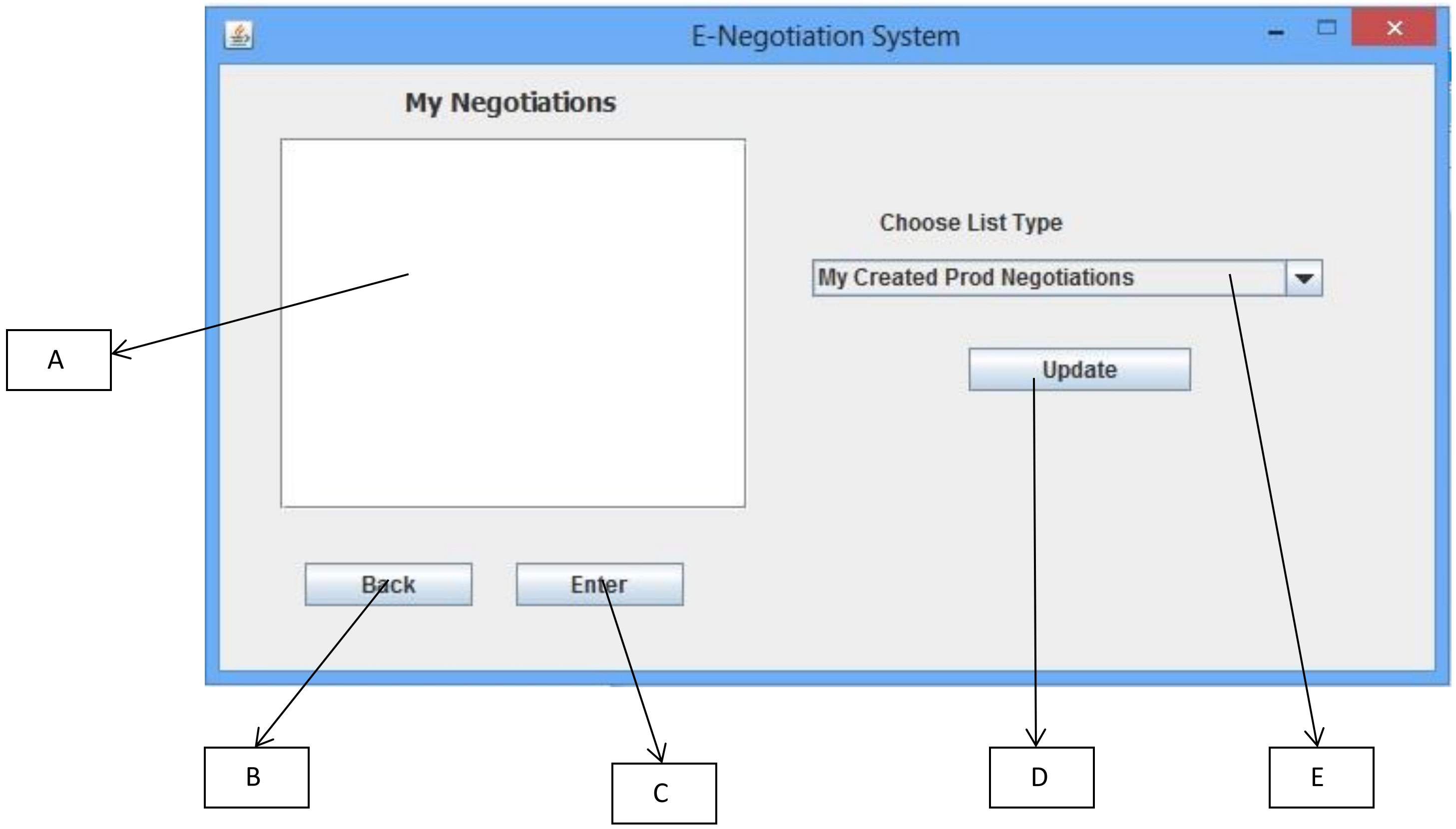
L) This button is used by the creator to commit to a user offer and thus conclude the negotiation

M) This button is used to update a user's values

N) Product Name

The Service Negotiation page is the exact same as Product Negotiation page. The only difference lies in the values the users define, instead of Price, Shipping, Shipping Time and Quantity, these values are replaced by Price of the service and time required for completion.

6) My Negotiations Page – This page allows users to easily keep track of their created and joined negotiations, users can enter any of these negotiation instances through this page. Negotiations instances are grouped into created and joined types.



A) This area is where the users listed negotiation instances are presented

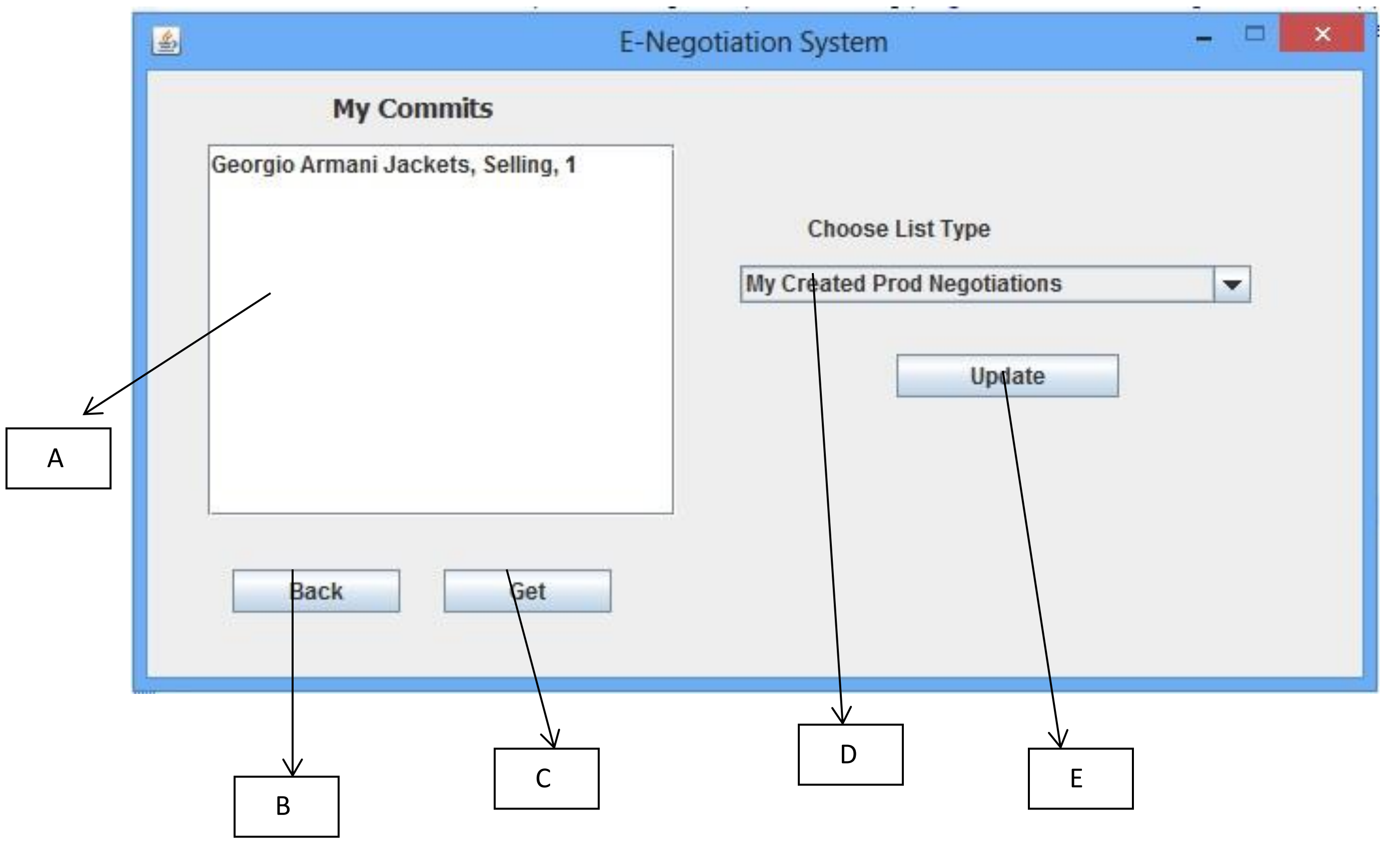
B) Back to Choice Page

C) This button allows the user to access the selected negotiation instance

D) This button updates the list according to the choice made using the drop down menu

E) This drop down menu allows users to choose between their created and joined product and service negotiation instances by updating the list using the update button.

7) My Commits Page – like the My Negotiations Page, this page keeps track of all of a user's committed/successfully concluded negotiations. It classifies them by putting them under the joined or created category of negotiation instances.



Georgio Armani Jackets, Selling, 1

A) Area where all commit listings are output

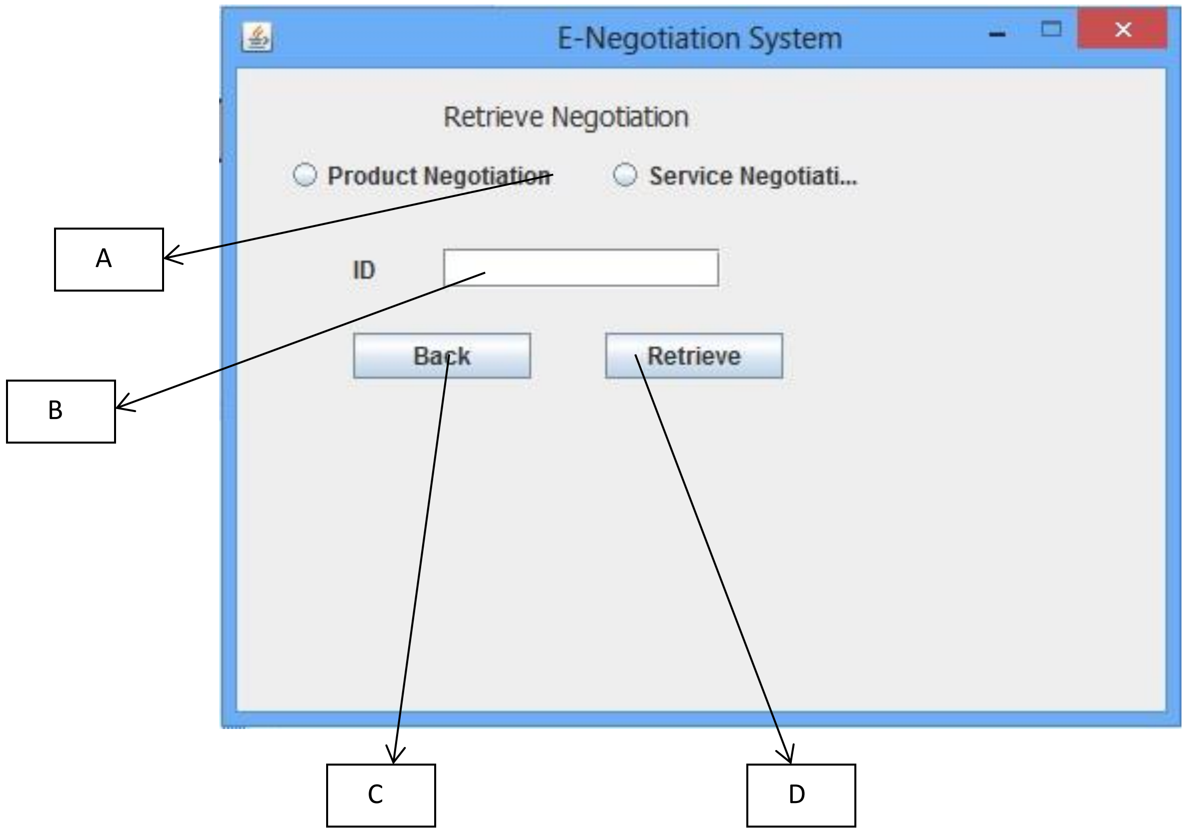
B) Back to Choice Page

C) Gets the information of the selected commit

D) Allows user to choose between created and joined Negotiation Instance types.

E) Updates the output list according to the user's choice

8) Retrieve Page – This GUI page simply allows for the quick retrieval of a negotiation instance using its ID.



A) Radio buttons for the user to choose between product and service negotiation instance types

B) Field to input the negotiation ID

C) Back to Choice Page

D) Retrieves the negotiation according to the provided ID. Outputs an error message if the negotiation is not found.

4.3) <u>System Architecture</u>

The system being designed will need for users to log in to the system. Transactions, negotiations and participants will need to be recorded and retrieved. Users should also be able to retrieve most any information available to them, changes made from user interactions will need notifications for all participants and records of important changes will need to be kept. All of these important points make a client-server architecture an obvious choice for the implementation of the prototype.

The client-side of the system will provide a graphical interface (the GUI) which the users will use to interact with. It will not need to be threaded as the client will only be responsible for the single user logged into the client. It will be reactionary to messages received from and sent to the server. Since the client will use a graphical interface for user interaction it will need to follow the standard Model-View-Controller architecture that is common in software engineering practices.

The server will be where the database is housed and it will act as a relay sending data from the database to connected clients as well as replying to messages received by clients. It will need to be threaded as to deal with as many clients that can be connected concurrently as required. The server will need to be efficient in its reply to messages and also handle notifications that need to be sent outside the system, such as a webserver.

4.4) <u>Objects and Relations</u>

The next step in the system design involved using the mock GUI and keywords from the requirements analysis to create objects and relations between said objects. Obvious keywords were chosen from the requirements analysis after discussion of particular scenarios of usage. The most recurring keywords were found to be: Users, Negotiation Instance and the negotiation process itself within the instance. Detailed explanation of these objects is provided below. Products and Services, even though being important parts of the system prototype are for the time being only represented as a name and a possible short description. As the goal of the system prototype is to focus on the negotiation process itself the products and services are just represented as variables within the Negotiation Instance they are attached to.

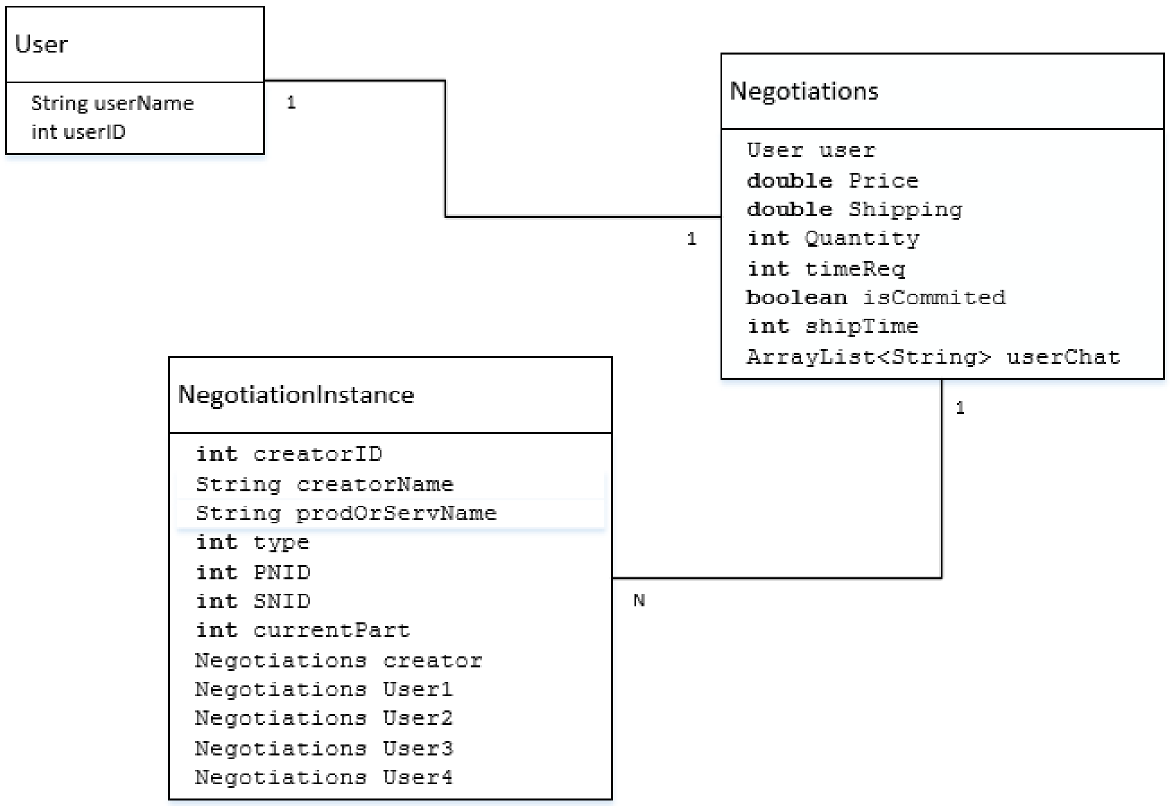
User – This object represents all the participants within the system, may it be creators of negotiation instances or participants within the negotiation instance, or simply users observing a negotiation process to familiarize themselves with the system.

NegotiationInstance – The heart and soul of the implemented project itself, the NegotiationInstance encapsulates the entire negotiation process, providing users with a platform to negotiate through as well as keeping track of all of the offers made by the participants. Each NegotiationInstance also holds one Negotiations object for each participant.

Negotiations – This object was added to represent the negotiation process itself, each NegotiationInstance holds Negotiations which keep track of the participants involved, and for each participant, their corresponding offers for each aspect of the product of service being negotiated on. Negotiations also keep track of a User's chat log with other participants. Each Negotiation object is corresponded with a single user and a single NegotiationInstance.

Class Diagram

This diagrams details the three main objects used by the system and represents visually, their variables as well as their associations.



4.5) <u>Database Model</u>

The objects determined above and their associations were used to model a relational database which would hold all the information required for overall system functionality. Each object was then mapped to identical database entities and relations were formed between the entities using the associations in the class diagram above. Each entity was then translated to a database table.

Below are the mappings used from object to database entities:

Users → Users

NegotiationInstance → ProductNegotiationInstance

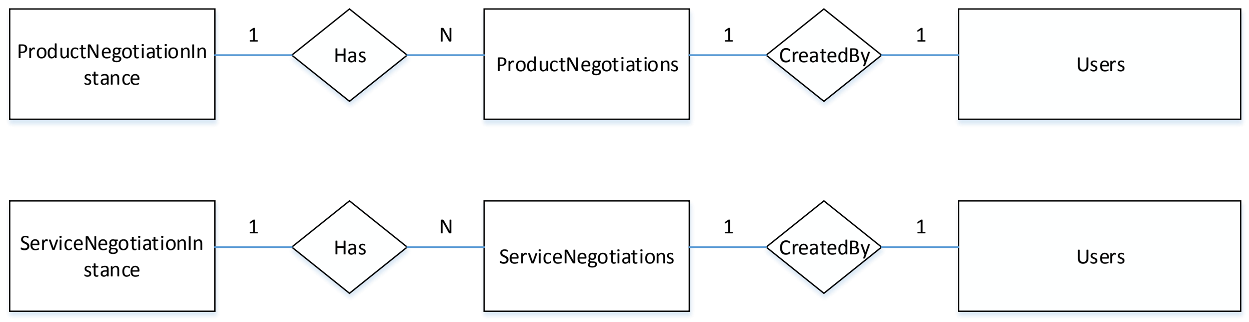
NegotiationInstance →ServiceNegotiationInstance

Negotiations → ProductNegotiations

Negotiations →ServiceNegotiations

NegotiationInstances and Negotiations were both mapped to their product and service counterparts. This does bring the issue of some redundancy in the design of the database tables but allows for concise compact tables with simpler joins which in turn speed up the overall processes within the system, as well as shorter lookup times. This design choice was also made to segregate negotiation instances of products and services to keep them separate in the system itself.

Below are two simple ER models detailing the relations between the entities within the database:



Database Tables:

Using the relations above tables were then created for each entity which are shown below:

Primary keys are denoted by * sign next to them.

Users

UID*	userName	EMail

ProductNegotiationInstance

CreatorName	Type	prodName	PNID*	creatorID	isActive	currParticipants

ServiceNegotiationInstance

CreatorName	Type	servName	SNID*	creatorID	isActive	currParticipants

ProductNegotiations

userName	isCommited	Quantity	ShipTime	Shipping	Price	PRNID*	UID	PNID

ServiceNegotiations

SRNID	SNID	UID	userName	Price	timeReq	isCommited

4.6) Subsystem Design

Once the database model has been set and the tables created, a proper subsystem design was created and for each subsystem, their classes were determined. Below is an explanation of all the subsystems and their classes for both the client and server sides.

Server Side Subsystems

The server side of the system should be able to initialize and accept connections from the client. The server needs to be threaded so as to provide concurrency between consecutive client connections, allowing many different clients to communicate and interface with the server at once. Sockets will be used for server to client connections.

The server should be able to accept, read parsed XML messages and create appropriate replies to the messages received. The server should also be able to push notifications for changes to all or certain groups of clients. Some notifications may require E-mails to also be sent out to respective users, thus the server should be able to communicate with a web e-mail server and push the notifications out. All XML messages are parsed to single strings before being sent out through a socket. The server is also responsible for retrieving, updating, inserting and deleting data into the database.

Below is a rough description of each subsystem and its corresponding classes:

Server Subsystem

This subsystem just holds the main server class and is responsible for initializing and opening a server socket which accepts threaded socket connections to the server. The server class also keeps track of all the clients connected to it and assigns IDs to each socket connection. This list of client connections and their corresponding IDs are then used to push notifications out to respective clients.

XMLParser Subsystem

This subsystem holds three main classes, SThread, MessageReader and MessageBuilder. This is the most essential subsystem on the server side and is responsible for receiving messages, translating them and then interfacing with the database and the e-mailing systems to fashion appropriate replies to be sent back to the client or notifications to be pushed out to several clients at once. All of the functionality of this subsystem is encapsulated by a single server thread.

SThread: The main server thread for a client connection which holds the socket from which messages are received and sent. The SThread class sends the received message string to the MessageReader class.

MessageReader: This class receives the parsed string received from the client by the server thread and then converts it to a proper XML message. It then reads the message and interfaces with the MessageBuilder class to create a proper reply to the message.

MessageBuilder: This essential class receives variables and objects from the message reader and then fashions a reply by first interfacing with the database wrapper to retrieve, persist or delete information in the database and then creating a proper parsed reply to be sent back to the client. If needed it also interfaces with the e-mail subsystem and pushes out e-mail notifications as necessary.

<u>Emailer Subsystem</u>

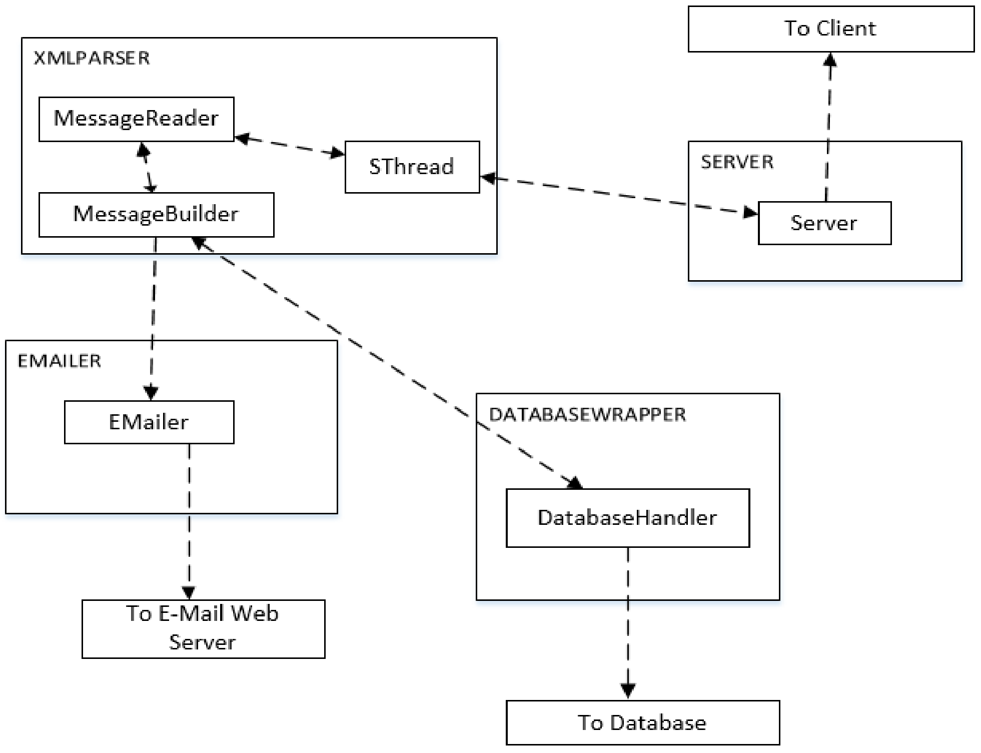
This subsystem contains a single class called EMailer which is responsible for sending e-mail notifications to users as required. The User's e-mail addresses are retrieved from the database when required and passed into the e-mailer through the MessageBuilder.

Database Subsystem

This subsystem is a simple database wrapper and holds the DatabaseHandler class which connects to the system database and provides a whole host of functions for retrieving, inserting, deleting and updating information.

Server Side Subsystem Diagram

This diagram shows all the subsystems and their inherent classes and how they communicate with each other on the server side.



Client Side Subsystems

The client side of the system should be able to initialize and connect to an open server through socket connections. The client is responsible for all direct user interactions with the system through a GUI. Most interactions yield a message which gets sent to the server, the GUI is then updated and the user is notified of any changes resultant from the interactions according to the reply from the server.

The client parses, reads and sends messages in a similar manner to the server using an XMLParser subsystem. The client holds lists of all of a User's concluded and active negotiations. If in case errors occur in the server side or the client side, the user is notified by the client and it is responsible for handling errors and invalid inputs in a graceful manner. The client side is not threaded as each client only deals with interactions from a single user and messages received by the server.

Client Subsystem

This subsystem contains just a single class called Client. It is responsible for initializing the sockets and creating a connection to the server. It also allows for sending and receiving messages through the initialized socket.

XMLParser Subsystem

This subsystem is very similar to its counterpart on the server side except that it isn't threaded. It contains two classes called MessageReader and MessageBuilder and is responsible for creating XML messages to be sent to the server as well as reading them. It is also responsible for initiating and notifying other subsystems and objects of changes that need to be done according to messages from the server.

MessageReader: This class receives messages in the form of strings from the client and composes XML messages/documents out of them. It then uses the information in the message to notify other parts of the system of changes. Changes can be done directly to objects as this class interacts with the ObjectWrapper subsystem or can interact with the GUI through the ControllerBridge subsystem.

MessageBuilder: This class provides many functions to build XML messages to be sent to the server. The XML messages are parsed into single strings before being returned.

ObjectWrapper Subsystem

This subsystem holds all the objects that were stated in the “Objects and Relations” section above. It also seeks to encapsulate all objects within a wrapper class called ObjectWrapper which is used to interface with the objects.

ObjectWrapper: This class holds all the objects required for the system to function. It also maintains a lot of lists used for caching and displaying information about Negotiation Instances. It also provides some static functions to update these lists and maintain some critical object changes if required.

GUI Subsystem

This subsystem holds all the separate GUI pages. It is simply the view portion of the model-view-controller architecture of the client side. Some GUI pages do hold some little amount of controllers used for very simple list updates.

ControllerBridge Subsystem

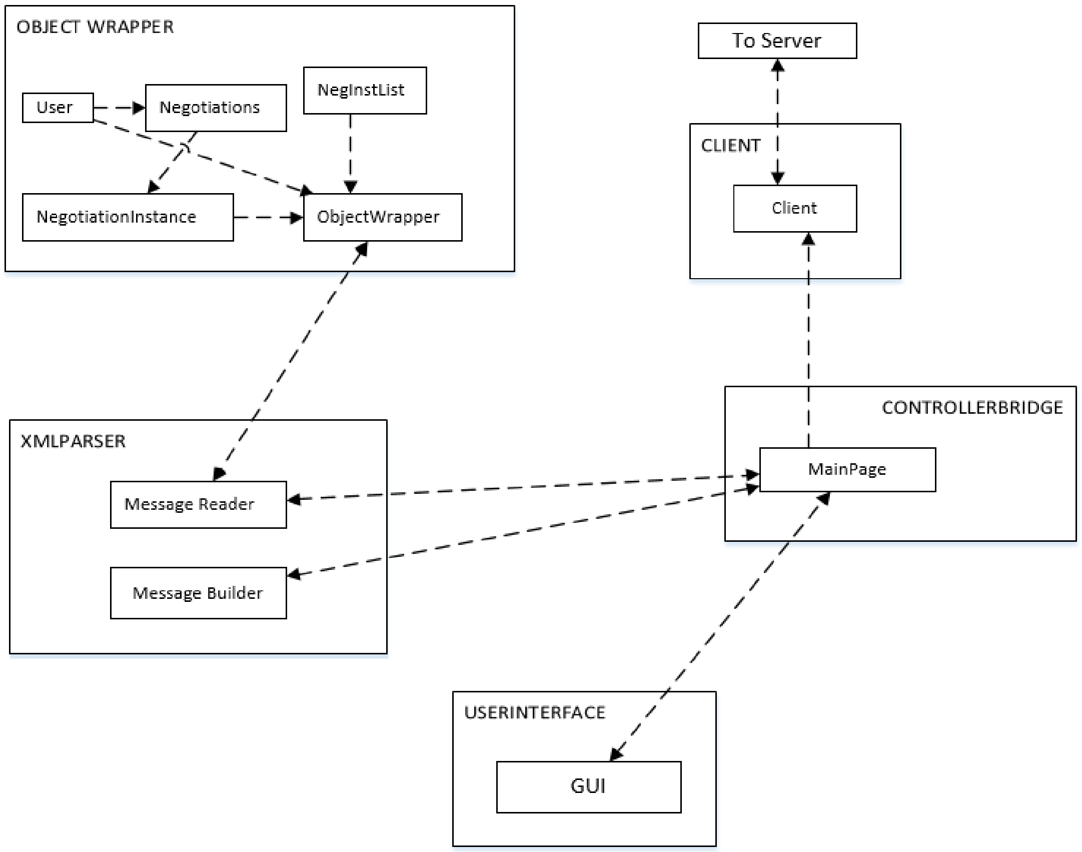
This is the most essential subsystem on the client side of the system; all the functionality of this subsystem is handled by a single class called the MainPage. The many different uses and functionality of the MainPage class are detailed below.

MainPage: This class is the main controller section of the model-view-controller of the client side. It also handles a lot of different types of functionality:

1) It has the main frame which initializes and interfaces with all the GUI pages.
2) It initializes almost all of the different sections of the client side of the system and its subsystems.
3) Most controllers and listeners for the GUI are located and handled by this class.
4) It provides a multitude of different static functions to update the GUI pages.
5) It also is responsible for trimming lists should they prove to become too large as to cause system downtime.
6) This class interfaces with almost every other subsystem on the client side.

Client-side Subsystem Diagram

This diagram gives a visual representation of all the subsystems on the client side and provides information on how they communicate.



4.7) Design Patterns

Software engineering based design patterns start to emerge from the subsystem diagrams upon more careful observations. The design patterns used in the system are:

The Observer Design Pattern

The observer pattern is a design pattern whereby an object/class notifies all other dependent objects/classes about changes that have occurred [8]. We see this design pattern being used specifically in the client-side of the system by the MessageReader class. This class waits and polls the server for incoming messages and once a message has been received it then notifies all other objects/classes on the client side about the incoming changes that need to be made. The MessageReader is then the observer subject and all the other classes/objects connected to it will be regarded as its list of dependents [8].

The Facade Design Pattern

The facade pattern is a software design pattern whereby an object/class provides simplified access to other classes and their functions [9]. The ControllerBridge Subsystem in the client side of the system and its class MainPage was designed to do just this as it provides many easy static functions to quickly update and change values in any of the GUI pages. It also provides static function which can indirectly affect objects that are used to update the GUI given changes. The façade also allows for easy interface of the client and provides static functions to send certain messages to the server should the need arrive. It also helps maintain a card layout of the entire collection of GUI pages, allowing to easily display any GUI page within a single frame, static functions are provided to interface with the card layout as well [9].

4.8) <u>Tools</u>

This section details the tools required for implementation of E-Negotiation prototype:

Java was used as the primary language to implement the prototype as a standalone, lightweight and extensible system. The Eclipse IDE was used to implement most of the system. Simple java sockets and server sockets were used to implement the network portions of the system. Some external JARs and libraries were required for the implementation, namely the sqlite-jdbc JAR for the implementation and Email and Activation JARS to provide access to mail servers to send out e-mail notifications.

The GUI pages were implemented and designed using the Window Builder for the Eclipse IDE. A card layout was implemented in java to hold all the GUI pages in a single frame and navigate through them.

SQL was used to make and maintain the database model and tables. SQLite was chosen to be the database of choice as it compacts the database into a small standalone file which can be packaged with the program.

4.9) <u>Data Structure and Algorithms</u>

Simplicity was the main goal of the implementation and system design. Thus no complex algorithms were used. There were some small data structures used other than the object classes for system functionality, details on these data structures and simple time complexities of some processes are detailed below:

Data Structures

Dom Documents in java were used to hold, parse and read XML data in and out of the system. The XML data and messages were simply converted into singular Strings to be passed between the client and server.

Simple Java ArrayLists were used where collections of objects or other data were necessary; these ArrayLists are maintained and trimmed as required so they never grow too large as to cause system slowdowns.

Java PreparedStatements were used to execute SQL queries to the database in a secure manner. Java ResultSets are used to temporarily hold information retrieved from the database before being packaged into objects or variables.

Algorithms and Time Complexities

The system was designed in such a manner that sorting was not required for most, if not all the processes, thus implementing expensive sorting algorithms were not necessary.

Simple loops (While and For loops) are used to find, extract and delete data from ArrayLists which hold most of the required information needed for functionality. These ArrayLists are trimmed when redundant data is present at certain points in the system. So even though the ArrayList lookups are done in constant time and deletions and insertions require linear time of O(n), the ArrayLists are never allowed to grow too large as to slow down system functionality [10].

Most database lookups are done in linear time of O(n), this did not seem to cause system slowdown even with about fifty thousand test rows inputted into the system [11]. Database joins can be extremely expensive and thus usage of joins is sparse in the system [11]. Currently database joins are only required during system startup where most of the information is cached locally on the client side of the system and if a user chooses to datamine on the given product or service. The database tables themselves were designed to allow for simple joins over a single column.

5) System Functionality

Upon completion of implementation, the testing phase for each feature began, each feature was tested in a myriad of ways to make sure that it represented the functionality proposed in the requirements analysis, and here is a list of tested functionality and results:

- The Server successfully starts listening for connections on a specified port
- The Client is able to connect to the server on a given IP and port, upon which the starting page of the GUI is initialized
- User is able to login with their unique username saved in the database, error message is displayed if username is incorrect or does not exist
- Users are able to search for both service and product Negotiation instances, and the system returns and displays a list of correct results, users are able to view Negotiation Instances by selecting items from the list.
- Users are able to create new product and service negotiation instances which are persisted into the database, they are notified of the created instance's ID.
- Users are able to join viewed Negotiation Instances upon which they take one of 4 participant slots and are able to withdraw at will.
- Creators have full access to all the offers provided by participants in a Negotiation Instance.
- Participants can only view their own offer and the offer requested by the creator.
- Creators are able to successfully chat with all participants in a Negotiation instance, chat logs are successfully trimmed once they get too large.

- Participants can successfully chat with the creator of the instance.
- The lowest/highest offer is always presented to the users of each aspect of Negotiation in the Negotiation instance.
- Users are able to successfully datamine on a given product or service within a Negotiation Instance, where averages are returned of previous transactions conducted on products of services with the exact same name.
- Creators are successfully able to commit to any offer provided by participants and upon doing so all participants and the creator are given a system notification of the commit, the Negotiation Instance is then closed and e-mail notification of the commit are successfully sent to each user's e-mail address.
- All fields are properly populated with their intended values on all GUI pages.
- If a creator withdraws from a Negotiation Instance, the instance is successfully closed and delete. E-mail and on system notifications are sent out to all participants.
- Users are able to successfully view all their previous and new commits. They can retrieve information about these commits upon selecting one from the list.
- Users are able to successfully view their joined and created Negotiation Instances and join them upon selecting them from the list.
- Users are able to successfully retrieve a Negotiation Instance using its ID.
- The system caches most information needed by the user at startup, reducing process times greatly while the system is being used.

Some bugs are however still present in the system and searching for and dealing with them is an ongoing process. The system rarely crashes on both the Server and Client side and seems fairly stable. Most major system bugs that could cause system crashes have been safely dealt with.

5.1) <u>Testing and Performance</u>

All functionality stated above was tested using 6 simultaneous clients on a test database which was populated with about 50,000 – 100,000 thousand test entries in each table. The system did not seem to show any sizeable slowdowns in processing speeds and the server seemed to respond to messages in a timely manner.

All types of messages that are possible with the system were sent back and forth between the server and client. No bugs were found in most cases and most database deadlocks were found and efficiently dealt with.

Tests were also conducted on both home and work networks and yielded the same results.

Most heavy processes in the system such as the data-mining feature, populating lists and caching takes at most O(n) time to complete. Most of the information that forms lists in the system (such as a user’s commits, or their active negotiations) is cached locally during system startup, greatly increasing performance. These lists are updated locally in response to user interactions as the system is run, thus reducing overall load on the server and required messages that need to be passed between the server and client, as well as retrievals from the database.

5.2) **Comparisons with Existing Systems**

Final comparisons were done with functionality in regards to existing systems. We will compare the Multi-Agent E-Negotiation System with SmartSettle One, SmartSettle Infinity, and the 4 "Invite" platforms stated in the "Existing Systems" section above, namely, SimpleNS, Inspire2, Imbins and Imaras [1 – 7].

Firstly and fore mostly, three of the six systems stated above (being SmartSettle One, SimpleNS and Inspire2) are not multi-agent negotiation platforms and are only useable for negotiations between 2 parties at any given time [1, 4 - 5]. The prototype suggested and designed in this paper is more powerful than these three systems simply for the fact that it allows negotiations to be conducted between multiple parties at once. The prototype allows for negotiations between one buyer and many sellers, or one seller and many buyers, thus establishes a one to many relationship in the negotiation process.

SmartSettle infinity is the multi-agent counterpart to SmartSettle One and requires a company quote to be filled out before it can be purchased and used, indicating that it is mostly a tool to be used in business to business negotiations [2]. Both Imaras and Imbins from the Invite set of platforms also boast the power of multi-agent negotiations between one buyer or seller and many buyers and sellers [6 -7]. The prototype is more powerful than these three systems because of the protocol enforcing the negotiation process. Whereas in all three systems (SmartSettle Infinity, Imaras and Imbins) the buyer/seller in control of the negotiation has power over everything and all information is available to them whilst the other participants are not given any information about their competitors [2, 6-7]. The protocol used in the negotiation process in the prototype tries to add a certain competitive element deviating from the protocol used in the three systems stated above, where it reveals to all other participants the highest/lowest offer currently being offered on all aspects of the product/service being negotiated on. Where and who is providing the offer is not available to the participants. This information allows the participants to tailor their negotiation practices to the best current offer provided by other parties while keeping the information as anonymous as possible.

The prototype is also more powerful than all the six other systems discussed in the paper because of its data-mining feature. At any point during the negotiation process a user can, by the press of a button, access statistical information about the product/service being negotiated on from the database. This information encompasses all values being negotiated on and provides averages of previous transactions logged into the system under the same product/service name. This provides some more information to participants about the expected offers of their prices and the current prices offered by other users in the system negotiating on the same product/service, giving them a slight edge in the negotiation process and balancing the power between the buyer and seller and vice versa. This information is also kept completely anonymous providing a sense of greater security to the participants using the system.

6) Conclusion

The end result of the design and implementation of the system proves satisfactory in achieving goals set at the beginning of this paper and their underlying motivations. The Multi Agent E-Negotiation System, delivers all promised functionality and seems to perform well on a small scale; providing a negotiation platform between multiple parties with a unique datamining aspect and a fresh protocol which tries not to hinder the competitive aspect of negotiations. The system prototype leaves many areas for extensions and improvements, and software engineering practices used make it easier to pursue such tasks. Once this system is thoroughly expanded and tweaked to be scalable it should provide a fascinating and unique alternative to E-Commerce and E-Negotiation platforms out in the market today.

6.1) Future Work and Issues

Although the designed prototype provides most if not all the functionality promised in the earlier sections, it is still just a base implementation of a theoretical system which needs to be much larger to provide better functionality and usage before it can be fully utilized. Thus the prototype was designed to be abstracted, reverse engineered and extended in many ways. Some important extensions are explained below:

More information on products and services

Currently, products and services in the system are merely identified by a product/service name. The users are not provided with much information on the product/service being negotiated on. Thus an extension to the system is required where product and services are abstracted into their own objects which provide a lot more information to the users. These objects would need to be logged into the system and saved into the database as more and more types of products and services are introduced. Surveys and research would need to be conducted on what aspects of a product or service users would want information on and the objects in the system as well as the database would need to be updated in response to this information. The system would also need to provide functionality to easily retrieve and display this logged information to the user before they choose to join the negotiation.

User Profiles

Currently users in the system are identified by just their username, e-mail and user ID. Users may wish to add in more information about them in their system, such as locations, their affiliated companies, a user interface page where they could build a store front possibly about product and services they need and are providing. The system will need to be extended to facilitate this extra user information. One way this could be done is to embed the system into a social networking platform. The system could then work in conjunction with said platform and thus pull out user information and profiles to be displayed. Another way to do this would be build an internal user profile system in the system, this would require some changes in architecture and views and an extension to the user object which would store a lot more information to be displayed, users could also be provided with option to upgrade this information at any given time. Moreover User profiles are also hard coded into the database, functionality to allow new users to register and change their information will need to be implemented.

Security

The system simply identifies and uses a username for users to log on to the system. An encryption algorithm could be used to encrypt passwords which the user sets which can then be used for a more secured login to a user account. Messages passed between the client and server can also be encrypted so that they cannot be hijacked or retrieved in anyway.

Datamining

The datamining feature in the system, though functional, is implemented in an extremely simple manner. When used, it merely retrieves averages of previous values obtained from transactions logged into the system database. This important feature can be extended in a myriad of ways. Statistical algorithms can be implemented to fine tune the datamining feature so as to avoid outliers. The datamining features also retrieves information simply using the exact name of the product/service negotiated on, similarity algorithms can be implemented so as to expand the set of data used to find and calculate information provided by this feature. The datamining feature can also be extended to interface with large online E-Commerce databases to retrieve world market prices on negotiated products and services.

Stress Testing

Even though most of the features in the system have been thoroughly tested, resources were not available to do large stress tests on the system. Stress tests with many users would need to be conducted on the system to check whether it is properly scalable, and if it isn't to change algorithms to support scalability.

Going Mobile

The system is lightweight enough that it can be ported as a mobile application to work with and interface with smartphones currently on the market. Since the system is implemented in java it can easily be ported as an Android application which mainly uses java. This will however take a revamping of the entire user interface to provide a comfortable environment for the users to interact with the system using their phones. The system can also be ported to work with other smartphone types such as the IPhone by Apple or a Windows based phone though architecture changes would be required for a successful port.

Web Application

Even though the system was designed as a standalone java application over a network due to resources and capabilities of the implementer, the main demand for such a negotiation system is on the World Wide Web. Thus porting the system to be a web application or web page will dramatically increase the potential of the system as well as quickly increase its user base. Some cross platform problems that could appear with the java implementation could also be avoided in this way.

7) REFERENCES

[1] SmartSettle One – Online Dispute and Negotiation System. SmartSettle One.

Retrieved July , 15th , 2013, from

http://www.smartsettle.com/home/products/smartsettle-one/

[2] SmartSettle Infinity – The World's Most Advanced Negotiation System. SmartSettle Infinity

Retrieved July, 15th, 2013, from

http://www.smartsettle.com/home/products/smartsettle-infinity/

[3] Invite Negotiation Systems. J. Molson School Of Business, University Of Concordia (2005).

Retrieved June, 15th, 2013, from

http://invite.concordia.ca/

[4] SimpleNS Negotiation Handout. Invite Negotiation Systems (2005).

Retrieved June, 17th, 2013, from

http://invite.concordia.ca/simplens/handout.html

[5] Inspire Negotiation Handout. Invite Negotiation Systems (2005).

Retrieved June, 17th, 2013, from

http://invite.concordia.ca/inspire/handout.html

[6] Imbins Handout. Invite Negotiation Systems (2005).

Retrieved June, 18th, 2013, from

http://invite.concordia.ca/imbins/handout.html

[7] Imaras Handout. Invite Negotiation Systems (2005).

Retrieved June, 19th, 2013, from

http://invite.concordia.ca/imaras/handout.html

[8] Observer Pattern. Wikipedia. Retrieved July, 30th, 2013, from

http://en.wikipedia.org/wiki/Observer_pattern

[9] Façade Pattern. Wikipedia. Retrieved July, 30th, 2013, from

http://en.wikipedia.org/wiki/Facade_pattern

[10] Class ArrayList<E> . Oracle Java Documentation (2013).

Retrieved August, 3rd, 2013, from

http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html

[11] Databases. Wikipedia. Retrieved August, 4th , 2013, from

http://en.wikipedia.org/wiki/Database

[12] Jack Loechner. E-Commerce Driving To 170 Bllion This Year. Research Brief (2006).

Retrieved August, 10th, 2013, from

http://www.mediapost.com/publications/article/51357/#axzz2bv0kv59M