Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS

Tomáš Helešic

Investment Strategies Simulator

Department of Software Engineering

Thesis supervisor: RNDr. Ondřej Šerý
Study program: Computer Science, General Computer Science

2010

I would like to thank my supervisor, RNDr. Ondřej Šerý, for his time and valuable guidance during my work on the thesis. Also I would like to thank my friends for their comments which forced me to look into my work from different perspectives and a big thank to my parents for their endless support.

I declare that I have written this bachelor thesis on my own and entirely using the cited sources. I agree to lending of this thesis and its publishing.

In Prague, July 10th, 2010 Tomáš Helešic

Contents

List of Figures

Název práce: Simulátor obchodních strategií
Autor: Tomáš Helešic
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Ondřej Šerý
e-mail vedoucího: ondrej.sery@d3s.mff.cuni.cz

Abstrakt: Předmětem této práce je vytvoření simulátoru obchodních strategií. Výsledný program umožňuje uživatelům stahovat aktuální i historická burzovní data, vizualizovat je pomocí grafů a implementovat na ně prostředky technické analýzi. Tyto komponenty jsou navrženy a propojeny tak, aby vytvořili plnohodnotné prostředí pro psaní, vyhodnocení a zobrazení uživatelských strategií.

Klíčová slova: Obchodních strategií; Technická analýza; Scriptování;

Title: Investment Strategies Simulator
Author: Tomáš Helešic
Department: Department of Software Engineering
Supervisor: RNDr. Ondřej Šerý
Supervisor's e-mail address: ondrej.sery@d3s.mff.cuni.cz

Abstract: The subject of this thesis is to create a simulator for investment strategies. The resulting program allows users to download current and historical stock data, visualize it using charts and implement on it tools of the technical analysis. These components are designed and linked to create fully worthwhile environment for writing, evaluation and representation of users strategies.
Keywords: Investment strategies; Technical analysis, Scripting;

Chapter 1

Introduction

1.1 Trading on the stock market

Trading on the stock exchange is an attractive way of investing our funds. We can trade in foreign, securities, commodity and future exchange. The stock exchange in general is not creating goods, but every profit is payed by loss of somebody else. It implies searching for better strategies to be over other traders. Because of the fact, that market is evolving rapidly, the secure strategy of gaining profit does not exists. Successfulness is gain by understanding the market and practicing. But it's not enough. The success in trading is a result of wisdom and common sense. *When we trade we have to try to achieve both realizing that wealth can be a result of wisdom* [6]. Technical analysis (Fig. 1.1) provides us only recommendation based on a survey of existing problems. We can call it as a general advice made by skilled brokers, but it still does not assures us a profit. The final decision has to be made by us own and here we come again to the matter of practice. Because even a good strategy applied in a wrong time can lead us to bankruptcy.
Another very important fact is that the majority of people who trade are not successful enough to justify being in the market. If we trade with the majority, then we will be good as the average and the average does not make money [5]. This note forces us not to be too dependent on analysis of somebody else. We will end this section with a quotation:

> Do something different, even if it's wrong
>
> Charles B. Goodman

In the next section we will describe the key parts of the project, which can improve our skills and be more successful in the real trading.

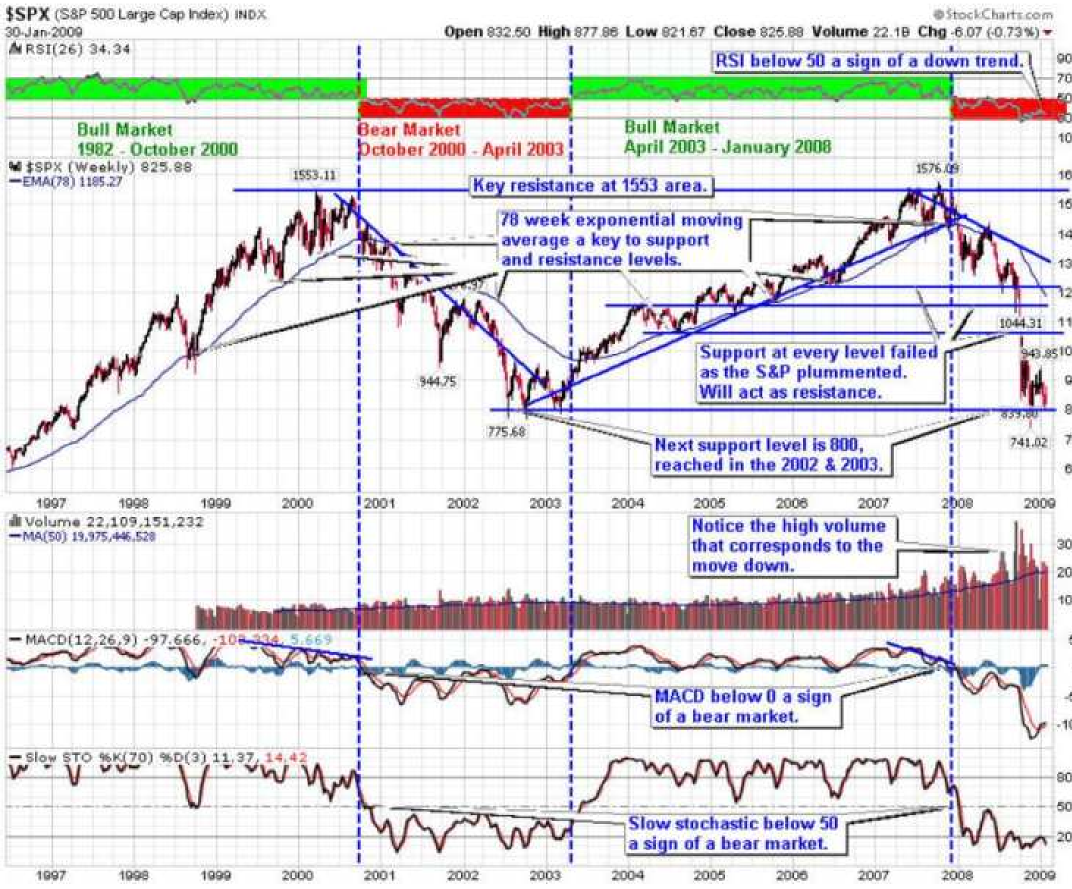


Figure 1.1: Example of a technical analysis in use(Taken from [1])

1.2 Project goals

The main goal of this thesis is to test our strategies on the historical stock exchange data. This is enabled to us by methods and structures, which allows us to:

- download and store stock exchange data, from various sources
- create and apply technical indicators

- create and evaluate strategies
- chart these records, technical indicators and results of strategies

Next chapter describes technical analysis and its purpose. How can indicators estimate market trends. It explains the difference between bull and bear market and their effect on opening and closing positions. It mentions also various types of representing data in a chart and about gaining access to stock exchange data. Chapter 3 explains us the installation process, how to run the program and instructions how to achieve maximum serviceability. Chapter 4 elaborates internal model of the project. External libraries, classes, plug-ins etc. This chapter gives us also information how to extent our program. The last chapter concludes and summarized our effort in analyzing and processing the task, compares with available programs of the same aim and engages in possible future expand of our program. Appendix A contains a list of contents on the accompanying CD.

Chapter 2

Analysis

This chapter contains definition of technical analysis, technical indicator, bull and bear market, opening and closing position, types of charts and stock exchange data providers.

2.1 Bull and Bear market

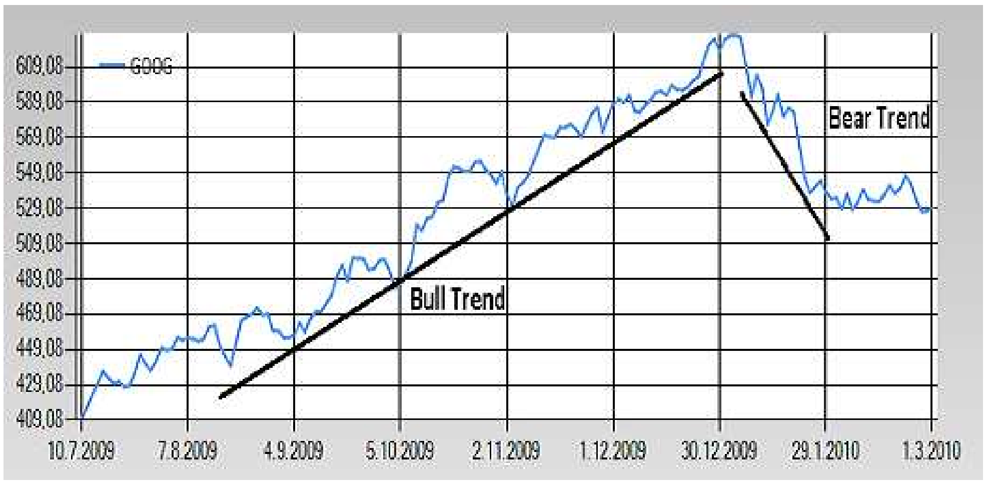


Figure 2.1: Example of Bull and Bear trend

Bull and bear market (Figure 2.1) are only synonyms for upward (bull) and downward (bear) trend. Important is the substance, which expresses the concepts of a price trend. We can use these names only for long-term upward and downward trend. Bull and bear market is for a trader most profitable strategy, because if a trader goes with this trend he continuously make money. For us is very important to indicate the beginnings of these trends and adapt our strategies to this conditions. The last trend which has not yet been mention is a doubtful trend. Soever speculation is very dangerous with a likely loss.

2.2 Positions

Position in financial trading is a commitment to buy or sell financial instrument for given price. When we are entering a position we have to specify a volume of the transaction. This volume in financial trading is called lot and represents the standardized amount of a traded financial instrument. We have two types of positions:

Long Position

- We will profit if the price of a security goes up
- When we buy an option that has not already been sold, we are opening a long position
- When we sell an option that we own, we are closing a long position

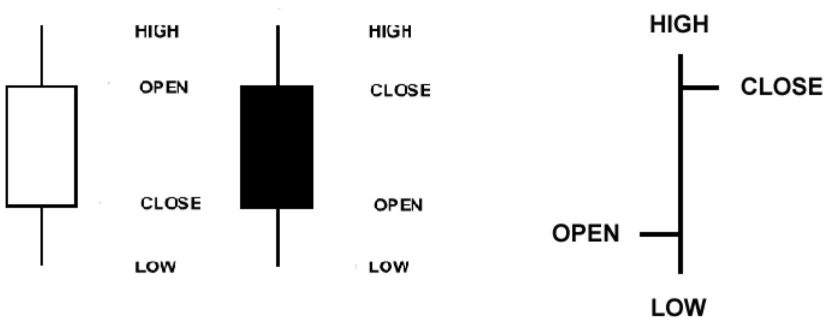
Short Position

- We will profit if the price of a security goes down
- When we sell an option that we don't have, we are opening a short position
- When we buy an option that we have sold, we are closing a short position

Speculate on downward trend of an asset is possible, if we find someone, who can lend us this asset. Usually the lender is a broker and we promise him buying identical asset back at a later date to return it, but we have to pay the lender during the whole time lending fee.

2.3 Charts

In our program we use 4 kinds of displaying chart:
Line chart displays only one value on Y-axis. We use it for displaying indicators and price of the asset.
Candlestick (Fig. 2.2(a))represents four values of an asset in a time moment. Open, close, high and low price. It also shows if the asset closed higher then it opened or closed lower then it opened.
OHLC is similar to Candlestick, it also displays open, close, high, low price over one unit of time. But it has a different shape (Fig. 2.2(b))



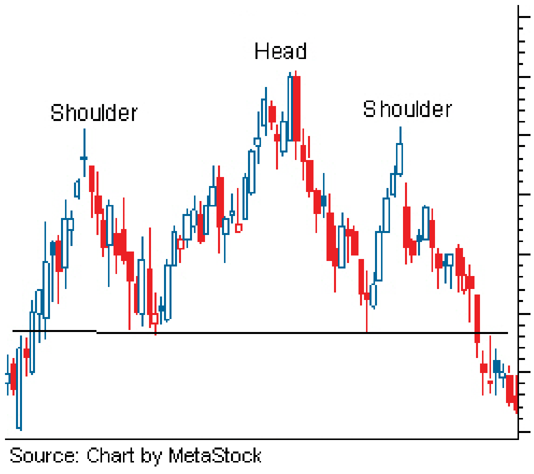
(a) Candlestick (b) OHLC bar

Figure 2.2: Two types of a chart representation

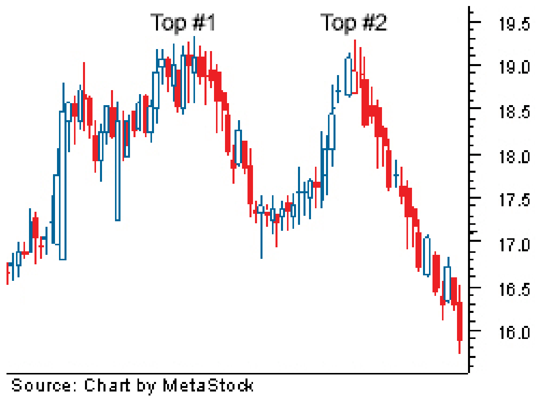
2.4 Technical analysis

Technical analysis is a discipline which studies the past market data and forecasts the direction of prices. It searches for the archetypal price patterns (Fig. 2.3), trends, cycles and tries to exploit them. These patterns are well-known such as Head and Shoulders (Fig. 2.3(a)) or Double top (Fig. 2.3(b)) reversal patterns.

Head and shoulder pattern has four main parts. Neckline (the horizontal line), 2 shoulders and a head. We have two inverse types, top and bottom. Both are sets of peaks and troughs. In a "*top*" version, it signals the change of a trend from upward into downward. After reaching the first maximum (left shoulder), the security reaches the new low. Its followed by the head



(a) Head and Shoulder Top



(b) Double Top

Figure 2.3: Well-known patterns (Taken form [3][4])

that has the new high, which is above the left shoulder. Again it retraces back near the low after the left shoulder. Then the trend changes again and reaches the right shoulder, which price is lower then the head. Head and shoulder pattern is confirmed when the price falls under the neckline, which is defined by two troughs (after the left shoulder and head).The "*bottom*" is an exact opposite, where peaks and troughs are reversed. The second pattern is Double Top and inverse variant Double Down. It indicates also the changeover in a trend. In the figure 2.3(b) the market has reached the new high and is followed by the so-called correction, that creates a trough with some fluctuations. After this period the market tries to break the previous high and continue in the upward trend, but this attempt failed due to the strong resistance. We can imagine the resistance as the highest price level that the majority of traders for the purchase of an asset are willing to pay. The traders concluded that the current price is high enough and further speculation on growth is not safe. This confirms us the Double Top, the market fails to achieve new high and the price bounces into the downward trend. Technical analysis in recent time endeavors to implement artificial neural networks. They can learn how to detect complex patterns in data. They are universal approximation method.

2.5 Technical Indicators

Technical indicators are a results of mathematical calculations. Indicators follow the relationship between price and volume. Volumes is used as a secondary indicator, which indicates money movement and activity. Indicators are filtering the small market fluctuation, which are confusing a mainly meaningless. We distinguish 4 types of indicators by their orientation and 3 of them are implemented in our program:

- Volume (On Balance Volume indicator)
- Oscillators (Relative Strength Index)
- Trends Indicators (Simple and Weighted Moving Average)
- Bill Williams

2.5.1 On Balance Volume

On Balance Volume (OBV) measures positive and negative volume flow. The concept of this indicator is that volume precedes price. The OBV is in a rising trend when each new trough is higher then the previous and each new peak is higher then the previous. In a bear trend each new trough is lower then the previous trough and it is the same with peaks. When the OBV is moving sideways with small fluctuations, the trend is doubtful.For bull trend we can translate it as the rising price is the result of an increased demand for the security. However, if the price is rising, but OBV is falling, it warns us that the trend is unhealthy and probably will not persist (Fig. 2.4). The important is here direction of the indicator not the value and we have to concentrate on the OBV trend and its relationship with the asset's price.
Calculation:

- If today's closing price is higher than yesterday's closing.
  OBV(i) = OBV(i-1) + Volume(i)
- If today's closing price is lower than yesterday's closing.
  OBV(i) = OBV(i-1) - Volume(i)
- If today's closing price is equal to yesterday's closing.
  OBV(i) = OBV(i-1)

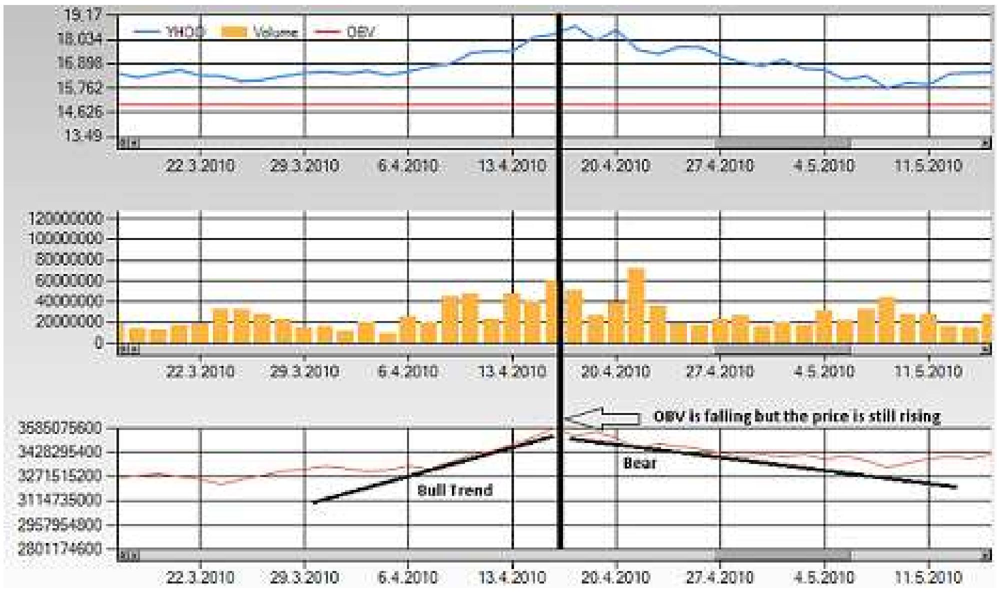


Figure 2.4: OBV detected an unhealthy bull trend

2.5.2 Relative Strength Index

Relative Strength Index (RSI) is classified as a momentum oscillator (Fig. 2.5). It compares the magnitude of recent gains to recent loss of the market (2.1). Typical time frame for RSI is 14 days. Gains and loss is given by comparing the current close price with the previous close price. RSI ranges from 0 to 100. The indicator levels are 30 and 70. If the RSI falls under 30 it is the signal that the asset is getting oversold and therefore predicts the asset to rice in price. Likewise, if it goes over 70, it is the signal that the asset is overbought and the market is going to fall. Large surges and drops in the price greatly affect the RSI causing false signals. This cause in long therm small effective of this indicator and it is better to check the rise or fall of the market with other indicators.

$$RSI = 100 - \frac{100}{1 + RS} \tag{2.1}$$

RS = The sum of days up closes / The sum of days down closes

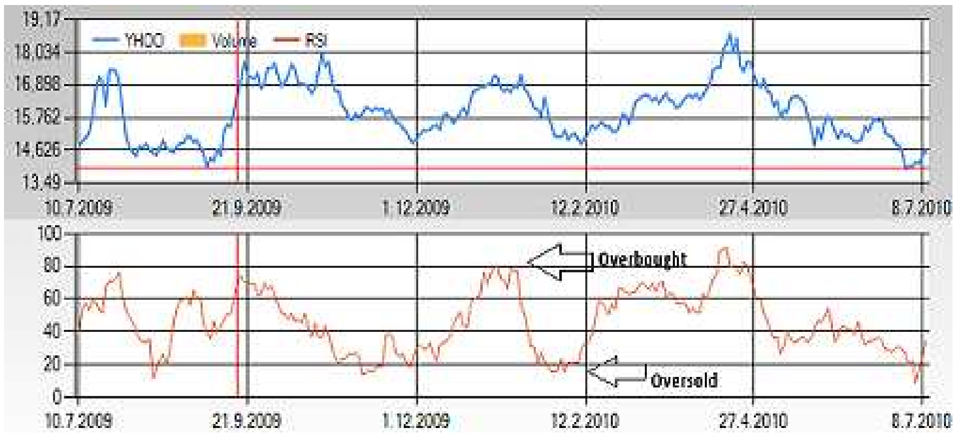


Figure 2.5: RSI detected oversold and overbought of the market

2.5.3 Simple and Weighted Moving Average

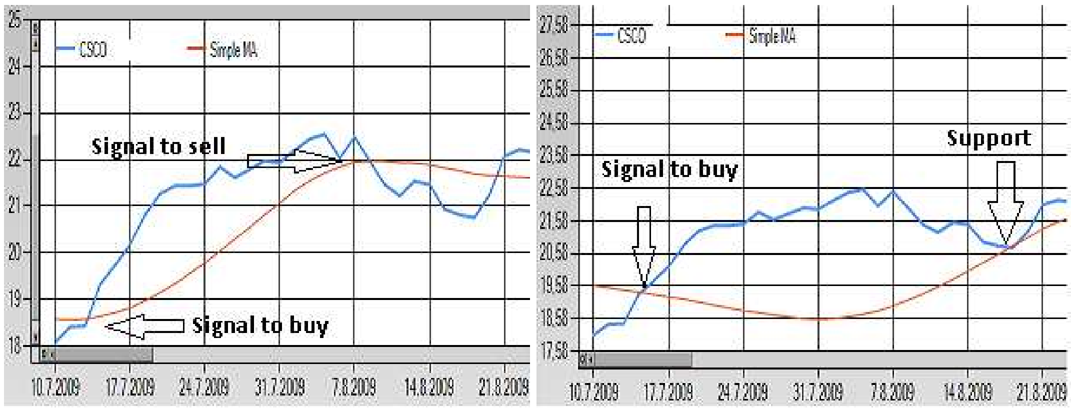
Simple and Weighted Moving Average shows the average value of a asset's price over a set period (typically 5, 14, 20, 50 etc.). MA does not predict the price direction but define the current direction with a small lag. Old data are subtracted and new data added. MA can be shifted couple of days in the future to increase the support method.
Simple n-days' MA is calculation:

$$SMA = \frac{p_M + p_{M-1} + \ldots + p_{M-n}}{n}$$
$$SMA_{today} = SMA_{yesterday} - \frac{p_{M-n}}{n} + \frac{p_M}{n}$$

The Weighted Moving Average is a special variant of the Simple MA. It has a multiplying factor to give different weights to different prices. This Multiplying method is the highest for the most recent data and decrease arithmetically. In an n-day WMA the latest day has has weight n, and it decreases by 1 to zero.
Calculation:

$$WMA = \frac{np_M + (n-1)p_{M-1} + \ldots + 2p_{M-n+2}p_{M-n+1}}{\sum_{i=1}^{n} i}$$

Moving Averages are used to generate simple signals with price crossovers signals. A upward trend is signaled when the moving average moves below the price and downward when the moving average moves above the price. Longer n-days' MA is used for long-term signals. Price crossover signals are generated when the MA crosses the price (Fig. 2.6(a)). When it happens it's a signal of changing trend. When the MA intersects the price from above, it signals an upcoming bull trend and if it happens from below, it signals the change into bear trend. MA can be also used as a support in a bull trend and resistance in a bear trend (Fig. 2.6(b)).



(a) Simple 14-days' MA (b) Simple 14-days' MA shifted by 14 days

Figure 2.6: Simple MA showing price crossovers and support

2.6 Data Providers

We have already discussed data representing and their analysis, but we have to somehow gain access to them. Every stock exchange provides "*up-to-date*" data for free, but with small delay, which is crucial in moments when the market collapse and we cannot act in time. Instant data generated by the stock exchange are highly charged and for our project unacceptable. Because our main goal is to test strategies, we do not need the most recent data. For us is more convenient to have an access to whole history prices of any asset. We have to find also a simple way how to automatically request data for an asset, process it and store it. The solutions are described in the Chapter 4.

Chapter 3

User Documentation

In this chapter we describe the end user documentation, steps how to install, run and operate the program.

3.1 Installation

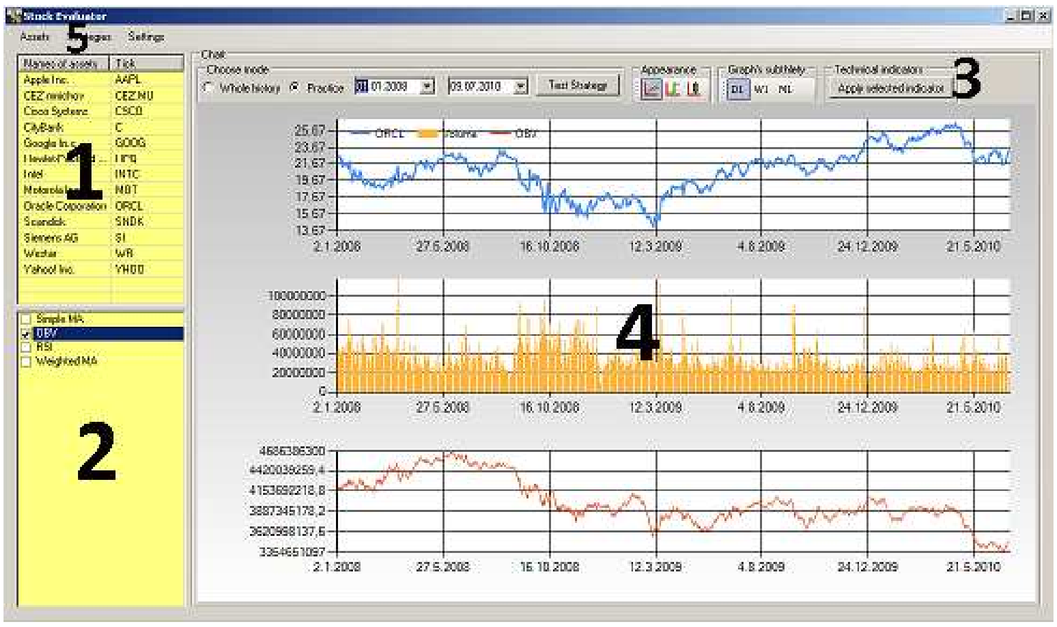
Target platform for this project is Windows. The installation folder *Setup* contains two files. *Setup.exe* and *StockEval.msi*. The installation file is *Setup.exe*, which checks the prerequisites *.Net Framework 3.5 with SP1 and Windows Installer 3.1*. If there are missing, the installation program tries to connect to the Internet, download these programs and install them. After this first step, installation process continues to install the main program. We can choose the target folder for the program and we can select who can use the program, just us or everyone who uses out computer. The installation program also creates a shortcut on the desktop. The installed program contains in the main directory folders with files:

- \Databases database files, new databases are created here
- \DataProviders libraries used to download stock market data from the remote server
- \Indicators contains technical indicators
- \Resources icons
- \Output primary folder for saving records of evaluated strategies in Excel and CSV files.

- \Scripts primary folder for loading scripts into the program

3.2 Starting the program

The program can be started with the shortcut on the desktop or with the file *StockEval.exe* which can be found in the main directory. After starting the program it searches for the data providers, indicators and loads the default database. If the database is not found, the program opens the Settings form and prompts user to select the database manually or to create a new one. How to change settings is described later on. The main program is shown in figure 3.1. The main form layout:



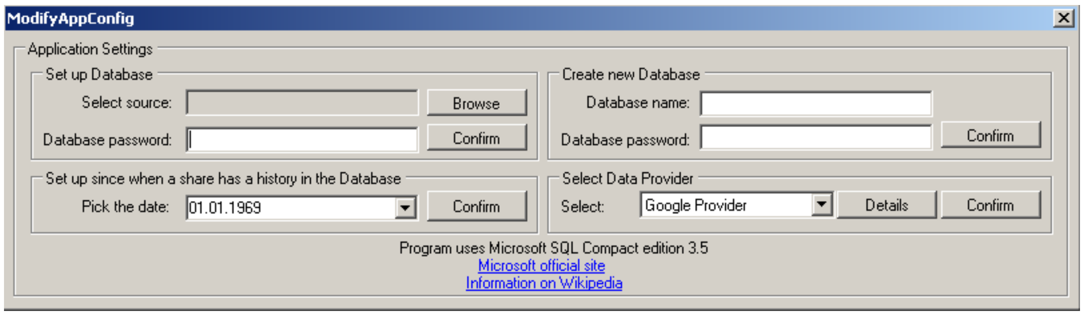
Names of assets	Tick
Apple Inc.	AAPL
CEZ mnichov	CEZ.NU
Cisco Systems	CSCO
CityBank	C
Google Inc.	GOOG
Hewlett-Packard ...	HPQ
Intel	INTC
Motorola Inc.	MOT
Oracle Corporation	ORCL
Scandisk	SNDK
Siemens AG	SI
Wachie	WR
Yahoo! Inc.	YHOO

Figure 3.1: The layout of the main program

1. displays all assets stored in the database
2. enumerates all available technical indicators
3. controls which operate the chart
4. chart area

5. controls to add/delete assets, select and apply strategies and change application settings

3.3 Change application settings


ModifyAppConfig

Application Settings

Set up Database
Select source:
Database password:

Create new Database
Database name:
Database password:

Set up since when a share has a history in the Database
Pick the date: 01.01.1969

Select Data Provider
Select: Google Provider

Program uses Microsoft SQL Compact edition 3.5
Microsoft official site
Information on Wikipedia

Figure 3.2: Application settings

In this form we can set up our application. Change and create a new database, select data provider and set up since when we want to have historical data from a provider.

3.3.1 Change a database

In the Group box named *Set up Database* we have to click on a *Browse* button. It opens a form, which displays a list of databases from the folder \Databases. We can then select one of these and confirm a selection by pressing the *Select* button. Then we have to type a password for the selected database and confirmed the change of the database by pressing the *Confirm* button. If everything has been successfully made a message box appears with the confirmation text that the database has been changed. If not an error message informs us, where is the problem.

3.3.2 Create a new database

In the Group box *Create a new database*, we have to input the name and password for the new database. Password is optional and can be empty if we don't want to protect the database. When we press the *Confirm* button program checks, if the database name contains valid characters and if yes, program creates a new database in the \Databases folder and selects it immediately.

3.3.3 Select data provider

We can select a data provider from the drop down box in a Group box called *Select Data Provider*. This drop down box is filled with names of libraries in the folder *DataProviders*, that are implementing the implementing IAnalyzer interface. After the selection we can click on the *Details* button, which shows us a small description of the selected provider. For example *GoogleProvider* library provides us only one year old historical stock exchange data. Again *Confirm* button set ups this provider as a default for out application.

3.3.4 Set up initial date for downloading

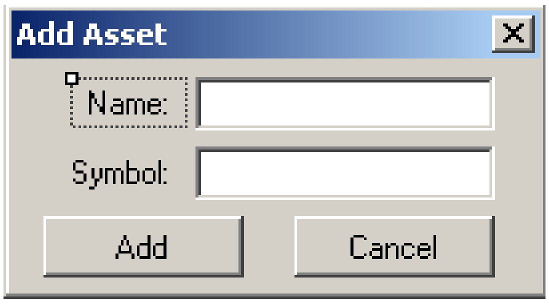
However the Google does not provide historical data older then one year, Yahoo does. We can set up a date since when we want to have historical data. It tells the provider to ignore possible older records, or data provider ignores this date. We can change this date be clicking on the date time picker and select a new date. Clicking on the *Confirm* button we set ups the date.

3.4 Add or delete assets

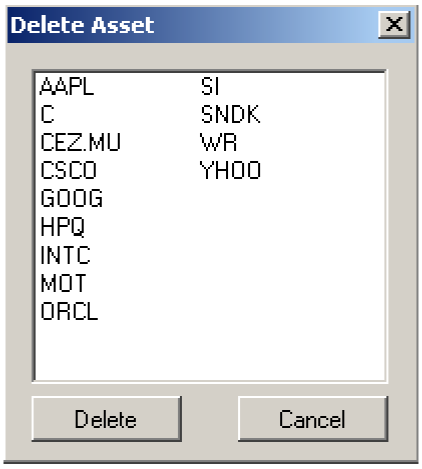
We can add or delete asset by clicking in the Menu on the *Assets* label and select the desired operation. We can open two forms. Add Asset form (Fig. 3.3(a)) or Delete Assets form (Fig. 3.3(b)). To add new asset we have to find the unique symbol, under which it's represented on the market. The name can be filled randomly by us. By clicking *Add* button, the application tries to download the historical data and add the asset to a database. If an error occurs, we will be noticed.
To delete assets just select in the *Delete Assets Form* the names in the list of these assets and click on the *Delete* button and all their records will be removed from the database.

3.5 Chart

In this section we discus how to draw a chart and work with it.



(a) Add Asset



AAPL
C
CEZ.MU
CSCO
GOOG
HPQ
INTC
MOT
ORCL
SI
SNDK
WR
YHOO

(b) Delete Assets

Figure 3.3: Add and delete assets

3.5.1 Display a chart

The chart of the asset is drawn, when we click on the name of the asset or on its symbol in the top left list view. The application loads the data from the database, filters them eventually if the practice mode is selected and displays them into the chart control see 3.1. We have 2 default charts representing assets price and volume, which will be always displayed.

3.5.2 Operate with a chart

As we can see in the figures 3.4(a) and 3.4(b), the application provides us functions to control the chart. Figure 3.4(a) describes us 2 possible modes:

- **Whole History** tells the program to display always whole history of an asset. In this mode we can apply indicators, change chart appearance and value grouping but we cannot test strategies.

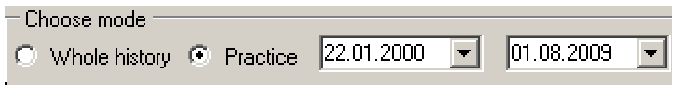
- **Practice mode** tells the program to display only certain history of an asset. Left date says is the start date and the right is the end date for displaying records. In this mode we can do the same things as in *Whole history mode* and besides we can run strategy scripts and test a simple strategy. By changing the start date and end date the chart will be redrawn to display this new time frame.

Figure 3.4(b) represents controls to change the chart appearance and group values. The assets price chart can be represented as:

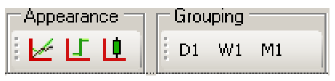
- **Line** chart shows the opening price
- **Stock** chart represents the records as a Open-High-Low-Close bar. See Fig. 2.2(b)
- **Candlestick** chart represents the records as Stock but in a different format. See Fig. 2.2(a)

The values can be group by:

- **D** no grouping, only displays the data from the database
- **W** groups the values into the week interval. For Line chart calculates an average of the opening prices. For OHLC and Candlestick calculates the average for open and close prices and from the highs takes the maximum value and from the lows minimum value. For the indicators value calculates the average
- **M** groups the values into the month interval in the same way as *W* does.



(a) Choose an application mode



(b) Change chart appearance and value grouping

Figure 3.4: Change application mode, chart appearance and value grouping

3.5.3 Zooming and scrolling

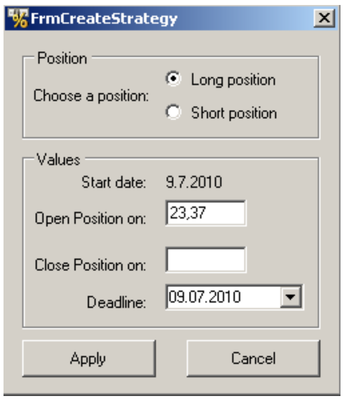
We can zoom in and out by using the Keys + and - or by using the mouse wheel. The chart is reset to its initial view by pressing the small circular button on the left side from the Y-axis scrollbar and over the X-axis scrollbar. We can zoom only a specific part of the graph by clicking and dragging the mouse. The actual mouse selection is shown in light gray color.

3.6 Apply indicator

In the list of available indicators we have to select the desired one. Our application supports only one indicator to be charted concurrently. To confirm the selection and to display the indicator we have to click on the button *Apply selected indicator* in the group box *Technical indicators*. We have to make sure, that an asset is picked and charted or the indicators will not be calculated and displayed. If an asset is charted, the application opens a settings form of the indicator if it is necessary. For example MA needs to set up time frame and shift but on the other hand On Balance Volume does not need any additional attributes than assets records and is calculated and displayed immediately. If we want to change some attributes of the same indicator we have to click again on the *Apply selected indicator* button and set up new values because the indicator needs to be recalculated. How to create a new indicator is described in Chapter 4.

3.7 Apply simple strategy

To apply a simple strategy we have to be in a practice mode and an asset's chart has to be drawn. Then we can click on the *Strategies* ⇒ *Simple*. It opens us the *frmCreateStrategy* form (Fig. 3.5) where we can set up our strategy. We can choose if we want to go in *Long* or the *Short* position. In


FrmCreateStrategy

Position

Choose a position: [✓ Long position] Short position

Values

Start date: 9.7.2010

Open Position on: 23,37

Close Position on:

Deadline: 09.07.2010

Figure 3.5: Set up a simple strategy

the *Value* group box we are informed about the start date. The *Open Position* *on* value holds the start dates' opening price, but it can be changed. Then we have to input the closing price and the *Deadline* date, which tells the application about the last possible date to close the position. By clicking the *Apply* button we start evaluating the strategy. The application will inform us about the results and eventually it will mark in the chart the opening and closing position.

3.8 Create and run strategy scripts

In the Chapter 4 we deeply discuss how to create a strategy script. In this section, we only inform how to load and run the script. As we have said in the previous section we have to be in a practice mode and an asset's chart has to be drawn. Then we have 2 possibilities for loading the script. The first is to click on the *Strategies* ⇒ *Scripts* and a form will occur (Fig. 3.6). By clicking on the *Browse* button a File input dialog will be shown and we

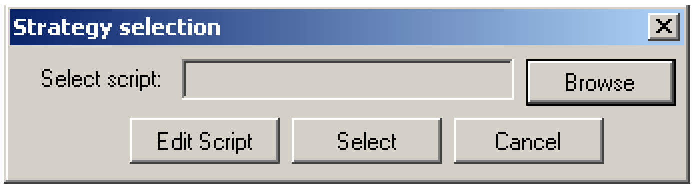
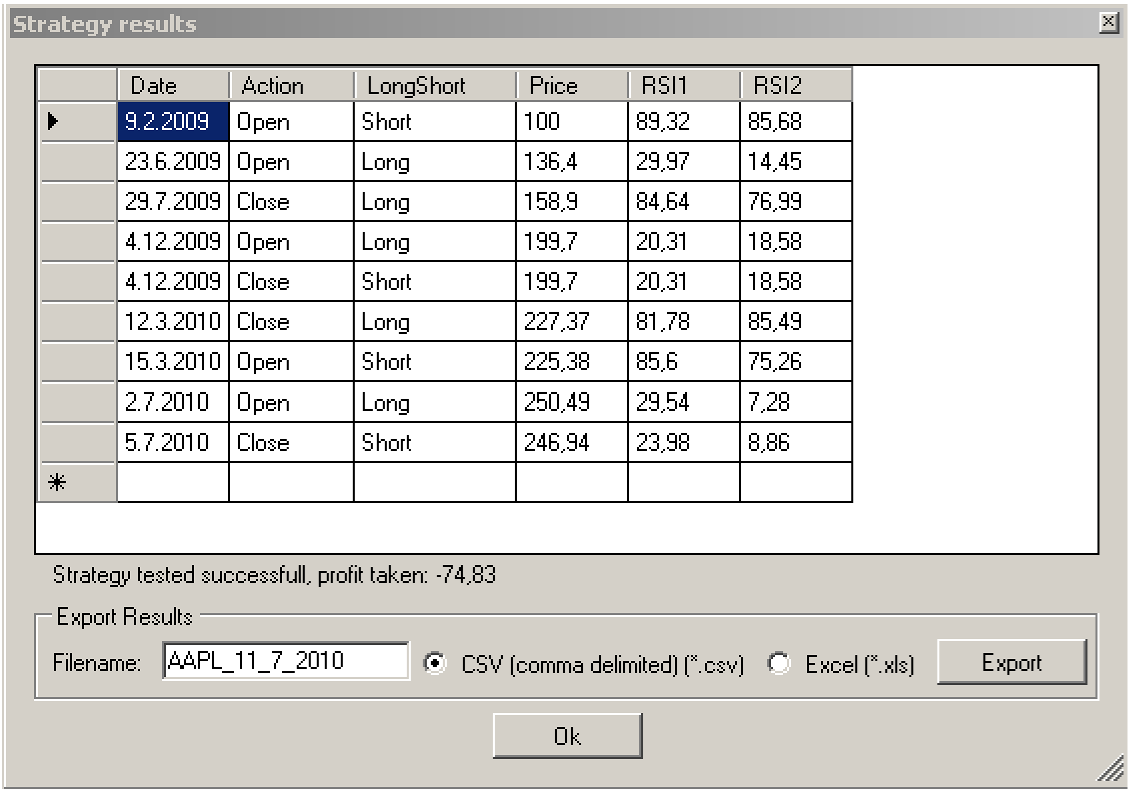


Figure 3.6: Select a script

can select a script to evaluate. The default folder for scripts is em \Scripts but we can select also from another location. After the selection we can edit the script by clicking the *Edit Script* button. It will start the default program for creating the **.cs* files in our computer. The *Select* button stores the path of our script. The second possibility to load a script is to click *Test Strategy* button. If the path to a script is missing it will open the same form as in a first example, but if the path exists this button runs the script. The script is compiled and evaluated. If no error has occurred we can see now the results (example in Fig. 3.7). We can see when the position has been opened and closed also if we have gone long or short, the price of each transaction, indicators values which has been used and below the table the final profit of the strategy. We can export the result into CSV or Excel files by selecting these options and by clicking the *Export* button. We can also change the name of the file. The files are exported into *Output* folder in the main directory of our program.



Strategy results

	Date	Action	LongShort	Price	RSI1	RSI2
▶	9.2.2009	Open	Short	100	89,32	85,68
	23.6.2009	Open	Long	136,4	29,97	14,45
	29.7.2009	Close	Long	158,9	84,64	76,99
	4.12.2009	Open	Long	199,7	20,31	18,58
	4.12.2009	Close	Short	199,7	20,31	18,58
	12.3.2010	Close	Long	227,37	81,78	85,49
	15.3.2010	Open	Short	225,38	85,6	75,26
	2.7.2010	Open	Long	250,49	29,54	7,28
	5.7.2010	Close	Short	246,94	23,98	8,86
*						

Strategy tested successfull, profit taken: -74,83

Export Results

Filename: AAPL_11_7_2010 [✓ CSV (comma delimited) (*.csv)] Excel (*.xls)

Figure 3.7: Example of a strategy result

Chapter 4

Implementation

In this chapter we discuss the internal model of the project, interfaces, external libraries and how the program initializes components, downloads and stores data, draws charts and uses indicators and applies strategies. The program is written in C Sharp under Visual Studio 2008.

4.1 Internal Model

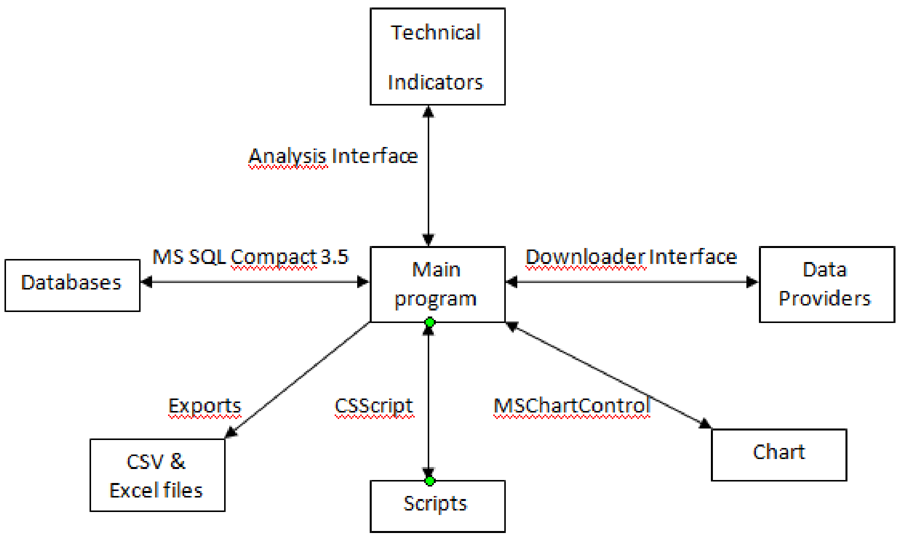


Figure 4.1: Internal model

In the figure 4.1 we can see the internal program's model. Every displayed object cooperates with the main program trough interface, external library or special methods. We describe these relationships with the main program in later sections.

4.2 External Libraries

4.2.1 Databases

In our program we use Microsoft SQL Compact Edition 3.5 [7]. This decision has been made for one very important reason. We do not have to install additional SQL server on the user's machine and it also enables user to work in a off-line mode. The application maintains databases using these Class libraries which are part of the installation:

- sqlcese35
- sqlceqp35
- sqlceoledb35
- sqlceme35
- sqlceer35EN
- sqlcecompact35
- sqlceca35

Every database contains two types of table. *CompanyTicks* stores assets' names and their symbols and the second type serves for storing an asset's stock data. These tables are named by the asset's symbol and the structure is:

- *Date*, date and time of the record, type DateTime
- *OpenPrice*, opening price, type Double
- *High*, highest price, type Double
- *Low*, lowest price, type Double

- *ClosePrice*, closing price, type Double
- *Volume*, transactions volume, type Long
- *AdjPrice*, adjacent price, type Double

4.2.2 Chart control

We have decided to use The Microsoft Chart Control for .NET Framework [8]. It provides rich data visualization and enables the programmer to serialize charts, group values, change appearance, set legends and tool tips, zoom and scroll etc. It is widely supported and the developers installation package contains very deep tutorial with code examples that saves a lot of time.

4.2.3 Compiling script

For compiling users' scripts on-the-fly we use the *CSScriptLibrary.dll* [2]. It's the C Sharp Script engine class library. The library enables the script to share assemblies with the main program. The usage is described in the section *Scripts*.

4.3 Interfaces

In our program we use interfaces to set rules in s communication between the main program and external files. We haves designed 3 kinds of interface where each of them fulfills a different task.

4.3.1 Downloader Interface

Set up an interface for data providers. Every new data provider has to implement this interface to be accepted by the main program. The crucial method is *GetHistory*, which process the request of the main program and returns table containing stock data.

4.3.2 Analyzer Interface

Specifies the interface for technical indicators. It contains the prerequisites for chart control. The *Calculate* method calculates the indicator's values from the asset's records. It also orders a indicator to implement *Settings* method that setts the indicator's attributes whenever it is called.

4.3.3 Strategy Interface

Sets the structure of every strategy script. The most important functions are discussed in the section Scripts.

4.4 Program's relationships, communications and functionality

4.4.1 Initializing components

When the program starts it creates 2 static variables in the class *Global.* They store all the instances of data providers' plug-ins and indicators' plug-ins. Data providers are loaded by using classes *DownloaderServices* and *DownloaderPlugins.* These classes search for the .dlls in the *DataProviders* folder that are implementing the *IDownloader* interface. If a .dll is found, class *DownloaderPlugins* creates the instance and adds it to the collection. Loading indicators is almost the same and they are searched in a *Indicators* folder. Classes *AnalysisServices* and *AnalysisPlugins* provides the desired functionality to find them and to load them. Afterwards the main form is created and in the *LoadList()* method is checked, if the database has been correctly opened and if not, the application setting's form will be opened.

4.4.2 Downloading

The main form creates a background worker, who informs us about downloading process in the background. This worker calls class *DB* in the *IUinDB.cs*, which has the methods to store new data in the database. It receives the assets name and symbol, or only a symbol. This class checks if the database should be filled in with new data and if yes, the activated data provider is called. In our program we can download from Yahoo finance or Google finance. They differ in providing the historical data. Google enables us only to download 1 year historical prices, however Yahoo unlimited. Both of these plug-ins are using the Webclient to downloads the data as one string. They parse the values and store them in a table. *DB* stores the table into the database and kills the background worker so we can again manipulate with the program.

4.4.3 Charting

Charting is implemented in the *Graph.cs* file. It is a partial class of *FrmMain*. Because the main form is processing the events, buttons and etc. which have effect on the chart, its easier to implement these functionalities in one class. When the asset is selected for charting, all the data are loaded into the program from the DB. When we change the time frame of the chart or we frequently load indicators, the traffic between the main program and database will slow us down. After we load the history, the new chartareas are created. One for the asset's price and the second one for the volume. Indicators can be added into the new chartarea or into existing one. It depends on the attribute Graph which is a part of the analyzer interface.

4.4.4 Indicators

As mentioned above, the indicators are loaded directly when the program starts and their instances are stored in a static variable in the *Global* class. The program searches for these indicators in the *Indicators* folder. The program calls the *Calculation* method of the indicator, filters data and calls the *InsertIntoGraph* method that draws the chart.

4.4.5 Scripts

In the folder \Scripts we have a detailed template to create a new script. The class ScriptHost serves as a Host for the scripts and provides them functionalities, for an easier writing and evaluating. The script is compiled and called to evaluate with this command:
IStrategy script =
(IStrategy)CSScript.Load(frm.ScripthPath).CreateObject("Script");
Then the functions specified in the *Strategy Interface*, are called. *LoadComponents()*, *Evaluate()*. The *ScriptHost* class creates the template for the result table of the strategy, so we don't have to focus on this problem and focus only on the logic of the evaluating. However we can add new columns to the result table if we want to. *ScriptHost* class provides *GetIndicatorsRecords()* method to get the indicator's records and *Filter* method to clear useless val-

ues from the tables. The script's *Evaluate* method has to be implemented by the user itself. Finally the *ExportResults* method is called to to mark the opening and closing position in the chart. Then *FrmStrategyOutput* formed is created and the results are displayed there. We can export the results into CSV or Excel. Excel export uses the Microsoft.Office.Interop.Excel.dll library, which enables us to create an instance of the program and add the values into the cells. CSV is implemented as a streamwriter.

4.5 Error processing

The application distinguishes 2 types of errors. The user's and runtime's. From user's errors the application can be recovered. User is informed what has gone wrong and can correct it. Runtime errors are not recoverable and causes application to exit. The user is also informed about what has happened and then the application immediately exits.

Chapter 5

Conclusion

We have managed to fulfill the goals of this thesis. Our program provides stock exchange data from the remote server only by writing the asset's symbol and furthermore we have managed to create plug-in based system for this functionality. If a new subject appears on the market and can provide us more frequent data and less latency, we can simply create a new class library which downloads this new data and add it to our program without additional changes in the code. Also plug-in based system is designed for indicators. So if someone creates a new indicators, they can be easily distributed among the users. Also scripts and databases are portable and not dependent on the user's locale settings. We have managed also to maximize benefits of the chart control, which is very flexible, but when we are operating with 30 years history it's getting slow and we are reaching the borders of our patience. We have to keep in mind that even if it's widely supported it's still a non business control, which is improved only a little. We have also created instruments to evaluate strategies using scripts. We have made a softly background support for creating and running scripts. We have to be still a little bit skilled in C Sharp to write a script. After summarizing our programming effort and mined knowledge about trading on the stock exchange, we can present the wishes, what could we do next. Probably for the end user is valuable to make writing a scripts more simplified in a some kind of natural language. It can be done by creating a grammar or by creating a library with additional functions and the script will be compiled together with this library. It will be very nice and probably it will make us very rich to automatize searching better strategies or just find the best attributes for indicators in our strategies to maximize the profit with minimum risk.

Artificial intelligence is the solution, but this is a topic for a dissertation.

5.1 Comparison with another available implementation

We can encounter on the market with various business application designed for trading in real time on the stock market. Some of them like XTB or MetaTrader are focusing on the novices. They give them fully functional program and a test account to trade with a fictive money. So they can make the beginnings more attractive for them because they can immediately see the money flow on their account. But still it's for them confusing, because they become overloaded by a new terminology, functions and unexplainable behavior of the market. But what we can envy is the beautiful chart representing and controlling. We can draw there our lines, levels, write notes etc. But it's only a representation of the market, not explanation. Our program tries to simplify and filters only valuable things for creating and testing strategies, but is limited to free of charge data providers and free chart controls.

Bibliography

[1] ARUN, Arun. *Market Trends [online].* 2009-02-07 [cit. 2010-07-09]. Stock Market Position,
`http://weblogsurf.com/february-2009-stock-market-position`

[2] CSScript `http://www.csscript.net`

[3] LANGAGER, Chad; MURPHY, Casey. *Investopedia [online].* [cit. 2010-07-09]. Analyzing Chart Pattern: Head and Shoulders,
`http://www.investopedia.com/university/charts/charts2.asp`

[4] LANGAGER, Chad; MURPHY, Casey. *Investopedia [online].* [cit. 2010-07-09]. Analyzing Chart Patterns: Double Top And Double Bottom,
`http://www.investopedia.com/university/charts/charts4.asp`

[5] DALTON, James F.; JONES, Eric T.; DALTON, Robert Bevan. *Power Trading With Market Generated Infromation.* 2 edition. USA : Traders Press, 1999. Introduction, p. 1-6

[6] Ross Joe. *Trading By The Book* 5 edition.USA : Ross Trading, 1985. How To Get Wisdom, p. 10-11. ISBN 976810824X

[7] Microsoft SQL Compact edition 3.5,
`http://www.microsoft.com/Sqlserver/2005/en/us/compact.aspx`

[8] MSchartControl,
`http://msdn.microsoft.com/en-us/library/3ks53324(VS.71).aspx`

Appendix A

CD contents

The accompanying CD has the following structure:

- \Txt
  - \Latex - Latex source files with used images
  - *Thesis.pdf*
- \Setup
  - *Setup.exe*
  - *StockEval.msi*
- \Doc
  - *StockEval.CHM* - documentation generated by the Sandcastle
- \Src
  - *StockEval* - Visual studio 2008 solution with all source files