DOROTHY: A FINANCIAL SIMULATOR

Salil Maharjan

CMPS 450-01: SENIOR PROJECT
Professor Miller
Ramapo College of New Jersey

06/05/2020

Table of Contents

1 INTRODUCING DOROTHY

Dorothy is a financial simulator written in C++. A financial simulator is a tool that uses trading algorithms to simulate the buying and selling of securities. It is useful for testing the performance of various trading strategies before deploying it into actual practice.

Financial simulator tests trading strategies by simulating trades over a period of time. Trading strategies help to predict the movement of securities so that profit can be made by buying and selling stocks at different prices. Dorothy uses the simple MACD (Moving Average Convergence Divergence) indicator as a trading strategy to trade securities. It is able to simulate trades with the help of this indicator using the available price data of the securities.

Dorothy's trading model can be configured by passing multiple configuration files that contain values for different trading parameters. It generates various reports and calculates the Sharpe ratio of the model after each simulation, which can be used to understand the performance of the trading model. These reports can be further used to optimize the configuration parameters and improve the performance of the trading model.

2 BACKGROUND INFORMATION

2.1 MACD (MOVING AVERAGE CONVERGENCE DIVERGENCE) INDICATOR

The MACD indicator is an oscillator-type trend-following momentum indicator that uses the relationship between two moving averages to determine a trade signal [1]. According to Investopedia, a moving average is a series of averages of different subsets of the entire data. For instance, a series of subsets of fifty data points gives the 50-day moving average of daily price data. Moving averages are commonly used for technical analysis because it helps to smoothen price data over a predefined period of time. It helps to gain helpful inferences like identifying the general trend direction of a security. Among the various types of moving averages, the Simple Moving Average (SMA) and Exponential Moving Average (EMA) are frequently used for technical analysis.

2.1.1 Simple Moving Average (SMA)

The Simple Moving Average is the simplest form of a moving average that weights all the data points equally [1]. It is a true indicator of the average price over a time period. It is calculated simply by taking the arithmetic mean of the data values. The formula for 'N'-day SMA is:

$$SMA_t = (P_{t-(N-1)} + P_{t-(N-1)+1} + \cdots + P_{t-1} + P_t)/N$$

Where:

t = The index of the current day, and P = Price data of a security.

2.1.2 Exponential Moving Average (EMA)

Exponential Moving Average assigns greater importance to the most recent data values and is more responsive to recent price changes than the SMA. The formula for 'N'-day EMA is:

$$EMA_t = EMA_{t-1} + \alpha * (P - EMA_{t-1})$$

Where:

t = The index of the current day.

$\alpha = \frac{2}{N+1}$ (A smoothing multiplier)

P = Price data of security.

The MACD indicator uses three different EMAs to generate a trade signal, which is used to identify potential buying and selling points. The typical settings for a MACD indicator are (12, 26, 9) and it is calculated as follows:

i)	Generate **MACD line =** (12-day EMA – 26-day EMA)
ii)	Generate **Signal line** = (9-day EMA of **MACD line**)
iii)	Generate MACD histogram = (**MACD line** – **Signal line**)

As the name suggests, the MACD indicator generates trade signals by looking at the convergence or divergence of the two moving averages: MACD and Signal line. The MACD indicator indicates a buy signal (entry point) when the MACD line crosses over the Signal line and it indicates a sell signal (exit point) when the MACD line crosses below the Signal line [1]. This helps to identify the market momentum and potential reversals so that trades can be made at the right time to make possible gains.

2.2 SHARPE RATIO

The Sharpe Ratio was developed by Nobel laureate William F. Sharpe and is used to understand the return of an investment compared to its risks [2]. Simple profit and loss is not a good way to rate the performance of a model because it does not give us an idea of the risk associated with the gain. As Investopedia explains, good investments are the returns that do not come with an excess of additional risks. The Sharpe ratio is therefore useful to gain a better understanding of the performance of a trading model as it takes both the gain and risk into account. It gives us a better idea about the strength and smoothness of the profit and loss in a model. There are several variations of the formula. This project uses the following formula to calculate the Sharpe ratio:

$$Sharpe\ Ratio = \left(\frac{252}{\sqrt{252}}\right) * \frac{Mean\ of\ daily\ ROR}{Standard\ deviation\ of\ daily\ ROR}$$

Where ROR (Rate of Return) is the net gain or loss of an investment over a time period. It is expressed as a percentage of the investment's initial cost and is calculated as:

$$ROR = \frac{(Current\ price - Initial\ price)}{Initial\ price} * 100$$

A Sharpe ratio of 1.0 or greater is typically considered good and a ratio of 2.0 is considered very good by investors [2]. This measure is used in Dorothy to rate the performance of trading simulations.

3 LANGUAGE CHOICE

This project is written in C++. Trading models can be built using many other programming languages but C++ is preferred by many as it is fast and has interfaces with many data provider APIs. As a statically compiled language, it is faster than other languages like C#, Java, Python, and so on.

C++ is widely used for simulation frameworks. Speed is an important factor to consider as running and optimizing simulations can be very time-consuming. It also has many libraries like the STL library which contains various containers, algorithms, and functions that are beneficial for creating a simulator.

4 INSTALLATION INSTRUCTIONS

Dorothy is a CMake-based project and can be installed with a minimum CMAKE version of 3.16. The project uses C++ 17 standards to compile. The file "*CMakeLists.txt*" contains all the information required to compile and run Dorothy.

Once in the project directory, the project can be built by using cmake to build an executable file named "*dorothy*". The following code can be pasted in the terminal in the working directory to create and run the executable:

```
cmake CMakeLists.txt
make
./dorothy
```

5 USER'S MANUAL

After compiling the project, the program can be started by running the executable file "*dorothy*" and passing configuration files as command-line arguments. Dorothy can handle multiple configuration files and can be passed as:

```
./dorothy config1.ini config2.ini …
configN.ini
```

5.1 CONFIGURATION FILE

Configuration files are used to initialize the parameters used in the simulator. A sample configuration file "*config.ini*" is shown in Figure 1.0 with comments beginning with '#' that explains the purpose of each parameter.

```
# [DIRECTORY DATA]
# DIRECTORY WHERE THE UNIVERSE CONSTITUENTS ARE LISTED.
# Universe constituents is the list of securities that will be used in the simulation.
UNIVERSE_DIRECTORY = ./data/constituents/universe.txt
# DIRECTORY WHERE THE PRICE DATA FOR ALL SECURITIES IS LOCATED.
TICKER_DATA_DIRECTORY = ./data/ticker

# [PORTFOLIO DATA]
PORTFOLIO_CAPITAL = 1000000         # TOTAL PORTFOLIO CAPITAL TO USE IN SIMULATION
MAX_CAPITAL_PER_STOCK = 250000      # MAXIMUM CAPITAL THAT CAN BE ALLOCATED FOR A SECURITY.

# [SIMULATION PERIOD]
# FORMAT: YYYYMMDD
START_DATE = 19970510               # SIMULATION START DATE
END_DATE = 20140510                 # SIMULATION END DATE

# [ENTRY CONDITION]
ENTRY_THRESHOLD = 0.42              # MAXIMUM SIGNAL THRESHOLD TO ENTER A POSITION

# [EXIT CONDITION]
EXIT_THRESHOLD = 0.32               # MAXIMUM SIGNAL THRESHOLD TO EXIT A POSITION (Smaller value is closer to a crossover)

# [TRADING PORTFOLIO]
MAX_POSITIONS_PER_STOCK = 6000      # MAXIMUM POSITION LIMIT PER STOCK
MAX_DAYS_IN_POSITION = 30           # MAXIMUM DAYS LIMIT TO STAY IN A POSITION
STOP_LOSS = 4.5                     # STOP LOSS PARAMETER

# [PUBLICATIONS]
# 1 TO GENERATE REPORT, 0 TO SKIP.
DAILY = 1                           # DAILY REPORT
MONTHLY = 1                         # MONTHLY REPORT
TRANSACTION = 1                     # TRANSACTION REPORT
SIGNAL = 1                          # SIGNAL REPORT

```

Figure 1.0: Sample configuration file.

5.2 CONSTITUENTS FILE

The text file with the universe constituents is simply a list of ticker names of the securities that will be included in the simulation. Figure 1.1 shows a sample universe constituents file. Dorothy checks if the ticker symbol is valid and if the data is available for the ticker symbol before running the simulation.

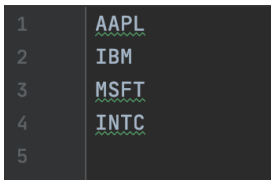


Figure 1.1: Sample constituent file.

6 PROJECT DESIGN AND DOCUMENTATION

This project consists of a utilities namespace, a precompiled header, and seven main classes. It follows an object-oriented, minimalistic, and functional design. A brief overview of all these components can be found in the sub-sections of Section 6. The detailed documentation of all the functions can be found in a separate PDF file named "dorothy_documentation.pdf" under the documentation folder. It also includes an HTML documentation version that is generated by Doxygen [3] using Latex. The HTML version can be viewed by going to the "html" folder under documentation and opening the "index.html" file.

6.1 NAMESPACES

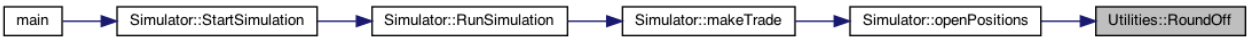
The project contains a single namespace to group utility functions that are used by the program.

6.1.1 Utilities

The utilities namespace encapsulates utility functions that the simulator requires for running various functions. The function documentation of all the functions including their caller graphs that shows the usage of these functions is below.

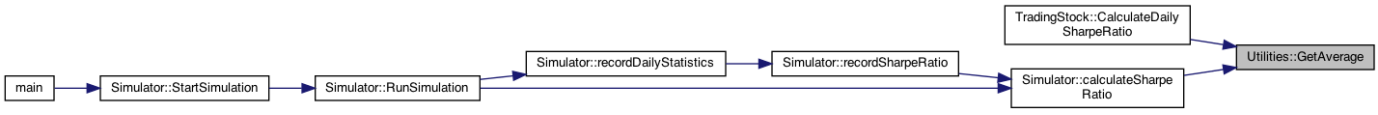
- **int RoundOff (double a_value)**

Function to round off value of '*a_value'* to the lower integral value.



- **double GetAverage (std::vector< double > a_list)**

Function to get average of the list with doubles '*a_list*'.

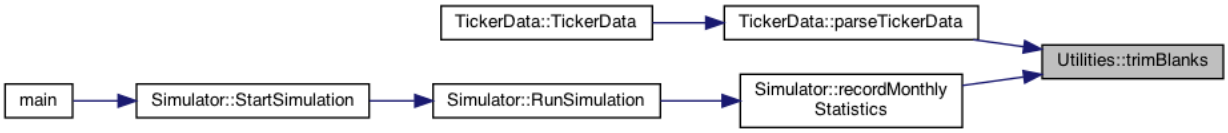


- **double GetStandardDeviation (std::vector< double > a_list, double a_average)**

Function to get the standard deviation of a list with doubles '*a_list*' that has an average value of '*a_average*'.

- **void trimBlanks (std::string &a_str)**

Function to trim leading and trailing blanks from string '*a_str*'.



6.2 PRECOMPILED HEADER

This project consists of a precompiled header file that consists of headers that are used in the program. Figure 2.0 shows the headers included in the precompiled header.

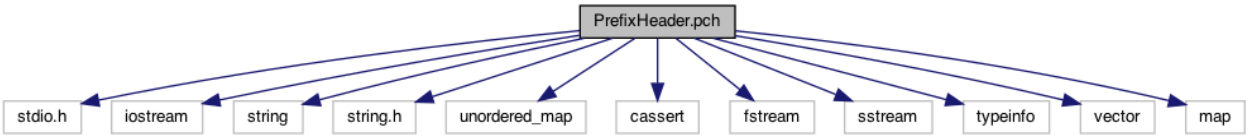


Figure 2.0: Headers included in the precompiled header.

The program utilizes the standard "*iostream*" for input and output to the terminal and the standard "*fstream*" for reading and writing from files. Vectors, maps, and unordered maps are used extensively to store data in various classes. The use of these containers in various classes is discussed in sub-section 6.3.

6.3 CLASSES

The documentation for the seven classes used in this program can be found in the subsections below.

6.3.1 Config

The config class is the main class that is responsible to load the configurations from the configuration file and store it so that it can be used in the simulation.

It contains a struct member variable called "ConfigValue" that contains a string variable to store the parameter value and a Boolean flag to record if the variable has been accessed or not. This is helpful to figure out if the parameters have been used in the simulation or not. The ConfigValue struct is further stored in an unordered map of string and ConfigValue, so that the access time is reduced. Figure 3.0 shows the collaboration diagram of the config class that shows the use of ConfigValue and unordered map. This class also has a template function called GetParameter

that can return the value of parameters as different types including string, int, short, long, float, double, char*, and booleans. This makes it easier to get the parameter values from the class and use them for simulation purposes.

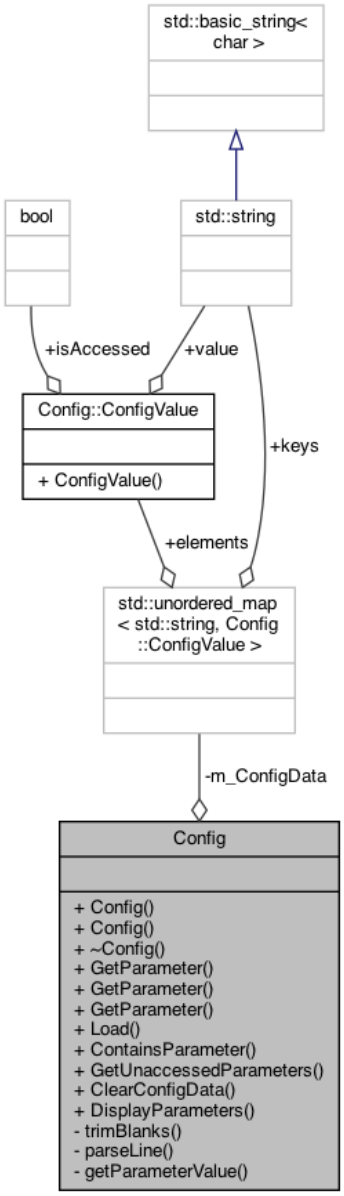


Figure 3.0: Collaboration diagram of Config class.

6.3.2 DateTime

Dates are used extensively for sequential data like price data. This is why we require a separate DateTime class that is able to handle and manipulate date values effectively. The DateTime class in this project stores the date internally as (Year * 10000 + 100 * Month + Day) as an integral value. This saves space as an entire date can be stored in four bytes. The class offers several parameterized constructors that can be used to create a DateTime object very conveniently. It contains an enumeration of the days of the week so that the days can be

identified and used efficiently. It provides several accessors and mutators by which date values can be used and changed easily. It contains a member variable that allows the class object to fake the date. This can be useful to fake the actual date for various purposes. The class also overloads all the major arithmetic and comparison operators including the unary operators. This adds a lot of functionality while working with date values. We can do various arithmetic operations with the date values and can also print out the date in a readable format using this class. The collaboration diagram for the DateTime class is shown in Figure 4.0

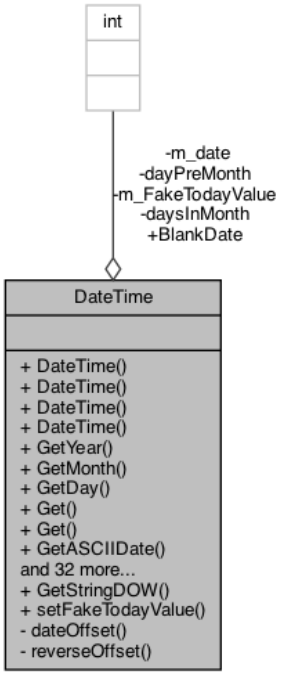


Figure 4.0: Collaboration diagram of DateTime class.

6.3.3 DataAccess

The DataAccess class is the main class that is responsible for accessing and storing the price data of the specified securities. The DataAccess class follows a singleton design pattern because price data can be very large and creating multiple instances of such classes can cause major problems. The class has private constructors and a designated static function called "*GetDbInstance*" to ensure that only a single instance of this class is created. The copy constructor is privatized and the assignment operator for the class is deleted for the same purpose. Figure 5.0 shows the collaboration diagram for the DataAccess class.

The DataAccess class is also responsible for loading the trading dates, which is used as a reference by the simulator to make trades. The IBM security is used to load the trading dates as a reference because it is one of the oldest securities with over a hundred years of trading data.

A map of string and the TickerData class object is used to store the price data for each ticker in this class. The price data for each security can be accessed using class the accessors provided in the class. The security is identified by its ticker name and the data can be quickly accessed by

using the map that has the ticker name as the string key and the price data as a TickerData object. The details about the TickerData class can be found in Section 6.3.4.

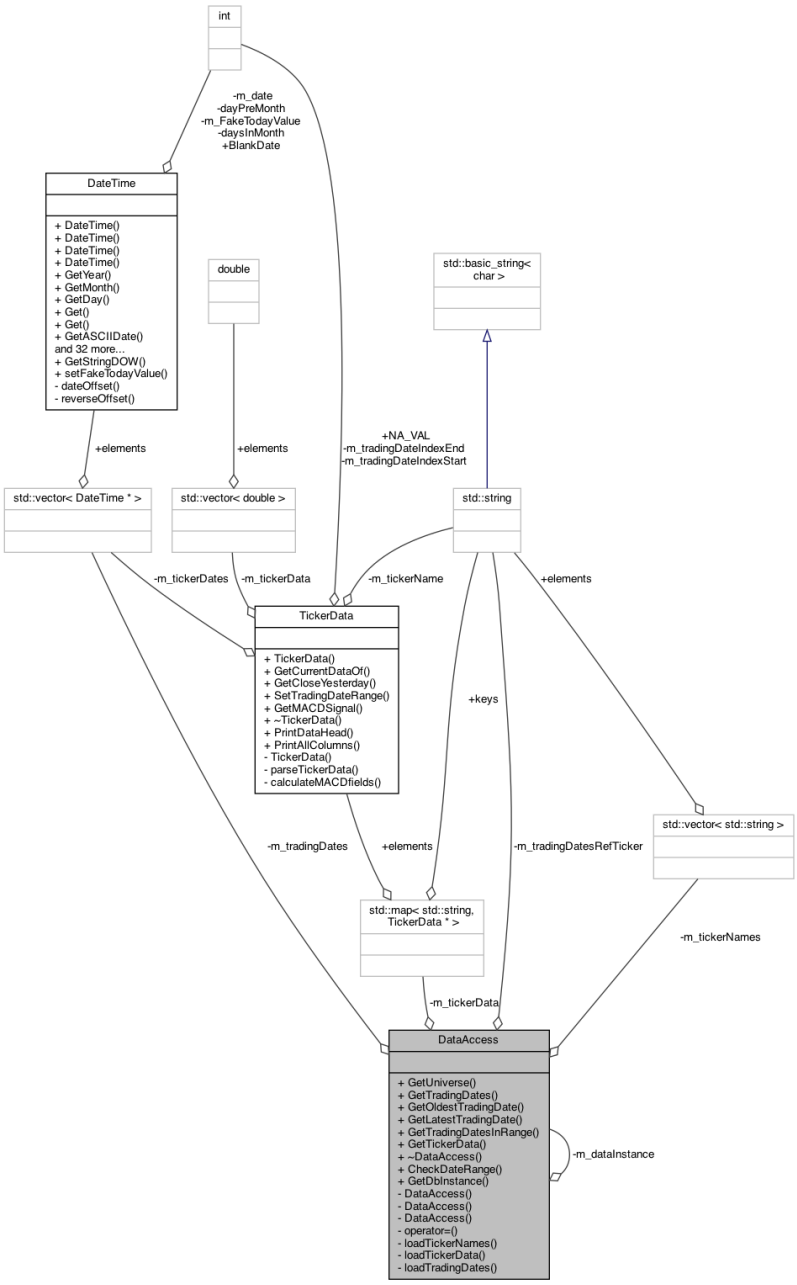


Figure 5.0: Collaboration graph of DataAccess class.

6.3.4 TickerData

The TickerData class is responsible for storing the price data for individual trading securities in the simulation. TickerData objects are stored in the DataAccess class to access the data for each ticker conveniently. As previously discussed in Section 6.3.3, a map with a string

key and TickerData value is used to access the price data for the individual securities. The actual price data in the TickerData is stored in an array of vectors with doubles. Initially, the program used a vector of vectors but it was replaced with an array of vectors as the row size is constant for all entries throughout the simulation. Since dynamic resizing is not required, the decision to change the vector of vectors was made. All of the data fields like open, close, adjusted close, etc. are stored in this array of vectors except the date field because it is a DateTime class object. The date field is stored separately in a vector of DateTime pointers. It is made sure that the index of the DateTime vector aligns with the actual price data while parsing in the price data so that the data is easily accessible. No mutators are provided by this class to make sure that the data is not changed. This class only provides two debugging functions to display a sample of the price data to make sure everything is loaded correctly.

TickerData is also responsible to fill in the calculated fields, which is used to get the trade signals. This class calculates the 12-day EMA, 26-day EMA, MACD Line, Signal Line, and the MACD Histogram values. The calculated fields are appended to the price data while creating a TickerData object. The moving averages are calculated while loading the price data so that the complexity of calculating the trade signal is reduced while trading. It is more time-efficient to have these values loaded along with the price data so that it is ready to use. This reduces additional code to calculate moving averages while performing trades and makes the trade functions cleaner. This design choice also reduces a lot of redundant recalculations as the calculated fields persist along with the price data while running multiple simulations. Even when there are ‘N’ configuration files, the price data and the calculated fields are only loaded once on the first simulation and the same object is used throughout the simulation. Object persistence is one of the major benefits of this design choice.

The array of vector is also squared by filling in unavailable data with the value of -999. This makes it easier to identify and skip unavailable data in the simulation. The collaboration graph for the TickerData class is shown in Figure 6.0.

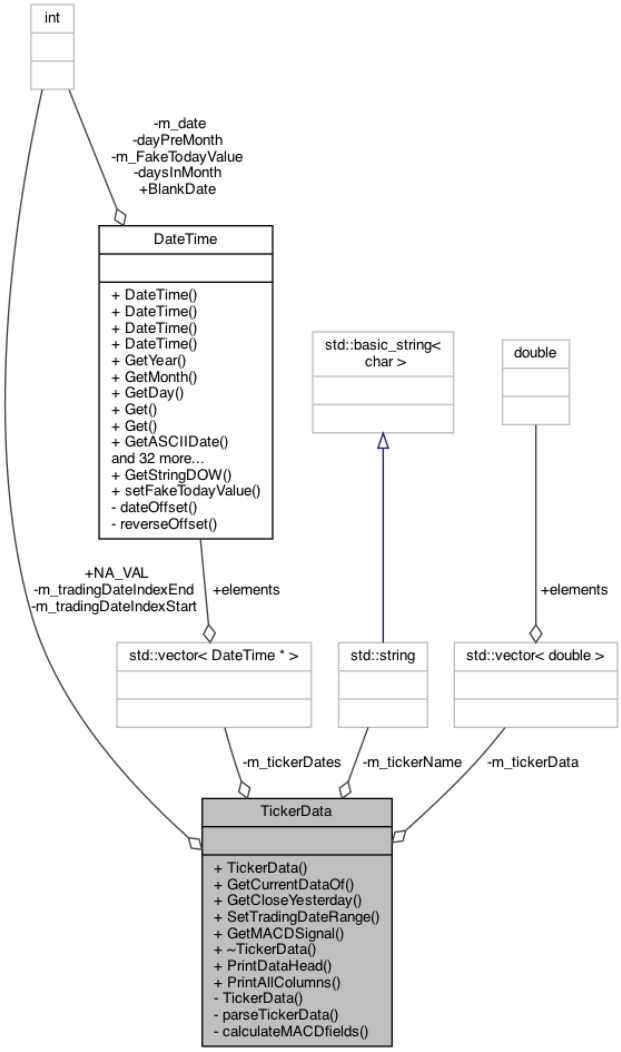


Figure 6.0: Collaboration graph of TickerData class.

6.3.5 Simulator

The Simulator class is the main container for the simulator that runs the simulation using all other components of the project. The comprehensive collaboration graph for this class is shown in Figure 7.0. The graph shows how all the components of the project are used by this class.

The Simulator class reads the configuration files and uses the Config class to store the values of different configuration parameters. It stores all the parameter identifiers that are used to identify configuration parameters as string constants. As previously mentioned, multiple configuration files can also be passed for simulation. It keeps a record of how many configuration files are passed and runs the simulation for all configurations passed. It runs the simulation from the specified date and uses the DataAccess class to get the price data for the simulation. It performs the actual trading by performing the necessary actions and sets a log of events that take place on each trading day. It is also responsible for generating publications like daily and monthly reports, transaction reports, and signal records by using the record of each trading day. It has a minimal

interface of three public functions including a parameterized constructor, using which a simulator object can be created and run easily. Figure 7.1 shows the call graph that shows how the simulator is initialized and Figure 7.2 shows the call graph that shows how the simulator runs using functions from all the components of the project. Figure 7.3 shows an extensive call graph of how the simulator is initialized, started, and restarted for multiple configuration files. The function StartSimulation is the main method that is called from the main entry point to run the complete simulation.

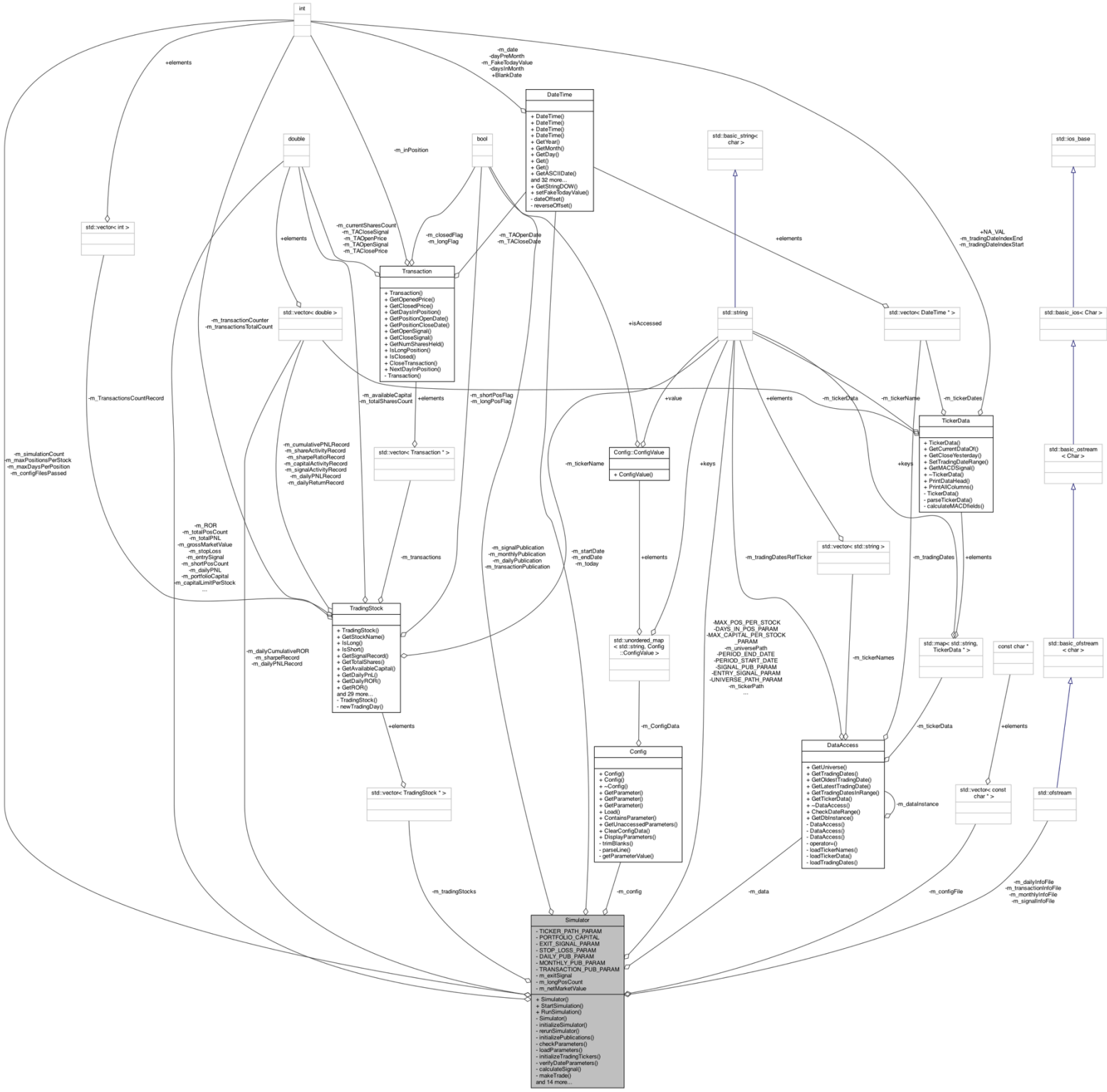


Figure 7.0: Collaboration graph of Simulator class.

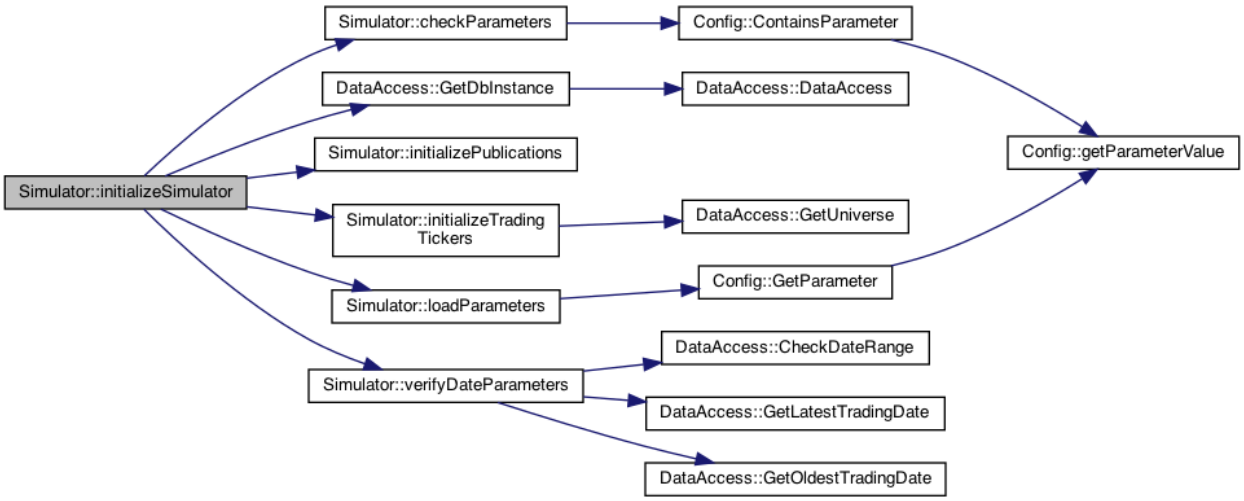


Figure 7.1: Call graph for initializing simulator.

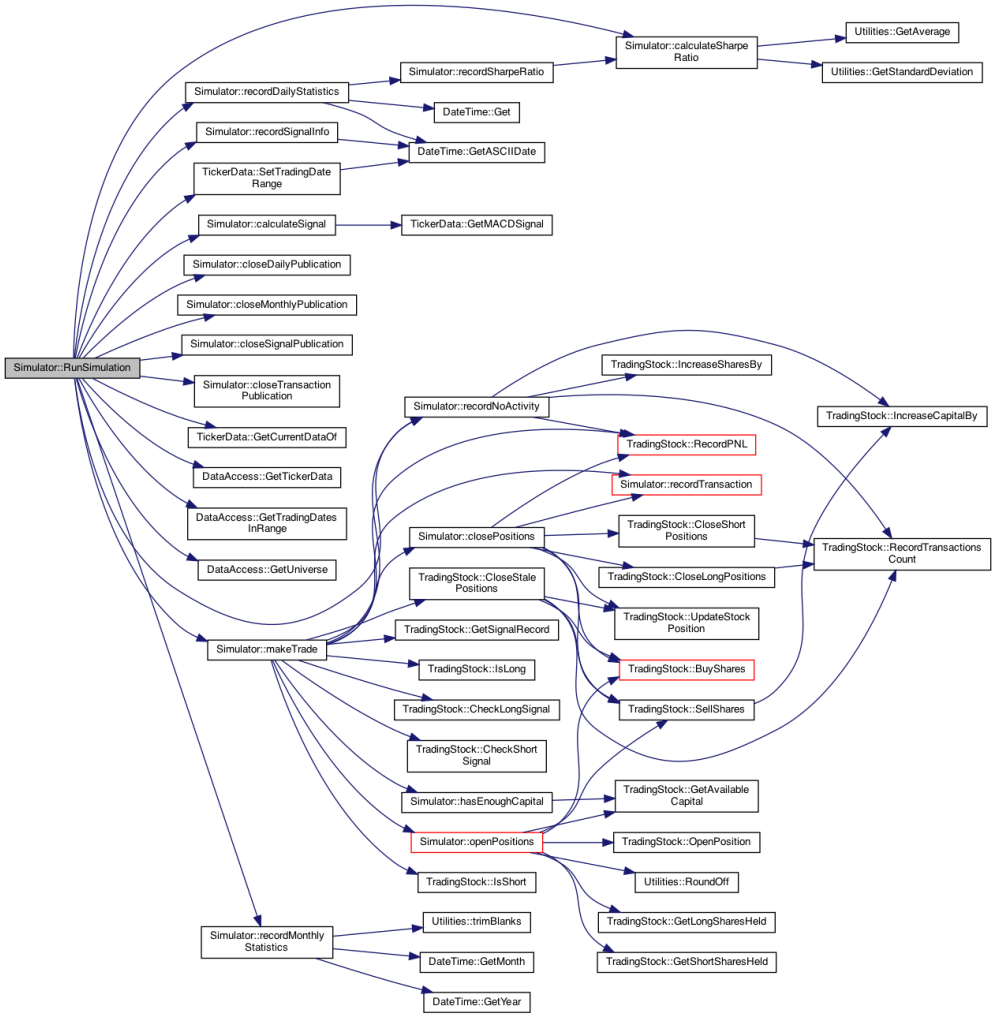


Figure 7.2: Call graph for running the simulator.

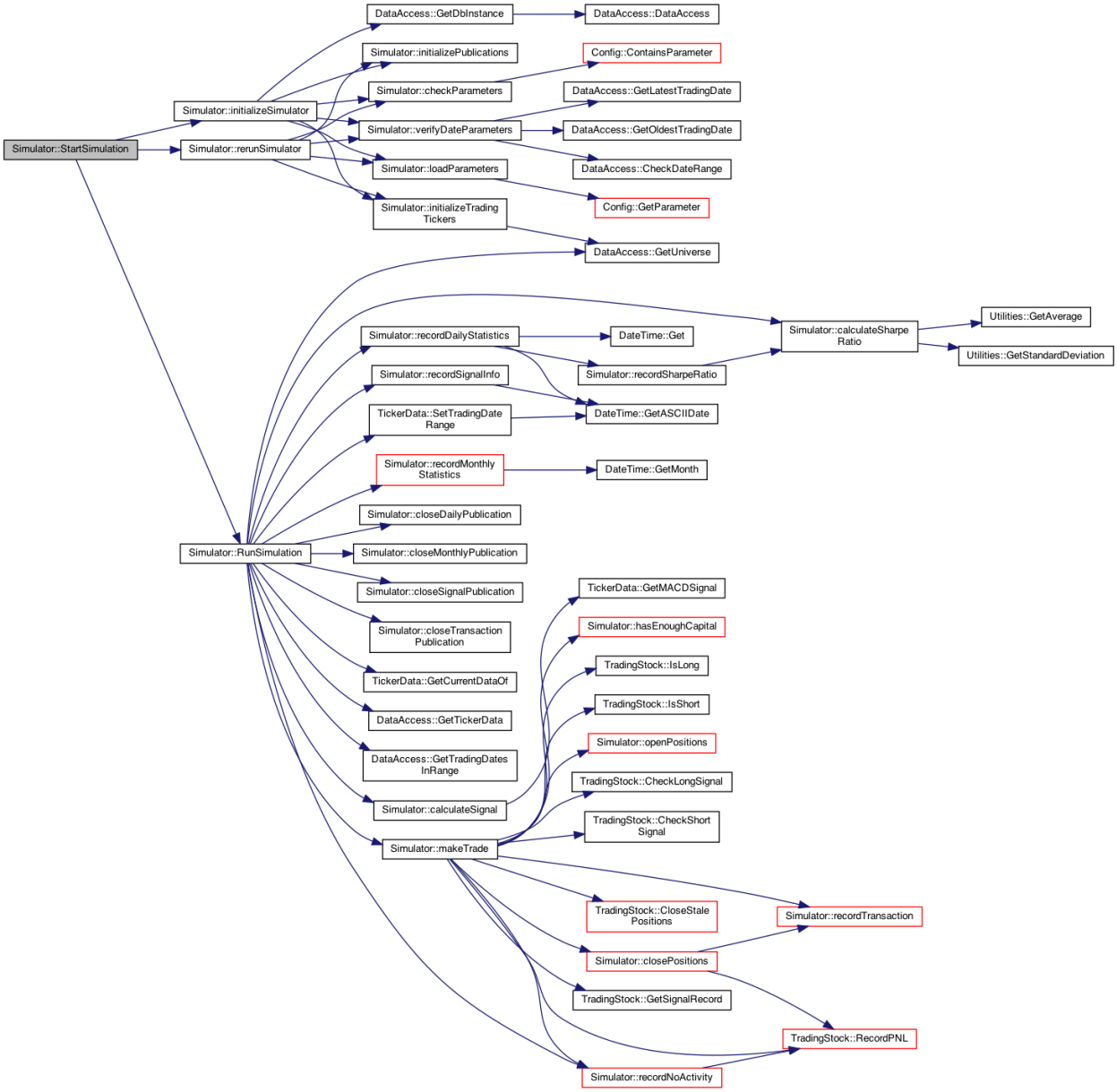


Figure 7.3: Comprehensive call graph from the main entry point to run the simulator.

6.3.6 TradingStock

The TradingStock class holds all the information about the security being traded in the simulation. It contains information about the investment portfolio of the trading security. It is initialized using the ticker name of the stock and the capital allocated for trading. It records all the details of the security like transactions, long/short positions, capital invested, total shares, etc. It also has a record of all the daily profit and loss, rate of return (ROR), and the trading signals. All of the daily records are stored using vectors and the class provides all the necessary accessors and mutators to get and store daily trading data for the security.

In order to keep transaction details like opening/closing price, date, or signal separate from security portfolio information, a separate Transaction class is introduced. The TradingStock object stores transactions using a vector of Transaction objects but the transaction details are handled separately by a Transaction class. This is discussed in Section 6.3.7.

The TradingStock class is responsible to open and close long/short positions when signaled by the simulator and it also closes positions that exceed the maximum day limit. It provides interface methods to buy/sell shares of the trading stock too. Additionally, a function to check the profitability of a trade signal is also included in the class. The discussion of this function can be found in Section 8.

TradingStock is an extensive class that stores all the necessary details required for trading securities. The collaboration graph of the TradingStock class is shown in Figure 8.0.

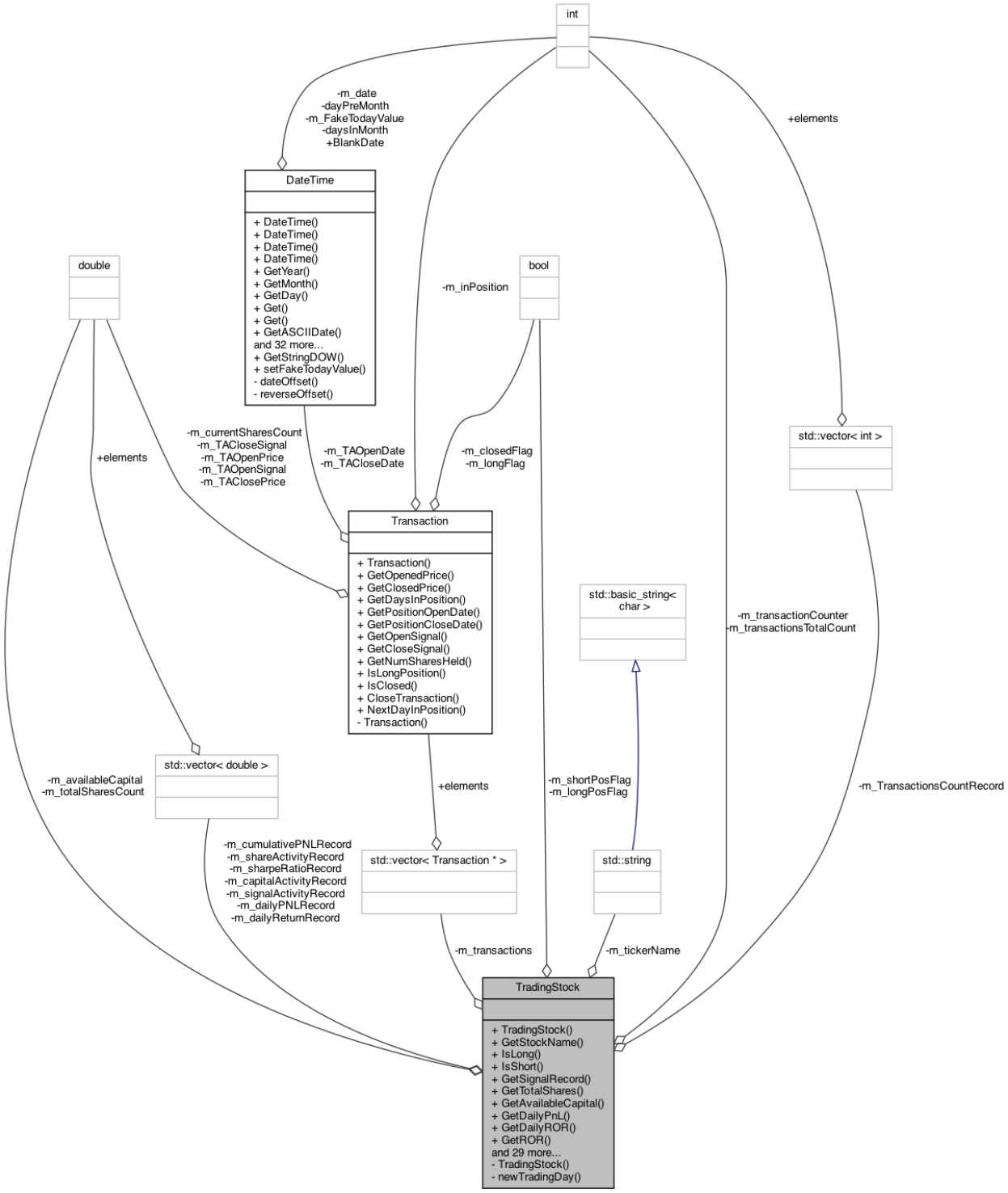


Figure 8.0: Collaboration graph of TradingStock class.

6.3.7 Transaction

The transaction class is used by the TradingStock class to record transactions when a position is opened or closed. It stores transaction details like opening/closing price, signals, date, etc. It provides accessors to view all the transaction details but no mutators to ensure that

transaction values are not changed. The member variables can only be initialized by opening and closing positions in the trading stock. The collaboration graph for the class is shown in Figure 9.0.

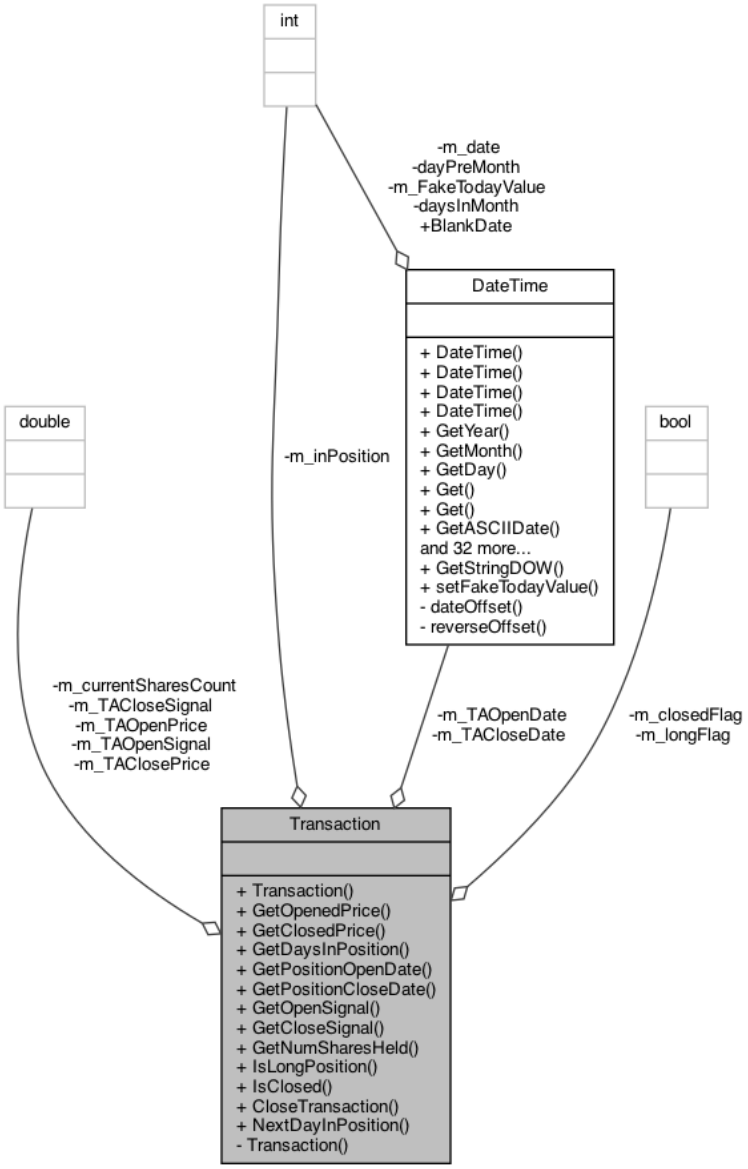


Figure 9.0: Collaboration graph of Transaction class.

7 PROJECT DEMO

The subsections below show a demo of simulating two different configuration files for four well-known stocks: Apple (AAPL), Microsoft (MSFT), IBM (IBM), and Intel (INTC). A false ticker (ASDF) is added to the constituent file to show how Dorothy handles it.

7.1 CONFIGURATION

Figure 10.0 shows the configuration parameters used in the simulation from the two configuration files. The first simulation is done for five years and the second for ten years. The main difference is that the first configuration allows 500 positions per stock and the second configuration only allows 100 positions per stock.

```
# [DIRECTORY DATA]
UNIVERSE_DIRECTORY = ./data/constituents/universe.txt
TICKER_DATA_DIRECTORY = ./data/ticker

# [PORTFOLIO DATA]
PORTFOLIO_CAPITAL = 1000000         # Total capital
MAX_CAPITAL_PER_STOCK = 250000
# ( maybe increase max capital per stock? )

# [SIMULATION PERIOD]
# FORMAT: YYYYMMDD
START_DATE = 20091205
END_DATE = 20141205

# [ENTRY CONDITION]
# Maximum strength threshold to enter
ENTRY_THRESHOLD = 0.32

# [EXIT CONDITION]
# Maximum strength requirement to exit. (Smaller value c
EXIT_THRESHOLD = 0.23

# [TRADING PORTFOLIO]
MAX_POSITIONS_PER_STOCK = 500
MAX_DAYS_IN_POSITION = 15
STOP_LOSS = 3.1

# [PUBLICATIONS]
# 1 for generation, 0 for not.
DAILY = 1
MONTHLY = 1
TRANSACTION = 1
SIGNAL = 1
```

Config1

```
# [DIRECTORY DATA]
UNIVERSE_DIRECTORY = ./data/constituents/universe.txt
TICKER_DATA_DIRECTORY = ./data/ticker

# [PORTFOLIO DATA]
PORTFOLIO_CAPITAL = 1000000         # Total capital
MAX_CAPITAL_PER_STOCK = 250000
# ( maybe increase max capital per stock? )

# [SIMULATION PERIOD]
# FORMAT: YYYYMMDD
START_DATE = 19941205
END_DATE = 20141205

# [ENTRY CONDITION]
# Maximum strength threshold to enter
ENTRY_THRESHOLD = 0.45

# [EXIT CONDITION]
# Maximum strength requirement to exit. (Smaller value closer to crossove
EXIT_THRESHOLD = 0.21

# [TRADING PORTFOLIO]
MAX_POSITIONS_PER_STOCK = 100
MAX_DAYS_IN_POSITION = 15
STOP_LOSS = 3.1

# [PUBLICATIONS]
# 1 for generation, 0 for not.
DAILY = 1
MONTHLY = 1
TRANSACTION = 1
SIGNAL = 1
```

Config2

Figure 10.0: Screenshot of two configuration files.

7.2 SIMULATION

Figure 11.0 shows the results of running Dorothy using the terminal by passing two configuration files to the executable. The first message shows how many configurations were passed and then the first simulation run is shown. Dorothy first prints out all the configuration parameters loaded from the configuration file. It then prints out the head of the price data that was loaded for verification purposes. We can see the error message printed out for the false ticker (ASDF) in Figure 12.0. The simulator continues to run for the remaining tickers.

```
**********************************************
salilmaharjan@MacBook-Pro dorothy % ./dorothy config1.ini config2.ini
----------------------------------------------------------------------
Dorothy: Running simulator for 2 different configurations passed.
----------------------------------------------------------------------

------------------------------------------------------------
Dorothy: Simulation #1
------------------------------------------------------------

Dorothy: Configurations loaded successfully.
****************************************
Dorothy - Configuration Parameters:
[1]   DAILY
[2]   END_DATE
[3]   ENTRY_THRESHOLD
[4]   EXIT_THRESHOLD
[5]   MAX_CAPITAL_PER_STOCK
[6]   MAX_DAYS_IN_POSITION
[7]   MAX_POSITIONS_PER_STOCK
[8]   MONTHLY
[9]   PORTFOLIO_CAPITAL
[10]   SIGNAL
[11]   START_DATE
[12]   STOP_LOSS
[13]   TICKER_DATA_DIRECTORY
[14]   TRANSACTION
[15]   UNIVERSE_DIRECTORY
****************************************
Dorothy: Loading data for constituents in universe...

****************************************
Dorothy - Tickers Loaded:
------------------------------------------------------------------------------------------------------------
[ 1 ]  AAPL
------------------------------------------------------------------------------------------------------------
DATE    OPEN    CLOSE   LOW     CLOSE   VOLUME  ADJ_CLOSE       DIVIDEND        SPLIT   VWAP    SHARE_OUTSTANDING
12/5/2014       115.99  116.08  114.64  115     3.78927e+07     115     0       1       115.495 -999
12/4/2014       115.77  117.2   115.29  115.49  4.20445e+07     115.49  0       1       115.63  -999
12/3/2014       115.75  116.35  115.11  115.93  4.30634e+07     115.93  0       1       115.84  -999
12/2/2014       113.5   115.75  112.75  114.63  5.90751e+07     114.63  0       1       114.065 -999
12/1/2014       118.81  119.25  111.27  115.07  8.3814e+07      115.07  0       1       116.94  -999
11/28/2014      119.27  119.4   118.05  118.93  2.48144e+07     118.93  0       1       119.1   -999
11/26/2014      117.94  119.1   117.83  119     4.07683e+07     119     0       1       118.47  -999
11/25/2014      119.07  119.75  117.45  117.6   6.88404e+07     117.6   0       1       118.335 -999
11/24/2014      116.85  118.77  116.62  118.63  4.74508e+07     118.63  0       1       117.74  -999
11/21/2014      117.51  117.57  116.03  116.47  5.71793e+07     116.47  0       1       116.99  -999


------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
[ 2 ]  IBM
------------------------------------------------------------------------------------------------------------
DATE    OPEN    CLOSE   LOW     CLOSE   VOLUME  ADJ_CLOSE       DIVIDEND        SPLIT   VWAP    SHARE_OUTSTANDING
12/5/2014       163.61  164.5   162.91  163.27  3.0104e+06      163.27  0       1       163.44  -999
12/4/2014       164.01  164.5   163.01  164.05  3.8594e+06      164.05  0       1       164.03  -999
12/3/2014       162.47  164.52  162     164.52  6.4291e+06      164.52  0       1       163.495 -999
12/2/2014       162.47  162.73  161.64  162.67  3.4641e+06      162.67  0       1       162.57  -999
12/1/2014       161.64  163.32  161.35  161.54  4.1684e+06      161.54  0       1       161.59  -999
11/28/2014      162.75  163.37  161.44  162.17  2.4055e+06      162.17  0       1       162.46  -999
11/26/2014      161.93  162.1   161.01  161.95  3.966e+06       161.95  0       1       161.94  -999
11/25/2014      162.65  163.5   161.56  161.76  4.0623e+06      161.76  0       1       162.205 -999
11/24/2014      161.54  163.86  161.06  162.15  6.6185e+06      162.15  0       1       161.845 -999
```

Figure 11.0: A terminal run of Dorothy by passing two configuration files.

The first configuration simulates for 5 years and generates $1,259,993.42 in profit with a Sharpe ratio of 4.72. This is shown in Figure 12.0. The second configuration simulates for 10 years and generates a lower profit of 709,671.00 but has a higher Sharpe ratio of 5.34. This is shown in Figure 13.0. One of the main reasons why the profits are lower for the longer time period is because the position limit is considerably less. We can see from the transaction report in Figure 18.0 that the position limit played a major role to limit the profits. However, this is also one of

the major reasons why the Sharpe ratio is considerably higher as the second simulation does not commit too many positions in a single security. Therefore, over time the profits become much smoother.

```
------------------------------------------------------------------------------------------------------------------
Dorothy Error (LoadTickerData): Unable to load price data for ticker [ASDF]
[ 3 ]  ASDF(FAILED)
------------------------------------------------------------------------------------------------------------------
[ 3 ]  INTC
------------------------------------------------------------------------------------------------------------------
DATE    OPEN    CLOSE   LOW     CLOSE   VOLUME  ADJ_CLOSE       DIVIDEND        SPLIT   VWAP    SHARE_OUTSTANDING
12/5/2014       37.57   37.9    37.52   37.67   2.0516e+07      37.67   0       1       37.62   -999
12/4/2014       37.4    37.46   37.04   37.46   2.33591e+07     37.46   0       1       37.43   -999
12/3/2014       37.68   37.89   37.41   37.43   3.06608e+07     37.43   0       1       37.555  -999
12/2/2014       37.18   37.6    37.18   37.6    2.83437e+07     37.6    0       1       37.39   -999
12/1/2014       37.21   37.62   36.9    37.17   3.0904e+07      37.17   0       1       37.19   -999
11/28/2014      37.04   37.69   36.94   37.25   1.91285e+07     37.25   0       1       37.145  -999
11/26/2014      36.37   36.99   36.28   36.9    2.406e+07       36.9    0       1       36.635  -999
11/25/2014      36.36   36.48   36.15   36.32   2.90885e+07     36.32   0       1       36.34   -999
11/24/2014      35.89   36.43   35.84   36.25   3.33179e+07     36.25   0       1       36.07   -999
11/21/2014      35.98   36.46   35.5    35.59   5.67204e+07     35.59   0       1       35.785  -999


------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------
[ 4 ]  MSFT
------------------------------------------------------------------------------------------------------------------
DATE    OPEN    CLOSE   LOW     CLOSE   VOLUME  ADJ_CLOSE       DIVIDEND        SPLIT   VWAP    SHARE_OUTSTANDING
12/5/2014       48.82   48.97   48.38   48.42   2.608e+07       48.42   0       1       48.62   -999
12/4/2014       48.39   49.06   48.2    48.84   3.03204e+07     48.84   0       1       48.615  -999
12/3/2014       48.44   48.5    47.81   48.08   2.35348e+07     48.08   0       1       48.26   -999
12/2/2014       48.84   49.05   48.2    48.46   2.5743e+07      48.46   0       1       48.65   -999
12/1/2014       47.88   48.78   47.71   48.62   3.11916e+07     48.62   0       1       48.25   -999
11/28/2014      47.95   48.2    47.61   47.81   2.15344e+07     47.81   0       1       47.88   -999
11/26/2014      47.49   47.99   47.28   47.75   2.71636e+07     47.75   0       1       47.62   -999
11/25/2014      47.66   47.97   47.45   47.47   2.80079e+07     47.47   0       1       47.565  -999
11/24/2014      47.99   48      47.39   47.59   3.54342e+07     47.59   0       1       47.79   -999
11/21/2014      49.02   49.05   47.57   47.98   4.28831e+07     47.98   0       1       48.5    -999


------------------------------------------------------------------------------------------------------------------
****************************************
Dorothy: Loaded 4 tickers successfully.
Dorothy: Simulator initialized for trading 4 stocks.
Dorothy: Running simulation...
Dorothy: Simulating for 1259 days.
Dorothy: Simulation completed successfully.
Dorothy run statistics:
Total profits: $1259993.42
Simulation Sharpe Ratio: 4.74
**************************************************
Dorothy - Publications Reports:
        * Daily Report successfully generated.
        * Monthly Report successfully generated.
        * Transaction Report successfully generated.
        * Signal Report successfully generated.
**************************************************
------------------------------------------------------------
Dorothy: Simulation #2
------------------------------------------------------------

Dorothy: Configurations loaded successfully.
****************************************
Dorothy - Configuration Parameters:
[1]   DAILY
[2]   END_DATE
```

Figure 12.0: A terminal output of the first simulation.

Figure 13.0: A terminal output of the second simulation.

In the second simulation, there is no price data head printed because the same data object is used for the second simulation and it does not need to be reinitialized as discussed in Section 6.3.4. At the end of each simulation, a publication report status is generated that shows the status of the recording the publication to files.

7.3 PUBLICATIONS

The subsections below contain screenshots of the publication files that are generated from the simulation.

7.3.1 Daily Publication

YYYYMMDD	Date	Short Positions	Long Positions	Total Positions	Daily PnL	Cumulative PnL	ROR	Market Value	Net Market Value	Sharpe
20091207	12/7/2009	0	0	0	0	0	0	0	0	0
20091208	12/8/2009	0	1000	1000	0	0	0	158335	158335	0
20091209	12/9/2009	0	1000	1000	4760	4760	0.0543049	24860	24860	11.225
20091210	12/10/2009	-1000	1000	2000	0	4760	0	187895	-138025	9.16515
20091211	12/11/2009	-1000	1000	2000	70	4830	0.00471222	187050	-137955	8.78494
20091214	12/14/2009	-1000	1000	2000	0	4830	0	188500	-137955	7.82236
20091215	12/15/2009	-1000	1000	2000	85	4915	0.00569514	186240	-137870	7.91666
20091216	12/16/2009	-1000	1000	2000	1015	5930	0.0119981	186610	-136855	8.76629
20091217	12/17/2009	-1000	1000	2000	0	5930	0	183965	-136855	8.12837
20091218	12/18/2009	-1000	1000	2000	400	6330	0.00621552	186665	-136455	8.32806
20091221	12/21/2009	-1000	500	1500	290	6620	0.0206536	178700	-146210	9.65346
20091222	12/22/2009	-1500	0	1500	150	6770	0.00982962	175165	-171490	10.1193
20091223	12/23/2009	-1500	0	1500	0	6770	0	176100	-171490	9.57382
20091224	12/24/2009	-2000	0	2000	0	6770	0	195470	-186990	9.10801

Figure 14.0: Daily report of the first simulation

YYYYMMDD	Date	Short Positions	Long Positions	Total Positions	Daily PnL	Cumulative PnL	ROR	Market Value	Net Market Value	Sharpe
19941205	12/5/1994	0	0	0	0	0	0	0	0	0
19941206	12/6/1994	0	200	200	0	0	0	13612	13612	0
19941207	12/7/1994	-200	200	400	0	0	0	23563	3649	0
19941208	12/8/1994	-100	200	300	75	75	0.020475	19537	7312	9.16515
19941209	12/9/1994	-100	300	400	0	75	0	23362	10937	7.93725
19941212	12/12/1994	-100	300	400	0	75	0	23062	10937	7.0993
19941213	12/13/1994	-100	300	400	13	88	0.00358621	22938	10950	7.72359
19941214	12/14/1994	-100	200	300	150	238	0.0412314	19388	7312	9.16357
19941215	12/15/1994	-200	200	400	0	238	0	22974	3600	8.48387
19941216	12/16/1994	-200	200	400	0	238	0	23025	3600	7.93597
19941219	12/19/1994	-200	200	400	38	276	0.00603175	23117	3638	8.26511
19941220	12/20/1994	-200	200	400	0	276	0	23038	3638	7.82535
19941221	12/21/1994	-200	100	300	312	588	0.0458108	16250	-3412	9.10016
19941222	12/22/1994	-300	100	400	0	588	0	23586	-10762	8.66799
19941223	12/23/1994	-300	100	400	0	588	0	23619	-10762	8.29207
19941227	12/27/1994	-300	100	400	75	663	0.0121951	23748	-10687	8.8409
19941228	12/28/1994	-300	0	300	-200	463	-0.0310078	17312	-17137	5.45905

Figure 15.0: Daily report of the second simulation

The daily report consists of two dates. The first date is formatted according to how date value is stored internally in the simulator and the second follows the standard MM/DD/YYYY format. It contains Short/Long and Total positions, Daily and Cumulative Profit and Loss, Rate of Return, Gross and Net Market Value, and also the Sharpe Ratio.

7.3.2 Monthly Publication

MM/YYYY	Short Positions	Long Positions	Total Positions	Monthly PnL	Cumulative PnL	ROR	Market Value	Net Market Value	Sharpe
12/2009	-2000	0	2000	6790	6790	0.114699	196255	-186970	7.86142
1/2010	-500	1500	2000	17420	24210	0.295813	181015	-7785	8.5791
2/2010	-2000	0	2000	-3770	20440	-0.207968	190490	-186640	1.82028
3/2010	-1000	1000	2000	18670	39110	0.330943	207415	48640	3.54306
4/2010	-500	1500	2000	7320	46430	0.210356	221735	-28800	4.26552
5/2010	-1000	500	1500	27375	73805	0.260239	201780	-67580	4.62026
6/2010	0	1500	1500	12420	86225	0.198107	199010	209180	4.64145
7/2010	-1500	500	2000	3140	89365	0.291448	216030	-187565	5.01634
8/2010	-1500	500	2000	11685	101050	0.0100622	203685	39050	4.54341
9/2010	-1000	500	1500	26035	127085	0.369493	88915	-63425	4.7202
10/2010	-1000	500	1500	23850	150935	0.27071	95160	47830	5.02245
11/2010	-500	1000	1500	13855	164790	0.295609	236885	98565	5.32127
12/2010	-1500	0	1500	7440	172230	0.124528	248615	-249120	5.30059
1/2011	-1000	1000	2000	9285	181515	0.129937	275255	93040	5.31406

Figure 16.0: Monthly report of the first simulation

MM/YYYY	Short Positions	Long Positions	Total Positions	Monthly PnL	Cumulative PnL	ROR	Market Value	Net Market Value	Sharpe
12/1994	-400	0	400	463	463	0.0983225	23750	-23575	5.1319
1/1995	-100	200	300	2895	3358	0.494115	17187	5812	9.4772
2/1995	-200	100	300	1339	4697	0.209171	17775	-5125	7.9059
3/1995	-200	200	400	2989	7686	0.502013	27337	-5613	9.43736
4/1995	-300	0	300	3536	11222	0.484665	22238	-21749	9.51141
5/1995	-100	300	400	2455	13677	0.345998	33150	16800	8.84848
6/1995	-100	200	300	-1485	12192	0.0477166	23282	4513	4.9064
7/1995	-200	200	400	6490	18682	0.782529	30938	-2788	5.62025
8/1995	-300	100	400	2434	21116	0.254932	30026	-17688	5.53465
9/1995	-300	100	400	3649	24765	0.524457	28237	-9312	5.93568
10/1995	-300	0	300	4674	29439	0.649957	26713	-26713	6.44691
11/1995	-200	200	400	2668	32107	0.385107	28275	3250	6.62328
12/1995	-200	200	400	732	32839	0.111266	26775	-2124	6.18137
1/1996	-200	200	400	2300	35139	0.318878	28386	-2913	6.05594
2/1996	-300	100	400	4219	39358	0.467152	30762	-11055	6.04596
3/1996	-100	300	400	3148	42506	0.480067	29581	9369	6.19176
4/1996	-300	100	400	2435	44941	0.322246	31313	-7550	6.24573
5/1996	0	100	100	3733	48674	0.625325	10675	11100	6.42338

Figure 17.0: Monthly report of the second simulation

The monthly report consists of the same columns as the daily report. The date is recorded in MM/YYYY format since it is a monthly report.

7.3.3 Transaction Publication

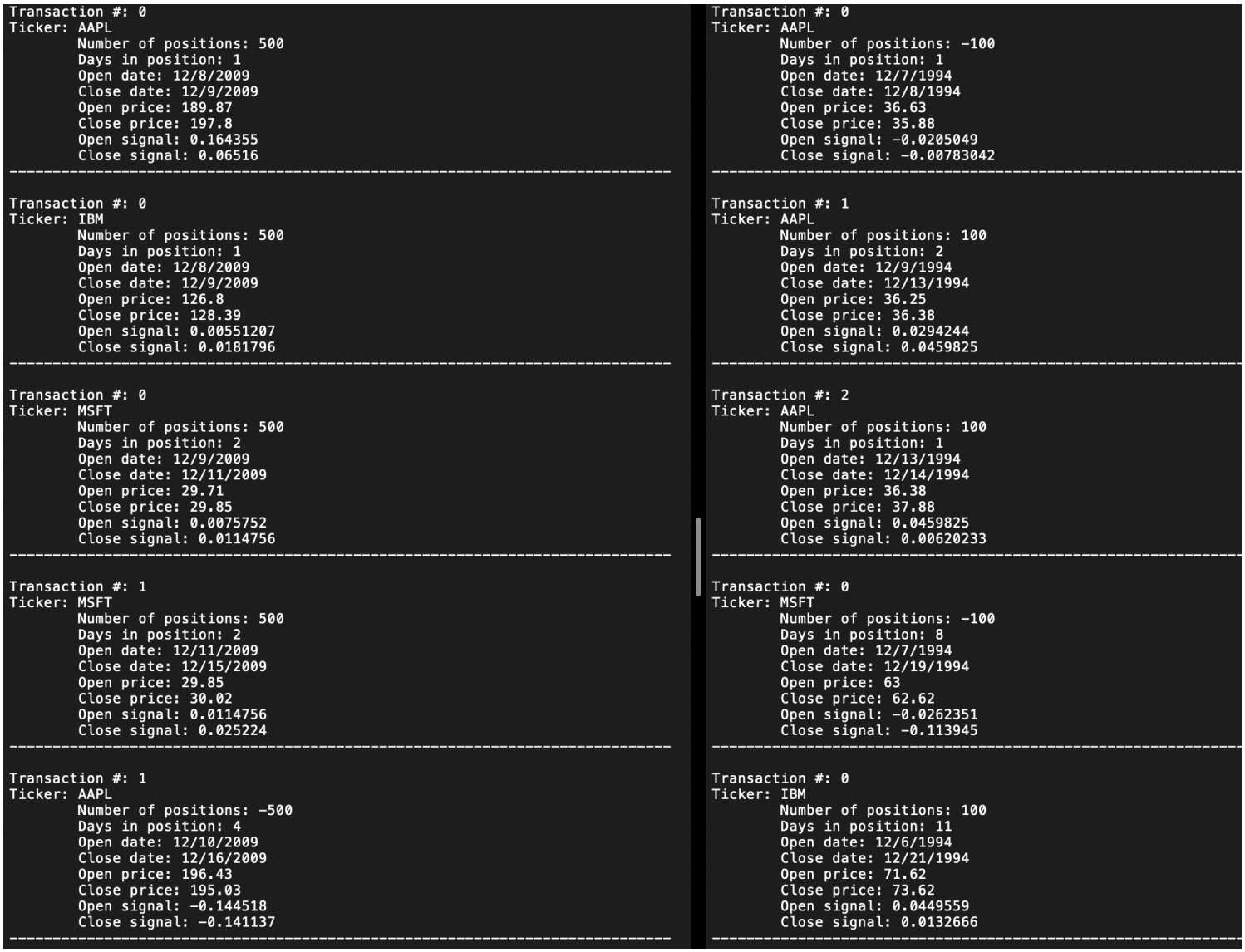


Figure 18.0: Transaction report of the first and second simulation.

7.3.4 Signal Publication

Date	Signal
12/7/2009	-0.22031
12/7/2009	-0.040222
12/7/2009	-0.00537829
12/7/2009	-0.00612889
12/8/2009	0.164355
12/8/2009	0.00551207
12/8/2009	-0.00717299
12/8/2009	-0.00328227
12/9/2009	0.06516
12/9/2009	0.0181796
12/9/2009	0.011444
12/9/2009	0.0075752
12/10/2009	-0.144518
12/10/2009	-0.0205235
12/10/2009	-0.00837056

Date	Signal
12/5/1994	0.0243389
12/5/1994	-0.00420909
12/5/1994	-0.0267673
12/5/1994	0.00412036
12/6/1994	0.00640212
12/6/1994	0.0449559
12/6/1994	0.0725278
12/6/1994	0.0141534
12/7/1994	-0.0205049
12/7/1994	-0.069959
12/7/1994	-0.0690193
12/7/1994	-0.0262351
12/8/1994	-0.00783042
12/8/1994	0.0411309
12/8/1994	0.0689936

Figure 19.0: Signal report of the first and second simulation

8 SUMMARY AND CONCLUSIONS

Dorothy has been tested on a variety of securities and has consistently shown positive results. The Sharpe ratio is significantly high for a simulator that simply uses the MACD indicator to generate trade signals. This is because of a feature that was added to the simulator to test the profitability of a trade signal. This feature is added to the TradingStock class and it checks whether the buy or sell signal generated by the MACD indicator is profitable or not. It does this by checking and comparing the opening price of the open positions to the current price whenever a trade signal is generated. The Sharpe ratio before adding this feature ranged from -0.5 to 1.0. The transaction reports showed that there were many false signals generated by the indicator. Therefore, this feature was added to the simulator because it was performing poorly while solely relying on the signal generated by the MACD indicator.

This project helped to identify how technical indicators such as the MACD indicator can show false trade signals. The trade signals generated by such technical indicators must be analyzed carefully before taking any actions. It is better to use a variety of technical indicators to assert the trade signals rather than solely relying on a single technical indicator to perform trades.

This project contains the basic features required to run a simple simulation to test trading strategies. One of the challenging parts of this project was gathering price data for securities. Many APIs were tested to gather real-time data including using the C++ libcurl library to get data from credible websites. Many of them required subscriptions and it was not possible for the scope of this project. This is why a locally gathered dataset was used for testing the project. For future work, real-time data can be used to test the simulator to trade on the most recent price data.

There are numerous features that can be integrated into this project. One of the possibilities is to incorporate multi runs for parameter optimization. We can use parameter optimization to find the best configuration parameters for trading a security using the simulator. By incorporating multi runs, we can automate the process, and this can make the simulation more effective. Similarly, hedging can also be incorporated in the simulator so that the model is cash neutral. Cash neutrality is a good goal to achieve to add more safety to the simulator. It would also be helpful to incorporate graphical aides to help understand the reports more easily. More importantly, the trading strategy can be further developed to get better trading signals. This can be done by tuning the parameters of the MACD indicator and also by introducing other technical indicators. It would also be interesting to test and compare the performance of other technical indicators using this simulator.

This project accomplishes all the major goals that were initially set. Dorothy considers more configuration parameters than what was initially planned and is also able to generate a better variety of publications. A Graphical User Interface (GUI) was planned at the beginning of the project but was discarded because it is much more convenient and functional to use configuration files instead.

The main goal of this project was to be able to make reasonable predictions and to gain a better understanding of developing and using financial simulators. I did not have much knowledge about financial simulators before starting this project. This project helped me understand many important concepts about financial modeling and it also helped me to develop my skills in software development. I look forward to working more on this project in the future to make it a more user-friendly, robust simulating software.

9 REFERENCES

[1] A. Hayes, “Moving Average Convergence Divergence – MACD Definition,” *Investopedia*, 06-May-2020. [Online]. Available: https://www.investopedia.com/terms/m/macd.asp. [Accessed: 07-Jun-2020].

[2] M. Hargrave, “How to Use the Sharpe Ratio to Analyze Portfolio Risk and Return,” *Investopedia*, 25-Apr-2020. [Online]. Available: https://www.investopedia.com/terms/s/sharperatio.asp. [Accessed: 07-Jun-2020].

[3] Doxygen, “doxygen/doxygen,” *GitHub*, 07-Jun-2020. [Online]. Available: https://github.com/doxygen/doxygen. [Accessed: 08-Jun-2020].