The Report Committee for Salim K. Amirdache
Certifies that this is the approved version of the following report:

TSS: A Trading Strategy System

Committee:

____
Adnan Aziz, Supervisor

____
Rajat Chaudhry

TSS: A Trading Strategy System

by

Salim K. Amirdache, BSEE

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May, 2010

For

Leah

Acknowledgements

I would like to thank my supervisor, Professor Adnan Aziz, for giving me the opportunity to work on this project, as well as providing me his patience and technical guidance. I would also like to thank Rajat for his input as my reader and technical expertise. Finally, I would like to thank my parents, Khaled and Ghozlan, and fiancée, Leah, for their encouragement in pursuing a Masters Degree, patience, and support.

TSS: A Trading Strategy System

Salim K. Amirdache, MSE

The University of Texas at Austin, 2010

Supervisor: Adnan Aziz

This report presents TSS - a Trading Strategy System developed to let traders define arbitrarily complex trading strategies in the Java programming language and evaluate them using historical stock information. In addition, TSS provides access to Google Trends data for use in meta-strategy definition, and has the ability to return the best strategy from a family of strategies using data mining algorithms. Finally, TSS is highly extensible - we can integrate new data feeds by simply extending the interface and database.

Table of Contents

List of Tables

List of Figures

Introduction

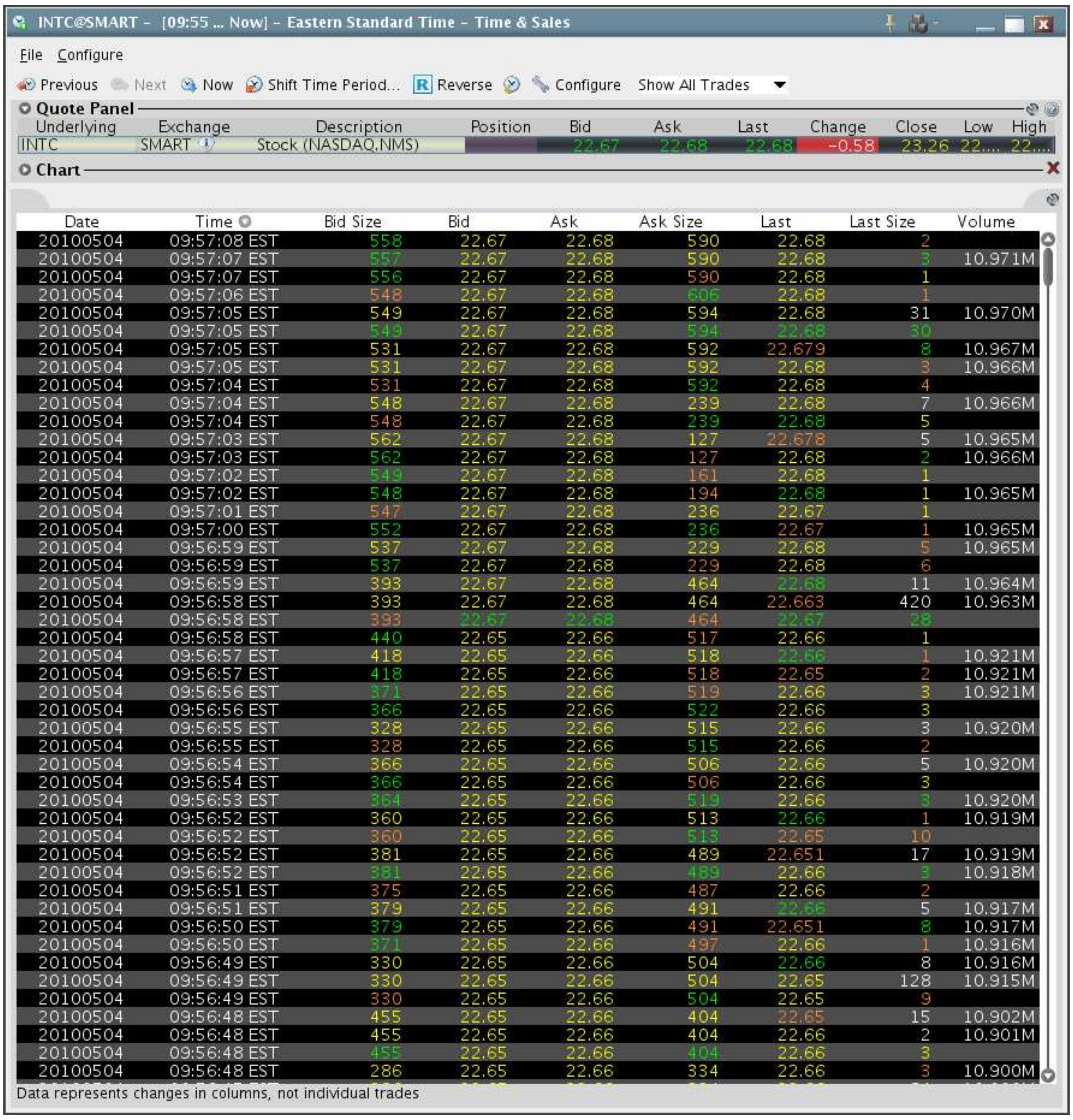
Trend analysis and prediction play an important role in stock trading. Traders often predict the future prices of stocks (and more generally commodities, bonds, currencies, options, and futures) using a variety of information sources including historical stock prices, technical indicators (functions of the underlying time series), fundamentals, and news sources. Traders use tools to analyze data for patterns, generate trading signals, and back-test trading strategies to support trade decisions. The accuracy and timing of these tools are vital to traders.

The stock market refers to the organized trading of company stocks and derivatives, including options and futures, on a public market. The stock price refers to the amount of money required to purchase a single share. The current price of the stock is the price the last trade was executed. The bid price is the highest price any brokerage firm will pay for shares of the stock; the ask price is the lowest price a brokerage firm is willing to sell the stock for. The bid-ask-spread concept is heavily rooted in supply and demand concepts. The price difference between the bid and ask is referred to as spread.

Over a given period, the number of stocks traded is referred to as volume. In this period, the stock price fluctuates depending on market conditions. The maximum price the stock reaches during this period is called the high, while the lowest price is referred to as the low. Furthermore, the price of the stock at the beginning of the period is called the opening price, while the price of the stock at the end of the period is called the closing price.

Figure 1 illustrates the bid and ask requests entering the market for the INTC stock symbol. Each request has a timestamp, size, and price. A number of methodologies are used to analyze the stock market.

Technical Analysis, a popular method of predicting future stock prices using time series data [1].



INTC@SMART - [09:55 ... Now] - Eastern Standard Time - Time & Sales

File Configure

Previous Next Now Shift Time Period... Reverse Configure Show All Trades

Quote Panel

Underlying	Exchange	Description	Position	Bid	Ask	Last	Change	Close	Low	High
INTC	SMART	Stock (NASDAQ.NMS)		22.67	22.68	22.68	-0.58	23.26	22....	22....

Chart

Date	Time	Bid Size	Bid	Ask	Ask Size	Last	Last Size	Volume
20100504	09:57:08 EST	558	22.67	22.68	590	22.68	2	
20100504	09:57:07 EST	557	22.67	22.68	590	22.68	3	10.971M
20100504	09:57:07 EST	556	22.67	22.68	590	22.68	1	
20100504	09:57:06 EST	548	22.67	22.68	606	22.68	1	
20100504	09:57:05 EST	549	22.67	22.68	594	22.68	31	10.970M
20100504	09:57:05 EST	549	22.67	22.68	594	22.68	30	
20100504	09:57:05 EST	531	22.67	22.68	592	22.679	8	10.967M
20100504	09:57:05 EST	531	22.67	22.68	592	22.68	3	10.966M
20100504	09:57:04 EST	531	22.67	22.68	592	22.68	4	
20100504	09:57:04 EST	548	22.67	22.68	239	22.68	7	10.966M
20100504	09:57:04 EST	548	22.67	22.68	239	22.68	5	
20100504	09:57:03 EST	562	22.67	22.68	127	22.678	5	10.965M
20100504	09:57:03 EST	562	22.67	22.68	127	22.68	2	10.966M
20100504	09:57:02 EST	549	22.67	22.68	161	22.68	1	
20100504	09:57:02 EST	548	22.67	22.68	194	22.68	1	10.965M
20100504	09:57:01 EST	547	22.67	22.68	236	22.67	1	
20100504	09:57:00 EST	552	22.67	22.68	236	22.67	1	10.965M
20100504	09:56:59 EST	537	22.67	22.68	229	22.68	5	10.965M
20100504	09:56:59 EST	537	22.67	22.68	229	22.68	6	
20100504	09:56:59 EST	393	22.67	22.68	464	22.68	11	10.964M
20100504	09:56:58 EST	393	22.67	22.68	464	22.663	420	10.963M
20100504	09:56:58 EST	393	22.67	22.68	464	22.67	28	
20100504	09:56:58 EST	440	22.65	22.66	517	22.66	1	
20100504	09:56:57 EST	418	22.65	22.66	518	22.66	1	10.921M
20100504	09:56:57 EST	418	22.65	22.66	518	22.65	2	10.921M
20100504	09:56:56 EST	371	22.65	22.66	519	22.66	3	10.921M
20100504	09:56:56 EST	366	22.65	22.66	522	22.66	3	
20100504	09:56:55 EST	328	22.65	22.66	515	22.66	3	10.920M
20100504	09:56:55 EST	328	22.65	22.66	515	22.66	2	
20100504	09:56:54 EST	366	22.65	22.66	506	22.66	5	10.920M
20100504	09:56:54 EST	366	22.65	22.66	506	22.66	3	
20100504	09:56:53 EST	364	22.65	22.66	519	22.66	3	10.920M
20100504	09:56:52 EST	360	22.65	22.66	513	22.66	1	10.919M
20100504	09:56:52 EST	360	22.65	22.66	513	22.65	10	
20100504	09:56:52 EST	381	22.65	22.66	489	22.651	17	10.919M
20100504	09:56:52 EST	381	22.65	22.66	489	22.66	3	10.918M
20100504	09:56:51 EST	375	22.65	22.66	487	22.66	2	
20100504	09:56:51 EST	379	22.65	22.66	491	22.66	5	10.917M
20100504	09:56:50 EST	379	22.65	22.66	491	22.651	8	10.917M
20100504	09:56:50 EST	371	22.65	22.66	497	22.66	1	10.916M
20100504	09:56:49 EST	330	22.65	22.66	504	22.66	8	10.916M
20100504	09:56:49 EST	330	22.65	22.66	504	22.65	128	10.915M
20100504	09:56:49 EST	330	22.65	22.66	504	22.65	9	
20100504	09:56:48 EST	455	22.65	22.66	404	22.65	15	10.902M
20100504	09:56:48 EST	455	22.65	22.66	404	22.66	2	10.901M
20100504	09:56:48 EST	455	22.65	22.66	404	22.66	3	
20100504	09:56:48 EST	286	22.65	22.66	334	22.66	3	10.900M

Data represents changes in columns, not individual trades

Figure 1: Bid and asks entering the market

A number of back-testing tools exist from major trading firms such as Fidelity [2], Ameritrade [3], and Scottrade [4]. All these tools have a common limitation: the user can define limited trading strategies using only historical stock prices and simple technical indicators. However, the Internet is rich with a variety of data sources. For example, Google Trends [5] provides users with a Search Volume Index - an index that describes the number of searches performed on the specified keywords daily, and News Reference Volume – describes the number of times the specified keywords appeared in a Google news story. Another limitation of the tools in [2-5] is that they do not provide the flexibility of allowing a user to define a strategy in a general purpose programming language.

TSS is a system that evaluates arbitrary trading strategies using historical price data. TSS provides users the ability to define a trading strategy using the Java programming language. Historical stock prices, technical indicators, and chart analysis functions are available to the user trading strategies. In addition, trading strategies have access to information from Google Trends, as well as data mining features such as the ability to build classification models. The results of the evaluation are presented on visual charts using Google's Visualization API [6]. Also, users can browse Google News [7] headlines related to the current symbol while viewing their trading strategy results.

The report organization is as follows: We begin by describing the implementation of TSS. After that, we will focus on user stories for two stock traders with different trading philosophies in the Case Study section. TSS is used to provide solutions to both stock traders and present results in the context of the two user stories. Finally, we will conclude with a list of future extensions for TSS. We provide low level documentation for TSS in the Appendix.

Implementation

We used a combination of technologies to implement the TSS Client-Server architectural model [8]. TSS's client, a web-based user interface, is hosted from the TSS custom web server. The web page communicates with the web server to submit trading strategies and retrieve strategy test results. Trading strategies are written in the Java programming language and are passed on to the Strategy Processor from the web server after a trading strategy submission. The Strategy Processor compiles the trading strategy and launches the Testing system. The Testing system runs the compiled trading strategy against the symbols available in the database. Once Testing is complete, the web server passes the results to the web page. The following section describes in detail the components of TSS and their interaction shown in Figure 2.

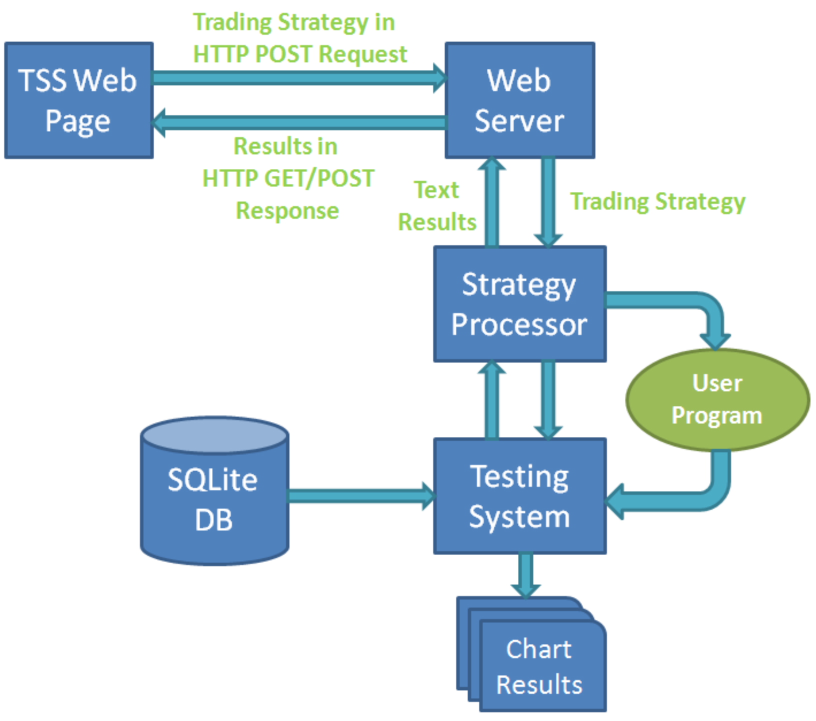
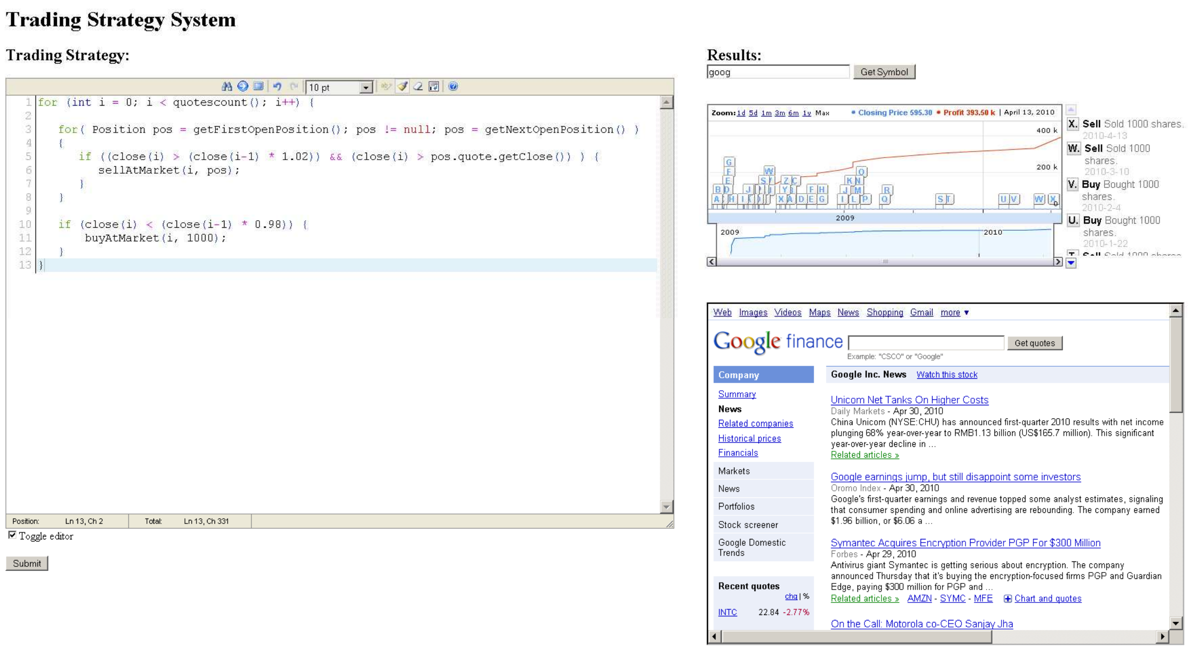


Figure 2: Diagram of TSS system components

WEB-BASED USER INTERFACE

Users submit trading strategies for evaluation and view their results using TSS's Web-based user interface, shown in Figure 3.



```
for (int i = 0; i < quotescount(); i++) {

   for( Position pos = getFirstOpenPosition(); pos != null; pos = getNextOpenPosition() )
   {
      if ((close(i) > (close(i-1) * 1.02)) && (close(i) > pos.quote.getClose()) ) {
         sellAtMarket(i, pos);
      }
   }

   if (close(i) < (close(i-1) * 0.98)) {
       buyAtMarket(i, 1000);
   }
}
```

X. **Sell** Sold 1000 shares.
2010-4-13

W. **Sell** Sold 1000 shares.
2010-3-10

V. **Buy** Bought 1000 shares.
2010-2-4

U. **Buy** Bought 1000 shares.
2010-1-22

Unicom Net Tanks On Higher Costs
Daily Markets - Apr 30, 2010
China Unicom (NYSE:CHU) has announced first-quarter 2010 results with net income plunging 68% year-over-year to RMB1.13 billion (US$165.7 million). This significant year-over-year decline in ...
Related articles »

Google earnings jump, but still disappoint some investors
Oromo Index - Apr 30, 2010
Google's first-quarter earnings and revenue topped some analyst estimates, signaling that consumer spending and online advertising are rebounding. The company earned $1.96 billion, or $6.06 a ...

Symantec Acquires Encryption Provider PGP For $300 Million
Forbes - Apr 29, 2010
Antivirus giant Symantec is getting serious about encryption. The company announced Thursday that it's buying the encryption-focused firms PGP and Guardian Edge, paying $300 million for PGP and ...
Related articles » AMZN - SYMC - MFE ⊞ Chart and quotes

On the Call: Motorola co-CEO Sanjay Jha

Figure 3: Screen shot of TSS web page

The code editor text box uses the EditArea [9] JavaScript library which provides text formatting, search and replace, and real-time syntax highlighting. Once the strategy is written, the Submit button is used to send the trading strategy for evaluation.

AJAX, Asynchronous JavaScript, and XML are used to communicate with the server using POST or GET HTTP methods [10]. The user is never redirected to another web page, and only the contents of the web page are updated providing a seamless experience to the user.

GET methods are commonly used by web applications to retrieve resources from a server. In TSS, the AJAX GET method is used to retrieve the results symbol names, the symbol chart data, and files required to view the web page. POST methods are generally used to submit HTML form data to the server. On a trading strategy submission, the web page sends the trading strategy within the message body of the AJAX POST request to the server for processing and evaluation. While the web server is processing the request, a loading graphic is shown to the user. After the server completes the request, the loading graphic is removed and a search box is presented to the user. Users can search for the results of a particular symbol by entering the symbol name in the search text field and pressing the Get Results button shown in Figure 4.

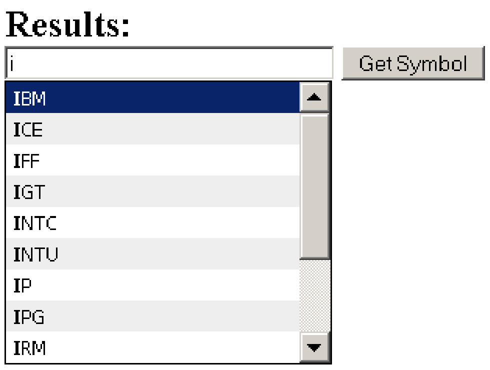


Figure 4: Searching the available symbols

Once the Get Results button is pressed, a request is sent to the server via AJAX for the specified symbols results. The symbols results are loaded into a Google Chart as shown in Figure 5.

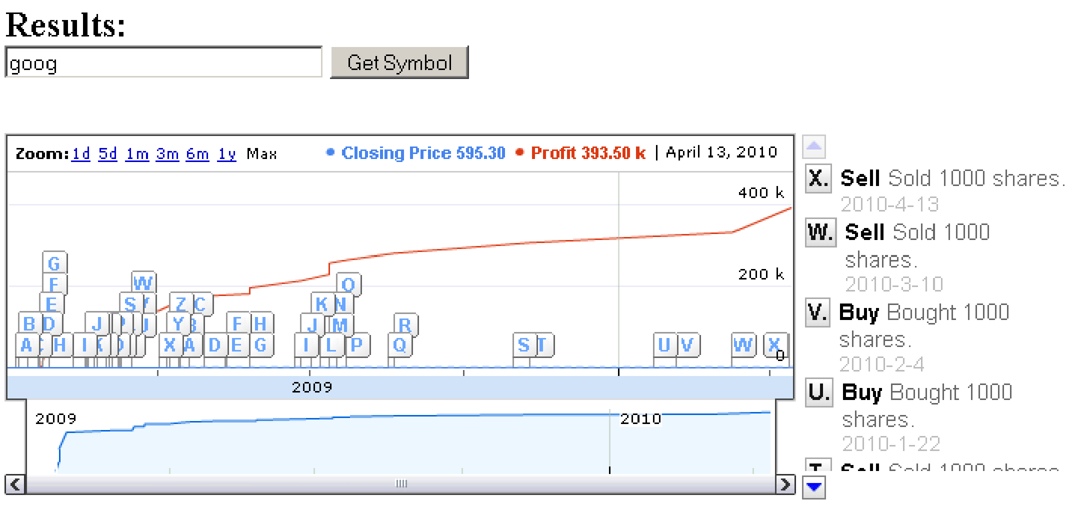
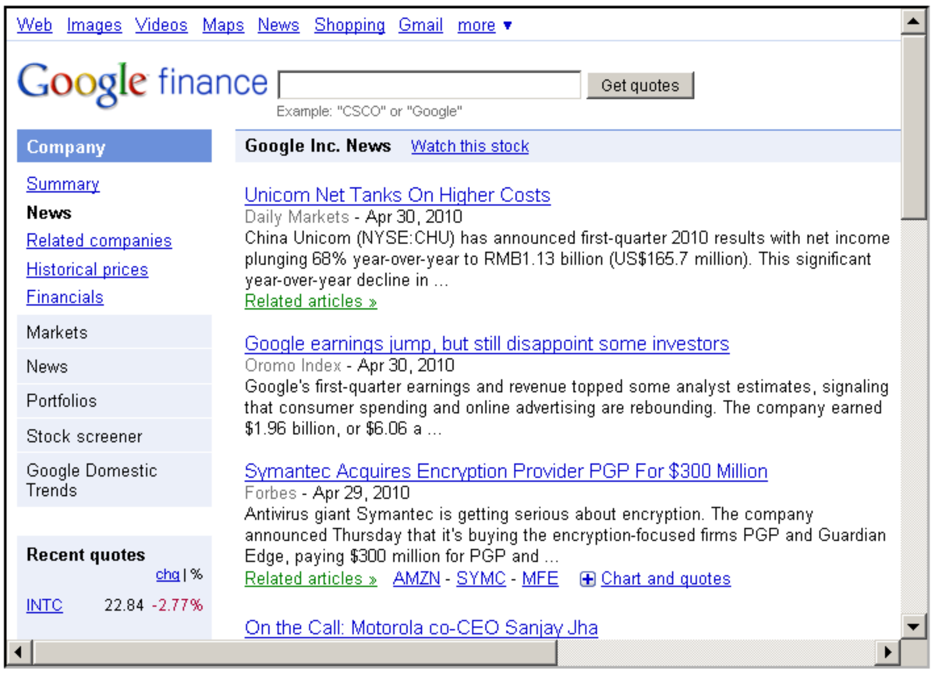

Results:
goog
Get Symbol
Zoom: 1d 5d 1m 3m 6m 1y Max
Closing Price 595.30
Profit 393.50 k
April 13, 2010
400 k
200 k
2009
2010
X. Sell Sold 1000 shares. 2010-4-13
W. Sell Sold 1000 shares. 2010-3-10
V. Buy Bought 1000 shares. 2010-2-4
U. Buy Bought 1000 shares. 2010-1-22

Figure 5: Google Chart displaying evaluation results for a single stock symbol

In addition, Google News is integrated via a HTML iFrame into TSS to show news related to the current stock selected, as seen in Figure 6.



Web Images Videos Maps News Shopping Gmail more ▾

Google finance [ ] Get quotes

Example: "CSCO" or "Google"

Company

Summary
News
Related companies
Historical prices
Financials

Markets
News
Portfolios
Stock screener
Google Domestic Trends

Recent quotes chg | %
INTC 22.84 -2.77%

Google Inc. News Watch this stock

Unicom Net Tanks On Higher Costs
Daily Markets - Apr 30, 2010
China Unicom (NYSE:CHU) has announced first-quarter 2010 results with net income plunging 68% year-over-year to RMB1.13 billion (US$165.7 million). This significant year-over-year decline in ...
Related articles »

Google earnings jump, but still disappoint some investors
Oromo Index - Apr 30, 2010
Google's first-quarter earnings and revenue topped some analyst estimates, signaling that consumer spending and online advertising are rebounding. The company earned $1.96 billion, or $6.06 a ...

Symantec Acquires Encryption Provider PGP For $300 Million
Forbes - Apr 29, 2010
Antivirus giant Symantec is getting serious about encryption. The company announced Thursday that it's buying the encryption-focused firms PGP and Guardian Edge, paying $300 million for PGP and ...
Related articles » AMZN - SYMC - MFE ⊞ Chart and quotes

On the Call: Motorola co-CEO Sanjay Jha

Figure 6: Google News search of current stock

WEB SERVER

A custom multi-threaded web server was developed for TSS to consume HTTP POST and GET requests. When a GET request is received, the contents of the file specified is returned. When a POST request is received the server assumes that a trading strategy was submitted. The trading strategy is extracted from the POST payload and sent to the Strategy Processor. Once the Testing is complete, the strategies output is returned in the client's POST request response.

TRADING STRATEGY LANGUAGE

Users write trading strategies using Java and are provided access to the Java Util package; therefore, users have the ability to use data structures such as Lists and Maps contained in the Collections framework, and useful classes such as Date and Calendar. In addition, users have access to a number of data structures for manipulating historical stock quote data, functions to make trades, and perform data mining tasks.

Series

This class provides a generic container for an array of values of the same type. A Series of the specified type is constructed from an input array. Values of a series are accessed using accessor functions, and various technical indicators can be applied to the series data. For example, if a user wanted to access the 10th value in the Series s, they would call the function `s.get(10)`.

The Series class uses TA-lib [11], a technical analysis library, to perform the technical analysis calculations. Table 1 in the Appendix provides a summary of the available methods of the Series class.

Quote

This class encapsulates a stock's historical data. TSS uses daily stock data; thus, a single stock quote refers to the set of information: open, close, low, high, and volume of a single given day. For instance, if a user required the opening price from the Quote class q, they would call the method `q.getOpen()`. Table 2 in the Appendix provides a summary of the available methods of the Quote class.

Quotes

This class contains a single stock's complete historical data available in the database. There are number of methods available to access the data. For example, a user can request a specific attribute of the stock such as Opening price for a particular day using the function `open(index)`, where index represents the day. Each daily quote corresponds to a numerical index, such that the first quote is 0, and the last quote is n. The quotes are sorted by date in ascending order. Table 3 in the Appendix describes the methods available for the Quotes class.

WekaClassifier

Weka [12] is an open source Java library implementing a set of machine learning algorithms for data mining tasks. The TSS WekaClassifier class provides access to the Weka's classification and regression analysis tools. For example, a user can build a classifier by supplying training data, classifier name, and range to `BuildClassifier()` method. After that, a user can classify test data using the `Classify()` method. Table 4 in the Appendix summarizes the methods of the WekaClassifier class.

We provide an example of the WekaClassifier in the Case Study section of this report.

Trading System Utility Functions

The trading strategy can submit various types of orders for entering and exiting positions. A position object contains the number of shares purchased and the daily stock Quote that was purchased. Table 5 describes the methods accessible for opening and closing positions.

Method	Description
`void buyAtMarket(int quoteIndex, int numberOfShares)`	Buy the specified number of stocks at the closing price of the specified trading day.
`void sellAtMarket(int quoteIndex, Position pos)`	Sell the opened position with the specified trading day closing price.

Table 5: Methods for entering and exiting positions

In addition, TSS provides the methods in Table 6 to manipulate opened positions.

Method	Description
`Position getFirstOpenPosition()`	Returns the first open position.
`Position getNextOpenPosition()`	Returns the next open position. If there are no more open positions, return null.

Table 6: Position manipulation methods

Default Data

For each stock symbol available in the database, the trading strategy is run. During that run, the default stock symbol's data is accessible without the need to create a Quotes class.

STRATEGY PROCESSOR

The web server invokes the Strategy Processor once a trading strategy submission is identified. The Strategy Processor inserts the trading strategy coded in Java into a UserProgram class, and compiles the created UserProgram object using Java’s Runtime class – essentially invoking a shell and running the Java compiler on UserProgram and then executing UserProgram on a JVM. The UserProgram class inherits the Quotes class such that default stock data is available within the UserProgram. In addition, the trading system utility methods are defined by the UserProgram class. If compilation is successful, the Testing System is launched; however, if the compilation fails, the error(s) are returned to the web server to be returned to the web page.

TESTING SYSTEM

The Testing System is launched by the Strategy Processor once compilation of the UserProgram is successful. The system is responsible for retrieving stock data from the database and executing the compiled UserProgram for each available symbol. Once execution is complete for a single stock, the transactions generated during the run are evaluated. The results of the evaluation are written to files stored on the web server. These files are retrieved by the web page when a user requests the results for a particular stock.

DATABASE

The UpdateDatabase class is used to manipulate a SQLite [13] database that's populated with historical stock quote data pulled in from Yahoo Finance [14]. SQLlite is a popular software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

The database schema is as follows:

- There is a table called `listofsymbols` - which has one symbol per row.
- For each symbol in `listofsymbols`, there is a table, which has a number of rows - each row corresponds to data for a particular day.

Table 7 in the Appendix describes the database manipulation methods for the `UpdateDatabase` class.

The `updateSymbol` method sends queries describing the contents of the requested data to Yahoo Finance to download a CSV file containing historical stock quotes. For example, consider the following query:

http://ichart.finance.yahoo.com/table.csv?s=INTC&a=00&b=1&c=2009&d=11&e=31&f=2009&g=d

When submitted, Yahoo Finance returns a CSV file containing the daily historical stock quote data for the symbol INTC from January 1, 2009 to December 31, 2009. Each row in the CSV file corresponds to 1 day of stock quote data. The default data returned contains the following information:

- Date - Trading day date
- Open – The stock's opening price
- High – The stock's maximum price for the time interval
- Low – The stock's minimum price for the time interval

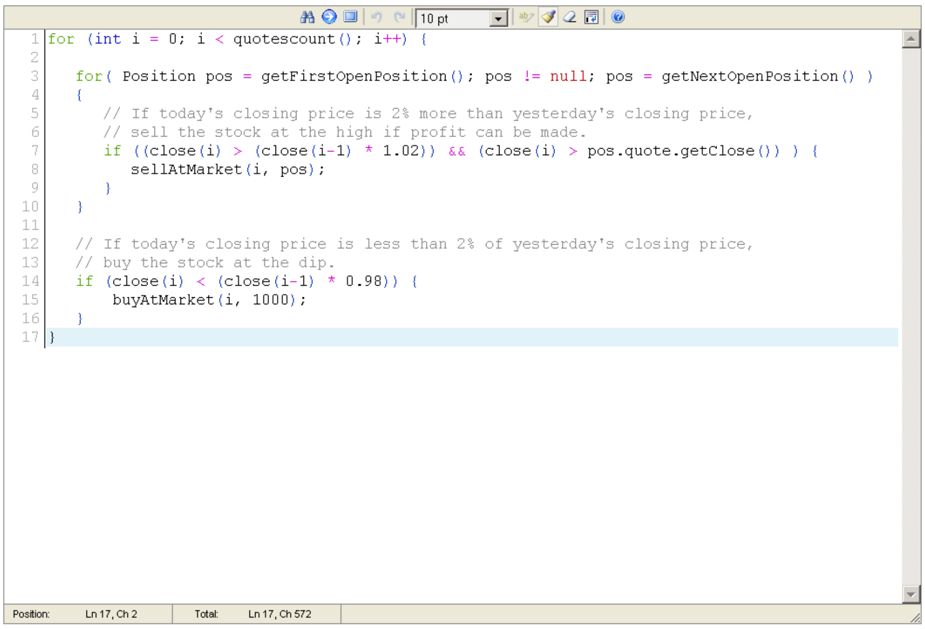
- Close – The stock’s closing price
- Volume – The volume of stock traded

Case Study

In this section, we will examine two trading strategies that take advantage of TSS's features.

THE DIP BUYER

A dip buying strategy generates a buy signal when the current price is a specific percentage below the previous day's closing price. A sell signal is identified when the closing price is a specific percentage above the previous day's closing price. Thus, the stock trader buys on the lows and sells on the highs. Figure 7 illustrates the dip buyer strategy in TSS.



```
for (int i = 0; i < quotescount(); i++) {

   for( Position pos = getFirstOpenPosition(); pos != null; pos = getNextOpenPosition() )
   {
      // If today's closing price is 2% more than yesterday's closing price,
      // sell the stock at the high if profit can be made.
      if ((close(i) > (close(i-1) * 1.02)) && (close(i) > pos.quote.getClose()) ) {
         sellAtMarket(i, pos);
      }
   }

   // If today's closing price is less than 2% of yesterday's closing price,
   // buy the stock at the dip.
   if (close(i) < (close(i-1) * 0.98)) {
       buyAtMarket(i, 1000);
   }
}
```

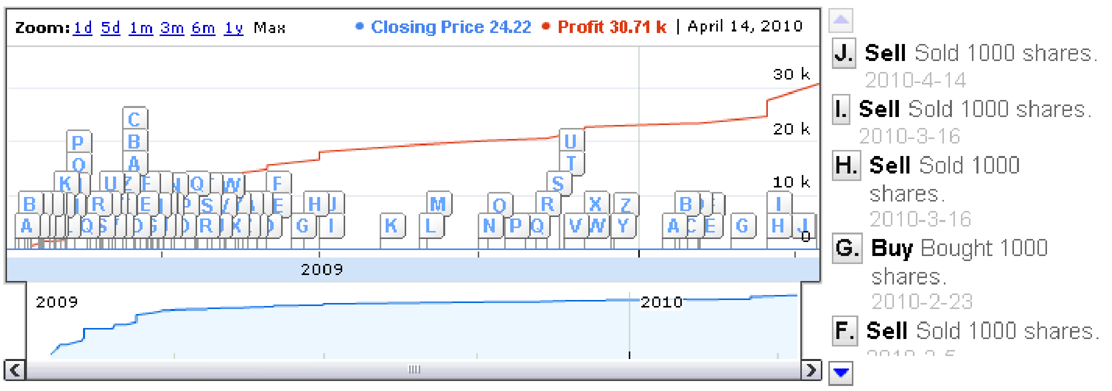
Figure 7: TSS Dip Buyer Strategy

The trading strategy can be broken up into two sections. In the first section, lines 14 to 17 in Figure 7, the system buys the stock at market price (the close price) if the current day's closing price (`close(i)`) is 2% below the previous day's closing price (`close(i-1)`). An open position of 1000 shares is created and added to the positions array.

In the second section, lines 3 to 10 in Figure 7, we loop over any open positions and sell if the position meets the criteria to sell:

1. Today's closing price is 2% above the previous day's closing price.
2. A profit can be made.

The results of the trading strategy run for the stock symbol INTC is shown in Figure 8. The strategy was run on 1 years worth of INTC data. We see that if the stock trader followed the trading signals indicated, we would have made $30.71 K in profit.


Zoom: 1d 5d 1m 3m 6m 1y Max
● Closing Price 24.22 ● Profit 30.71 k | April 14, 2010
30 k
20 k
10 k
0
2009
2009
2010

J. **Sell** Sold 1000 shares. 2010-4-14

I. **Sell** Sold 1000 shares. 2010-3-16

H. **Sell** Sold 1000 shares. 2010-3-16

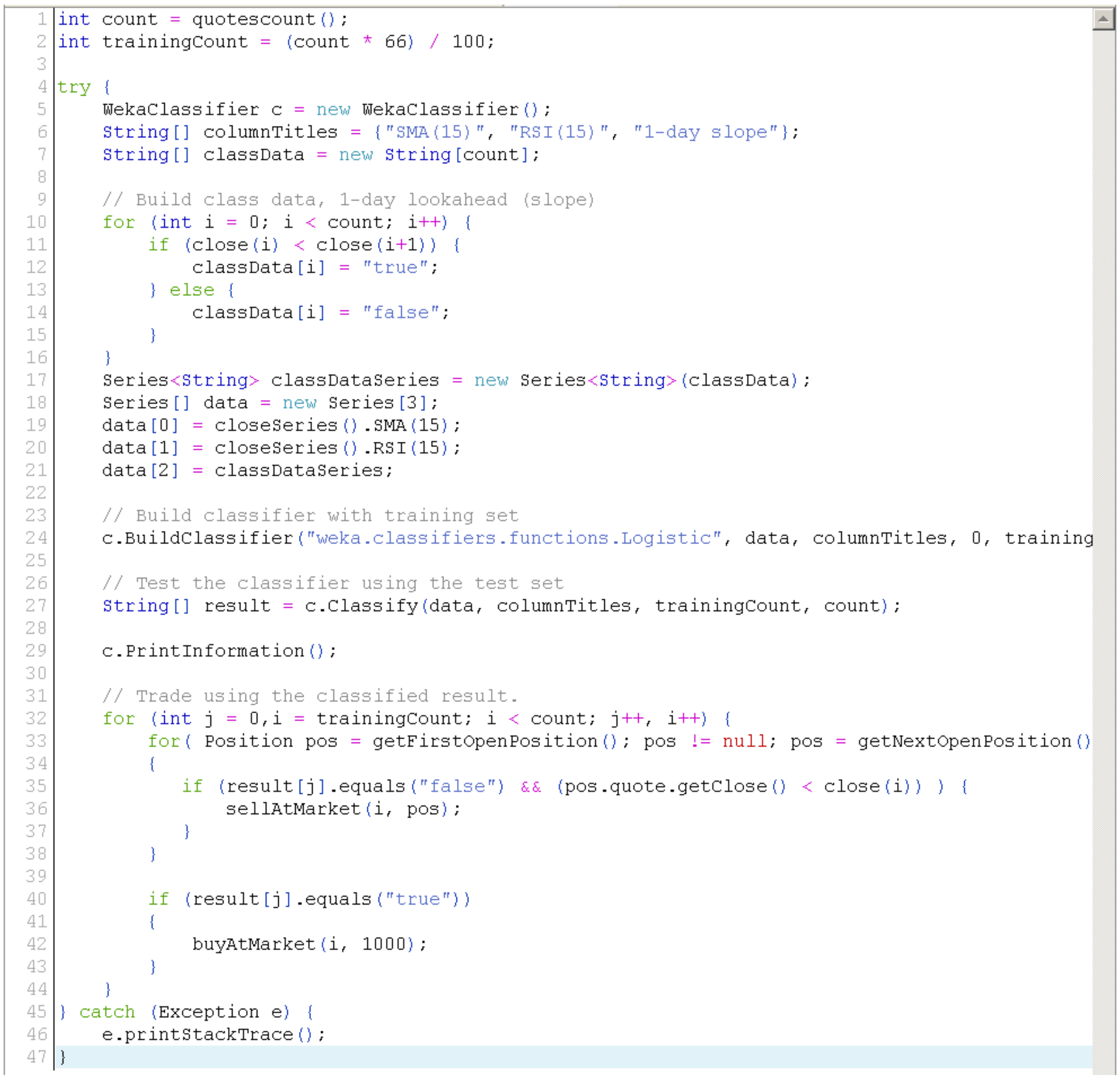
G. **Buy** Bought 1000 shares. 2010-2-23

F. **Sell** Sold 1000 shares. 2010-2-5

Figure 8: Dip Buyer chart for INTC

THE DATA MINER

In this example, we will look at a more complex trading strategy using TSS's data mining functions. The data miner uses Logistic regression [15] to classify input data to derive future buy and sell signals. Logistic regression is a popular method to describe the relationship between explanatory variables and a Boolean outcome. The Boolean outcome variable is expressed as a probability between two possible outcomes. In this case, the explanatory variables are technical indicators, and the outcome is an up or down trend. A subset of the available historical stock data is used as training data to generate the regression coefficients. These coefficients are stored in the built classifier for use in future classification of test data. The data miner uses the generated trend signals to make trading decisions. In Figure 9, we show an example trading strategy using the data mining functions.



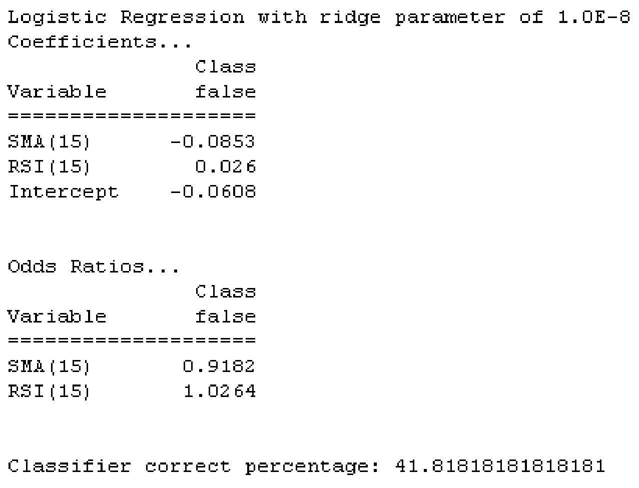
```
 1 int count = quotescount();
 2 int trainingCount = (count * 66) / 100;
 3
 4 try {
 5     WekaClassifier c = new WekaClassifier();
 6     String[] columnTitles = {"SMA(15)", "RSI(15)", "1-day slope"};
 7     String[] classData = new String[count];
 8
 9     // Build class data, 1-day lookahead (slope)
10     for (int i = 0; i < count; i++) {
11         if (close(i) < close(i+1)) {
12             classData[i] = "true";
13         } else {
14             classData[i] = "false";
15         }
16     }
17     Series<String> classDataSeries = new Series<String>(classData);
18     Series[] data = new Series[3];
19     data[0] = closeSeries().SMA(15);
20     data[1] = closeSeries().RSI(15);
21     data[2] = classDataSeries;
22
23     // Build classifier with training set
24     c.BuildClassifier("weka.classifiers.functions.Logistic", data, columnTitles, 0, training
25
26     // Test the classifier using the test set
27     String[] result = c.Classify(data, columnTitles, trainingCount, count);
28
29     c.PrintInformation();
30
31     // Trade using the classified result.
32     for (int j = 0,i = trainingCount; i < count; j++, i++) {
33         for( Position pos = getFirstOpenPosition(); pos != null; pos = getNextOpenPosition()
34         {
35             if (result[j].equals("false") && (pos.quote.getClose() < close(i)) ) {
36                 sellAtMarket(i, pos);
37             }
38         }
39
40         if (result[j].equals("true"))
41         {
42             buyAtMarket(i, 1000);
43         }
44     }
45 } catch (Exception e) {
46     e.printStackTrace();
47 }
```

Figure 9: Trading Strategy using Logistic regression

The trading strategy must first assemble the input data used to train the classifier model. For the independent or explanatory variables, the strategy uses the simple moving average with a 15 day period, and a Relative Strength index with a 15 day period shown on lines 19 and 20 respectively. The 3rd column of the input data is the classification we want to achieve. To build this series, we simply compare the current day's close price with the next day’s close price. If the slope is positive, we classify the {SMA, RSI} tuple as true; otherwise, the slope is negative, and we classify the tuple as false.

On line 24, we make the call to build the Logistic classifier. Note that we classify the input data from 0 to trainingCount. That is, we split the available historical stock data into a training set and a test set. In this case, we use 66% of the available data to train the classifier model. Even though we passed in the entire array of generated data, the BuildClassifier function will only look at the data within the specified start and end range.

Once classification is complete, we test the classifier using the remaining 33% of historical stock data. Note that the data array is the same for building and classifying. This allows us to generate a percentage that describes how accurate the classifier predicted the test data. On line 29, the classifier model and the prediction percentage, shown in Figure 10, is printed.



```
Logistic Regression with ridge parameter of 1.0E-8
Coefficients...
                  Class
Variable          false
===================
SMA(15)         -0.0853
RSI(15)           0.026
Intercept       -0.0608


Odds Ratios...
                  Class
Variable          false
===================
SMA(15)          0.9182
RSI(15)          1.0264


Classifier correct percentage: 41.81818181818181
```

Figure 10: Output of Classification

We see that during this run the model correctly identified 41% of the test data. Using the classification results, we make trading decisions for buying and selling the stock. The outcome of these decisions is shown in Figure 11.

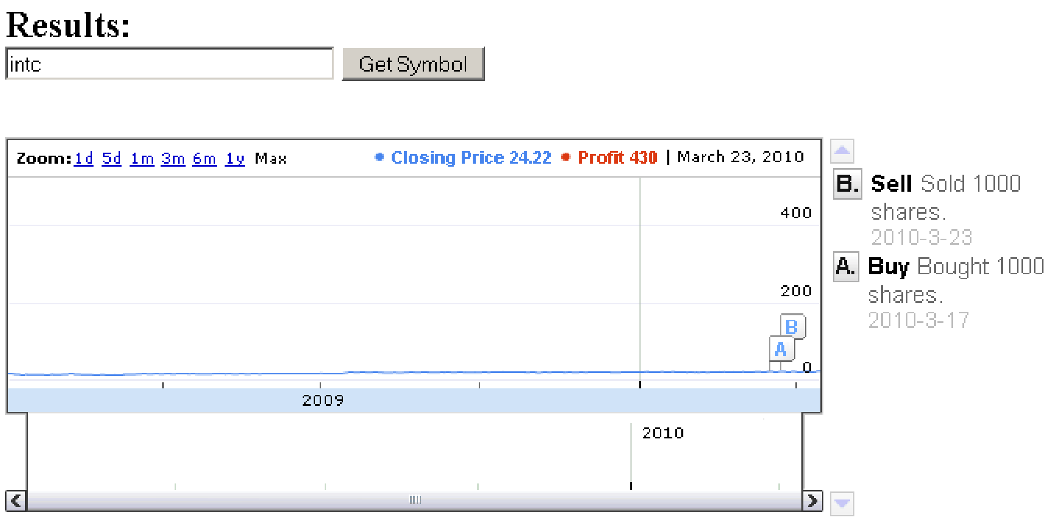

Results:
intc
Get Symbol
Zoom: 1d 5d 1m 3m 6m 1y Max
● Closing Price 24.22 ● Profit 430 | March 23, 2010
400
200
0
B
A
2009
2010
B. Sell Sold 1000 shares. 2010-3-23
A. Buy Bought 1000 shares. 2010-3-17

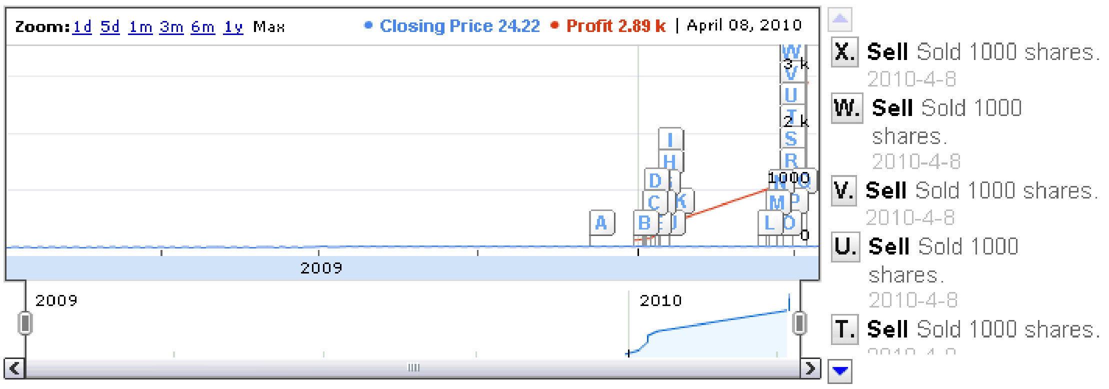
Figure 11: Chart using classified trade signals

From the chart, we see that if we followed the trade signals generated, we would have made $430. Note that, TSS assumes that the user has infinite income available.

TSS's strength lies in how easily we can modify the trading strategy and launch another run. Instead of using a 15 day period for SMA for one of the explanatory variables, let us change it to SMA(5). In this run, we see that only 39% of the test data was predicted correctly. However, we see in Figure 12 that we would have made $2,890 in profit.

Results:

intc Get Symbol



X. Sell Sold 1000 shares. 2010-4-8

W. Sell Sold 1000 shares. 2010-4-8

V. Sell Sold 1000 shares. 2010-4-8

U. Sell Sold 1000 shares. 2010-4-8

T. Sell Sold 1000 shares.

```
Logistic Regression with ridge parameter of 1.0E-8
Coefficients...
               Class
Variable       false
====================
SMA(15)      -0.0018
RSI(15)       0.0069
Intercept    -0.3836


Odds Ratios...
               Class
Variable       false
====================
SMA(15)       0.9982
RSI(15)       1.0069


Classifier correct percentage: 39.09090909090909
```

Figure 12: Chart using slightly modified trading strategy

Summary

In this report, we described the architecture and implementation of TSS - a system to evaluate trade strategies written in Java. We demonstrated TSS's viability through two case studies: one using technical indicators, and the other data mining tasks.

TSS can be extended in many ways.

FUTURE WORK

- Realistic Settings:
    - Liquidity: TSS ignores some fundamental trading issues. For example, we assume that there is an unlimited amount of stock volume as well as money to purchase the stocks. Thus we would like to add features for specifying the initial cash pool and only making trades when funds are available.
    - Slippage: When an order is filled, there can be a difference or spread between the price of what the user put the market order for, and the price that fills the order. This slippage is a feature we would like to add to TSS.
    - Trade Commissions: We would like to add the ability for a user to specify a trade commissions in TSS.
    - Cash Interest: In reality, funds that are held in cash will accumulate interest.
- When a trade strategy evaluation is complete, TSS does not report any positions that are still open. TSS should report any open positions to the user, and factor them into the profit.
- Multiple User Accounts: At the moment, a single user may use the system to generate evaluations. We would like to extend this such that multiple users may

submit trading strategies to TSS, and their performance can be reported in sortable order.

- TSS only supports the types of orders: buy at market value, and sell at market value. We would like to add additional order types such as buy at limit.
- TSS uses daily historical stock quote data. We would like to extend the model to use real-time intraday quote data, and allow us to run a trading strategy on it automatically.
- Once TSS has the ability to automatically run trading strategies, we would like TSS to alert users when a trade signal is identified.

Appendix

Method	Description
`void Series<T>(T input[])`	Creates a Series of type T from the inputted array.
`Series<Double> SMA(int period)`	Returns a Series of type Double that contains the Simple Moving Averages of the contained Series data using the specified period.
`Series<Double> EMA(int period)`	Returns a Series of type Double that contains the Exponential Moving Averages of the contained Series data using the specified period.
`T get(int index)`	Returns the value found at the specified index position.
`int size()`	Returns the size of the Series.
`Series<Double> RSI(int period)`	Returns a Series of type Double that contains the Relative Strength Index of the contained Series data using the specified period.
`Series<Double> TRIMA(int period)`	Returns a Series of type Double that contains the Triangular Moving Average of the contained Series data using the specified period.

Table 1: Methods for Series class

Method	Description
`Date getDate()`	Returns the trading Date for the quote.
`double getOpen()`	Return the trading day's opening price.
`double getHigh()`	Return the trading day's maximum stock price.
`double getLow()`	Return the trading day's minimum stock price.
`double getClose()`	Return the trading day's closing price.
`long getVolume()`	Return the trading day's volume.
`double getAdjClose()`	Returns the trading day's closing price adjusted for dividends and splits.

Table 2: Methods for Quote class

Method	Description
`Date date(int index)`	Returns the trading day's date for the specified trading day index.
`double open(int index)`	Returns the trading day's opening price for the specified trading day index.
`double low(int index)`	Returns the trading day's lowest price for the specified trading day index.
`double high(int index)`	Returns the trading day's highest price for the specified trading day index.
`double close(int index)`	Returns the trading day's closing price for the specified trading day index.
`long volume(int index)`	Returns the trading day's volume for the specified trading day index.
`double adjClose(ind index)`	Returns the trading day's adjusted closing price for the specified trading day index.
`Series<Date> dateSeries()`	Returns a Series class of all the Date's of the stock.
`Series<Double> openSeries()`	Returns a Series class of all the Opening prices of the stock.
`Series<Double> lowSeries()`	Returns a Series class of all the low's of the stock.
`Series<Double> highSeries()`	Returns a Series class of all the high's of the stock.
`Series<Long> volumeSeries()`	Returns a Series class of all the volume's of the stock.
`Series<Double> closeSeries()`	Returns a Series class of all the closing price's of the stock.
`public Series<Double> adjCloseSeries()`	Returns a Series class of all the adjusted closing price's of the stock.
`Quote getQuote(int index)`	Returns a Quote class for the specified trading day index.
`int size()`	Returns the number of trading day's available.
`String getSymbol()`	Returns the stock symbol for the Quotes class.
`int quotescount()`	Returns the number of Quotes (trading days) available.
`void Quotes(String symbol)`	Creates and populates a Quotes class with data for the specified stock symbol.

Table 3: Methods for Quotes class

Method	Description

`void BuildClassifier(`
`  String classifierName,`
`  Series[] in_data,`
`  String[] columnTitles,`
`  int start,`
`  int end)`
	Build the specified classifier with the provided training data.

`String[] Classify(`
`  Series[] in_data,`
`  String[] columnTitles,`
`  int start,`
`  int end)`
	Return the results of the classification on the provided test data.

`void PrintInformation()`	Print information about the built classifier.
`void PrintSource()`	If the classifier is sourcable, print the source code that implements the build classifier.
`void setDebugOutput(Boolean value)`	Enable or Disable printing of debug information during the building of the classifier.

Table 4: Methods for WekaClassifier class

Method Name	Description
`createDatabase`	Creates the database, deleting data that existed previously.
`addSymbol`	Add a symbol into the listofsymbols table. Create a table for the symbol and populate it with the symbol’s historical stock quote data.
`deleteSymbol`	Remove a symbol from the listofsymbols table and drop the table for the symbol.
`updateSymbol`	Populate the symbol’s table with historical stock quote data.
`GetDatabaseAsMap`	Get the database as a map from symbols (e.g., "XOM") to a list of QuoteRecords. These records are sorted in ascending order by date.

Table 7: Database manipulation methods

Bibliography

[1] Lo, et al., The Econometrics of Financial Markets

[2] Fidelity Back-Test Trading Strategies,

http://eresearch.fidelity.com/backtesting/landing

[3] Ameritrade StrategyDesk,

http://www.tdameritrade.com/tradingtools/strategydesk.html

[4] Scottrade Back-Testing Tool,

http://www.scottrade.com/scottradeelite_online_trading_platform/backtesting.asp

[5] Google Trends, http://www.google.com/trends

[6] Google Visualization API,

http://code.google.com/apis/visualization/interactive_charts.html

[7] Google News, http://news.google.com/

[8] Client-Server model, http://en.wikipedia.org/wiki/Client-server_model

[9] EditArea, http://www.cdolivet.com/editarea/

[10] HTTP Made Really Easy, http://www.jmarshall.com/easy/http

[11] TA-Lib, http://www.ta-lib.org/

[12] Weka, http://www.cs.waikato.ac.nz/ml/weka/

[13] SQLite, http://www.sqlite.org/

[14] Yahoo Finance, http://finance.yahoo.com/

[15] Logistic Regression, http://en.wikipedia.org/wiki/Logistic_regression

Vita

Salim K. Amirdache graduated from the University of Texas at Austin with a Bachelor of Science in Electrical Engineering in December of 2004. He began his career as an OpenGL Driver Developer at 3DLabs for workstation 3D graphics cards. Shortly after, he joined the Visual Computing Group at Intel Corp to work on the OpenGL Graphics Driver for the Larrabee project. In January of 2007, he entered the Graduate School at the University of Texas at Austin and enrolled in the Software Engineering Program.

Permanent Address:	8203 Ganttcrest Dr
	Austin, Texas, 78749

This report was typed by Salim K. Amirdache