Performance and Functional Enhancement of Artificial Market Psychology Simulator

By
MOHAN KUMAR AKULA
B. Tech, Civil Engineering, Institute of Technology, Banaras Hindu University, 2000
M. S, Civil and Environmental Engineering, University of Missouri, 2003

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN CIVIL AND ENVIRONMENTAL ENGINEERING
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Author.....
Department of Civil and Environmental Engineering
May 14, 2004

Certified by.....
Andrew W. Lo
Harris and Harris Group Professor of Finance
Thesis Supervisor

Certified by.....
Dmitry Repin
Postdoctoral Associate, Laboratory for Financial Engineering
Thesis Supervisor

Certified by.....
George Kocur
Senior Lecturer of Civil and Environmental Engineering
Thesis Advisor

Accepted by.....
Heidi Nepf
Chairman, Committee for Graduate Students

Performance and Functional Enhancement of Artificial Market Psychology Simulator

By
MOHAN KUMAR AKULA

Submitted to the Department of Civil and Environmental Engineering
on May 14, 2004, in partial fulfillment of the requirements for the degree of
MASTER OF ENGINEERING IN CIVIL AND ENVIRONMENTAL ENGINEERING

Abstract

Artificial Market Psychology Simulator (AMPS) is an adaptive and programmable simulation system designed to assist researchers in the study of psychology of securities traders. It forms a part of the larger system that includes the MIT Web Market – a double-auction engine for securities trading and RStudio – a physiology data acquisition software. The AMPS system controls price patterns and induces market events in an adaptive fashion.

This research improves the performance and functioning of AMPS by generating realistic market scenarios in response to the trading subject's physiological feedback during the simulation process. Despite the subject's knowledge of the simulated system, AMPS generates market scenarios such that the trading subject can attach credibility to these events and respond accordingly. The adaptive nature of AMPS can be attributed to its ability to adjust market response based on observed physiological characteristics of the research subject. Another improvement is the integration of AMPS with RStudio. The simulation system uses real-time market and physiology data provided by the MIT Web Market and RStudio respectively. A prototypical scenario that represents a research subject's typical emotional states in response to market volatility during the simulation process has also been modeled.

One specific software challenge is the integration of RStudio – written in C++, and AMPS – written in JAVA. Another challenge is that AMPS system needs to be modified to reflect different rule sets for different class of users based on their physiological response. The logging system for errors and events has been improved while the error and exception handling has been structured and streamlined as well.

Thesis Supervisor: Andrew W. Lo
Title: Harris & Harris Group Professor of Finance

Thesis Supervisor: Dmitry Repin
Title: Postdoctoral Associate, Laboratory for Financial Engineering

Thesis Advisor: George Kocur
Title: Senior Lecturer of Civil and Environmental Engineering

Acknowledgements

I dedicate this thesis to my parents for their love, inspiration and guidance during my entire academic career; my brother for always being supportive and encouraging and finally my fiancée for being very patient and understanding with me during my graduate study while enduring many months of separation.

This thesis is a culmination of my efforts and the guidance and support of several people at MIT. I extend my sincere gratitude to:

Prof. Andrew Lo for providing me the wonderful opportunity to work on this research project and agreeing to supervise my thesis.

Dr. Dmitry Repin for constantly guiding me through all the critical phases of the project through his insightful feedback. His invaluable guidance and support in understanding the functional requirements and design of the system were critical to the success of the project.

Ms. Svetlana Sussman at the Lab for Financial Engineering for crucial administrative support and encouragement.

Dr. George Kocur for his perceptive guidance in walking me through the technological challenges I encountered.

Dr. Eric Adams, Prof. John Williams & Prof. Gerbrand Ceder for helping me throughout my stay at MIT.

Lawrence Wang and Eric Ho for the remote support they provided in understanding AMPS and RStudio respectively.

Adlar Kim for helping me understand with the MIT Web Market. Dr. Itzhak "Gingi" Aharon for visualizing the prototypical scenario of emotions.

Finally, friends and colleagues at MIT whose support and encouragement were invaluable.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Research Overview

There are two schools of thought among financial researchers with differing views on the rationality of investors. One believes in the Efficient Market Hypothesis and is unflagging on the view that investors are rational provided that the information dissemination is efficient. In such a market, there exists no ambiguity on the price of the securities and the market can almost instantaneously move the price of securities to its perceived economic value. One variant to this belief is the Random Walk Hypothesis which states that the securities prices follow a random pattern and it is impossible to use historical prices to predict future prices. Researchers in behavioral finance, who have been critics of the above belief, argue that there are predictable biases in the responses of investors to what is happening in the markets and that their responses are not always rational from the traditional economic perspective. The irrationality of these responses is mostly attributed to psychological factors that are beyond the domain of fundamental economic value.

While much of the earlier research has provided a qualitative picture of the relationship between decision making process and psychology of investors, the research at the MIT Lab for Financial Engineering (LFE) headed by Prof. Andrew Lo and Dr. Dmitry Repin is engaged in systematically investigating this relationship. A study conducted at the LFE earlier used securities traders as research subjects because

of the real-time nature of their decision making in scenarios involving risk and the inherent transient nature of their psychological responses. The study showed that there was a significant statistical correlation between physiological responses that may be manifestations of emotional states and market events.

1.2. Goal of the Thesis

To further strengthen the above argument, a research project at LFE is investigating this link through a methodical simulation process. Artificial Market Psychology Simulator (AMPS), implemented by Lawrence Wang, is a product of the research at the LFE that provides the investigator with the basic framework for real-time monitoring of trading activity of a selected human trader. RStudio, implemented by Eric Ho, is another product that is used for real-time physiological data collection, analysis and monitoring. The aim of this research is to combine the capabilities of these two systems to work together as an adaptive simulation system that can facilitate controlled market movements with the help of AMPS in response to real-time monitoring of the trading subject's emotional states, using RStudio.

During a typical simulation experiment, the researcher can observe and record the trader's decision making process as his/her response to specific market events. These market events are simulated by this tool incorporating physiology information reflecting the trader's transient emotional and cognitive states that are collected and analyzed in real time. A prototypical scenario will involve monitoring of physiological variables and specific market events (for example, a sharp rise or decline of the

security price) will be generated if physiology values exhibit a predefined pattern (for example, muscle tension going above a certain threshold). In other words, in such a scenario, the magnitude and type of market movement is determined by the trading activity and the physiological state of the research subject. These market movements may in turn elicit another physiological response from the research subject and thus the simulation progresses.

Due to the adaptive nature of this tool, it may be used for training inexperienced traders as well as for monitoring a group of traders. This will involve allowing traders to practice under the settings of a prototypical scenario described above. This scenario serves as a basic framework for inexperienced traders to learn about the trading process. This simulation system can also be tailored to provide alerts to a trading supervisor in a trading environment.

1.3. Requirements Overview

The first version of AMPS, eponymous with its name, provides several promised features including the basic design framework, process outline, response generation sequence among many other characteristics required in a simulation setting. RStudio also provided the basic features needed for independent monitoring of real time physiology data. However, the two applications did not completely address several important requirements that include the following:

1. Implementation of integration modules for AMPS and RStudio as represented in the overall architecture.

2. Generation of AMPS market response in proportion to size of the trader's current order;
3. Generation of AMPS market response as a feedback to the current physiological states of the trader because the applications were not integrated;
4. Generation of AMPS responses that seem indistinguishable compared to actual trading environment;
5. Generation of relevant test-case scenarios for testing of the simulation system

The above-mentioned characteristics are the ones that require utmost attention at this stage of the project and require a clear understanding of the entire setup giving rise to the following set of requirements

1.3.1. Data synchronization requirements

The AMPS and RStudio are designed to work in unison such that the simulation uses the physiology data and trading data for creating market moving events that are generated in real time. The fleeting nature of the physiological responses bears special attention indicating the need for a real-time nature of the application. It must be noted that the different physiological responses under study do not necessarily have the same response time. This data must be collected and analyzed by RStudio, calibrated and transferred to AMPS.

1.3.2. Data Calibration Requirements

The sampling rates as well as the response parameters for each of the physiological variables collected by RStudio are different. Although there is a generic range associated with the level of response for each of the variables like temperature, skin conductance and electromyographic signals that reflect muscle activity, these levels differ from person to person. The application must also be able to capture the baseline physiological state to scale these variables. There must be a method for standardizing and calibrating these variables so that they can be interpreted by the AMPS.

1.3.3. Order Parameters Computation Requirements

The physiology data interpreted by AMPS is used to generate market events that correspond to the expected emotional state of the research subject. AMPS must respond to changes in the physiology variables by creating market events that depend not only on the current physiological state but also the trading activity of the research subject. AMPS creates these market events by connecting to MIT Web Market as a "trader" and submitting large enough orders to move the market in the desired direction. The exact size and nature of these orders must be driven by the two types of monitored variables, namely physiology data and trading data.

1.3.4. AMPS and RStudio Integration Requirements

AMPS is written in JAVA and RStudio is written in VC++. This poses a technological challenge of integrating the two applications considering that both are multi-threaded real time applications. The primary aim of this integration would be a seamless data transfer where both applications could work independently as well. Several alternatives for integrating inter-process and inter-language applications must be considered before choosing the one appropriate for our application.

1.3.5. Relevant Test Case Scenario Requirements

The design of a prototypical scenario is essential to testing the application against the backdrop of a real trading scenario. The scenario must incorporate typical emotional fluctuations experienced by a trader in a real trading scenario.

1.4. Solution Methodology

The scope of the project covers the development lifecycle of the second version of the AMPS prototype. The basic steps of a software development lifecycle provided a useful guideline to progress in the system enhancement process. However, an iterative process of development with constant feedback on the requirements and methodology considering time and effort constraints was followed.

1.4.1. Requirements Gathering and Specifications

Since this effort required building over existing components, the overall architectural framework and high-level design features were made available. Based on research of

the earlier work done and interactions with the research staff at the Lab for Financial Engineering, a detailed description of the existing problems and expected improvements in the application was documented as described above. Modifications and additions to the existing design were presented in the form of process flow diagrams and use-case scenarios. A general idea of prioritizing the critical features, level of effort and timeline was established being well aware of the iterative development process.

1.4.2. Iterative Development

Addition of a new developer to any software development project entails cost in the form of time and effort to learn and understand the features of the existing system in great detail to the extent he/she can modify it. The iterative development was approached in the learning mode until sufficient expertise of the application was obtained. Enhancement of features was identified by challenging the existing design through code review and iterative testing process. The additional features were established by identifying the need and scope for glue code as well as modification of existing code.

1.4.3. Testing

The development effort must be constantly tested for conformity with the overall architecture and the requirements specifications. The iterative nature of the

development process and the need for building a prototypical scenario serve as helpful guidelines in the overall testing process.

1.5. Conventions

The following typographic conventions and terminology is used in this thesis to describe sections of code and the role of participants in the experiment respectively. It may be noted that the writing convention is very similar to Wang's [1] description.

- `Constant Width` type is followed for Java and C++ code, with language keywords italicized in the same font.
- `Constant Width` type is used for grammar and configuration file contents.
- *`Constant Width italicized`* type is used for filenames and descriptive expressions like filenames, in variable names, and in values. The practice of enclosing between '<' and '>' is reserved for descriptive expressions. For instance, SESSION<*number*> is used to represent SESSION1 or SESSION2.
- **`Constant Width bold`** is used for command line input, variable names, variable values and other symbols. When these expressions appear inline with the body text, they are enclosed in single quotes (') for clarity.

The terminology used in this thesis is described below to participants and simulations states:

- **Simulation or Experiment**: These terms refer to an instance of running the code for AMPS, RStudio, MIT Web Market, Machine and Informed Traders

by involving a person to act as a subject for the experiment and another person overseeing the running of the experiment.

- **Research Subject or Trading Subject**: These terms represent the trader or the subject participating in this simulation experiment.
- **Trader**: This term refers to human or simulated traders that can connect to the MIT Web Market as clients and perform trading transactions.
- **Financial Researcher, Research Administrator, User or End User**: This person oversees the running of the experiment.
- **Developer**: Person or group involved in the implementation of the code for AMPS, RStudio, MIT Web Market, Machine and Informed Traders.
- **Time Step and Clock Tick**: This term represents one increment of the internal timer thread in AMPS during the experiment.
- **Runtime**: This term refers to the time when the experiment is being performed or the simulation is in progress.

1.6. Thesis Outline

The thesis is divided into three sections as described below. The first section in Chapter 2 and 3 describes the earlier research done, the basic experimental setup and the design level specifications of the existing code base and usage model specifications. Chapter 4 describes the enhancements to the AMPS and RStudio system including the integration framework and the relevant test-case scenario in.

Chapter 5 describes the scope for future work to improve the working of the experimental system. Chapter 6 presents a brief discussion on the current version of AMPS and challenges faced in its implementing changes to the existing system

Chapter 2

Background

The primary aim of this research is to provide the behavioral financial researchers with the software tools necessary to understand the relationship between the physiological factors and financial decision making in a systematic and adaptive setting of a simulated environment. This is accomplished by enhancing existing components of AMPS and RStudio. This chapter provides the basic framework for understanding of the overall simulation setup which includes MIT Web Market, AMPS, RStudio and Machine and Informed traders.

2.1. MIT Web Market

The MIT Web Market is an electronic market developed in collaboration with the Artificial Intelligence Laboratory and the Center for Biological and Computation Learning at MIT. It is a web-based system where several simulated and human traders can connect to a central server as clients. The central server incorporates a double-auction engine and serves as a market platform for securities trading. When several traders are connected to the system and constantly transacting with the market, the simulated reflects the dynamics of a trading market. The application, written in JAVA, provides the server and the client as applet-based user interfaces. The connecting machine traders are allowed to access the built-in functions to continuously trade therefore creating trading activity noise in the market. The market-

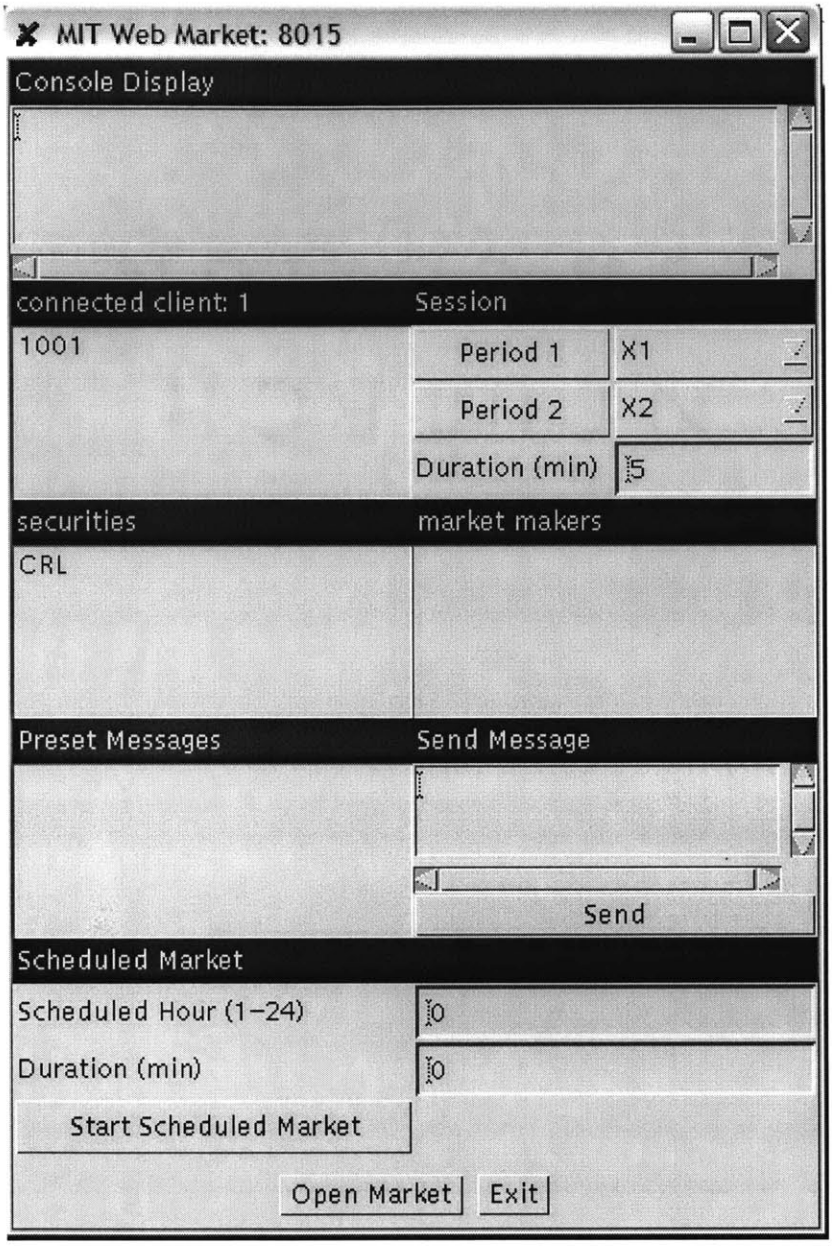

Figure 2-1: User Interface for MIT Web Market

making functionality of this market is helpful in automating trade submission activity of a simulated trader. The market making functionality gives the research subject or automated traders the ability to submit trades with certain rules on its own clients for transactions placed through the market. Typically, these rules include limit/market

buy orders or limit/market sell orders and canceling order before order execution. It must be noted that since the MIT Web Market was enhanced at the same time that AMPS was developed, the version of the market integrated with AMPS and the machine traders is not the latest. The MIT Web Market logs all activity to the database which can be accessed by other components in the simulation system that act as market makers. Figure 2.1 shows the user interface for the MIT web server and the client applet.

2.2. Artificial Market Psychology Simulator or AMPS

AMPS, developed by Lawrence Wang at the Lab for Financial Engineering, is a software application for financial researchers for monitoring the trading activity as

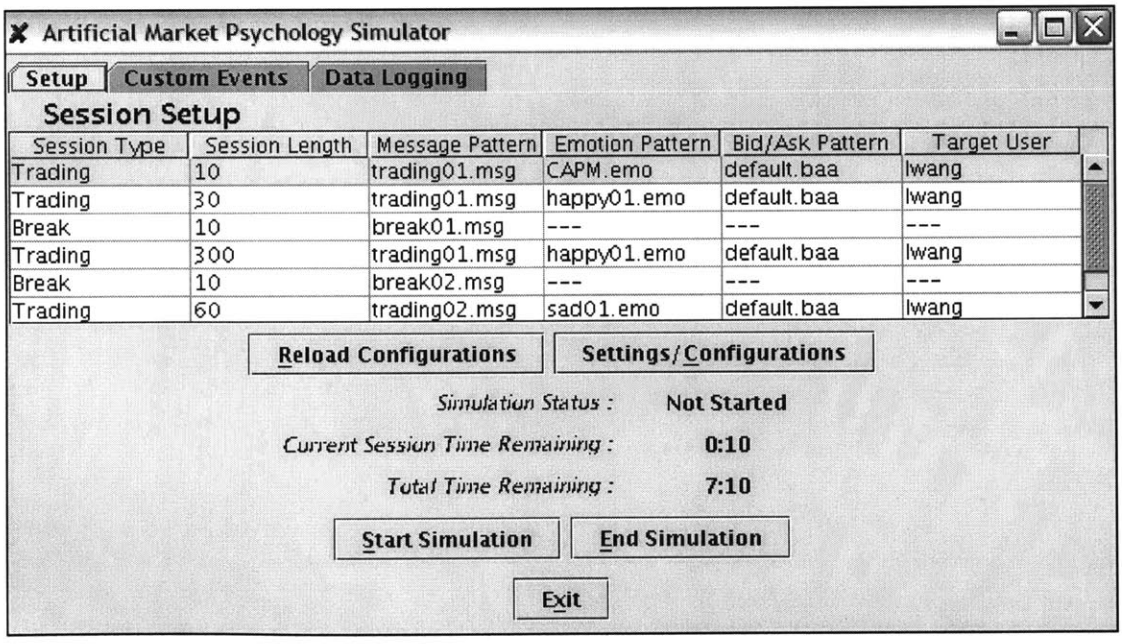

Figure 2-2: User Interface of AMPS Client

well as the physiological data of a selected research subject in real time and also simulating controlled market movements. The tool must have an adaptive capability to create market events as a response not only to the monitored trading activity variables like type and volume of the current transaction but also the constantly updated physiology variables like skin conductance, temperature, and electromyographic signals.

The AMPS connects to the MIT Web Market as a client and has access to the MIT Web Market database. The AMPS operates with MIT Web Market and RStudio for monitoring and response generation. The data monitoring is done by constantly querying Web Market database for trading activity and RStudio for physiology data. The response from AMPS is in the form of market-moving trades within a certain time-horizon, which is a function of these two monitored variables, namely trading activity and physiology data.

This thesis addresses certain functional and performance issues with the earlier version of AMPS. The issues include, but are not limited to, unfinished integration with RStudio, the nature and size of trade submitted being independent of the subject's trading activity and physiological state, inadequate control of market noise and no presence of a relevant test-case scenario. These are explained in greater detail later.

AMPS uses several features of object oriented programming (JAVA) including multi-threading, networking and database access. Figure 2.2 shows the user interface for AMPS.

2.3. Real Time Studio or RStudio

RStudio, developed by Eric Ho at the Lab for Financial Engineering, is a real time physiology data collection software package. The software package can independently collect the physiology data, analyze and relay the analyzed data over a network.

The activity of the autonomous nervous system (ANS) can be measured in terms of its responses which can be detected by measuring the physiological variables. The collected variables include the skin conductance response (SCR), blood volume pulse (BVP), heart rate (HR), electromyographical signals (EMG), respiration (RES) and body temperature (TEMP). These variables are measured using portable biofeedback equipment – ProComp data acquisition unit - by placing sensors at various points on the body of the research subject. Optical fibers are used to transfer data from ProComp unit to the serial port of a computer. RStudio collects this data, scales it down from a frequency of 256 Hz to about 30 Hz depending on the channel frequency. This data is then analyzed and made available for monitoring and network access. The user interface of RStudio is designed for graphically viewing data variation with time, saving and logging data and connecting to a network server - RStudioServer – that can relay data across the network. The RStudio system also provides for RStudioMonitor, which is an independent monitoring application that can be trained to identify the baseline physiology data patterns for a research subject.

It must be noted that RStudio is an MFC VC++ application written using Visual Studio 6.0. Despite RStudio's claims of network connectivity and data relay, it is of little use to AMPS because the network implementation works for other C or

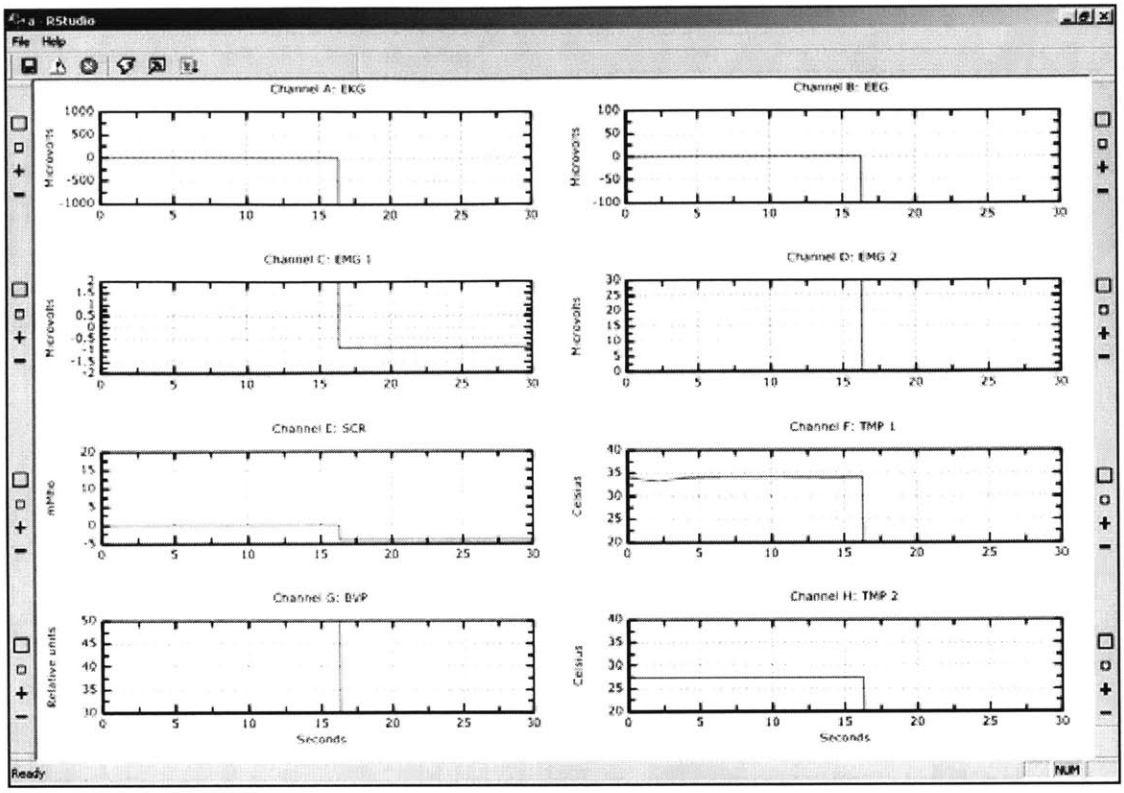

Figure 2-3: User Interface of RStudio Client

VC++ applications only. It must be modified in a way such that it may be accessed by AMPS, which is a JAVA application. Moreover, the data it could relay across the network cannot be easily interpreted by AMPS. Figure 2.3 shows the user interface for RStudio.

2.4. Machine and Informed Traders

This is a set of automated traders written in JAVA that can connect to the MIT Web Market as traders. These traders have access to the MIT Web Market database and can act as market makers. The distinction between these two types of traders lies in their manner of determining the price of an order before submitting it to the market.

The machine traders query the database for the best available orders on the market and randomly decide on placing limit buy, limit sell, buy or sell orders. The informed traders follow a random price pattern in placing their orders with the market. This random price pattern generation is indicative of the market efficiency and it can be supposed that these informed traders have their own information which guides their judgment of prices. While each informed trader, guided by its own information of prices, contributes to market noise, the machine traders constantly follow the market in the direction of best available prices. Though the machine and informed traders arrive at prices in a different manner, they decide on the kind of order to be placed in a similar manner, that is, by placing orders randomly. The machine and informed traders are the necessary components of the simulation system and guide the market price change.

Chapter 3

Design Specifications & Usage

This chapter presents the design changes and the specific instructions to use and modify the configurations of the system. This chapter also serves as a useful addendum to the design described by Lawrence Wang and Eric Ho to understand the overall architecture of the entire system when AMPS and RStudio are made to work together.

3.1. Changes to Architecture

The three types of components used to build the architecture of the system are user interface components, data components and business logic components [1]. These major components are illustrated in the system architecture shown in Figure 3.1. The figure also indicates the changes that were applied.

The Web Market provides two user interfaces (or simply UI) – MIT Web Market Server UI and Trader UI. The Trader UI is of interest in this architecture because of direct interaction with the trader in collecting market making activity. The simulation UI of AMPS, used by the financial researcher, has been slightly modified to reflect the presence of RStudio CPP Client and its connection to RStudio as a data channel.

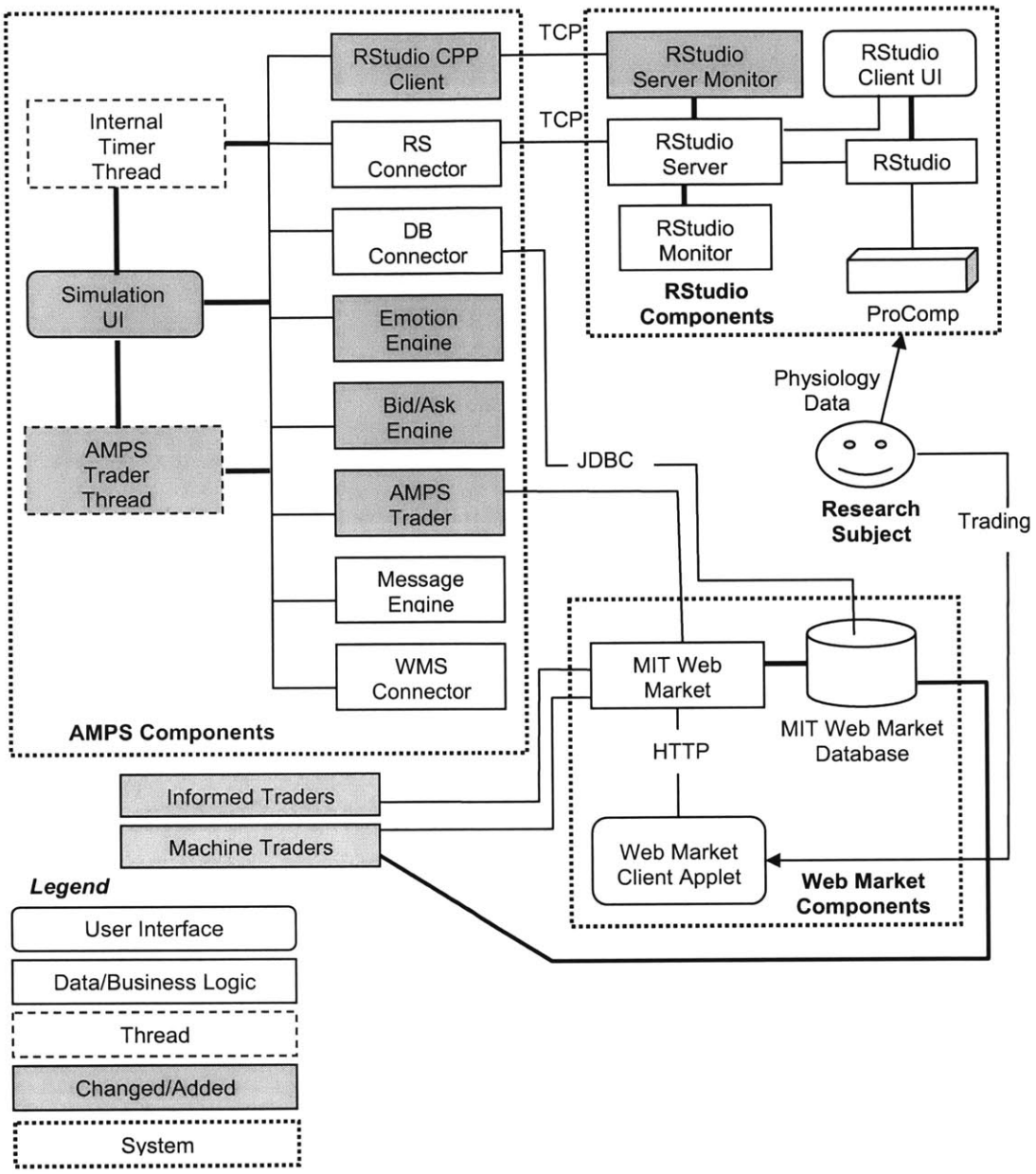

Figure 3-1: AMPS and RStudio Component Architecture

The system is coordinated using several business and data components as reflected in the figure and described by Wang[1] and Ho[2]. Several coordinated internal changes were made in the Emotion engine and Bid/Ask engine of AMPS to

satisfy order parameters computation requirements described in chapter one. The AMPS order parameters, as described earlier, must be adaptive in nature.

The notable changes are the addition of "RStudio CPP Client" to AMPS and "RStudio Server Monitor" to RStudio. While RS Connector is used to send timestamps from AMPS and RStudio, the channel is not configured to transfer data back and forth. These two components help to establish inter-process data communication channel between RStudio and AMPS. RStudio CPP Client is a java based client for C++ applications built into AMPS. RStudio Server Monitor is a C++ application that acts both as a server and as a client built into RStudio system. While this component acts as a client to the RStudio server as a variation of RStudio Monitor, it also acts as a server for the JAVA-based RStudio CPP Client.

The fluctuation of share price observed in the market due to constant trading by traders connected to the simulated market is termed as market noise. Machine traders and informed traders have been slightly modified to control the market noise parameters. The simulation model is very sensitive to market noise and choosing an appropriate value was necessary to the success of the simulation. Wang[1] also describes low-level data structures like "Session" and "Rule-Sets" that were used for modularity. These data structures have also been modified to accommodate changes in Emotion and Bid/Ask engines.

3.2. Changes to Process and Data Flow

Figure 3.2 shows the modified process and data flow diagram. The internal timer thread runs the simulation session by invoking different business logic components in a sequential order during the simulation. Earlier, this thread invoked the DB Connector to retrieve the latest order placed by the research subject. This real time market data is sent to the Emotion Engine and this data is used for computation of

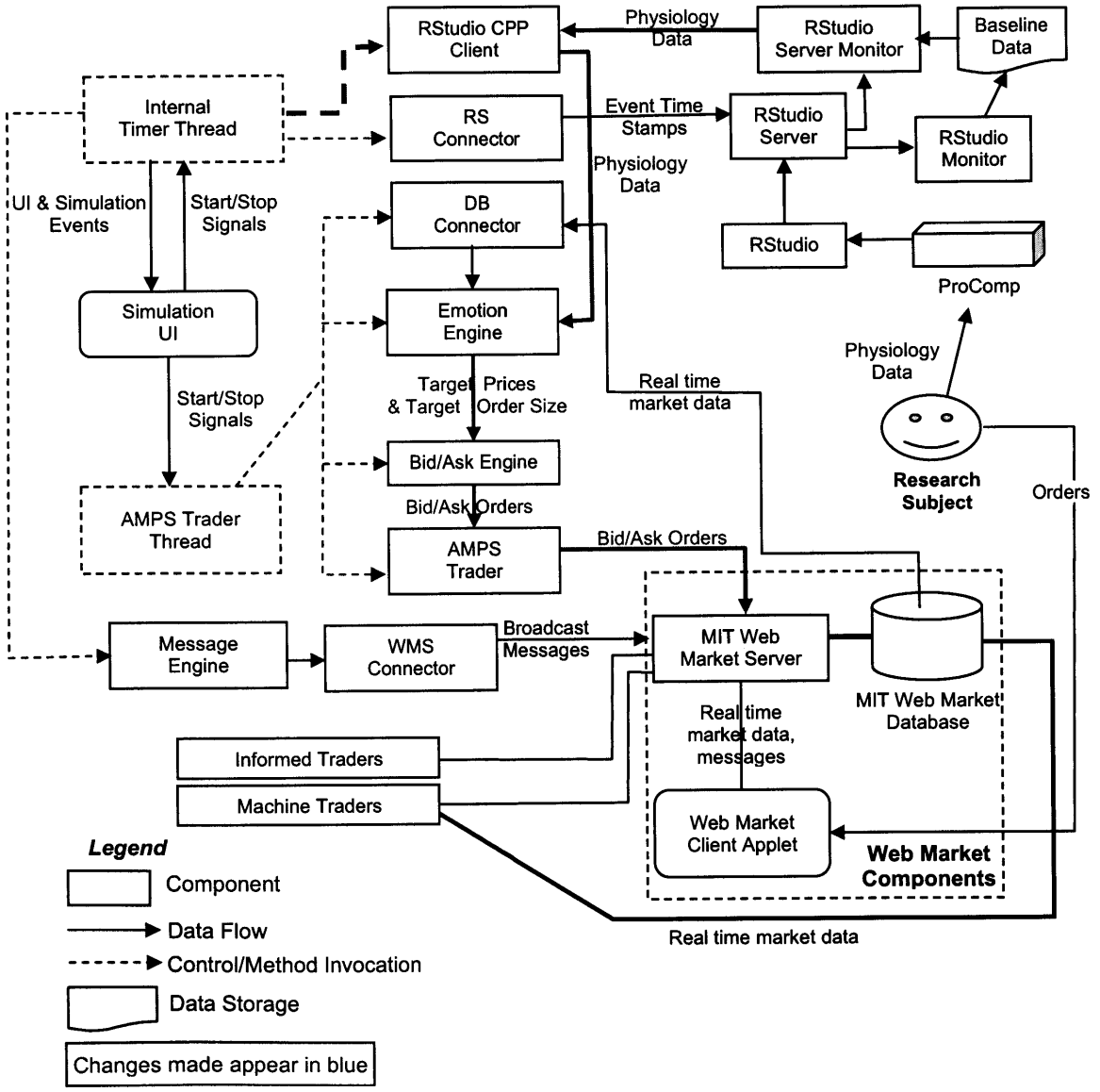

Figure 3-2: Simulation Process and Data Flow Diagram

target prices based *only* on the real-time market data. Currently, this process has been modified such that the physiology data is also considered in determining the target price.

The internal timer thread is modified to initialize the RStudio CPP Client and collect physiology data apart from invoking several other business logic components. The collected physiology data is used by the Emotion engine to derive or interpret the physiological state of the research subject. The target order price and size are then determined by the Emotion Engine and sent to the Bid/Ask Engine. The target price and size are such that they can affect the market price; and the direction of market movement is decided based on the underlying rule-sets of the current session. The Bid/Ask Engine then generates the bid and ask prices by establishing a randomly determined spread around the target price. These final bid/ask orders are then submitted to the market.

Changes have also been made to the underlying rule sets and the manner in which sessions are accessed. Currently, the use of multiple sessions during a simulation has been relegated because it was found that the glue code in the JEP parser scripting language could be utilized to mimic different sessions in a single generic session. This generic session could accommodate all the differentiating characteristics of each emotion session without loss of generality.

The next chapter describes specific enhancements applied to AMPS and RStudio components and verifications of these changes on the simulation. However,

a brief incursion to describe how to use and modify the initial configurations of the simulation is presented below.

3.3. System Usage Description

This description on usage and modification of system configurations can be taken as an addendum to a similar discussion by Wang [1]. A brief summary of the general system usage is followed by maintenance details not documented earlier.

3.3.1. General Usage Model

The general usage model is broken down into three kinds of tasks to be taken care of by the research administrator. Prior to the simulation, the researcher has to set up the configuration files. One important step is to initialize the share price, the number of shares and the initial cash to be given to the trader at the beginning of the simulation. This is explained in greater detail later in this chapter. The AMPS configuration files can be edited using a text editor or the configuration editor in AMPS. The reference to these configuration files can be changed in the `SimUI.java` file.

Filename	Location	Tasks
`dbadmin.sh`	~/afm/rst/market/	1. Shell script to run commands for database administration. Also includes command for setting the initial share price and initial number of shares.
`amps.ini`	~/afm/rst/amps/config	1. Setting folder path for emotion & Bid/Ask files, message & RStudio log files

		2. Configuration of simulation sessions 3. Setting up RStudio connection settings 4. Setting up database connection settings
<emotion>.emo *<emotion>.ini*	See *amps.ini*	1. Set up the .ini file for configuration of the emotion variables 2. Set up the .emo file for the runtime configurations of the emotion file.
<message>.msg	See *amps.ini*	1. Set up and configure the message files
<trader>.dat	~/afm/rst/trader	1. Configuration file for market and informed traders

Table 3.1. Configuration files for AMPS simulation

During the simulation run, the administrator has to exercise controls and precautions to check for the smooth progress of the experiment. The administrator has to check that there are no network routing problems to ensure that AMPS connects with RStudio. An easy check is to telnet to the *<ip address>* and *<port>* of the computer on which RStudio is running and listening from the computer on which AMPS is running to ensure that there are no routing problems. The administrator also has to reload configuration files in case there is a change in any of these files. Please note that these configuration files can be reloaded only before the simulation is started. The post-simulation processing will involve the analysis of the log files and cleanup of the database files.

3.3.2. Summary of Startup & Exit Instructions

3.3.2.1. Running MIT Web Market

Before the experiment is run, the very first step for the administrator is to determine the initial share price, number of shares and cash to be given to traders. This is a three-step process. The first step is to empty or clean the database and change the initial cash and the number of shares in the database. This step is necessary because every time a trader connects to the market, it uses the values in the database to assign initial number of shares and initial cash to the connecting trader. The administrator is provided with a convenient functionality to act as a database administrator. The shell script used to access this functionality in **csh** environment is given below

$>./dbadmin.sh *<server name>* *<database prefix>* *<initial cash>* *<initial # of shares>*

$> initcrl

$> quit

While the ***<initial cash>*** and ***<initial # of shares>*** were variable, the following settings applied during the development process.

<server name>	*Wang*
<database prefix>	*Testmarket*

These sets of commands are sufficient for the database to be cleaned up and reconfigured with the given values. It was observed that the time taken to execute the **'initcrl'** command is proportional to the number of rows in the database tables. AMPS and machine traders constantly query the database for the subject trader's latest trades and the best available trades respectively. Database cleanup before each run of the simulation experiment ensures that the queries do not return illegal values,

in the form of trades submitted during previous experiments. In order to ensure that the commands have emptied the database, a quick check may be performed using the following commands.

```
$> bash
$> . oraenv
$> ?>OMKT
$> sqlplus
```

At the sqlplus prompt, enter the following query

```
> select count(*) from <database prefix>_orders;
```

where ***<database prefix>*** is the same value that was used in the database administration command earlier. This command must return 0 if successful.

Once the database was ready, the server may be started by running the following shell scripts in the **csh** environment on a UNIX platform.

```
$> csh
$> cda
$> ./runserver.sh
```

Internally, this shell script runs the java file using the following command:

```
$> java rst.market.ServerMain jdbc:<subprotocol>:<subname> <jdbc
driver> <db username> <db user password> <database prefix>
<market port>
```

This assumes that the environment variables and paths are correctly set. The market applet loads and can accept connections from machine and human traders. The trading simulation cannot start until the market is opened. The market can be opened clicking the "Open Market" button. After the simulation, the market must be closed clicking the "Close Market" button and then close the market using the "Exit Market" button. Since the MIT Web Market is a multi-threaded application, it is

necessary that all the threads be stopped, the port be freed and the application be closed gracefully using the above-mentioned buttons. As a caveat, it must be remembered that forcefully closing the application may leave the port open and not stop all the threads.

3.3.2.2. Running AMPS

Once the MIT Web Market is run and the steps required prior to simulation are performed, AMPS is ready to be run. One should be aware of the values for the username of the trading subject and the initial share price. If these values are changed at any time, the configuration file *amps.ini* and the *<emotion>.emo* should be changed accordingly. A convenient shell script is made available to run AMPS. By running the following shell script in the **csh** environment on a UNIX platform, AMPS can be started:

```
$> csh
$> cda
$> ./runamps.sh
```

Internally, this shell script runs the java file using the following command:

```
$> java rst.amps.AMPSMain <server name> <port number>
```

The AMPS window appears and connects to the MIT Web Market. At this point, clicking the "Start Simulation" button will begin the experiment. The administrator must ensure that the research subject is also connected to the market. Starting the experiment before the research subject connects will only reduce the available time allotted in the current session and increase the data logging overhead. The

administrator must also connect with RStudio components to collect physiology data and must ensure that RStudioMonitorServer is ready to accept connections before attempting to connect. At each time step, AMPS queries the MIT Web Market database for latest order parameters submitted by the research subject and gets the latest available physiology data and analyzes these two data sets using available rule-sets. The result of this analysis is used to determine the next market moving trade to be submitted by AMPS. After the end of simulation, click "Exit" button or close the application to exit. Exiting AMPS when the simulation is running will close the session and exit AMPS.

3.3.2.3. Running Machine and Informed Traders

The administrator must configure the machine traders before the experiment is run. Since the machine and informed traders control the market noise, it is necessary that the initial orders from machine traders reflect this initial share price. The configuration file for machine traders - *<trader>.dat* – must be edited to include the initial share price.

If the initial share price is $X, then the price is reflected as $X * 16, because

```
#Editing <trader>.dat
# price processes
# InitPrice interArrivalTime  mu      sigma
400          15000            0.001   0.001
380          15000            0.0001  0.0025
# special character "-" to indicate the end of  price processes
```

Figure 3-3: Editing the *<trader>.dat* file

the price is reflected in 32nds. For example, a share price of $10 would be reflected as $10*16 = 160. The value to be to be edited in *<trader>.dat* is shown below: A value of 400 in the figure above represents $25 since 400 = $25*16. This value must be replaced with 160 if the initial share price is set to $10.

The machine and informed traders can be initialized running the shell script - *populateMarket.sh*. The shell script is in the same directory as the *<trader>.dat*. Internally, this shell script runs the java file using the following command:

java rst.trader.SimOrderFlow <server name> <trader>.dat <market port>

The machine and informed traders can be stopped from trading using the Ctrl+C keys.

3.3.2.4. Running RStudio system

Before the RStudio application is started, the administrator must ensure that the research subject is prepared for the experiment. The research subject must be totally unaware of the internal workings of this experiment so as to not affect his trading characteristics. Collection of physiology data feeds must be verified using RStudio.

RStudio is a VC++ MFC application. `RStudio.exe` can be used to start the application. The server application for C++ based clients - RStudioServer can then be started using `RStudioServer.exe`. Then RStudio must connect to RStudioServer to send data feeds about the trading subject's physiology. At this stage,

RStudioServer is ready to send these physiology data feeds to any client connecting as RStudioMonitor.

The next step is to start the RStudioMonitor (`RStudioMonitor.exe`) and connect to the RStudioServer to collect data feeds transmitted. It is necessary to collect the values or physiology data during minimal autonomic activity for about five minutes for the purpose of establishing the baseline.

When the baseline values are available, RStudioMonitorServer – an application that acts as a client for RStudio and server for AMPS - must be started. It immediately connects to RStudio. Since RStudioMonitorServer mimics RStudioMonitor in its connection with RStudioServer, it gets data feeds similar to RStudioMonitor. It also waits for a client connection from AMPS. This application analyzes data packets and calibrates the data before transmitting it to AMPS. When the AMPS connection is established, the calibrated data is transmitted to AMPS. The application can be closed similar to any Windows application.

Chapter 4

System Enhancements

This chapter details the implementation of improvements and rectifications applied to the experimental system in accordance with the design changes described in the earlier chapter. The specific technological challenges faced while effecting additional features and applying modifications to an already existing system using ongoing regression testing are the focus of this chapter.

4.1. AMPS Improvements

The first version of AMPS was replete with several features including basic data structures, process outline framework, response generation sequence, timer threads to access central server and database functionalities and many other features required to create a responsive simulation environment. However, the implementation of AMPS was subject to time constraints due to which some of the key requirements were left for future developers to work with.

The requirements clearly state that AMPS must be an *adaptive* response system. The adaptive nature of the system depended on its information gathering and analysis capabilities provided by two distinct data sources. While one data source was MIT Web Market and provided trading activity information for the research subject, the other data source was supposed to be RStudio which could constantly provide physiology data. The first version of AMPS depended *only* on the trading activity data

to create a market moving response. It must be remembered that the market moving responses that AMPS generates are in the form or target orders that are submitted to the market. These target orders are submitted by AMPS itself which connects to the market at a client level like a trader. Moreover, the size, price and type of these target orders generated by AMPS must directly correspond to the nature of trading activity and the physiology state of the research subject at every time step. This provided sufficient opportunity to design, develop and implement changes to improve the existing AMPS system. The design changes and cursory description of the changes were described in chapter 3.

4.1.1. Session Implementation

The simulation is capable of running multiple sessions. Each of these sessions is an instance of the `Session` data structure that has attributes and methods which are invoked to run a series of simulation activities for a selected research subject over a finite period of time. Typically, a session was characterized by *only one* particular expected emotional state referenced by the emotion file attribute or `<emotion>.emo` of the particular session instance. For example, a "happy" session had `happy.emo` as its emotional file attribute representing happy emotional state during the session and market movements are targeted towards achieving this state in the trading subject. The rule sets included in `<emotion>.emo` are used to determine the target order price and target order size depending on the characteristics of the session.

The idea behind achieving the adaptive nature of AMPS was to dynamically change the session depending on changes in the physiology data. The problem associated with making this change was that the simulation was tightly integrated with the running session object. This session object was not available for access by other threads until the session was complete or in other words the session remained active for the finite time period as specified by its attribute. Any attempt to modify the current session variable was fraught with errors and exceptions. A deeper understanding of these errors and exceptions could not be performed due to time constraints and need for speeding up implementation. However, a solution to this problem was established by loading emotion data for different sessions into a single generic session. This was done by modifying its emotional file attribute or `<emotion>.emo` to include all emotion rule sets in the emotional file attributes of the different sessions. The modifications were applied using JEP parser [5] scripting as a glue code to alter sessions in real time. Though the session data structure was not modified, this approach worked effectively for the required simulation settings. The two types of active sessions that have been provided an enhanced implementation as part of this version are "happy session" and "sad session". The break session, though available, has not been included as part of the trading session.

4.1.2. Emotion Engine

Wang[1] describes emotion engine as a component of AMPS that is used to generate target prices that are market moving. The internal workings of the emotion engine have been changed to accommodate the requirements described earlier in the chapter.

At any instant during the simulation, the emotion engine is guided by the underlying session instance. It was described earlier that each session is driven by emotion rule sets that are described in the `<emotion>.emo` files. Each of these files is loaded when a session is loaded and the rule sets in these files are used for target price computation as long as the session remains active.

The emotion engine component is driven by `EmotionRuleSet.java` and `EmotionProcessor.java`. An important modification to the emotion engine was the availability of processed physiology data derived from communication with RStudio which necessitated some important internal changes. These physiology variables were first processed using the prototypical scenario constraints and the appropriate emotion to be elicited was determined. This emotion is represented by its own emotional rule sets scripted as part of the generic session file. Thus, it is known from this data as to which subset of rule sets in the generic session file will be used to process the real time trading data to generate the target prices.

In the earlier design, emotion engine queried the database during each time step for updated market variables. Then, the updated market variables were processed using the corresponding emotion rule sets specific to the session domain and passed to the Bid-Ask engine in a forever loop. It was observed that a better method to

process the updated market variables was to limit the emotion rule set based data processing to a finite time horizon instead of an infinite loop.

A prerequisite to this step was to check during each time step if the queried market variables, in fact, reflected a new order submission or not. The new orders are now processed using emotion rule sets for a randomly designated finite time horizon with a range of 20 ± 10 seconds. As described earlier, the emotional rule sets that must be used to process this data are determined using the physiology data. Within this time period, the target prices are generated and sent to the Bid-Ask engine. If the database query returns trading activity variables that have already been processed, the emotion engine does not process the market variables again. However, within the limited time horizon, the variables are passed to the Bid-Ask engine for processing of market moving orders again.

The target prices are determined based on the fraction of the order size to the total number of shares initially provided to the research subject multiplied by the current share price. If P is the current share price, N is the initial number of shares provided to the research subject initially and the order size of the most recent order placed by the trading subject is n, then the target price is given by the following equation.

$$P_{target} = P \pm P\frac{n}{N}$$

The target price is increased or decreased depending on the type of order submitted and the underlying session, which is executed by the Bid-Ask engine and illustrated in

Table 4.2. Finally, the net effect of these changes is that the target prices are determined for a finite time horizon based on the current physiological state of the trading subject. However, the emotion engine does not determine the size and type of the target order.

4.1.3. Bid-Ask Engine

The Bid-Ask engine generates actual buy or sell orders as well as bid and ask spreads. The changes made to the Bid-Ask engine accomplish the task of computing the order size based on the trading activity variables. After the emotion engine determines the target price, the Bid-Ask engine gets control of the target order variable and its attributes.

The earlier version of AMPS randomly determined whether a buy order or a sell order was placed. The randomness associated with determining the type of order to be placed was due to unavailability of physiology data which could be used as a determinant for target order size. Since this version accommodated the gathering and processing of physiology data, the randomness associated with determining the order price could be eliminated. The determination of target order type depended not only on the values of physiology variables but also on the trading activity data for the most recent order. The Bid-Ask engine did not directly process the physiology data but relied on this data processed by the emotion engine. The Bid-Ask engine derived information about the underlying session for the current order from the Emotion Engine which used the prototypical test-case scenario to determine the target

emotional state. The session could be "happy session" or "sad session". In addition to the underlying session, the only information needed about the most recent trading activity was the type of order placed. The type of order placed could be buy or sell. Using this information, the target order type was determined as shown in Table 4.1.

Underlying Session	*Happy*	*Sad*
Type of Order		
Buy	Buy	Sell
Sell	Sell	Buy

Table 4.1: Determination of target order type from physiology and trading data

In a similar fashion, the market movement is determined as shown in Table 4.2. Increasing indicates that the target orders are placed such that the market movement is used to increase the share value and vice versa.

Underlying Session	*Happy*	*Sad*
Type of Order		
Buy	Increasing	Decreasing
Sell	Decreasing	Increasing

Table 4.2: Determination of market movement from physiology and trading data

The size of the target order is such that it is market moving and is a random large number compared to the order size placed by the research subject. The order of the type described in Table 4.1 is such that the market movement is of the type shown in Table 4.2.

The implementation of this functionality was performed through ongoing regression testing. Figure 4.1 illustrates the trading scenario during repeated selling when the underlying session is a happy session using the screenshot of Web Market

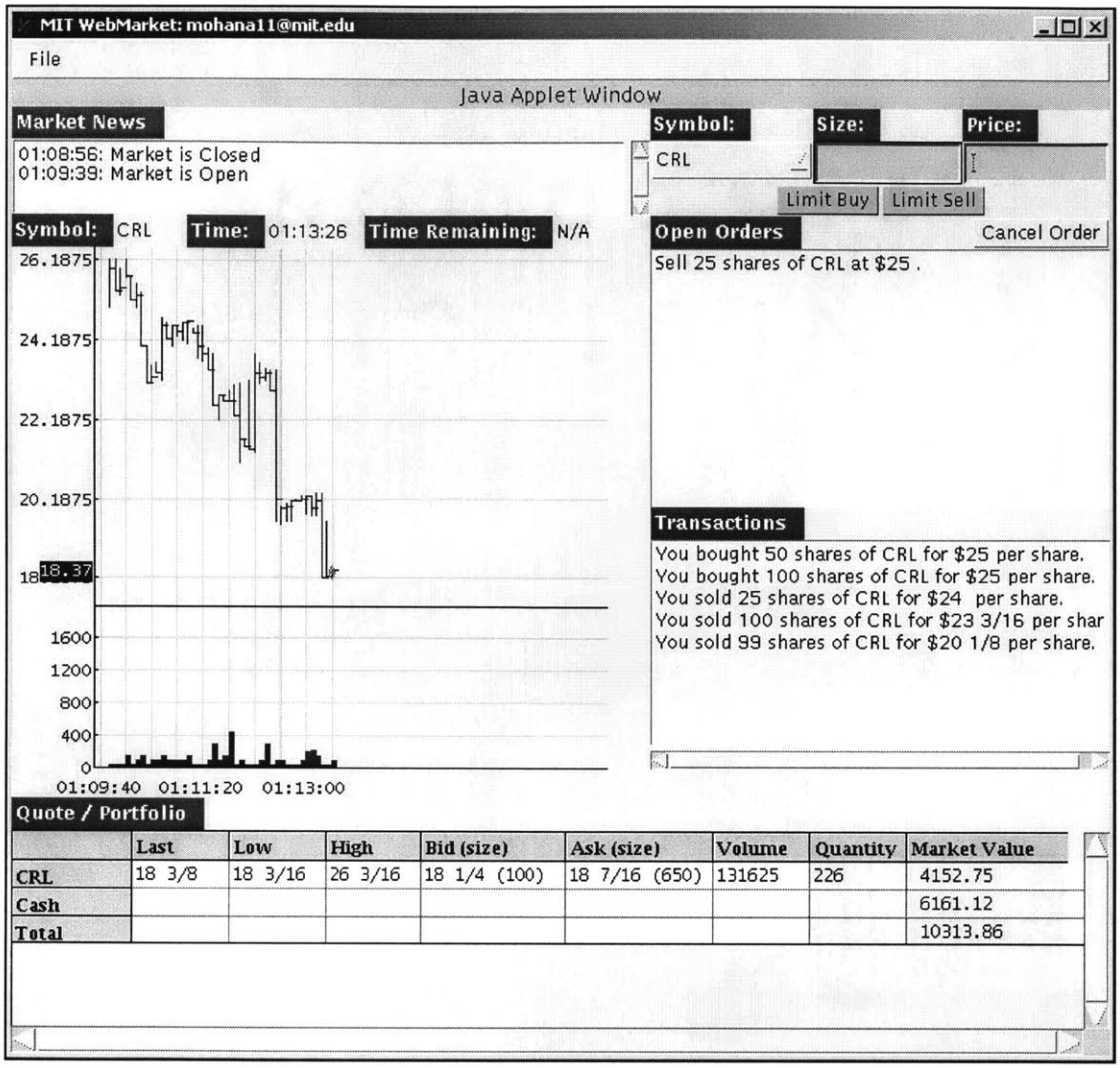

Figure 4-1: Illustration of a Happy-Sell scenario

client for the trading subject. It can be seen that the net effect on market movement for selling in a happy session is to decrease the share price as seen in the figure.

The effectiveness of this functionality was verified by querying the Web Market database for the bid-ask spread values of the share price over the simulation period. This behavior must be verified using the trade submission timestamp data for the trades submitted by the trading subject. Several runs of the simulation were performed for checking consistency of the results obtained and one such scenario

was used for verification. The focus of this verification was first to establish that the observed price increase was proportional to the ratio of the current order size (n) to the total number of shares (N) initially provided. This also provided an opportunity to double-check that the market events were not random and were a direct result of the orders generated by AMPS in response to the subject's trading activity. After the simulation, the Web Market database was queried for the bid and ask values of the security price and the timestamps associated with these values over the time period of the simulation. The database was also queried for the orders submitted by the research subject and the times at which each order was submitted.

This verification process was aimed to verify that the price increase is driven by the equation described earlier.

$$P_{target} = P \pm P\frac{n}{N}$$

where the increase or decrease are guided by information in Table 4.2. The graph shown in Figure 4.2 denotes current security price (P) and projected change in security price ($\pm P\frac{n}{N}$) guided by the formula above for the orders placed during the simulation run. The total number of shares initially provided is N = 300.

The graph shown in Figure 4.3 denotes the actual price variation as observed during the simulation run. It may be observed that not only the market movement is guided in the expected direction but also the magnitude of movement is within the expected limits. It can also be seen that the market movements are dependent on the trading activity of the research subject. It can be inferred from the analysis proposed

in this section that a similar analysis for a sad session would yield results which are very similar in nature but differ in the type of the response. We must remember that we have so far assumed that we know which underlying session is active and described the computation of target order parameters. The underlying session is determined dynamically depending on the physiological state of the trading subject. The procedure to determine the underlying session is detailed in Section 4.4 in which the prototypical test-case scenario is described.

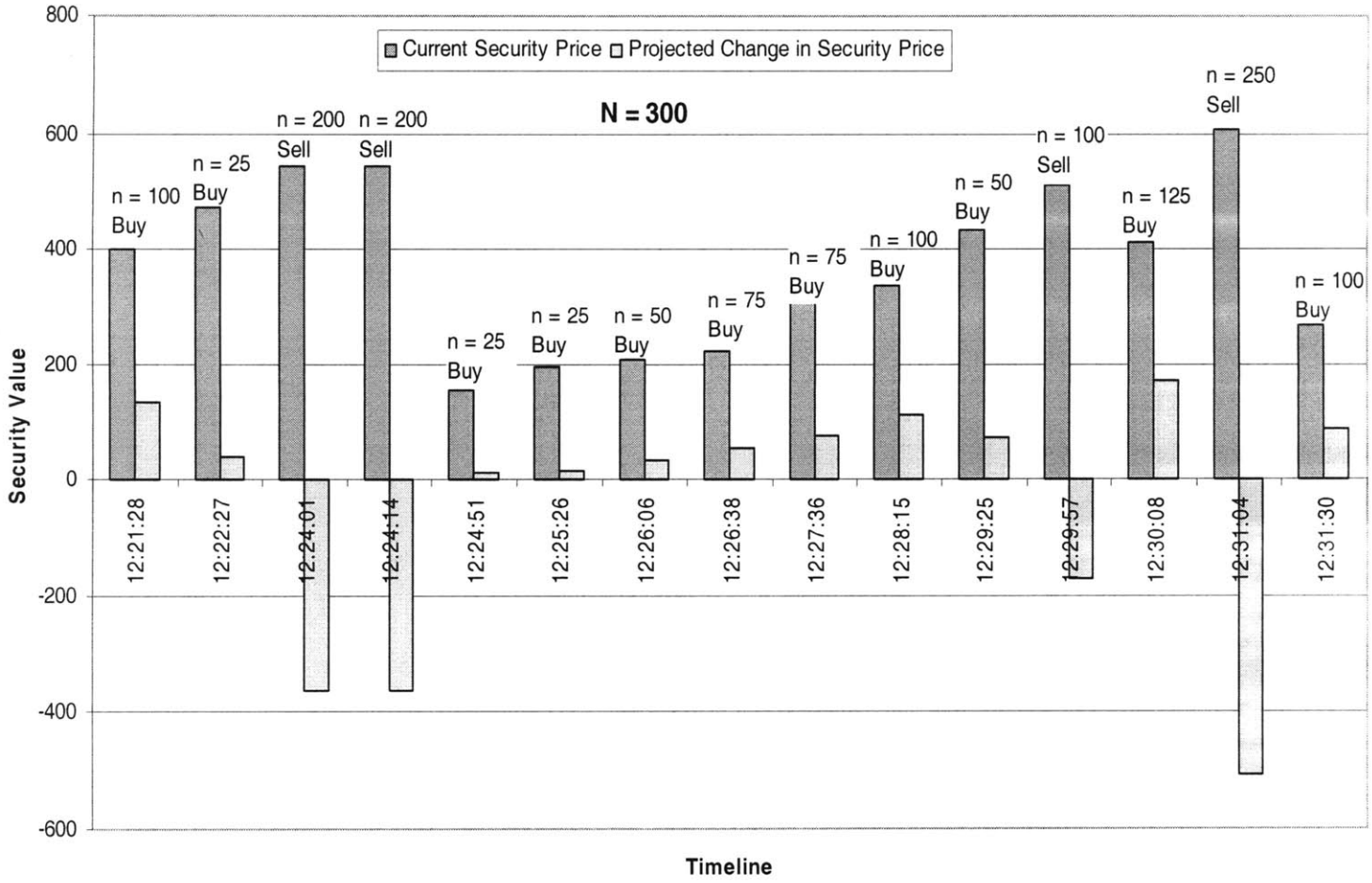

Figure 4-2: Projected change in security price in a happy session

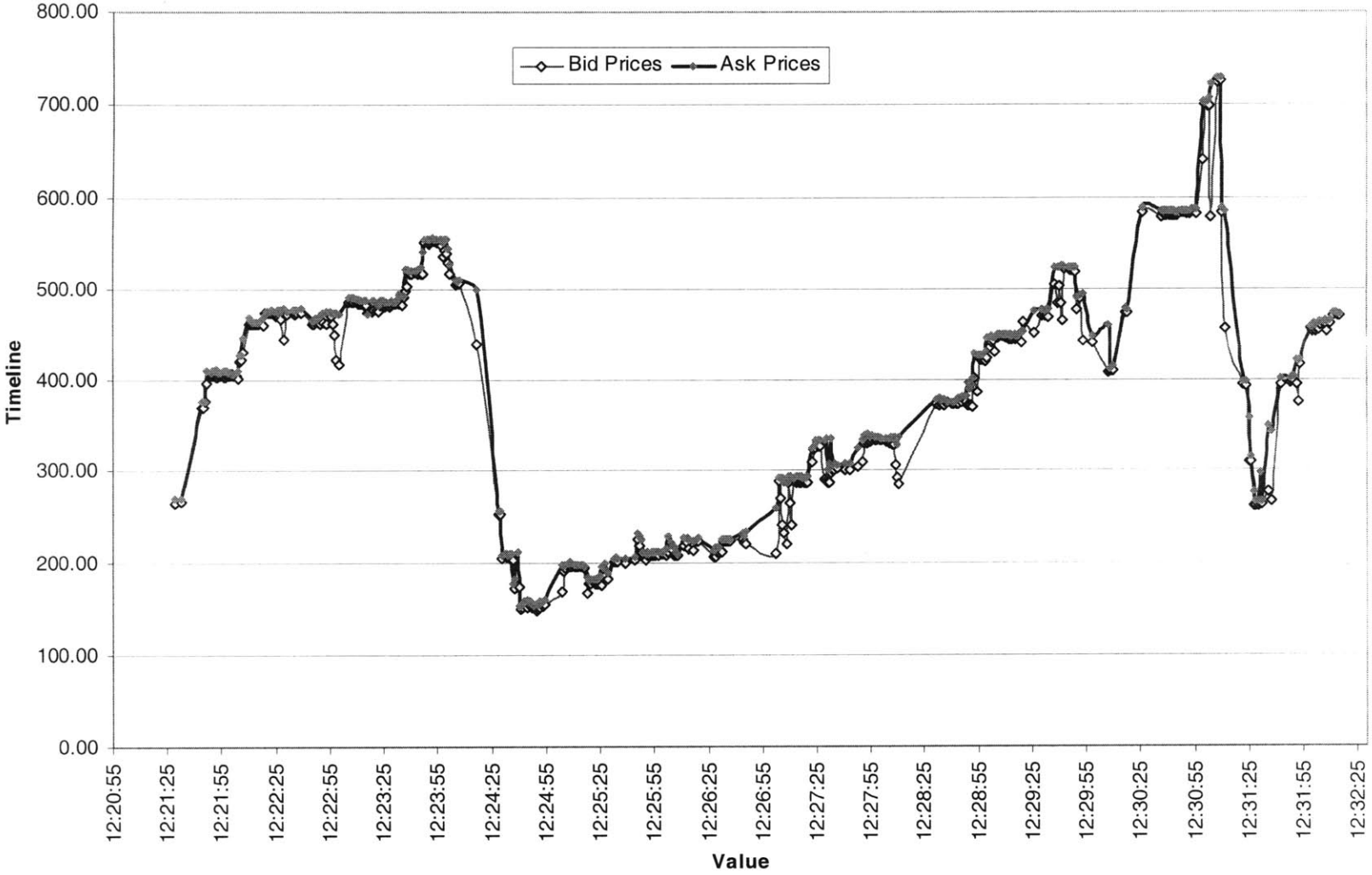

Figure 4-3: Observed movement in security price (bid & ask prices) in a happy session

4.1.4. RStudio CPP Client

An important change to the AMPS system is the addition of RStudio CPP Client, which is a JAVA based client for C++ server applications. This component is responsible for inter-process data communication between AMPS and RStudio. The applications communicate with each other using sockets that use the tcp protocol.

As proposed in the Chapter 2, the internal timer thread which is implemented using the `AMPSTraderThread.java` is modified such that it initializes the `cppClient` data component. An instance of this type is created when the researcher tries to connect to the RStudio System. The specific challenges associated with establishing the communication are explained later.

4.2. RStudio Improvements

RStudio is an independently built physiology data collection software that used ProComp for the data collection process and used its own algorithms for data processing and relaying across the networks. It is clear from the earlier discussion that RStudio system's networking was built for communication with other C++ applications. The data processing algorithms were also not representative of the actual physiological state of the trading subject under study because they considered a default set of values for the physiological variables to scale the responses.

The improvements to be applied to the RStudio system had to be such that it could also work independently. As described in Chapter 2, the design change had to incorporate changes in the form of additional modules which used the existing

functionality and could also build on top of the existing code base without affecting its basic functionality. The data processing algorithms required careful attention because of the need for an easy interpretation of the processed data when transmitted to connecting clients. Additional networking components for RStudio had to be provided to deal with inter-process communication, specifically with Java based clients like AMPS.

4.2.1. Physiology Data Calibration

Though RStudio could work independently and transmit data across the network to many clients, the primary client for which it had been designed was AMPS. It follows that the data to be transmitted across the network should be easily interpretable on the AMPS side. RStudio processes the data on its end and provides some basic algorithms to scale down several variables.

The scaling algorithm provided by Ho [2] in the earlier version of RStudio considers a default set of values for the minimum and the maximum of the values for each of the variables that are collected and scales each variable to a score of 100. This score is then averaged for all the variables taken together. The average score is then taken as representative of the physiological state of the trader.

This scaling method can be further improved to represent the true physiological state. We also consider the fact that the different physiological states have different sensitivities and the actual value of each variable is not uniformly consistent with a particular physiological state. For example, a positive value for skin

conductance may only indicate physiological arousal while the value of forehead temperature does not mean anything as long as we do not know its change over a small period of time which could indicate a change in emotional state. Some of the variables like skin conductance are transient in nature and a constant value of this variable does not necessarily mean that the subject is not being attentive. It is also possible to include individual sensitivities into the scaling methodology on a person-by-person basis and it was not reasonable at the time to assume a default set of maximum and minimum values.

A better scaling methodology would consider each physiology variable separately and consider individual sensitivities. The current RStudio system incorporates baseline calculations which compute the physiological sensitivities of the person. A simple modification of this baseline methodology can provide us with statistical measures like the mean and standard deviation of the physiology variable. The modified baseline calculations for each physiology variable over a period of time could be used as a useful benchmark to scale the variables. For the purpose of baseline calculations, the trading subject is first connected to the ProComp data collection kit and his physiology data is collected over a five-minute period. The physiology state must be maintained such that there are no sudden fluctuations in his/her emotional state during the baseline period. The baseline values, mean and standard deviation, are collected for each physiology variable and stored in a log file.

When the experiment begins, the physiology data begin to show fluctuations and are recorded at a sampling rate approximately equal to 32Hz and 256Hz

depending on the data channel and are processed into one data packet per time step. A time step is approximately one second and each data packet is a `struct` of the type '**`data_transfer_packet`**' that includes processed values of all collected physiology variables for the time step. `RStudioServer` collects these data packets from RStudio. The data packet from RStudioServer is transferred to `RStudioMonitorServer`. Details about `RStudioMonitorServer` are provided later in this chapter. For now, let us consider `RStudioMonitorServer` to be similar to `RStudioMonitor` in terms of its connection with `RStudioServer` and data collection. These data packets contain information about all physiology variables as described in Ho[2]. The values of the variables are extracted from the data packet and these values need to be scaled on an individual basis to gauge their sensitivity. We use the mean and standard deviation computed during the baseline calculations to scale these values. The formula for scaling vector (k_{var}) derived for each of the variables as a result of the scaling calculations is shown below.

$$k_{var} = \frac{X - \bar{\mu}_{var}}{\sigma_{var}}$$

where $\bar{\mu}_{var}$ denotes the mean of the variable as a result of the baseline calculations; σ_{var} is the standard deviation; X is the value of the variable extracted from the data packet for this time step. The subscript "var" indicates the particular physiology variable under study. This could be temperature (TEMP) or electromyographic signal (EMG) or any of the variables being collected. However, for the purpose of this

thesis, the three variables that have been used for scaling are temperature, electromyographic signal and skin conductance.

The scaling vectors are easy to interpret on an individual basis by considering the variations from the mean in terms of multiples of standard deviations. For example, if the mean ($\overline{\mu}_{var}$) is 3 and the standard deviation (σ_{var}) is 2, it can be stated that a value (X) of 2 for the variable is 0.5 or k_{var} times the standard deviation below the mean. The scaling vectors for each of the variables are now ready to be transmitted to AMPS.

4.2.2. RStudioMonitorServer

The networking architecture provided for the initial version of RStudio system accommodated only C++ based clients in terms of its data dissemination. The process of data dissemination was accomplished by the RStudioServer data component. It was observed that other networked clients could connect to the RStudioServer as RStudioMonitor and collect the data packets which contained the information about the physiology data variables. This is assuming that RStudio client is connected to RStudioServer and feeding it with processed physiology data.

There was also need for a server application on the RStudio side which could communicate with Java based clients. It was thought that an amalgamation of a RStudioMonitor's client-like functionality to extract data and an additional server to communicate as mentioned above could handle the data transfer from RStudioServer to AMPS. The result was RStudioMonitorServer which incorporated supported two

functions. One was that it behaved similar to RStudioMonitor in its connection with RStudioServer and extracted the required data packets, while the other was the server like behavior in transmitting calibrated data over to AMPS. It must be noted that the process of calibration happens after the data packet extraction and before data transmission to AMPS. Details about integration with `cppClient` – a part of AMPS - are described in the following section.

4.3. Integration of AMPS and RStudio

The implementation of RStudio CPP Client (`cppClient.java`) on the AMPS side and RStudioMonitorServer (`RStudioMonitorServer.cpp`) on the RStudio side were the building blocks to data level integration of the two applications. The communication is using tcp based sockets and the data transmitted is processed by these modules in these data components on each side for interpretation.

Several other alternatives were examined before choosing a socket based tcp communication methodology. The other alternatives that were studied include Java Native Interface (JNI) and Named Pipes. The Java Native Interface is used to allow a Java application to operate with applications and libraries written in other languages, such as C/C++, and assembly [7]. The JNI implementation is specific to the platform, in which it has been implemented. For example, in the future, if the AMPS system is moved to Windows or any other platform from its current UNIX platform, it will require developers to implement the section of JNI code all over again.

Moreover, several online communities suggest that programmers have been averse to using JNI when the application was to be made scalable over multiple platforms.

The trade-off between Named Pipes and TCP based communication was based on their performance considerations. Khambatti [6] mentions that the performance of Named Pipes over a network is interactive in nature. This interactive communication can become costly over slow networks. The TCP based communication has advantages of using socket performance enhancement mechanisms like windowing and delayed acknowledgements [6]. A major criterion for selection was also developer familiarity which can be critical to driving development speed. In view of the reasons mentioned above, a socket based communication using TCP was used.

The tasks of establishing a connection and data transfer were the primary challenges during the implementation of `cppClient.java` and `RStudioMonitorServer.cpp`. The establishment of connection happens through a blocking call to accept() function to initialize a socket object on the RStudioMonitorServer side. The cppClient object establishes a connection by initializing the socket and sending a message which conveys to the server side that the client has requested byte reversal. Before proceeding further, a small interlude to explain why byte reversal was needed is presented below.

When any data is represented with multiple bytes, the actual ordering of those bytes in memory, or the sequence in which they are transmitted over some medium, is subject to a convention called endianness [8]. The endianness can be either big-

endian or little-endian. C++ is little-endian meaning that the least significant byte is stored at the memory location with the lowest address. However, Java is big-endian meaning that the most significant byte is stored at the memory location with the lowest address. When data transfer takes place between C++ and Java applications, the data will be understood only when it is in a consistent format. For this reason, the little-endian data from C++ must be converted at the byte level to big-endian data and then transferred to the Java application. The process of conversion of data from one form to the other, in our case big-endian to little-endian, is called byte-reversal. This byte reversal in our application takes place on RStudioMonitorServer side before transferring data to cppClient object. The data transfer takes place using array of double to pass the scaling factors (k_{var}) for each of the physiology variables for which data is collected from C++ to Java side. The array of double are received on the AMPS side and stored in an instance of the `ScaleFactor`. The `AMPSTraderThread` component stores an array of `ScaleFactor` instances for processing of the prototypical test-case scenario.

4.4. Prototypical Scenario

It was mentioned earlier in the document that AMPS computes the target order parameters based on the trading activity and the physiological state of the trading subject. It was also described, in Section 4.1.3, how the target order parameters were computed assuming that we know which underlying session is active at each time step in the simulation. In this section, the details of determining the target underlying

session for the current time step of the AMPS internal timer thread are provided. The information about the underlying session is needed to determine which rule sets are used to process the target order price parameters and hence in which direction the market movement must be adjusted to elicit a particular physiology response. In the case, where we use a generic session file, this implies that we determine which subset of rules listed in the session file is to be used.

The prototypical test case scenario is the key to determining the underlying session or, in particular, the subset of rules to be used. The test case scenario has certain rules to compare the values of the physiology variables to the baseline physiology variables and determine the target emotional state. The rules used in prototypical test case scenario are described below.

Rule 1: If $k_{EMG} \geq 1$ for more than 3 seconds and current session is a happy session, then switch the current session to sad, where k_{EMG} is the scaling factor for electromyographic signal.
Rule 2: If $\bar{k}_{SCN} \geq 0.5$ and $T_t - T_{t-5} \geq 0$ and current session is happy session then switch the current session to sad session, where $\bar{k}_{SCN}$ is the average scaling factor for skin conductance over 5 seconds and T_t is the temperature at time 't'.
Rule 3: If $\bar{k}_{SCN} \geq 0.5$ and $T_t - T_{t-5} \leq 0$ and current session is sad session then switch the current session to happy session, where $\bar{k}_{SCN}$ and T_t are as described in Rule 2.

Though the rules above point to happy and sad as discrete sessions, the generic session includes these two sessions and we are referring to the specific subset of rules. These rules can be changed as and when more physiological variables are available for analysis. The use of the test case scenario helps to test the operation of the whole AMPS system.

The rules in the test case scenario can be adjusted to suit the expected tolerance in the emotional fluctuations of the research subject. The test case scenario presented here provides a basic framework to use this application as a training tool for inexperienced traders. Considering the risk and expense of allowing inexperienced traders to trade on the exchange during their training period, this test case scenario in the simulation provides an inexpensive alternative. Moreover, the simulation with this test case scenario can also supplement other learning material used by them.

Chapter 5

Future Work

This prototype addresses some of the key issues of immediate interest to the research at the MIT Lab for Financial Engineering. This chapter identifies the scope for future work in terms of providing additional functionality, improving error and exception handling, scalability and other performance relates issues.

5.1. Logging System Enhancements

The existing system of logging stores information about orders placed by the trading subject, system information and other events in local files. These files may be accessed after the simulation is run. These files as they stand need to be processed before getting useful information about the orders placed by the subject.

A systematic logging system can be provided based on the current requirements of the research administrator. The simulation is based on the set of rules provided by the research administrator and the logged data must be made custom viewable to cater to the administrator's need for understanding the trading activity of the subject.

5.2. Control of Multiple Traders

The business model of AMPS can be made scalable to provide for controlled monitoring of multiple traders in a trading environment. An enhanced version of

AMPS built on these business rules can be used by a manager in trading firm controlling several traders and monitoring their trading activity using their physiology data.

5.3. Provision for Multiple Securities

The current version of the AMPS system uses only one security to consider the trading activity of the subject. It is interesting to note that in a real trading environment, the price movement of several securities in the trader's portfolio interacts with the trader's physiological characteristics. An updated version of MIT Web Market which includes additional functionality for market making activity can be used.

5.4. Additional Logic Constructs

The JEP parser constructs are implemented using the if-then constructs. This parsing language includes several other mathematical and logical constructs which can be used to build several intuitive test constructs. The test constructs might include applying conditions on any of the physiology or trading activity variables. The mathematical functions like sums, averages or polynomial functions and boolean constructs that are provided by this scripting language can also be used.

Chapter 6

Discussion

The ability of traders to take rational decisions in their professions is often blurred in the face of risk. However, this behavior arises from an evolutionary process and the major factor that contributes to this is the emotional state of the trader. This disciplined effort to develop the experimental setup that includes the MIT Web Market, AMPS, RStudio and Machine and Informed Traders is to perform a systematic study of the link between the emotional state of the trading subject and his/her trading activity. Not only is this experimental study useful in understanding this link but also helpful in applying this setup as a training tool for inexperienced traders.

This experimental setup, which includes AMPS, MIT Web Market and RStudio is a prototype simulation system designed to facilitate research on trading and psychology by providing tools to systematically influence the trading experience of the research subject by predictably manipulating his emotional state in an adaptive setting. AMPS is a response generation system that considers the trading activity and physiological state of the trading subject in generating its market moving orders or responses that in turn may elicit physiological changes as well as changes in the market making activity of the trading subject.

Given the set of requirements, the changes to be made to the existing system design and the need for inter-process communication between applications posed

several implementation challenges. The major challenge was to reduce the cost of introducing a new developer to the existing code base, who needs to understand the existing system to the extent that it can be modified without compromising the required functionality. Ongoing regression testing was used frequently to test each and every change incorporated into the system. Modifications and enhancements were planned and implemented by challenging the existing design components and process flow sequence. Iterative development was another challenge in estimating the effort considering time and development speed constraints.

The major changes that were applied include modification of AMPS to include physiology variables into its decision variables used in generating market moving response, especially considering the real time nature of the project. The redesign and implementation of the target order parameters computation was also challenging, taking into account the factors which contribute to the sensitivity of this simulated environment.

AMPS has been installed and is being continuously tested at the MIT Lab for Financial Engineering. The simulation system can definitely be improved by testing several runs using different research subjects and preparing a clear documentation of the improvements needed which can serve as requirements for the next version.

Bibliography

[1] Wang, L., "AMPS – A Simulation System for Modeling and Analyzing the Psychology of Risk-Taking," diss., Cambridge: Massachusetts Institute of Technology, 2003

[2] Ho. E., "A Real-time System for Processing, Sharing, and Display of Physiology Data," diss., Cambridge: Massachusetts Institute of Technology, 2003

[3] Lo, A. W. and Repin, D. V., "The Psychophysiology of Real-Time Financial Risk Processing," Journal of Cognitive Neuroscience, 14, pp 323-339, 2002.

[4] Gamma, E. Helm, R., Johnson, R., and Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software. Boston, MA: Addison-Wesley, 1995.

[5] Funk, N., "JEP – Java Math Expression Parser," [Online Document], 2000, [cited 2004 May 07], Available HTTP: http://www.singlarsys.com/jep/doc/index.html

[6] Khambatti, M., "Named Pipes, Sockets and Other IPC," [Online Document], [2004 May 07], Available HTTP: http://www.public.asu.edu/~mujtaba/Articles%20and%20Papers/cse532.pdf

[7] Stearns, B., "Java Native Interface," [Online Document], 2004, [cited 2004 May 07], Available HTTP: http://java.sun.com/docs/books/tutorial/native1.1/

[8] Quinn, B. Shute, D., Windows Sockets Network Programming. Boston, MA: Addison-Wesley, 1995