The design considerations and development of a simulator for the backtesting of investment strategies

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Kevin Gounden
February 2011

Supervised by
Dr. Gary Marsden

This thesis is dedicated to my wife Sudasha who has unconditionally supported me throughout this research and I acknowledge the time that we have had to sacrifice together in order for this to be completed. To my father Krish, mother Rani, and brother Nevin, all of whom have been pivotal to my success and who will remain my muse.

Abstract

"Most of the time, common stocks are subject to irrational and excessive price fluctuations in both directions as the consequence of the ingrained tendency of most people to speculate or gamble... to give way to hope, fear and greed." – Benjamin Graham

The skill of accurately predicting the optimal time to buy or sell shares on the stock market is one that has been actively sought by both experienced and novice investors since the advent of the stock exchange in the early 1930s. Since then, the finance industry has employed a plethora of techniques to improve the prediction power of the investor. This thesis is an investigation into one of those techniques and the advancement of this technique through the use of computational power.

The technique of **portfolio strategy backtesting** as a vehicle to achieve improved predictive power is one that has existed within financial services for decades. Portfolio backtesting, as alluded to by its name, is the empirical testing of an investment strategy to determine how the strategy would have performed historically, with a view that past performance may be indicative of future performance. Whilst the existence of backtesting may span generations, the collective knowledge on this technique remains limited. A reason for this, lies in the fact that in order to accurately backtest a strategy, one must simulate how the strategy would have performed had it been deployed at a particular point in history, taking into account the various factors that influence the strategy. These range from the macroeconomic and geo-political situations of the time to the micro-fundamental detail of the companies to be invested in. It must also include the expert knowledge of the fund manager who would have been deploying the strategy and how he/she would have reacted daily throughout the holding period of the strategy. The process of simulation is further complicated by the need to mimic reality in terms of the mechanics of investing and the various costs that are associated with investing.

To successfully accomplish such realism is an arduous task, requiring skills in finance, software development, databases and core concepts within software agent design. The field of computational finance was the natural evolution in this impedance mismatch of skills, with the use of computational power and programming logic enabling more complex financial models and simulators to be developed.

This thesis, beginning with the collation of research into the body of knowledge available on portfolio backtesting as a financial concept, extends this knowledge by investigating the considerations in converting this financial concept into a software simulator. The reader is provided with the philosophical and technical challenges that arise, the alternatives available and the merits of each alternative. The emphasis of the research is on determining a flexible software architecture, accurate enough to simulate how a particular strategy would have performed in the market during a given historic period. The thesis discusses the complexities of backtesting that a software architecture would need to accommodate and succeed against. It also attempts to offer measures for success from the results achieved.

The intended audience for this research is the computer science practitioner that is unfamiliar with portfolio backtesting and the considerations one must take in designing a software architecture to achieve simulated backtesting. Core themes within the research relate to the following questions:

a. Does the simulator accurately implement the philosophical nuances of financial backtesting?
b. Does the simulator accurately profile a portfolio's return series?
c. Does the simulator architecture provide a framework that is sufficiently extensible to accommodate the varied investment portfolio techniques that could be deployed?
d. Does the simulator aid the portfolio optimisation process?
e. Does a comparison of simulated results to that of an existing portfolio compare favourably?

Acknowledgements

A special thank you must be extended to Dr Gary Marsden, whose experience, patience and wisdom are deservedly appreciated. I would also like to take this opportunity to thank Mohendra Moodley for his teachings, guidance and support in the field of finance. A further thank you is also extended to my wife and family for their support.

CONTENTS

List of Tables

List of Illustrations

Chapter 1

Introduction

"There is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success than to take the lead in the introduction of a new order of things." – N. Machiavelli

1.1 Motivation

The past two decades have borne witness to the transformation of a finance-based industry welcoming the disciplines of physicists, mathematicians, and computer scientists to apply their domain-specific knowledge in the pursuit of profitable stock-picking – thereby creating the field of computational finance. From this fusion of finance and science, the quantitative investing technique was born, attempting to mathematically quantify the problem domain of determining the most profitable stocks to buy or sell, devoid of human emotion. Today the finance world has two camps of thought, those that follow the traditional qualitative method of investment finance where in-depth company specific knowledge and "gut instinct" drive investment decisions and those that follow the quantitative or "quants" methodology where statistics and other mathematical techniques are used to make investment decisions.

This dissertation investigates the quantitative technique of portfolio backtesting and details the architectural considerations in developing a prototype software package to simulate, via

the principles of backtesting, the potential outcome of an investment strategy prior to implementing the strategy on the stock market.

A fund manager is the investment professional ultimately deciding in which financial instruments (stocks or bonds for example), to invest, given assets (money) invested by the general public or other institutions such as banks and pension fund administrators. Whilst a fund manager may have the conviction that a particular basket of stocks will perform successfully when invested in the market, the ability to simulate the historic performance of such a portfolio on a daily basis over the last ten years or more is a highly-valued, yet extremely complicated and an arduous task to achieve. Whilst many fund managers perform rudimentary backtesting, the ability to accurately test a strategy in a flexible framework for periods greater than two years often becomes cumbersome in tools such as Excel, which is often the tool with which fund managers are familiar.

Research into the development of a portfolio simulator for a hedge fund manager was triggered by an absence of any such product available within South Africa, tailored for the South African hedge fund industry. In order to develop a prototype suitable for the investment company to which this product would ultimately be utilised, research was conducted into both the finance and computational theory in developing a suitable simulator. Whilst several financial methods exist for the simulation of a portfolio, this research focuses only on the technique of backtesting as a consequence of the mandate for the prototype.

The initial phase of the project began in 2004 with extensive research into products available globally. Cost-return feasibility studies were conducted in conjunction with the analysis of functionality, particularly since most products required an annual licence fee and were dollar-based. The outcome of the research confirmed the need to develop a simulator in-house that could cater for the nuances specific to the stakeholder. Through a process of system analysis and design, much knowledge was gained into the development of a simulator for portfolio backtesting. Such knowledge is rarely documented for public consumption as it often represents the proprietary efforts and competitive edge of financial

services companies. The objective of this research is to introduce backtesting and its pitfalls, as well as bridge the divide for non-investment professionals such as developers, who will ultimately have to develop the system.

Having surveyed the market, completed a functional analysis and developed sufficient knowledge on the issues facing such a project, a prototype was developed. This research details the performance of the prototype with particular reference to the architecture of the simulator.

The process evolved through various iterations to yield an object-oriented (OO) design that reduced the processing time from approximately twenty four minutes to five seconds per simulation. The knowledge, whilst seemingly obvious in hindsight, provides research insight for any person considering such a development project.

1.2 Research Domain and Scope

The development of this simulator relies on the disciplines of finance and software engineering. Whilst each field is vast, only the relevant concepts from each that pertain to this development have been described in chapters two, three and four which serve as the knowledge foundation for the rest of the thesis.

As explained in section 1.1, whilst several techniques are available for portfolio simulation, this research, as mandated by the stakeholder, focuses only on the technique of backtesting to simulate the return profile of a portfolio. The research path for this thesis began with a study of the challenges that arise from backtesting an investment strategy. In order to understand these challenges, the first iteration of the study concentrated on investing a portfolio of stocks as at today. This study revealed the various mechanics of investing and the variables involved in accurately reporting the portfolio daily. The second phase of the study drew on the knowledge gathered in investing as at today, and attempted to identify the nuances to simulating a portfolio historically. Chapter four highlights these nuances which are considerations that do not affect a portfolio being invested today, but are critical

considerations to achieve realism when simulating a historic portfolio. The scope of this portion of the research is limited to simulating equity-based hedge fund portfolios and does not deal with the numerous additional asset classes which possess their own nuances.

The majority of the research concentrates on developing a software agent architecture that allows the stakeholder the ability to simulate a variety of stock portfolios. This architecture had to provide the functionality to amend the investment logic to generate alternate investment scenarios, whilst also being capable of dealing with the increasing breadth of data available for financial analysis. The scope of this research was limited to development on the Microsoft platform due to the experience of the researcher in this field. In order to be successful, the architecture to be designed had to satisfy the mechanics and nuances of portfolio investing and backtesting. As such, several design criteria were determined at the outset of the research phase and these were used to determine the eventual outcome of the research.

In evaluating the thesis, the research and simulator prototype had to achieve the following:

1. Design and implement portfolio backtesting software which is able to accurately model a stock market portfolio

2. Show that the software is able to recommend an investment portfolio that performs as well as, or better than portfolios generated from currently available techniques.

1.3 Evaluation Methodology

The two evaluation criteria introduced in section 1.2, whilst detailed in chapter ten, are summarised in this section.

If one accepts that the premise of a portfolio simulator is to mimic, as closely as possible, the return profile of a portfolio of stocks that had been invested in the stock market, then one measure of evaluation of the research simulator is the comparison of its output to that of an

actual invested portfolio. An audited invested portfolio represents a verified and controlled dataset against which comparisons can be made. It also represents a dataset in which the mechanics of investing are inherently built in. As expressed by Bernstein [BF98], the covariance of one portfolio to that of another is a reasonable guide to the investor's risk exposure, i.e. "*Covariance with the market portfolio is a surrogate for risk exposure"*. Taking covariance one step further, correlation between portfolios is both a measure of the investor's risk exposure as well as a measure of the similarity in the return profile, and consequently a measure of the expected reward. In accepting that the return profile represents the logic and mechanics of investing, the higher the correlation, the more likely the mechanics and nuances have been correctly implemented. This assertion is made on the basis that software unit testing of each logic module, and the cumulative testing of all modules successfully passed the functional testing phase. Further, the higher the correlation, the more similar the simulation becomes to reality, which is ultimately the goal of the modelling. As a consequence of the number of independent variables affecting a portfolio's return profile, including the discretionary human element of the fund manager, the research acknowledges that a perfect simulation is improbable. The objective of the research was a simulation that was sufficiently representative of reality. Since each fund manager has a discretionary risk tolerance, an acceptable level of correlation for one manager may differ from another. For this research, the acceptable correlation level was a consensus decision between several fund managers of the investment firm for whom the research was commissioned and who employed the researcher. This correlation level was determined to be a minimum of 0.7 between the simulator results and those of the existing simulator or to a market-invested portfolio. As shall be discussed, the final results of the simulator demonstrated a 0.76 correlation to a market invested portfolio which was accepted by the fund managers and confirmed the success of the research. As shall be discussed, this correlation level was achieved through improving the business logic of the simulator to more accurately represent the mechanics of a portfolio.

Whilst the primary goal of the simulator was the accurate generation of a simulated portfolio return profile, the second criterion in evaluating the simulator, was the use of the simulator to generate an investable portfolio. Whereas a regular simulator would be provided a

portfolio of stocks to simulate, this research extended the definition of the simulator such that it was capable of modelling an investment strategy in order to generate a portfolio of stocks. This resulting portfolio would then be simulated to determine its return profile and profitability. As a consequence, investment strategies could be iteratively modelled and simulated and the various results compared in the search of a portfolio which met the manager's investment objectives. In the investment management landscape, there exists no single investment strategy to determine a portfolio of stocks; rather, each investment manager has proprietary approaches that distinguish them within their profession. As such, one objective of the simulator architecture was the ability to model a proprietary investment strategy through the development of a framework which allowed such strategies to be described. Once modelled, several iterations of the strategy could be executed, along with the corresponding simulation of each strategy in order to determine a strategy which satisfied the requirements of the particular investment manager. Chapter ten details the modelling of a proprietary investment methodology and then provides a real-world implementation of a simulator-generated portfolio for the period 2005-2006 where the simulator objective was to generate a portfolio, using the proprietary methodology, to exceed the return profile of the South African All Share Index. As shall be discussed, the resulting portfolio achieved the objective successfully.

1.4 Dissertation Outline

Chapter 2: Investment 101 - Prior to investigating the nuances of calculating the performance of a historic portfolio, it is critical to first understand the mechanics of a portfolio had it been invested today. This understanding reveals the basic principles of investing and accounting for such investments. It also reveals the inputs required to invest a portfolio as at today and to account for its performance each day thereafter. This chapter introduces investment areas such as risk management, portfolio modelling, liquidity, portfolio construction and behavioural finance, all of which are pivotal to portfolio management regardless of whether the portfolio is invested as at today, or simulated historically.

Chapter 3: Portfolio Simulation 101 – As a preamble to the detailed discussion of the backtesting methodology investigated in this research, this chapter provides the reader with alternative methodologies, both in the qualitative and quantitative investment approaches.

Chapter 4: Backtesting - In understanding the detail of chapter two, one understands the mechanics of investing a portfolio of stocks as at today and accounting for its performance as at today. The result of this understanding is the ability to realistically profile the performance of the portfolio. Chapter four enhances this knowledge by introducing several nuances to these mechanics and accounting methods which are the direct result of simulating a portfolio with the benefit of hindsight. These nuances had to be catered for in the logic of a backtesting simulator in order to accurately profile a historically invested portfolio.

Chapter 5: Design Overview – This chapter provides a high-level overview of the simulator architecture to be discussed in later chapters. General design objectives are outlined in this chapter as well as an overview of the logic flow of the simulator.

Chapter 6: Database Construction – Prior to executing successful simulations, a database of historic stock market data must be constructed. Apart from collecting the necessary data, challenges regarding the importing of this data into the database as well as the removal of inaccuracies and inconsistencies within the data are discussed in this chapter.

Chapter 7: Scripting and Parameter Configuration – A simulation environment is defined by the configuration of environment variables. The result of a simulation is a function of these variables. This chapter discusses the improved flexibility realised by the architecture as a result of a scriptable input process.

Chapter 8: Database Access Layer (DAL) – This chapter discusses the logic residing at the database layer. This is the first step of the simulation process, the output of which is a

selection of stocks from a user-specified investment universe which have met the various investment criteria being tested.

Chapter 9: Business Logic layer (BLL) – The BLL houses the logic of the simulator. This chapter discusses the logic and provides further detail on the object-oriented design of the software. The successful implementation of the business logic is determined through the correlation between a simulated portfolio and the similar portfolio which was invested on the stock market.

Chapter 10: Evaluation – The chapter evaluates the thesis by analysing whether the research question was answered and whether the suggested architecture was able to accurately simulate an investment portfolio historically.

Chapter 11: Conclusion – The chapter concludes the thesis by summarising the findings of the evaluation, confirming that the research questions have been addressed and provides suggestions for future work that may improve the research and simulation results.

Appendix A: User Interface layer – The appendix highlights the various user interfaces developed for the software.

Appendix B: Automation – This appendix discusses the various tools and modifications that allow for end-to-end automation of the simulation process.

Appendix C: Database Normalisation – This appendix provides a detailed explanation of database normalisation theory and its application to the schema used in this research.

Chapter 2

Investment Management

"You've got to crawl before you can walk"

2.1 Introduction

As a prelude to understanding the scale and complexity of developing a simulator to mimic the decision-making ability of a portfolio fund manager, prerequisite knowledge of the fundamentals of portfolio management and the mechanics of investing are essential. The objective of this chapter is to provide broad-based, high-level coverage of the fundamentals and domain terminology that accompany portfolio investing, specifically targeted to the reader with limited experience in equity investing. This chapter is not recommended for readers with financial services industry knowledge of the stock markets, however, although limited in scope, it specifically covers those areas of investment finance most directly related to the development of the simulator logic.

2.2 Financial Instruments

In order to invest, one must choose an instrument to invest in. A common example of an investment instrument is property. Property is the instrument into which one invests with the expectation that the value of the instrument will appreciate with time, yielding a gain on the initial investment. One therefore realises a profit or a loss on the sale of the asset. Several

instruments or vehicles exist for investment on the stock markets and are categorised according to asset class. These categories or asset classes vary in the degree of risk and return they offer and cater to the varying risk appetites of investors.

Marx [MMV03a] provides examples of asset classes and instruments within each class:

- Real Assets
    - Property
    - Commodities (Gold, Diamonds)
    - Art and Antiques
- Financial Assets
    - Fixed Income Securities
        - Bonds
            - Treasury Bonds
            - Government Bonds
            - Corporate Bonds
            - Convertible Bonds
        - Preference Shares
    - Equity
        - Ordinary Shares
        - Derivatives
            - Warrants
            - Futures
            - Options
    - Other investments
        - Unit Trusts
        - Hedge Funds
        - Investment Trusts
        - Participation Bonds Schemes

This list of instruments represents a subset of instruments available to investors. This research is focussed on the Hedge Fund asset class with particular focus on the trading of Equity Ordinary Shares.

2.3 Risk-Return Profile

Marx [MMV03a] defines risk as the level of doubt from the investor, that an investment decision will not generate its expected return. The more doubt one has in a particular stock's ability to generate a profit, the more risky the decision is considered to be.

Risk factors are defined as the contributing factors towards the adverse outcome of an investment case. An example of a risk factor may be geo-political risk. For example, an investor who may wish to purchase shares in a Darfur-based company (Sudan), would need to assess the impact of the political instability of the region on the success of the company and its share price before making an investment decision. In this example, the geo-political risk is one of the risk factors in this investment decision.

Return, on the other hand, is defined by Marx [MMV03b] as the value generated or lost as a result of the investment decision; in other words, the profit or loss as a result of the purchasing decision. To continue the Sudan example, the investor in such an asset would expect higher rates of return for assuming higher risk. However, such assumptions are flawed by the fact that higher risk may just as equally translate to increased losses over more conservative alternatives.

Different financial instruments exhibit varying degrees of risk and return. Traditional long-term government treasury bonds exhibit very low risk

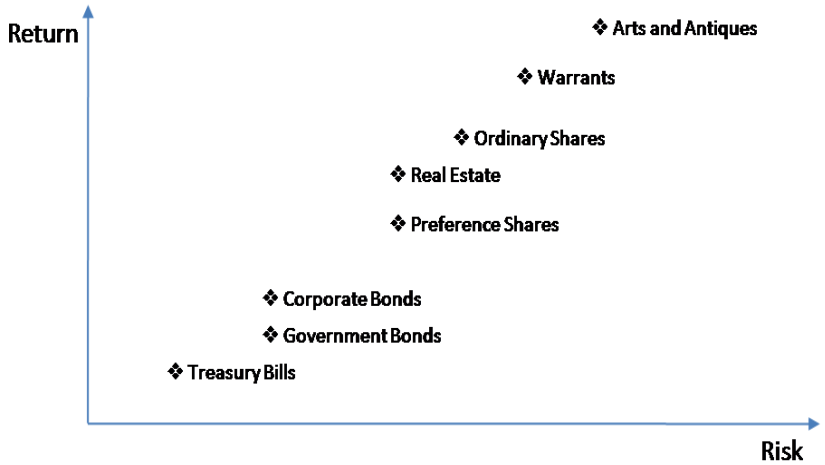


Fig 2-1: Risk-Return profile of financial instruments [MMV03a]

characteristics and therefore offer stable but lower returns than much riskier hedge funds for example, whose expected return could be multiples of the bond return, but whose risk is equally higher. Figure 2-1 highlights the positioning of various instruments based on their risk-return profile.

2.4 What is a Portfolio?

As a first approximation towards a definition, a portfolio may be viewed as a single stock that is owned by an investor. The decision to purchase the stock is based on a view taken by the investor e.g. "*I expect that the price of gold will increase, therefore I will purchase shares in a company whose share price will increase as the price of gold increases, thereby yielding a positive return on my investment when I sell my shares at the higher price."* The inherent risk of such a single-stock portfolio lies in its directional bias. The question that must be asked is "*What happens if the price of gold drops? Surely one should hedge one's bets against such an eventuality?"*

The answer to this question introduces the concept of portfolio diversification, a basic tenet of money management, wherein the investor understands that in order to partially negate the directional bias of a single stock portfolio, it may be necessary to own several stocks, some of which will, in the case of this example, counter the loss effect of a decreasing gold price. Research conducted by Statman [Sta87] suggests that the statistically significant number of stocks in a sufficiently diversified portfolio is approximately 20, provided the stocks themselves are uncorrelated in terms of return profile and business profile. The research concludes that the risk of a single-stock portfolio was 49.2% and the addition of sufficiently diverse stocks reduces the risk to a maximum of 19.2% by the addition of the 20th stock. The addition of between 20-1000 more stocks only reduced the risk by a further 0.8%. In keeping with the above example, the investor should therefore choose several stocks in an attempt to be less directional by hedging certain decisions with the purchase of alternatively behaving stocks, and in so doing, create a *portfolio* of stocks.

2.5 The Investment Universe

The "investment universe" is the term adopted in financial services referring to the domain of investment securities that are applicable to a certain investment decision. An example would be the following: assume the investor has a view that the price of oil/barrel will increase; his/her investment universe of securities would refer to all stocks, derivatives, futures and other investment vehicles within the oil sector that are influenced by the oil/barrel ratio. Examples of such stocks include Shell and BP; Toys 'R Us however, would not necessarily fall into this particular universe.

Whilst Toys 'R Us may seem an obvious exclusion from the investment universe if the investor was focussed on companies that were directly affected by the change in the price of oil, one may argue that the increase in oil price would result in increased cost of fuels (petrol and diesel) which are used by the trucks that transport Toys 'R Us products across the country. The impact of the increased cost of transportation is reflected in the landed cost of the goods, forcing the company to increase the price of the toys to maintain margins. This could ultimately make the company less competitive, reduce earnings and therefore have a negative impact on its share price. This example therefore highlights that Toys 'R Us could legitimately be a part of the investment universe.

The determination of what is eventually included in the investment universe ultimately depends on the conviction held by the investor towards the impact of the investment idea. It is common practice to limit the universe to elements that are directly affected by the investment idea.

2.6 Schools of investment analysis

The three common investment schools to model companies are fundamental analysis, technical analysis and quantitative analysis.

Fundamental analysis deals with modelling the company based on a thorough knowledge of the company's financials, its management, its products, its suppliers and its prospects. The

problem with this analysis technique is that it is not scalable. An investment analyst expecting to provide detailed coverage would be limited to less than twenty companies, simply due to the time and effort required to complete such analyses. One could not cover the global resources universe which contains in excess of 35,000 companies using this technique.

Technical analysis refers to the branch of financial modelling which uses known pattern-recognition methods to determine expected change in the data. An example of technical analysis would be the concept of a moving average, which attempts to "walk the path" of a curve in an attempt to predict local maxima and minima.

Quantitative analysis, the newest of the three forms of analysis, may be classified by some as higher-order technical analysis. The more accurate description though would be to recognise that technical analysis considers the data to be repeatable and attempts to recognise patterns within the repetition, whilst quantitative analysis recognises the randomness of the data and attempts to apply any method of pattern recognition that may aid in extracting value from the data. Whilst an example of technical analysis may be linear regression, a typical financial tool, quantitative analysts would explore the knowledge of such diverse fields as voice pattern recognition to the DNA decoding process of the genome project in order to extract a pattern from the random data. The focus of this paper is on quantitative analysis within a hedge fund framework.

2.7 How do stock exchanges work? What is Liquidity?

The stock exchange, as its name implies, is the facility available to investors to exchange their shares either for alternate instruments or for cash. The stock exchange relies on the fact that there are willing buyers and sellers in order for the exchange mechanism to operate. A new investor, a buyer, will purchase stock from an existing investor, the seller. The price at which an investor purchases and sells a stock is determined by the collective supply and demand for the stock. The more people who wish to buy the stock (greater demand), the higher the price will become and the inverse is equally applicable. The aim of

the investor is to buy the stock at a lower price and sell it at a higher price, yielding a positive return on the investment.

However, it should be noted that a single investor has the ability to affect the price of a share. If, for example, a single investor were to significantly increase demand for a particular stock through consecutively large purchases of the stock, he/she would ultimately have impacted the share price of the stock. The reason for this is based on other investors recognising that a new buyer has entered the market and therefore increased their selling price of the share to this new investor. Supply and demand economics implies that increased demand therefore results in an increased share price. The price at which the investor acquired the first share will probably be lower than the price at which the last share was acquired.

Liquidity refers to the saleability of investment instruments. Saleability is determined by the number of willing buyers and sellers of the particular instrument. A highly liquid asset is one that can be bought or sold without necessarily impacting the price of the share. The danger of investing in illiquid assets is the difficulty in selling the asset should one need to exit the position in a hurry. Liquidity is an important factor in the investment decision.

The number of stocks exchanged on a particular day is referred to as the volume of the stock. By way of example, if a stock trades an average volume of 1000 shares a day, an investor wishing to buy 100,000 shares without impacting the share price may elect to buy approximately 333 shares per day (30%) which would take approximately 300 days of trading. By the time the last share is purchased, the investment case to buy this assed may have expired. It is important to note that it will also take 333 days to then exit the position. Had the investor opted to trade 100% of the daily volume of the stock, that investor would essentially be creating the market. As such, willing sellers would recognise the opportunity to increase the price of the share to such a buyer. The situation is very different if one were to buy 1000 shares in a stock that trades an average of 100,000 daily. The 1000 shares would be purchased almost immediately without much impact on the market price. It is therefore important to factor both liquidity and the volume of daily transactions in a stock into the universe selection in order to remove stocks that are highly illiquid.

2.8 Going Long

In the traditional sense of stock trading, the investor usually purchases x number of shares of a stock for a price of y per share, and realises a profit P on the sale of the shares, should the price rise from y to z.

$$P = x*z - x*y$$

At this stage, the investor may continue to remain an owner of the shares or sell them on the exchange for the asking price of z. This method of investing with the expectation to make a profit due to the increase in a stock's share price is termed *going long* on a stock.

2.9 Going Short

If one can *go long* a stock, can one then *go short* a stock? Yes. The concept of *shorting* a stock implies the ability to profit from the decrease in share price of a stock, a counter-intuitive concept. It is easier to explain the concept using the following analogy:

Assume person A purchases a brand new vehicle for R100,000 on 1 Jan 2007. Person B, the investor, has a view that the price of the same vehicle, brand new, will be R70,000 on 1 April 2007. Assume a perfect world, where one could sell an unused vehicle for the same price that was paid to purchase the vehicle. Person B then borrows the unused vehicle from person A on 1 Jan 2007 with interest of 1%. Since it is unused and just out of the showroom, it is still valued at R100,000 and Person B sells it immediately, accumulating a positive cash balance of R100,000 as at 1 Jan 2007.

As expected by Person B, the price of a brand new vehicle did drop, and on 1 April 2007 a vehicle of identical specification was purchased by Person B and delivered to Person A along with the agreed interest owed. Person A is none the wiser that this is not the original vehicle.

How has Person B benefited?

Cash into bank on 1 Jan 2007 due to sale of the vehicle	*:*	*R 100,000*
Interest earned on R100,000 over 3 months at 5% interest rate	*:*	*R 1,250*
Cash out of bank on 1 April 2007 due to purchase of new vehicle	*:*	*R -70,000*
Interest paid on R100,000 over 3 months at 1% interest rate	*:*	*R 84*
Net cash in Person B's account after complete transaction	*:*	*R 31,166*

In summary, Person B made a profit of R31,166 from taking a view that the price of the vehicle would decrease with time. Further, Person B was able to achieve this profit without significant outlay of collateral or capital.

So if the investor has a mechanism for making money on both the upside and downside of an investment, is this not the holy grail of investing? Though shorting is often used to mitigate risk associated with investing in the stock market, it is important to realize the concept of "unlimited losses" that are possible when shorting investments. When "going long" on a stock, the maximum an investor can potentially lose is the total amount paid for the stock. In the case of shorts though, the investor has not yet paid for the stock, whilst still being obligated to return the borrowed stock to its owner. In the analogy described above, assume that the price of the vehicle rose from R100,000 in Jan to R500,000 in April. Person B is still obligated to purchase the vehicle at the April price as per the contract, thereby incurring a loss of R400,000. It is important to note that there is technically no upper limit to the price that the vehicle could have risen to, hence the term "unlimited losses". Portfolio managers therefore employ far more stringent risk management when evaluating and managing short positions.

An additional term worthy of mention regarding shorting is that of marking-to-market. In the analogy of shorting the vehicle, marking-to-market means that at the end of each day, Person B is required to pay the difference of the borrow price and the current price to the broking bank account of Person A. In simple terms this means that, suppose on day two of borrowing the vehicle the price went from R100,000 (original price) to R110,000. Person B is required to transfer R10,000 to the suspense account of the broking bank. If on day three

the price of the vehicle went to R90,000, the money in the suspense account is transferred back to Person B. The obvious risk of marking-to-market is that should the price continuously rise, Person B is required to have cash on hand each day to manage this *margin call*. The term *short squeeze* is used to denote the action of a continually rising short position which puts such severe cash obligation on the holder that the holder will eventually surrender the position, accepting the losses. One possible method to reduce the risk when shorting stocks is to set aside capital to manage margin calls – capital that cannot be utilised for any other purpose.

It is important to note that the ability to short a stock is not applicable to all types of funds. Unit trusts for example, may not make use of shorting; hedge funds on the other hand are unrestricted in their use of shorting. This thesis and the suggested software development are concentrated on hedge fund investment strategies and fund management.

2.10 Portfolio Risk Management

Portfolio risk management is the natural evolution of the risk-return concept discussed in section 2.3 and as its name implies, is the implementation of financial structures to mitigate the risks identified with a particular investment strategy. The process begins with an identification of the risks, for example, market risk or geo-political risk, and thereafter the selection of appropriate hedging mechanisms to mitigate such risks. By way of example, let us assume that we wish to short a stock (with the view that the price will decline). One scenario to be considered by the investor is the possibility that the price may rise by 10%, against the original view that the price will decline. In such an event the investor would require 10% of the value to be paid to the lender as per the mark-to-market requirement. This money should therefore be set aside for this particular event at the time that the stock was shorted and should only be used for this particular event. It is however not uncommon for the money that should have been set aside for the repayment of the short position or the managing of mark-to-market requirements to be used instead for the purchase of additional stock, thereby potentially worsening the situation and exposing the business to a bankruptcy scenario should the short positions outperform the long positions.

The majority of investment portfolios are governed by investment mandates which dictate the universe and concentration of stocks in the portfolio, amongst other criteria. The objective of the investment mandate is to provide a framework within which the fund manager must operate in an attempt to mitigate risk. The mandate also offers the investor intangible peace of mind that the fund is not being *recklessly managed*.

For example, suppose one were wishing to invest in a fund that offers low risk with consistent, but low returns – one that invests in American government bonds for instance. The mandate for such a fund would potentially stipulate that the fund manager may only invest in bonds. It may also stipulate that no single bond within the portfolio may hold a weight of more than 10% of the value of the fund. These two simple mandate stipulations protect the investor by ensuring that the manager cannot then invest the money in the highly volatile gold market, nor can the manager take a view that the Chinese bond market is more attractive than the American bond market and invest the majority of invested capital into Chinese bonds, thereby unnecessarily exposing the investor to the economic and stock market fluctuations of the Chinese market.

2.11 Investment Factors

An investment factor may crudely be defined as the independent variable in an investment equation. A hypothetical model for the forecasted share price of a stock may be:

Forecast price of share = x + y + z

Where *x = Impact of an oil price increase,*
y =Market perception of the CEO being fired and
z =Winning a huge government tender

Variables x, y and z are considered to be the independent variables that contribute to the movement of the share price. By way of example, let us consider that this equation applies to a company that imports furniture. The impact of a global oil price increase could result in the

price of fuel in the country being increased. This consequently increases the cost of transportation which naturally results in the price of the goods being increased. An increase in the price of goods could result in fewer sales, hence reduced earnings and a reduced share price due to a less optimistic profit expectation. The impact of market perception due to the CEO being fired may be severe, depending on the severity of the crime. Investors will shy away from the company by selling shares, thereby reducing the share price. This would be a result of uncertainty in the management structures of the company to survive such a crisis. The impact of the company winning a huge government tender would be an increased share price as investors assume that the tender will result in significant earnings growth.

In the previous example, one important aspect must be highlighted. Whilst all three variables are investment factors, they may be further classified into macroeconomic and microeconomic or company-specific factors. Macroeconomic factors, such as the increase in global oil price in this example, are factors that govern or are a function of the economic environment. They are also usually out of the control of the company and investor, but do impact both the company and investor. Company-specific factors are those factors which are directly related to the operation of the company. Variables y and z are examples of these.

The investment factor is an important concept as it directly impacts the development of the simulator by being pivotal to the development of the investment model to be discussed in section 2.12. This research revolves around the backtesting of multi-factor models whose objective is to accurately forecast the price of a share and thereby determine the appropriate investment stance to generate a positive return.

2.12 Investment Models

In keeping with the example of section 2.11, suppose the investor had different levels of conviction as to how each of the three variables would impact the share price. Suppose that the investor was less worried about the impact of a rising oil price, but deeply concerned about the management issues. He/she also acknowledged that the tender would be a strong motivator for an increased share price.

The equation could therefore be modified to:

$$Price\ of\ share = x - 3y + 2z$$

The individual factor coefficients represent the conviction of the investor. If the current share price was R1.50 and the equation above yielded an answer of R1.90, the case can be made that this stock is worth investing in (going long).

This equation is a crude representation of an investment model. An investment model is therefore a representation of a financial entity such as a company through the decomposition of its parts, allowing the user the ability to tweak the composition in order to view the effect on the whole. The success of a model may be judged by the realism of its outputs as a consequence of valid, varying inputs.

This combination of factors allows the investor to ultimately *benchmark* one company against another, thereby determining whether a company's share price is over-valued or under-valued, making it a sell or buy decision respectively. Models are therefore engineered investment strategies avoiding conceptual parsimony with heuristics that are composite equations to represent a complex system.

2.13 Mechanics of investing

The cradle-to-grave process of investing on the stock market is a multi-step process which is often made significantly complex as the number of stocks within a portfolio increase. In order to invest on the stock market, an investor must first open a brokerage account with a brokerage firm who will trade the stock on his/her behalf – for a fee which is known as a brokerage fee. Brokerage fees are paid to the broker upon entry into a stock position as well as upon exit of the position. Assuming 0.5% brokerage fees on entry and exit, one understands that the stock must exceed 1% in return in order for the investor to pay the brokerage fee and retain the remaining profits. Given that the stock has been purchased, the investor then monitors the stock to determine the optimal time to exit the position in order to extract maximum profit or negate least losses. In the event that the position is a short

position, the mechanics also require that the investor manage the daily margin calls as per marking-to-market.

For the investor who manages several portfolios, each with a large number of stock positions, the mechanics of investing can be arduous and costly, in terms of losses and brokerage. Too-frequent trading may result in excessive brokerage whilst infrequent monitoring may result in the investor not exiting at the optimal time, thereby sacrificing profit.

2.14 Behavioural Finance

Behavioural finance, an independent field of study, examines the effect of human psychology on investment decisions and the impact on the stock markets. Whilst this vast topic extends beyond the scope of this paper, it is worthy of mention by way of the following examples: Kahneman and Tversky [KT91] define *Loss Aversion*, an example of behavioural finance, by suggesting that the investor is more inclined to sell a stock when he/she has made a profit than when he/she is currently making a loss. The psychology of the investor in the latter case is the hope that the situation will reverse and the loss will either be negated or eventually yield a profit. The danger of such a psychology is that the investor may ultimately lose the complete investment. Another example of behavioural finance is that of *Herding* or *Crowding*. Shiller [Shi01] defines herding as "*The behaviour, although individually rational, produces group behaviour that is, in a well-defined sense, irrational. This herd-like behaviour is said to arise from an information cascade"*. The obvious impact of these scenarios is that a few investors can influence the majority, and in so doing, could create a panic in the market regarding a specific stock, sector or the markets in general.

Behavioural finance is introduced into the chapter to indicate that it can be used as part of the simulator by being an input into the investment model – an independent variable. For example, the share price could be a function of how much volume has been traded in the sector for the past three months versus the past three years; a dramatic increase may be interpreted as herd-like movement and decisions can be made thereupon.

2.15 Summary

The various topics discussed in this chapter represent a microcosm of the volumes of knowledge in the portfolio management space. Each topic was discussed at a novice level, with the view to introducing the logic that one must attempt to mimic in the simulator. Each aspect discussed in the chapter will ultimately result in a separate software model in the simulator to be prototyped.

With the knowledge of financial models and the understanding of the complexity of the mechanics of investing, it is evident that prior to investing, an investor may wish to simulate the model using historical data to determine its probability of future success – assuming obviously that the investor believes that past performance is indicative of future performance. This simulation using historical data is termed backtesting. Whilst this chapter may have introduced several areas of consideration from an investment perspective, backtesting is an equally complex topic with pitfalls that are often unbeknownst to the uninitiated. Chapter four details the complexity and nuances of backtesting in an effort to create a realistic simulation of the investment model.

Chapter 3

Portfolio Simulation

3.1 Introduction

As introduced in chapter one, an investment manager, having determined a potential investment strategy, may wish to simulate the strategy before deploying it on the stock market. The objective of the strategy, as discussed in chapter two, is two-fold; improved returns coupled with reduced risk. As such, the manager has several options in order to generate such a simulated portfolio. This chapter provides an introduction into some of these portfolio generation options and provides the preamble to chapter four where a detailed discussion is presented on portfolio backtesting; the simulation method selected for this research.

3.2 Portfolio Profiling

One of the objectives of a simulation is to determine the profile of a portfolio in terms of both risk and return. Two important criteria to consider are:

1. One can choose to either simulate the potential future profile of a portfolio or one may choose to simulate the historic profile of a portfolio. The former method

requires forecasting of stock performance, whilst the latter describes a portfolio using known historic data.

2. As evidenced by Madhavan [Mad02], one can choose to consider using only the valuation at the start and end points of the simulation; or one may choose to consider the volatility of the path taken, *in addition*, to the values at the start and end of the period when determining the performance of the portfolio.

3.2.1 Forecasting

Bearing in mind the previous paragraphs, possible methods of determining the outcome of a portfolio that are currently in use today are:

3.2.1.1 **Stock Analyst Forecasts** – As described by Domash [Dom02], financial stock analysts, using a myriad of financial models and qualitative assessments of a stock, determine an intrinsic target price for the stock. Referred to in chapter one as the fundamental approach to stock picking, this target price, used in conjunction with the current price, can aid the investment process by ranking stocks in terms of their potential profitability or lack thereof. The target price is a result of detailed modelling of the company in terms of items such as forecast earnings, forecast growth rate, potential takeover opportunities. Domash [Dom02] continues this description by acknowledging one of the disadvantages of this method being the limitation of the stock analyst to cover a significant breadth of stocks in sufficient detail. Further, the qualitative basis for the target forecast implies that the portfolio profile is biased to the view of the individual analyst and his/her interpretation of a stock's return profile. Finally, this method of using target forecasts does not take into consideration the pricing path of the stock/portfolio in reaching the target. Such a path may potentially violate the investment mandate and be unfeasible. Domash [Dom02] recommends the use of the analyst Sentiment

Index and analyst Consensus data, with the expectation that the average rating across analysts may improve the accuracy of the target price forecast.

3.2.1.2 **Monte-Carlo Simulations (MCS)** – Seydel [Sey02] describes the purpose of the MCS as being the calculation of a large number of trajectories of the stochastic differential equation, the averaging of which provides information on the probable behaviour of the process. Glasserman [Gla03] explains that through the modelling of percentage changes in the stock price as the increments of a Brownian motion, where σ represents the expected volatility of the stock price, μ, the expected return and the derivative as the mean rate of return, the practitioner is able to generate, with the use of a randomizer coefficient, several price paths for a particular stock. This method differs from section 3.2.1.1 since the target price is not being qualitatively inferred, but rather the path is being modelled. A primary disadvantage of this method is the complexity of the mathematics and the computational resources required to generate several thousand price paths.

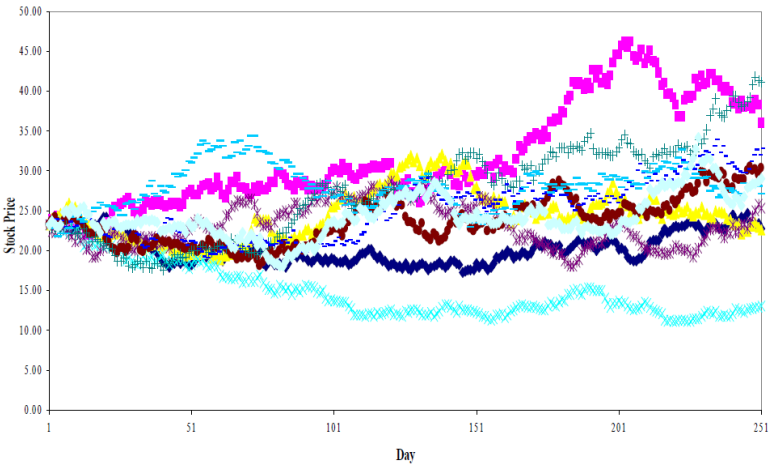


Fig 3-1: 10 possible price paths for a stock over a one-year period using the Monte Carlo Simulation

3.2.1.3 **Technical Analysis** – defined by Stevens [Ste02] as "*the study of any market that uses price and volume only in order to forecast future price movements and trends*". Achelis [Ach00] extends this definition by adding "*... with charts being the primary tool*". Using a variety of trend analysis techniques, technical analysis supports the identification of:

- Direction of the trend
- Strength of the trend
- Support and resistance levels in a trend
- Potential trend reversal

Two primary tools for trend analysis are moving averages and oscillators. The former is used to *follow* the trend and is a lagging indicator that is used to identify a new trend once the price momentum has been established. The latter, such as the moving average convergence-divergence (MACD) oscillator, is used for shorter term trading to signal buy and sell opportunities for prices that trade in a narrow price range.

Technical analysis, much like fundamental analysis, is a forecasting toolset which does not support the testing of portfolios as comprehensively as it supports the testing of trends. It is further disadvantaged by the hindsight employed in applying the various trend analysis techniques.

Whilst this section has only discussed three potential methods of forecasting pricing of stocks, these methods are disadvantaged by the human bias introduced in the forecasting process. As such, the quantitative approach attempts to remove, when possible, the effects of human bias by employing strategies on known historical data. The underlying philosophy for the quantitative manager remains that future performance is potentially described through past performance.

3.2.2 Historical Performance Modelling

Whilst the topics of section 3.2.1 represented the fundamental approach to stock price analysis, quantitative managers approach the problem of simulation through the use of non-linear approaches in an attempt to identify patterns within the historic data upon which to base future decisions. This section highlights some of the possible methods of quantitative portfolio simulation. In the majority of these techniques, historic data is split into in-sample and out-of-sample data sets, where the technique is optimised using the in-sample data and tested for accuracy when presented with the out-of-sample data.

3.2.2.1 **Neural Networks** (NN) – (An in-depth review of neural network theory remains outside the scope of this research) – Karaali [Kar01] describes this technique of back propagation neural networks as the tool utilised by quantitative managers to classify data through the learning process. The classification boundaries may be interpreted as regime changes in the data, allowing the manager to determine the most appropriately bound plane that is representative of the current investment environment. Whilst Sadeque [Sad96] emphasises the danger of overfitting the data as well as the complexity and knowledge required to implement these solutions, Karaali [Kar01] points out the benefit of NN over linear regression as being not assuming the normal distribution or homoscedasticity, multicollinearity and autocorrelation, all of which are inherent in the linear solutions used previously. An issue with NNs is that the sigmoid functions and weightings adopted by the neural network through training often cannot be directly translated to a real-world relationship or explanation, making it difficult to both understand and explain the logic utilised by the NN.

3.2.2.2 **Arbitrage/Pair-Trading** – Vidyamurthy [Vid04] and Karaali [01] both explain that following the implementation of a market-neutral strategy, where the manager would buy long a position and short sell another position of equal dollar value, opportunity exists to deploy a pair trading or arbitrage strategy. This strategy involves capitalising on the spread in price as the two stocks in question diverge

from the mean. Again, through the use of tools such as the Relative Strength Indicator (RSI), Moving Average Convergence Divergence and Correlation, the manager can use historic data to determine the strength of the relationship, the degrees of divergence in history and the relative probability of profitable trading on this pair of stocks.

3.2.2.3 **Backtesting** – Ford [For09] defines backtesting as "*the process of checking a trading strategy on previous time periods rather than applying the strategy for the current time period*". As an example, suppose the manager determined that stock X would outperform the market if purchased today and held for a period of six months. To backtest this theory, the manager would use actual historic data for stock X, looking back several years and consider how it would have performed had it been bought and held at different points along the historic time line. From the results, the manager may infer that, *in general*, the majority of purchases of this stock, when held for six months after purchase date, yielded a positive return, and the probability exists that the stock, if purchased today, would yield a positive return over the future six months.

Whilst this is a simplistic example of backtesting, it provides insight into the need to accurately simulate the performance or a single or multi-stock portfolio using historic data. However, inherent in achieving accuracy, is the requirement to embed expert knowledge of the domain, as well as capture the mechanics and transaction costs of the process. Aside from these considerations, several philosophical issues arise such as data timing, survivorship bias and normalisation amongst others.

3.3 Summary

Whilst this chapter has provided a brief description on a subset of the tools available to profile a portfolio, it is acknowledged that several techniques exist, both for the fundamental and the quantitative fund manager. For the purposes of this research, the researcher was

mandated to concentrate specifically on implementing the backtesting methodology in order to historically profile a portfolio to determine its profitability and risk profiles.

Chapter four concentrates on the theory of backtesting, with specific emphasis on the numerous considerations in order to achieve accurate simulation results. The remainder of the thesis describes the software design considerations and design choices made in order to accurately simulate the mechanics and nuances of backtesting.

Chapter 4

Backtesting

The Devil in the detail

4.1 Introduction

The core of this thesis lies in the concept of portfolio strategy backtesting and its associated complexities. As a first approximation towards creating a simulator, one must first understand the theoretical concept of backtesting in a financial services context. This prerequisite knowledge will ease the knowledge divide in chapter six when the reader is introduced to the architectural decisions taken to achieve backtesting.

4.2 What is backtesting?

Chapter two introduced the concept of model construction and highlighted the considerations and risks associated with developing a model. Having constructed a model, how does the manager now determine its scientific and financial soundness?

Backtesting is one technique utilised by quantitative managers to determine the predictive power of an investment strategy or model. As described by Ford [FOR09], "*Backtesting is the process of checking a trading strategy on previous time periods rather than applying the strategy for the current time period forward.... gauge performance of the strategy based on*

previous market data... may give an indication of how the strategy will perform when exposed to future market data". The objective of backtesting is to produce the risk/return profile of a portfolio as accurately as possible, had it been deployed in the past. This is achieved by building logic that accurately reflects the mechanics of a portfolio and subjecting this logic to historical data series.

Jaeger [Jae02] highlights a critical factor when considering backtesting which is that past performance of a portfolio cannot definitively project its future returns. It does however provide an understanding of the behaviour of the portfolio through historic investment scenarios. For example, in a currently declining oil price economic environment, a portfolio manager may opt to deploy a portfolio whose backtesting results have produced positive returns during historic periods of significant oil price decline.

Continuing the discussion on model construction from the previous chapter, we recognize that a model represents a relationship between a characteristic of a stock and some external factor. Rutherford [RA91] provides a brief explanation into the scientific principles of first establishing credibility of a theory by establish the legitimacy of the suggested relationship; but argues that validation through observation of current data is not enough to confirm such a relationship. Rutherford [RA91] suggests that one form of validation is the demonstration of the predictive power of the theory using evidence/data from the past. An example of this could be: A fund manager believes that there is a relationship between weather patterns and motor vehicle sales in an attempt to determine when best to buy motor company stocks. Whilst the hypothesis for the model may be sound in logic when based on current data, the scientific approach would require that this theory be tested to determine the correlation of the suggested relationship. In standard fashion, the scientist would now test this hypothesis using known, historic, factual data in order to draw a conclusion as to the reliability of the hypothesis. In his/her testing process, an important consideration would be that of realism; is the testing environment similar, if not identical, to the environment for which the hypothesis was developed. The desirable objective is to have the test environment identical to the real-world environment in order to generate the most realistic outcome of the hypothesis.

So what does it mean to backtest a strategy against historical data? Briefly, the objective is such that, given a particular strategy and a portfolio start and end date, it will mimic the day-to-day activities of a quantitative manager, analysing the market on that day, and making decisions, using the strategy as its decision framework, as to whether to buy, sell or hold a particular stock. Its decision will be influenced by the logic of the strategy against the available historic data. With every transaction, all the hidden costs of implementing the strategy are calculated and the end-of-day totals are as accurate and realistic as possible within expected error bands. These totals represent the start-of-day totals for the next day's simulation. This cycle continues until the portfolio end date has been reached. The delta in the start and end values of the investment determine the return of the strategy, which is one possible measure of its effectiveness.

Three items in this section must be stressed: One is the fact that only historic data is used, no forecast data is made available to the simulator, hence the term backtest; two, the backtest must be as realistic as possible; and three, there exists more than one measure to ascertain strategy effectiveness.

4.3 Why is backtesting necessary?

Whilst a quantitative manager may have a firm conviction in the logic of his/her model to generate positive returns, the implementation of the strategy on the stock market is equally critical in the determination of strategy success. The reason for this is that return is not merely a product of the strategy, but inherently includes, amongst others, transaction costs and risk management costs which are incurred as a result of the implementation of the strategy. Examples of such costs could be brokerage and tax paid in the process of buying or selling shares. These "hidden" costs would not necessarily have been factored into the model during construction because they are more a result of the trading environment than a competitive edge. Whilst this may be considered the unavoidable costs of doing business, and one that all funds in the industry are subject to, this is not reason enough to choose to ignore them. Including this information in the research process can only offer a more

thorough and realistic outcome of a strategy, particularly since these costs vary between funds and brokers. The inclusion of this information is made more prominent when one considers that the difference in transactional costs between a high-frequency trading environment and a low-frequency one is material.

Implementation therefore, represents the "reality" variable. It includes the aspects of the investment process which are not catered for in the model itself; and it also offers an additional layer of comfort and confidence in a particular model. For this reason, amongst others, backtesting has become an integral part of the investment process.

4.4 The need for extensibility and automation

Assume one wished to backtest the simplistic model suggested earlier which related weather patterns and car sales. Assume also that one only rates an investment model to have predictive power if it would have consistently achieved its objective of positive returns over the last ten years, had it been deployed since then.

In order to comprehensively test such a strategy, a portfolio manager would have to first obtain the daily weather data for the previous ten years. He/she would then have to obtain the daily data for all car sales on the market, which would generate a significantly large database. Having all this data, the manager would begin looking for changes in the weather pattern on a day-to-day basis and attempt to relate this to the daily rate of car sales. Without the advent of computer automation, this task could not be achieved without severe impact on the manager's time and productivity. Even if it could be achieved to some extent, it would still only be a test of the relationship suggested by the strategy, not a test of the behaviour of the strategy in the market. To compound this, reality shows that fund managers usually generate several strategies daily or weekly. Could they realistically expect to test each of them against data for the last ten years without a simulator, particularly since tools such as Excel which are most familiar to the manager are often inadequate for such a task?

To answer this question, we look at the traditional manager of yesterday. He/she would choose one of two paths after constructing a model. On the first path, the manager could justify that the model does not require backtesting because it is based on a collection of academic white papers or it is based on a strong "gut feeling" that has evolved through years of experience in the industry. Or he/she may choose the second path requiring the scientific stance of conceptual parsimony wherein one may test a hypothesis on the most basic of all assumptions for the hypothesis. The manager would therefore conduct a series of backtests whereby the model is stress tested against a small subset of the market and for possibly one or two short time periods, to achieve a crude idea of the models performance over time. Neither of these approaches would satisfy the diligent scientist.

Hence the need for a solution which would allow the manager the ability to test different strategies over extended time periods with a focus on realism and accuracy.

4.5 Adding additional complexity to traditional backtesting

Backtesting is a computationally expensive, number crunching exercise which relies heavily on finding patterns of recurring relationships within data. As such it is a test of relationship strength using historic data.

The simplest form of backtesting a portfolio would be to compute the change in price of each stock between the start and end date of the simulation period. This process is simple to compute, requires minimal input data and can be easily modelled in a spreadsheet tool such as Excel. However the correlation of this portfolio valuation to the actual accounting data that results from deploying the same portfolio on the stock market would be low. The reason for this is that the actual portfolio incurs ancillary costs and revenues during its lifetime which are not captured when simply comparing the prices at start and end date. By way of example, peaks and troughs during the lifetime of the portfolio may have resulted in portfolio reweighting or closing of positions, both of which incur transactional costs which reduce the overall return of the portfolio.

In attempting to improve the backtesting process, this research suggests the development of a simulator that can produce portfolio analytics on a daily basis, taking into account as many of the factors that influence the portfolio valuation as possible. This would result in an accurate depiction of the performance profile of the portfolio. The research further improves the process by extending the knowledge of software agents.

The alternative to the development of such a rules-based interactive system would be the use of a neural network or genetic algorithms. However these techniques were considered inappropriate to the design of a process that is required to be deterministic. The explanation for this lies in the fact that the outcome of each step in a deterministic system can always be explained through the embedded algorithm or logic and is a mathematical function of the input. Skepticism around non-linear processes such as neural networks arise as a result of the complexity in explaining the sigmoid functions or the path taken in the decision-making process within the different layers of the network. The bias towards deterministic systems whose outputs can be fully explained is also driven by the need to satisfy the investor, who expects to understand the process before committing capital to it.

4.6 Considerations when backtesting data

Although it may appear to be easier to test a strategy on given data rather than forecast or extrapolated data which has the added complexity of extrapolation itself, there are several considerations that make backtesting a science on its own. Like any other field, when testing a particular theory, the testing process must operate within strict guidelines to ensure that the results are not contaminated. This section attempts to explain some of these considerations.

4.6.1 Realism with costs

Coggin [CF98] highlights that an investment strategy simulator for backtesting "*should incorporate as many of the real world delays, constraints and influences into the simulator as* possible". Given that the objective of a simulator is to mimic that which it is simulating, one

acknowledges that the desired simulator output has a correlation of one to that of the actual output. Such a correlation is only possible if all factors which influence the actual object being simulated, are catered for in the simulation. As such, a simulator is only as good as the degree of realism it achieves, given that it may not be possible to incorporate every influential variable into the simulation. Coggin's [CF98] reference to "*real world*" may be translated to mean that the simulator must therefore attempt to, as much as possible, mimic stock market reality, by including such items as transaction costs, taxes, dividend payments, rights offers and de-listings. These items would not have been included in the model construction which only focuses on mathematical relationships between the investment factors. These are costs imposed by the "real world". The simulator also needs to provide a realistic pricing environment with sufficient detail for the simulator to make informed decisions. If one were to simulate a portfolio of stocks and compare its results to that of a portfolio of identical stocks actually invested in the market, the investment process path and the results between the two portfolios should be highly correlated.

4.6.2 Look-Ahead Bias

This bias refers to the use of data which was not available at the time of implementation, aptly described by Narang [Nar09] as "*getting yesterday's news the day before yesterday*". Suppose the following strategy: use the week's average closing price of a stock to determine whether to buy or sell. When being backtested on Wednesday, November 18 1998, it cannot use the average of that week which would include the 19th and 20th, even though this data resides in the database. This would not represent reality because the investor cannot use information today that is only available tomorrow. Chincarini [CK06] further explains the need for "*vigilant avoidance*" of look-ahead bias in backtesting in the attempt to mimic reality. In order to alleviate this problem, the simulator must discard, at start-up, all data that falls outside the test range time line. Obviously, as we move along the time line, that data which was discarded yesterday is included in today's simulator knowledgebase. In appendix D, "AI Agents", the concept of artificial learning will be further explained, in particular the ability to rationalise decisions through the use of historic cause-effect relationships and an evolving percept sequence.

4.6.3 Time Lag

Time Lag is another critical consideration to achieving realism in backtesting. There are two types of data that may be contained in a database for backtesting. One is market data, such as the daily closing price of the stock; the other would be company specific data which is released annually, bi-annually and sometimes quarterly. The latter is usually disseminated via the company's release of their financial statements, but becomes available by the data vendor often three months after the event. Such a lag must also be modelled in the system with the environment not having access to such data until the day it became available via the vendor. Any calculation using this data must use the appropriate data. An example would be the price-earnings ratio, which is the ratio of today's price to the last released annual earnings of the company. O'Shaugnessy [OSha98a] uses the example of a company that releases its earnings on 1 January of each year and we are doing the P-E calculation in February 2003. The earnings figure to be used as the denominator should be that of January 2002. The reason for this is that although the company would have released earnings in January 2003, it only becomes available on 1 March 2003 by the data vendor, so any calculation between the periods January 2002 to March 2003 should still use the earnings as of 1 January 2002 to maintain realism, accommodate the time lag consideration and avoid the look-ahead bias.

4.6.4 Survivorship Bias

O'Shaughnessy [OSha98a] describes survivorship bias as a statistical bias introduced as a result of determining results based only on data that is current, without regard for historic data. Bernstein [BD98] states that "*of the known sources of [statistical] bias, survivorship bias is by far the most important*".

An example of this would be: suppose a Company X was listed on 1 January 2000 and de-listed on 1 January 2001. If we tested a strategy from 1 February 2000 to 1 August 2000, this company could possibly have been selected and would have subsequently influenced the

model's predictive power. However, by working on the basis that this stock no longer exists on the exchange, and therefore removing it completely from the database, the results for the same test may, during the same period, prove to be significantly different, possibly even enhancing the predictive power of the model. On 1 February 2000, there was no way for a manager to know that the company would be de-listed in eleven months time. To assume so, would be to violate the first and second rules of this section, realism and look-ahead bias.

Gregoriou [GK04] goes on to further explain that "*When only living funds are considered, the data suffer from survivorship bias because dissolved funds tend to have worse performance than surviving funds*". As a result of the removal of the worse performing data, the average of the data is essentially increased, in essence producing a more positive representation than would actually have existed had all the data been incorporated. Bernstein [BD98] comments that "*the longer the historical time period involved in a comparison, the worse the impact of survivorship bias*".

Unfortunately, as explained by Fabozzi [FFK06], the solution to survivorship bias is not often a simple one. Fabozzi explains that the "*addition or deletion of companies from the portfolio introduces a nonlinearity in the model and precludes the use of tools such as OLS*". Companies de-list for a large variety of reasons: amongst other things, either they no longer meet the criteria for a listing on the exchange, have declared bankruptcy or are bought out by another company. The list is lengthy. Whilst the first two examples mean complete de-listing and the portfolio, both in the simulator and in reality, are liquidated of this de-listed stock, the third example of a company buyout/merger requires specific software code for each type of company de-listing as different companies exhibit different characteristics during a buyout. The fact, however, is that the shares still remain in the portfolio and on the exchange, but in an alternate form. For example Company X buys Company Y, and the deal is structured such that for every ten shares of Y, the shareholder now has one share of X. The specific deal structure for this buyout would need to be part of the knowledgebase of the simulator which would have to make the necessary adjustments on that simulation date to remove stock Y and acquire the necessary volume of stock X along with the costs that are associated with

such a transaction. Whilst the list of de-listings is lengthy, so too is the list of scenarios, unique for each transaction.

To gather de-listed, company specific information, for a historic period of, for example, thirteen years, is an arduous and time-consuming task and is thus often overlooked.

4.6.5 Data Timing Consistency

Leading on from the time-lag problem discussed earlier, it is important for the simulator to be consistent in the data it contains. An example would be pricing data. Whilst data vendors have the ability to stream data "live" from the exchange, most research houses choose to store snapshot pricing in the database. However, a common error occurs if the pricing is not captured consistently. An example of inconsistent data timing would be where some stock prices are captured at midday whilst others are captured at the close of the day. The variance in pricing throughout a day may be extreme and this variance will be captured if consistency is not maintained. What is meant by this is: Stock X may have prices at midday and at close of 123c and 100c respectively and Stock Y may have prices of 100c and 123c respectively. If the database contained midday pricing for Stock X and closing prices for Stock Y, the accuracy of the outcome is intuitively inconsistent because the simulator, looking back to that day, would use 123c for X and 123c for Y. The price for X has been unrealistically inflated. A decision should be taken as to which time will be used and that should be maintained throughout the data. Further, consistency of data fields should be a primary objective across geography/sector/instrument.

4.6.6 Data Mining

O'Shaugnessy's [OSha98] quote "*Torture the data enough and it will confess to anything*" embodies the essence of the data mining bias. Backtesting, by the nature of its iterative processing, generates volumes of data. The data are statistically prone to containing periods where performance was remarkably consistent. It is an error to forcibly rationalise this end result in order to convince oneself that this strategy satisfies the predictive power being

sought. O'Shaunessy [OSha98a] goes on to explain that it is best to test a strategy over several periods of data to determine it's predictive power and consistency. This is acceptable and is in direct contrast with alternative methods to find a pattern and then try to make sense of the answer.

4.6.7 Result Interpretation

In attempting to rationalise results, the manager must be wary of statistically over-fitting the models to the data. An example of this would be multi-factor models where several of the constituent factors are highly correlated and therefore skew its overall predictive power.

The units of data should also be checked when interpreting the results of a backtest. If the average monthly price of a stock is being calculated and compared to another stock's average price, ensure that the average for both is calculated the same way; both should either have total price divided by number of calendar days in the month or total price divided by number of trading days in the month.

It is imperative that the results of a backtest be analysed, not tortured. Since a backtest is essentially selecting a universe of stocks that have met a particular investment hypothesis that is to be tested, the results of the tests should be viewed with suspicion if they contradict or over-estimate a particular phenomenon. If accounting principles dictate that a variable is meant to decrease in a particular scenario, but the test indicates otherwise, and yields exceptional results, it is prudent to question the rationale of the simulator in that instance rather than attempt to rationalise the results.

4.6.8 Representative Data

Whilst it is agreed that statistical significance increases with the number of data points, the availability of data is a significant issue for simulators. Having 30 years of daily price data would be ideal; however, most exchanges did not collect data, or at least, did not collect

electronic data, that far back. Tortoriello [Tor08] explains that "*reliable backtests work through a variety of economic and market conditions*".

When assessing the period of data being used, some of the questions Tortoriello [Tor08] asks are:

- is the available data representative of the environment – are all the necessary financial indicators being captured;
- does it contain periods where the markets have been volatile; does it contain both bull and bear markets as well as economic recessions?
- does it contain sufficient data to represent the various regime shifts or inflection points where the environment changes and is indistinguishable from its history?
- does it sufficiently represent the nuances of each company's stock price history?

Within the South African context, data representation prior to 1996 is sparse and inconsistent, due to the non-electronic form of trading used by the JSE. It was only in March 1996 [JSE96] that the JSE moved onto an electronic platform which allowed the collection and storage of stock exchange data on digital medium. As such, managers are forced to accept that daily data post-1996 is sufficiently representative for backtesting. This however is a compromise which should otherwise be avoided. One primary reason for this is that the data does not incorporate the 1987 stock market regime shift which should be one scenario that all portfolios are backtested against. The years 2001-2002 also marked a structural shift in the global financial system due to the dot-com bubble bursting. These structural shifts allow for strategies to be stress tested in periods of non-performance on the market. Essentially, the more representative the data is of different scenarios, the more explanatory the return profiles of backtested portfolios.

Any variable that may be used in the investment decision-making process or that influences the outcome of the investment decision should be housed within the database and used in the simulation model. Examples of such variables could range from one as obvious as interest rates to one as obscure as a potential war on foreign soil whose impact on the oil price would then influence the price of a share, thereby influencing the investment case.

Whilst every effort should be made to capture as much data on as many of these variables as possible, with an almost endless list of share price drivers, one must be pragmatic in selecting only those that most affect the investment decision. In the example of war on foreign soil, whilst it is acknowledged that this will impact the share price indirectly, it is often not possible to quantify such an event making it difficult to generate, store and utilise such data within a model.

This section has highlighted the fact that a stock price is influenced by a variety of factors made up of macroeconomic variables and company specific variables. The sheer numbers of factors that potentially influence a stock price make the process of capturing all representative data unfeasible. As such, it is important to determine the primary drivers for stock price movement and at minimum ensure that the data coverage includes these variables. Principle component analysis (PCA) is one suggested technique to accomplish this exercise. Jollifer [Jol02] defines PCA as "*The central idea of PCA is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the dataset. This is achieved by transforming to a new set of variables, the principal components which are uncorrelated and which are ordered so that the first few retain most of the variation present in all of the original variables.*" Such analysis is useful in whittling down the myriad variables that one may consider to influence the investment decision.

4.6.9 Deflation

Another criterion to be considered in the implementation of backtesting is that of deflating values. Suppose one were to run a simulation from 1990 to 2007 with R1,000,000.00. The

first question is whether this amount represents the starting amount for the first simulation in 1990, or the amount used for the last simulation in 2007. The reason this is important is because inflation/deflation must then be catered for. If this amount represents the starting value in 1990, each simulation thereafter needs to inflation-adjust this amount, which will, in the case of South Africa, realise a 2007 amount which is significantly higher than R1,000,000.00. The converse is true if this amount represented the final value in 2007.

4.6.10 Clean data

Data forms the foundation of the backtesting method and therefore needs to be as comprehensive as possible to achieve the simulator objective of market reality. Comprehensiveness, however, is only one of the factors by which the data is evaluated. Another would be the quality of the data in terms of errors and inconsistencies that it may contain. To paraphrase Kumiega [KV08], if one accepts that the accuracy of the output of an equation is contingent on the accuracy of the input data viz the simulator output is only as accurate as the data used in the simulation. For the purposes of this research, all future references to c*lean* data refer to data time series that are error-free, accurate and consistent.

Share price data is made available through stock exchanges, which provide systems to allow individuals, investment companies and data vendors to purchase and extract data from them. Common practice for most investment companies is the use of data as provided by the data vendor, who in turn take the responsibility of purchasing and cleaning the data. For global investment companies, whose portfolios may include stocks from various geographies and exchanges, it is often more economical to pay a single data vendor licence than to pay and interface with, each individual exchange. This also makes sense for the data vendor, for whom, economies of scale allow for a discounted price to be offered to the investor. Kumiega [KV08] provides a detailed explanation of criteria when selecting a data vendor and these have been surmised below.

The collection and dissemination of stock market data is predominantly an IT service. As such, most financial services companies would have to deliberate as to whether this

functionality should be bourne in-house. In determining whether the responsibility of data collection and verification should be outsourced or retained internally, the following issues were raised:

1. The data vendor, liasing with all global exchanges, is able to offer companies a single, reduced cost of ownership to global data feeds due to their economies of scale. If a company were to attempt to replicate the functionality of the data vendor, they would incur the full price for data retrieval from each exchange globally.

2. Financial services companies do not enjoy a competitive advantage in setting up and maintaining IT infrastructure to warehouse global stock market data

3. The cost of building tools to capture and distribute real-time data from the exchange to the investor is unjustified for a financial services organisation

4. **Ensuring the accuracy of data in a global stock market database is more an IT responsibility than a financial role.**

5. **Standardising the data across companies globally and consolidating corporate event calendars is a time-consuming and labour intensive task.**

From this list, it is evident that the role of collecting and disseminating stock market data is best outsourced to IT data vendors.

Bullet points four and five are of particular importance as they directly impact this research project. Investment teams often do not have the luxury of dedicated software developers and data warehousing knowledge and therefore rely on making decisions based on the data received from the data vendors. Reality, unfortunately, has proven that the reliability/accuracy of data from vendors is often not acceptable. As an example, the closing price of a stock for a specific data differed across the various data vendors.

There are several reasons why such an error may occur. These include, but are not limited to the following:

1. How the vendor defines and thereafter calculates the value – in the example above, all the closing pricing from all the vendors were actually correct. The distinction was drawn regarding the method of determining the price. For example, one vendor used the average price during the auction period of the exchange close (usually last five minutes) whilst another vendor simply used the last price publicly reported by the exchange. Both are correct as they have consistency in their methodology and across their respective databases and they have justified reasons for their choice of definition.

2. it is not uncommon for the exchange to have provided the incorrect data feed and the subsequent correction may not have been corrected by one of the vendors

3. an event, such as a currency redenomination in a country, may force a historic revaluation of share prices within that country

4. different vendors "rebalance" a share price differently following a corporate event such as a merger, acquisition or stock split

5. The majority of exchanges only became electronic in the early 1990's and as such, data pre-1990 was manually captured and as a result error-prone. In such cases, vendors have had to manufacture, in accordance with best practice, historic price series to correct or extrapolate time series

6. most exchanges only have data available since becoming electronic, unlike the US where data is available from as early as 1960

7. techniques for standardisation of data vary amongst the data vendors

8. Each vendor uses a different exchange rate based on the time of capturing the data

With this knowledge of the inconsistencies and potential errors with data from data vendors, many investment houses have opted to set up internal teams to further "clean" the data received from data vendors and create their own historic databases. Over the years, these teams have had to implement complex techniques to remove errors from the data. An example of such would be the expert system implemented by Reuters in 1993 [Bla93] which attempted to use a combination of expert knowledge gathered from investment professionals and human intervention to improve the quality of data distributed by the company.

Paraphrasing Morgan's article *Cleaning Financial Data* [Mor02], several problems are encountered in attempting to clean historic financial data:

1. Either no data exists
2. Impossible values input
3. Inconsistent values input
4. Unlikely values input

Morgan's [Mor02] article goes on to further explain these four problems as follows:

1. Missing data can be a result of two very different issues – either the data was not meant to exist, for example there is no data on Christmas day or a national holiday as the stock exchange in that specific country was closed – this is referred to as structural missing; versus observational missing which is simply data that should have been input, but is missing for one of several reasons, potentially listed in the *list above*.

2. Impossible values are the easiest to clean as they breach the rules for the data. An example of such could be the stock price which can never be a negative number or zero. A database query could very simply extract all prices that violate these rules.

3. Inconsistent values, along with point four, unlikely values, are possibly the two most difficult problems to remedy as they require domain knowledge in order to determine whether the values are actually inconsistent or not. An example of a valid inconsistency could be a sudden price change in a share price time series which could be attributed to a currency redenomination. This very example could also be used to illustrate an inconsistent value error where the value was once provided by the exchange in ZAR cents as 1200 pre-1995 and then as ZAR rands as 12,00 post-1995.

4. Unlikely values are equally difficult to remedy. As in point three, unlikely values are those that could be theoretically correct, but, as the name suggests, are unlikely. Morgan suggests by way of example, a value such as the market capitalisation which is often represented as a factor of '000 000– so a value such as 200 in this field may seem unlikely for most situations, except for the very rare examples of companies who actually do fulfil this criteria. Without fore-knowledge of the company, an algorithm would not be able to distinguish whether the value was valid or not for a specific company.

Whilst these problems do not appear insurmountable in isolation, the complexity arises when one attempts to develop algorithms to fit large data sets, particularly ones that include multi-geography, multi-currency stocks. Chapter six will further explain the implementation details and complexities as well as potential solutions that have surfaced through further investigation into this aspect of the pre-processing.

4.7 Is the simulator holistic?

With specific focus on backtesting within the investment framework, the manager must ensure that the backtesting is as realistic but also as holistic as possible. Holistic, in this context, may be evaluated by how broad the coverage of the various investment mechanics has been incorporated in the logic of the simulator.

Rudimentary backtesting of a portfolio will generally focus on the stock's price at the start and end of the period being tested, without emphasis on the portfolio performance during the period. A simple percentage change in stock prices provides a snapshot of the performance of the portfolio being backtested.

Kumeiga [KV08a] provides a detailed breakdown of the various aspects of the investment process which must be considered for the process to be holistic. These are summarised as:

1. Daily risk management
2. Daily liquidity monitoring
3. Daily stop loss monitoring
4. Daily investment case validation
5. Portfolio rebalancing and re-composition

The more mechanics of investing that the simulator incorporates, the higher the accuracy and correlation of the return profile of the simulated portfolio to the real-world portfolio.

Commercially available simulators may elect not to incorporate this level of investment mechanics into their simulator logic due to the increased complexity. However, it can be argued that any and all aspects of the investment process must be incorporated in order to improve simulation accuracy.

4.8 Summary

In summary, chapter two introduced the basics of investing in a portfolio as at today and accounting for the portfolio performance. This chapter has introduced several nuances that are applicable to the historic simulation of a portfolio and add to the mechanics introduced in chapter two. Such nuances as look-ahead bias and time lag are examples of issues which need not be catered for if one were to implement a portfolio as at today, however these issues, if ignored, have material impact on the reporting of portfolio performance for historically simulated portfolios. The objective of chapters two and three are to provide the reader with the basic logic that will have to be accommodated in both the database design and business logic of the simulation engine to be developed.

Chapter 5

Architecture Overview

"A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away." Antoine de Saint-Exupéry

5.1 Introduction

Chapters two, three and four served the purposes of introducing the reader to the prerequisite knowledge required to simulate a portfolio of stocks. This, and subsequent chapters, are dedicated to explaining the integration of the prerequisite knowledge into the software architecture as well as providing a high-level decomposition of the resulting architecture. Specific emphasis shall be placed on how:

- Simulation time was minimised
- Extensibility and Flexibility were achieved via the architecture
- Accuracy and Realism were achieved via the architecture

These chapters shall not provide programming-level detail as this is considered proprietary as well as unnecessary to the research question.

5.2 Objectives of the Simulation Engine

As the point of departure for the architectural design process, the outcomes of the development were determined. These outcomes were determined in consultation with the primary stakeholder, the portfolio manager and primarily relate to some of the shortcomings of the existing backtesting model which was built in Excel. These shortcomings will be discussed in more detail in chapter ten during the evaluation and comparison of this research to the legacy system. For the development of a backtesting simulator, the following initial objectives were identified:

- Accurate, realistic results which avoid the pitfalls of backtesting as discussed in chapters two, three and four
- Flexible, extensible agent framework built with a modular design to cater for varying investment models
- Scriptable and command-line driven – user should be able to configure all the parameters of the system via a text editor and run the system via the command line
- Universe agnostic – the logic of the simulator should not require modification should the underlying database content be modified from, for example, Brazilian coffee companies to South African Gold companies
- Millisecond response mindset – one measure of success of any simulation performed by the system should ultimately be simulation time
- Investment Instrument agnostic
- Handling of large datasets, outside the capacity of Excel
- Must satisfy the finance and backtesting issues highlighted in chapters two and three

5.3 System Lifecycle

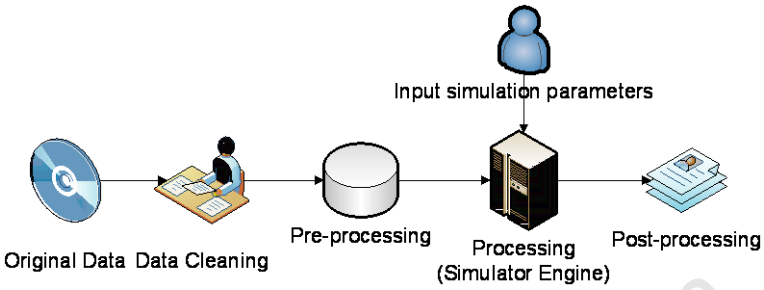


Fig 5-1: System Lifecycle

Following on from Madhavan's Investment Cycle [Mad02], the system lifecycle of the simulator is not limited to the operation of the simulation engine, but extends further to the ancillary activities that need to be performed in order to achieve a successful simulation. This process may be divided into three distinct steps:

1. The construction of a database with accurate/error-free data
2. The process of simulation
3. The analysis of simulation results

Step 1, the construction of a database with verified data, essentially begins with the collection of historic data from the various data vendors. Whilst it is desirable to receive all the data from a single vendor, this is generally not possible as vendors specialise in different areas of the financial spectrum. An example of this would be financial analyst forecast data. This data is primarily the property if IBES and owned by Thomson Financial. Whilst a similar dataset is available from a vendor such as Bloomberg, one cannot achieve the same level of comprehensiveness as you would get from collecting data directly from Thomson for example. One challenge in this process is the stitching together of the data from the various

vendors due to the fact that each vendor has their proprietary database schemas and methods of stock identification. A second challenge arises from the volume of data to be stitched together.

Once the data has been collected from the various vendors, the challenge becomes how to import this data into the database. Data is most often provided in the form of CSV files which need to be parsed before being imported into the database. The reason for vendors providing data in this format is due to the schema-agnostic nature of the format. It is simpler to provide data in this format and allow the user to reformat for their purposes than to provide custom schema outputs for each client request. Parsing has to ensure that the data is correctly formatted in terms of data types, data sequence and column sequence amongst others. SQL Server Integration Services (SSIS) in SQL Server 2005, the evolution of Data Transformation Services from SQL Server 2000, was used in partial fulfilment of this challenge. Additional software had to be developed with the necessary logic to first parse the file before SSIS could perform the SQL bulk insertion task. A more detailed analysis of this parser will be provided in chapter six.

Following the importing of the data into the database, it must then be sanity checked for consistency and accuracy. As discussed in chapter four, and further discussed in future chapters, this task is often the most complex of the pre-processing tasks in the design and implementation phase. Whilst several data verification tools were developed, the quantity of data combined with the variety of potential errors makes this task significantly complex.

At this stage in the lifecycle, the data has been collected, inserted into the database and verified for consistency and accuracy. The lifecycle now proceeds to Step 2, the Simulation Engine.

The Simulation Engine operates in two phases, the database phase and the simulation phase. When a simulation is executed, the engine accepts as its input the parameters of the simulation as configured by the user, as well as the definition of the universe that is to be considered the relevant pool of stocks from which it may construct the final portfolio. Using

these inputs, the engine then queries the database attempting to first reduce the universe based on stock availability, and then executing the investment model that further reduces and ranks the resulting universe. The output of the database query is a portfolio of stocks that have been both normalised and ranked based on the investment criteria being tested. It is this ranked portfolio of stocks which will then be simulated. At this point, the object-oriented software simulator accepts the database result and simulates the daily performance of the portfolio. It takes into consideration the various pitfalls and considerations of backtesting as discussed in chapter four, whilst implementing the necessary financial constructs as discussed in chapter two, to ensure realism of the simulation. The output of the simulation is a detailed portfolio decomposition for a particular scenario for a particular investment model.

A single simulation of an investment model is generally not the objective of the development. In practice, the fund manager may simulate several hundred or thousand permutations of varying investment models. The result of bulk simulation is bulk file creation which cannot be analysed manually. Step 3 was introduced to the lifecycle to automate the process of analysing the simulation results by ranking the results according to a weighted score method based on key performance indicators that were identified in assessing a portfolio's performance.

The relevant highlights of each of these integral phases of the processing sequence will be identified and addressed in the sections and chapters to follow.

5.4 Simulator timing

As outlined in the objectives of section 5.2, one criterion in the evaluation of the success of the simulator is the round-trip time from simulation execution to resultant portfolio.

Assume the manager has developed a single investment model and wishes to simulate its performance on a monthly basis for the past ten years. This means that the investment model will be executed at the start of each month for ten years; equating to 120 months of

simulation. If the *total time* for a single simulation is on average five seconds, the total time to simulate a **single** investment model monthly over ten years will be

$$120 \times 5 = 600 \text{ seconds} = 10 \text{ minutes}$$

Realistically, managers will not backtest a single investment model. A manager will create several investment models and multiple variations of each. Assuming the average number of investment models to be simulated monthly over a ten year period is 1000; this translates to 10,000 minutes of simulation time or 166 hours / 7 days continuous operation.

To emphasise the importance and impact of simulation time to the overall design process, consider that the impact of a marginal increase of 2 seconds to the simulation time for the above scenario yields the following:

- 5 seconds + additional 2 seconds = 7 seconds
- 120 tests x 7 seconds = 840 seconds = 14 minutes
- 1000 iterations x 14 minutes = 14,000 minutes
- 14,000 minutes = 233.3 hours = 9.7 days
- Therefore, a 2 second increase = 9.7days – 7 days = 2.7 days increase in simulation time

In order to determine the maximum acceptable simulation time, the primary stakeholder for this research, the portfolio manager, provided the following mandate:

- Monthly tests across a ten year historic window = 120 tests per model
- Approximately 1000 variations of the model will be tested
- Results to be provided within one week = by the end of 7 days of continuous operation

Working backwards from this mandate:

- 7 days = 168 hours = 10080 minutes
- 10080 minutes / 1000 variations = approx 10 minutes per variation
- 10 minutes / 120 months of simulation = 5.04 seconds per test

For this reason, it was critical that the following restriction be placed on the design:

- Total simulation time per iteration: max 5 second

5.5 Software architecture

Several critical design criteria were decided at the outset of the software architecture phase. The design was biased by default towards the Microsoft platform for three reasons:

1. this was the skill set of the researcher
2. the stakeholder organisation only supported Microsoft solutions
3. the stakeholder organisation had developed both people, processes and tools to aid development on this platform

As a consequence, the primary guidance for architecture design was based on the Application Architecture Guide [MHH+08] provided by Microsoft's Patterns and Practices division. This guide represents recommended best practice to optimise the use of Microsoft technologies in architecture design and solution deployment.

Three critical design recommendations from the guide [MHH+08] which align directly with the policy of the stakeholder organisation were:

1. **Three-Tier Design** - Best practice recommends this layered design pattern, as illustrated in figure 5-2 because:

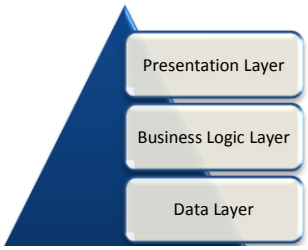


Fig 5-2: Three-tier architecture

   a. it attempts to isolate functionality to encourage reusability

   b. in an application where different layers may require substantially different computational power, each layer can be executed on separate, appropriately-specified hardware

   c. it simplified debugging by first allowing individual classes to be tested and thereafter, individual layers to be tested

   d. it enables modification of a single layer without modification or compilation of other layers (provided the API of the layer is not modified)

   e. it affords flexibility by allowing multiple forms of a layer to co-exist, for example, one could have both a web-based and a windows forms-based presentation layer coupled to a single business logic layer

   f. allows different teams of developers to work independently of each other on the different layers, being constrained only by the interface specifications of the layers

   g. prevents a layer from communicating with layers above it, and only permits communication to the most immediate below it – significantly aids debugging by narrowing the surface area of the layer

2. **Object-Oriented Design** – the guide recommends this programming technique:

a. If there is a desire to model an application based on real-world objects and actions

b. There is a need to encapsulate logic and data of common entities within the system to encourage reusability

c. If the business logic is complex and requires abstraction and dynamic behaviour

d. If a detailed business analysis has been conducted to identify, actors, entities, properties and behaviours

e. Specific to this research, the OO-design style compliments the software agent approach which serves as the primary orchestration object (to be discussed in chapter nine)

3. **Loosely-coupled, Modular, Stateless Design** – the guide recommends:

a. loose-coupling between layers with communication being governed by well-defined interfaces – this facilitates abstraction and allows reduces dependencies

b. Modularity is encouraged throughout the design process to ensure that functionality is encapsulated and the self-sustaining modules can facilitate a pluggable architecture, allowing the design to be flexible by accommodating varieties of modules for the same plug-in interface

c. Stateless design ensures that the various modules and independent utilities can operate without knowledge of the state of the rest of the system. This facilitates the development by allowing disparate teams of developers from building functionality without the need to depend on the state of another

module. As such, debugging is simplified because errors are isolated to a module and side-effect type bugs as a result of interaction with other modules can be eliminated immediately.

4. **Software Agents** – as a guiding principle, the concept of the software agent and agent environment were used to design the simulator. As such, each object involved in the simulation was developed as an autonomous object that was both goal and utility based. These objects interacted with each other within a simulated trading environment which was continuous and sequential. Appendix D, a summary of the work of Norvig [RN03], provides detail on the strict definitions of agent design within Artificial Intelligence. Whilst the scope of this research does not include such development, the overarching concept of software agents has been utilised in the design.

5.6 Simulation Process

The simulation process, Step 2 of section 5.3, is composed of several process blocks, some of which reside at the database layer and others at the software development layer. The principal components of the simulation process are illustrated in the process view below.

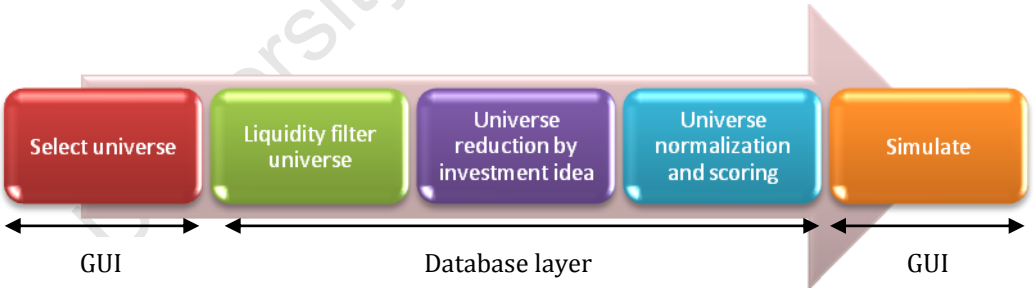


Fig 5-3: High-level process flow

The simplest explanation of the process begins with the simulation engine (SIMULATE block). In its simplest form, it may be considered as an I/O (input/output) black box. The input to the engine is a selection of stocks, the

resultant universe from the database layer; the output is the performance of that portfolio of stocks over a predefined time period. Whilst the output is a direct function of the logic of the engine, the input is a function of the pre-processing phase of the simulation.

The pre-processing phase for input into the simulation engine comprises the first four blocks in the illustration above. An investment decision begins with determining the investment universe, the pool of relevant stocks from which to construct a portfolio. The investor has the choice of picking one or many specific stocks, or selecting a particular group of stocks based on a specialisation; for example, the investor could either want to specifically buy AT&T shares, or all shares in the telecommunications sector. Whilst these are trivial examples, the universe selection process may become increasingly complex; for example, the selection of all gold stocks that have a market capitalisation > $2million and have traded above their average volume for the past 6 months. The output from this process block is a universe that is significantly reduced in comparison to its input. Whilst this process refines the universe stock count, not all of the stocks in the output universe may be applicable for the particular investment idea being evaluated. The next three blocks will further refine this universe for input into the simulation engine. A primary challenge to achieve a successful implementation of this process is the development of a framework that is flexible enough to categorise the stocks across several dimensions; for example, by sector, by country or by a specific fundamental factor. The secondary challenge is to ensure that the interface to the next process block accommodates the complexity of this variable output.

The next consideration focuses on the liquidity of the stocks being sought. Once a reduced universe has been generated from the preceding process block, one must then query this basket of stocks for their liquidity at the historical point in time. As explained in chapter two, liquidity may be crudely defined as the availability of stock on the exchange for both buyers and sellers. In terms of determining a stock's liquidity, once may calculate the average volume of a stock over a variable historical period, for example, the average volume for the sixty days prior to the historical investment date in question. The first challenge encountered by this step pertains to the initial design

objective of query speed, and hence the need for clustered indexing on the stock/date combination to improve the query time in reaching the appropriate database page for this historical date. The second challenge pertains to that mentioned in the previous paragraph, a flexible interface to accept the parameters that define a particular universe selection methodology.

In terms of the process blocks that function at the database layer, this process block serves the most important role. Once a general universe definition has been filtered to only those stocks that the simulator can legitimately purchase because of sufficient liquidity, it is time to apply the investment idea for this particular simulation to this reduced liquid universe. Assume that the investment idea being tested is the simulation of all stocks based on their Debt-to-Equity ratio with a minimum ratio of 1. This represents the major challenge presented by this block; the development of a flexible framework which can implement virtually any type of investment model to a particular portfolio of stocks. In this example, the input universe of liquid stocks is then queried to establish the Debt-to-Equity ratio for each of the stocks in the universe at that point in time, and all stocks whose ratio is below 1 are subsequently removed. Whilst this example rates stocks on the basis of a single investment criterion, the architecture had to be flexible enough to cater for complex rating algorithms.

Continuing the scenario from the above paragraph, suppose that only three stocks remained; BP, BHP Billiton and Bambi's Toys with ratios of 1.8, 15.9 and 35 respectively. In order to determine which of these stocks to *short* and which to go *long* on, one must rank these stocks. The problem however, is that the ratios in this example, are not representative of the company size and therefore cannot be compared directly. The problem is exacerbated should one wish to use multiple investment criteria of varying magnitudes. This process block serves the purpose of normalising these numbers using the z-score statistical method, thereby allowing for the numbers to be compared, scored and ranked. Whilst the exercise may appear trivial for a single criterion, it is critical when evaluating multiple criteria per stock, and creating an aggregate score. The aggregate score can only be comparable between stocks if it is

constituted as the sum of normalised numbers. The outcome of this process block is a filtered universe of liquid stocks which have been ranked in order of preference, as a consequence of the investment criteria to which they have been subjected.

Following normalisation and ranking, the portfolio is then constructed for the input interface of the simulation engine or I/O black box as alluded to at the beginning of this section.

5.7 Summary

This chapter served to introduce the high-level overview of the system as well as highlight some of the design objectives and considerations to be met by the software architecture. Separate chapters are dedicated to each of the building blocks described in section 5.6 to detail the complexities and solutions to achieve the architectural objectives.

Chapter 6

Database Construction

"Make everything as simple as possible, but not simpler. "
Albert Einstein

6.1 Introduction

This chapter details step one as introduced in section 5.3 which referred to the construction of a database of historical stock data against which the simulator would be executed. The construction of the database required the following sequential steps:

1. the collection and quality assessment of historic vendor data,
2. the selection of a database engine and the creation of a database schema,
3. the importing of the vendor data into the database and,
4. the subsequent cleaning of the data to ensure consistency and accuracy

Architectural design decisions had to be considered for each of these steps as each step contained challenges which voided one or more of the design objectives outlined in chapter five. This chapter shall discuss the challenges, compromises and subsequent decisions taken to successfully complete these steps and generate a consistent and accurate database for simulation purposes.

6.2 Collection and Quality-Assurance of Vendor Data

In order to simulate a global portfolio consisting of stocks from a variety of exchanges across multiple geographies, the underlying historic stock market database must incorporate a comprehensive replica of all relevant data for every stock for every exchange defined by the investment universe. Several challenges arose in the attempt to achieve this objective.

The database to be constructed had to house three types of data:

1. **Exchange data** that was generated from the daily trading of the stock on the stock exchange. These included data such as stock price or number of shares traded for the day.

2. **Audited financial statement data**, which is reported quarterly, semi-annually and annually directly from the company, through the release of financial statements. An example of a financial statement data item would be Revenue for the period (Earnings).

3. **Derived data** from the combination of data from (1) and (2) above; an example of which is the Price/Earnings ratio which takes the daily stock price (1) and divides it by the Earnings for that period (2)

The collection of both (1) and (2) above are complex tasks. The options available for the creation of the database were either to collect and compile the data for (1) and (2) directly from their sources or to purchase the data from a data vendor. This decision had to be made on the basis of price/effort feasibility as explained below.

Stock market or exchange data, indicated by (1) above, is the result of trading on the stock exchange. As such, the most appropriate source of historic stock market data would therefore be from the stock exchanges directly. The intended audience of this research, the financial services investment company would deem this unfeasible for several reasons:

1. The majority of stock exchanges were not computerised until the late 1980s and therefore would not be able to provide electronic data

2. Each stock exchange imposed a cost for the provision of data

3. The number of stock exchanges globally combined with geographic dispersion and language barriers increased the difficulty in sourcing the data

4. The technical detail, data warehousing, IT and data validation of such quantities of constantly evolving data would prove to be non-core to a financial services business and therefore more economical to outsource

Unlike exchange data, which is centralised for many stocks by geography and instrument type, Financial Statement data, as indicated by (2), is more disparate. This type of data is usually issued directly by the company. Financial statements are often highly detailed accounting documents containing several hundred line items within their Income Statement and Balance Sheets that need to be data captured and collated. The major challenges in this exercise that made it unfeasible were:

1. Quantity of data collection and capture; in the case of this project, twenty years of annual financial statements for six thousand stocks would result in one hundred and twenty thousand financial statements that would need to be gathered and captured electronically

2. Sourcing the data would be challenging because not all six thousand companies would still be in existence; and for those that were, each company in turn would need to be individually contacted to collect its financial statements. This is a consequence of there being no single source of this data that is freely accessible to the public

3. Reporting standards over the years have differed as new accounting standards have been introduced and evolved; an example being the Sarbanes-Oxley Act of 2002 [SO02] which revolutionised the financial reporting methodology. One would need the appropriate knowledge to assess the financial statements which differ across domiciles.

4. Companies represent their income statement and balance sheet items differently from each other; this makes it difficult to compare companies directly from their financial statements. An item such as Cash On Hand may be used as an example; Company A may have defined this to be only Cash in the bank ($5,000,000), whilst another company may define this as Cash in the bank ($2,000,000) plus any cash owing to the company by debtors that could be collected within twenty four hours ($5,000,000). The second company, with a total of $7,000,000 cash on hand appears to be more liquid than the first on this metric, however there is no guarantee that the debtors will actually pay on time, hence the reflection is skewed. As a solution, one needs to determine a generic framework for representing financial statements with very narrow definitions for each line item, and then re-interpret each financial statement for each company to fit this template; the result being standardised financial statements that are comparable by line item. This is a very time-consuming exercise as one needs accounting-specific knowledge to thoroughly examine and understand each business in order to understand how to transform the reported numbers.

In summary of section 4.6.10, there exist a limited number of data vendors that provide comprehensive historic global stock market/financial statement data. Examples of such vendors include Bloomberg, Reuters, Thomson DataStream and Quantitative Analytics, amongst others. The benefit of utilising such data vendors is that they have consolidated the data from the various exchanges, incorporated missing data for years where computerised data was not available and reduced the royalties payable to each individual exchange through economies of scale. These companies have teams of resources dedicated to cleaning data as well as teams of auditors to re-interpret and standardise financial statements. Being

a data vendor rather than an investment firm, they are best served by implementing the complex IT infrastructure required to collate, manage and disseminate this data.

For the investment services company which requires the data for financial analysis and simulation, these steps to collect both exchange and financial statement data are non-core to their business and the data vendor serves as a central data collection point; divorcing the finance company from the technical and logistical issues experienced in assimilating and validating the data as well as the ongoing maintenance of the data.

In summary, the effort required in collecting data directly from the various sources in order to construct the historical database outweighed the price to purchase the data and hence the decision was made to sample several vendor databases with the objective of purchasing data from one of them. Data from Bloomberg, Thomson DataStream, Quantitative Analytics and Reuters were used in this project to evaluate their effectiveness as a data vendor of historical data for quantitative backtesting purposes. In selecting a data vendor for this project several considerations had to be satisfied by the vendor:

Criteria	Bloomberg	Reuters	DataStream	QA
Cost of Bulk Data service	Exceeds budget	Within budget	Within budget	Exceeds budget
Cost of continuous data updates	Exceeds budget	within budget	Within budget	Free
Access to technical help regarding data issues	Limited	Limited	Direct contact	Direct Contact
Ease in retrieving bulk and daily updates	Cumbersome	Cumbersome	Simple	Simple
method of providing data	FTP/DVD	FTP	FTP	Custom Utility
Format of data	CSV	CSV	CSV	SQL DB
Programmability for bulk services	Provides API	Provides API	None	None
Quantity of data downloadable per day	Limited - Quota	Limited - Quota	Unlimited	Unlimited
Cost of exceeding daily download limit	Very expensive	Very expensive	Free	Free
Breadth of data universe	Excellent	Excellent	Excellent	Excellent

Table 6-1: Criteria for selecting data vendor

DataStream was selected as the vendor due to both its cost feasibility, direct access to technical help and most importantly its unlimited bulk data provision. Note that this offering from DataStream was compelling because it was also a promotional offer within the South African market in order to create a presence. As such, they offered a 70% discount on the price of the service whereas the other providers charged for their services at standard rates.

If one adopts the view that the simulator is a shell within which one can simulate an investment model, then the importance of the development shifts from the software agent logic to the underlying data. As explained in chapter five, the simulation engine may be described as an input/output block with the output being a function of the input. Therefore, regardless of the accuracy of the simulator logic, or the realism achieved by the simulator, if the underlying data (input) is dubious, the same can be said of the resulting output. The reader is referred back to section 4.6.10 which introduced the various reasons why vendor data may be inaccurate or inconsistent. The challenge faced by these data vendors remains the quantity of data and the practicality of verifying the data's consistency and accuracy. Whilst most data vendors have developed proprietary data validation tools to eradicate the simpler problems as detailed in chapter four, guaranteed data consistency and accuracy is not a proviso of the service.

Having acknowledged that inaccuracies and inconsistencies exist in the data collected from the vendor, it was the objective of this phase of the research and simulation process to eliminate them through the use of proprietary data validation logic applied at the database layer. Whilst this may appear to be a redundant process since the vendor had done the same exercise, the sample data from the vendors all contained errors, making the exercise necessary. It is important to understand that whilst the onus is on the vendor to ensure that obvious errors such as missing stock prices are corrected, most errors cannot be fixed by the vendor because it would require a methodology to be defined for the correction which may not suit every client. This is a result of the numerous statistical techniques available to fix inconsistent/inaccurate data problems. For example, in the case of missing data, one client may wish to Windsorize [Wil99] the dataset whilst another may wish to apply a Bayesian

Outlier Filter [KB06] to determine how to deal with missing data. As such, the vendor leaves the option available for the client to process and clean the data unless instructed otherwise.

6.3 Data Quantity

The volume of historic data to be imported into the database is a technical consideration that must be architected for. For this research, the scope of the universe of stocks was limited to global *Resources and Related Resource* equities with a market capitalisation greater than $250,000,000. The reason for this universe selection was based on the fact that it coincided with the industry sector being analysed by the employer of the researcher; the employer having tools and a knowledge base that would aid in the design and testing of this project, as discussed later in chapter ten.

In effect, the $250,000,000 limit refers to large capitalisation companies that are involved directly or indirectly in the resources sector. These may range from the companies involved in the mining of gold to those involved in the sale of jewellery to those that sell earth-moving equipment to aid the mining process. This universe filtering process reduced a potential global universe of thirty six thousand stocks within the Resources sector to a manageable six thousand stocks.

As discussed earlier, the historical database that must be constructed consists of a variety of investment factors for each stock, from exchange data, to financial statement data, to derived data. If one were to construct a database to house one hundred and fifty investment criteria for twenty years of daily financial data for each of these six thousand stocks, the following data storage is required:

Stocks	6,000.00
Investment Factors (fields)	150
Years	10
Days/year	252
Total number of data points	2,268,000,000.00
Avg Char/Data point	5
Total chars	11,340,000,000.00
Raw data (Gb)	88.59
Approx SQL Server compression ratio	90%
Database size (GB)	8.86
Row Count	15,120,000.00

Table 6-2: Calculation of Database Storage

The calculations of table 6-2 are based on the actual database used in this research. As a consequence, the SQL Server compression ratio has been derived as a result of knowing the final database file size. More importantly is the size of the raw data CSV files which had to be zip-compressed for FTP download. The eighty-eight gigabyte files were ultimately compressed to 1.5Gb which was a manageable once-off download.

This calculation highlights critical design considerations:

1. Bearing in mind the restricted universe being used in this research, the physical storage requirements from this calculation pose a challenge. One can assume that a real-world implementation of the simulator with a more comprehensive universe would be multiples of this calculation.

2. The calculation highlights the complexity in manually reviewing the two billion data points for inconsistencies and inaccuracies. Even if software were to scan for potential

irregularities in the data, a 1% failure rate is equivalent to twenty million errors, an equally complex task to remedy.

3. The database schema, indexing and table partitioning need to be cognisant of the software objective for millisecond query response time.

4. Hardware constraints would need to be architected for and all queries would need to be optimised to meet the constraints of simulation time on this hardware specification.

6.4 Database Engine and Schema Design

In architecting the database schema, several factors had to be considered. These included:

- the design objective of millisecond query response time,
- the quantity of data,
- the degree of normalisation,
- the partitioning of tables,
- indexation,
- data integrity and
- the type of database to be designed (relational versus object oriented)

6.4.1 Database Engine Selection

The first decision to be taken was the type of database management system to be used, the choices being between a Relational Database Management System (RDBMS) and an Object-Oriented Database Management System (OODBMS).

The OO model was considered due to:

- **Composite objects and relationships** - Due to its ability to store complex objects when compared to relational databases, the OO model allows for the storage of an

arbitrary number of atomic types as well as other objects; an object can hold multiple smaller objects, which each in turn could hold multiple smaller objects, ad infinitum. The equivalent representation of such a model in the relational world would require numerous, normalised tables which are all linked via keys. The major problem with this approach is that the more granular the data, the more tables that need to be joined, resulting in less query execution efficiency as compared to that of the OO model.

- **Query language** – Unlike the RDBMS, the need for a specific query language is no longer necessary as the data is queried when being accessed from the programming language. The current relational model requires that the developer either be fluent in both SQL/PSQL and their programming language, or rely on a Database Administrator who can perform the SQL specific tasks.

- **"Impedance Mismatch"** – when using an OO programming language with a RDBMS, three problems often occur; the first is the requirement to map tables between the database system and the language in an attempt to match the objects; the second being the treatment of data types that do not map natively; the third being synchronicity of the schema to the software mapping in the OO language

- **No primary keys** – Unlike traditional RDBMS's which rely on the primary-foreign key constraints to ensure data integrity, the use of Object ID's which is invisible to the user means that there is no limitation on what can be stored in an object.

- **One data model** – As discussed in the coursework material provided for this degree [UCT03], "*Capturing Semantics*", the modelling of entities and relationships between the OO application language and a RDBMS requires an Entity Relationship model. This however cannot comprehensively model every operation and behaviour between these entities and detail how the data may be modified. The same is not

true for the OO model which allows the entire system to be comprehensively modelled in one UML diagram since it is simply an interaction of system objects.

However, as compelling as some of these reasons were, the OO model was less favourable in the following aspects:

- **Not mainstream** – Kriegel [KT08] cites the research report of the IDC which states that the RDBMS market at the end of 2006 constituted 84% of the DBMS market with an expected annual increase in market share of 10% the following year. This 84% comprises the five major database vendors, Oracle, IBM, Microsoft, Sybase and Teradata in order of market share. The remaining 16% of the DBMS market is not solely attributed to the OODBMS market either. This 16% is further decomposed to include smaller RDBMS vendors as well as OODBMS vendors. As such, the OODBMS, being a small segment of the market, has not reached critical mass in terms of third-party vendors developing the necessary infrastructure and tools for it.

- **Solution-specific** – As indicated in the coursework for this degree, "Problems with the OO Model" [UCT03a], one of the major disadvantages of the OO model is that the result is very solution specific with an insistence that the results of the design stage are fully descriptive without the need for changes; which is often not the case. As a result of such specificity, the range of queries of the data is further limited as a particular design may render a query either impossible or query-speed unfeasible because the design may not have been optimised for it.

- **Schema changes** – For a design that is in flux, with specifications in constant evolution, the modification of the schema must be efficiently implemented with minimal impact on the architecture and codebase. A schema change in the RDBMS is usually independent of the actual application, whereas in the OO model, a change to a persistent class usually means that other classes will also require modification and a system-wide compilation is required. Further, all instance objects within the

database and their inner-objects need to be updated and for large datasets, this could be very time consuming.

- **Language dependence** – Unlike a RDBMS, an OODBMS is usually inextricably tied to an application language due to its reliance on the language API. This makes the architecture language dependent, making the system less extensible should new languages be created. It also means that the data within the database is only accessible via this particular language through this API.

- **Lack of query language** – Whilst this was also highlighted as a possible feature of the OO model, in designs that are still evolving, the flexibility to be able to query the data for different requests and differing solutions is imperative. The lack of this flexibility within the OO model is a serious design consideration. Rudimentary RDBMS concepts such as the creation of new datasets through the joining of existing tables cannot yet be replicated through the "joining" of classes within the OO model.

- **Not justified for simple data types** – As explained in the coursework for this degree [UCT03a], the OO model may be justified if the design is well specified, but more importantly, if the object description is complex. Where the object description is fully described through simple native data types, there are no compelling advantages to the OO model.

As a consequence, the RDBMS was selected for:

- Existing skill set and familiarity

- Advanced, optimised query engines

- Advanced tools for querying, profiling, optimising and managing the database

- Comprehensive list of native data types that could describe the stock data

- Flexible enough to accommodate an evolving feature set and its direct requirement for schema modifications and query flexibility

- Mature query language

- Extensive knowledge base in terms of online and print literature

- Extensive high-level programming language support through either native API's or ODBC connectivity

Having decided on implementing a relational database, a database engine had to be selected for the project. The options considered and the reasons for selection/rejection were:

- The engine had to operate within the Windows environment due to the researcher's limited operational experience using alternate operating systems

- SQL Server 2000/2005 – **selected** because of availability of licences, in-house expertise, proven ability to handle large datasets without compromising query time, 64-bit compliant, existing codebase, mature toolset and query language

- Oracle 9i – **rejected** due to the cost of licensing the solution and in-house inexperience

- IBM DB2 – **rejected** due to the cost of licensing the solution and in-house inexperience

- MySQL – **rejected** due to not supporting triggers, stored procedures and user defined functions; also rejected following a trial performance test on the large

dataset which proved significantly slower than SQL server on the same query against the same dataset; lack of advanced management tools for query analysing and profiling

- SQL Server Express – **rejected** due to its inability to handle large datasets and its reduced query engine capability

- MS Access – **rejected** due to its inability to handle large datasets and its reduced query engine capability

6.4.2 Data Quantity Considerations

Data Quantity was addressed at two levels; via the hardware specification and the schema design. With the reduced costs of storage, physical data storage has become less of an issue in recent times. At the time of writing, one terabyte of storage on a SATA-300 hard drive could be purchased for USD $296. Whilst this made the issue of data storage less material, optimising the data structure on the physical storage medium would still yield benefits. The coursework for this degree introduced records and record types, and with that, the associated benefits of fixed-length versus variable length record data types [UCT03b]. In the design of the schema, the choice of data types were forced to meet the minimum requirements of the data; for example, if the maximum string length for a stock ticker was ten characters, the data type was forced to this constraint, rather than the default fifty characters. Whilst this reduces the flexibility of the schema, it could be easily amended in the future, should the specifications change. The primary benefit however is realised in the reduced footprint of the database coupled with increased speed of query execution.

The final selection of data types was determined by first using generic data types for each field and then analysing the data in each field independently to determine what the minimum requirements would be to satisfy the storage of that field to which the schema was

consequently amended. In addition, the storage of the value zero as a null in the database also reduced the storage footprint.

6.4.3 Schema Normalisation

Unless the table schema of the stored data was optimal, an increase in data quantity would result in a reduction in query response time, one of the design objectives that had to be minimised. To address the structuring of the data and thereby reduce data redundancy, the schema had to be normalised. Relational modelling offers several degrees of normalisation, with each degree offering a more concise representation of the data, removing redundancy and reducing the potential for data integrity issues. A welcome side-effect of this normalisation process is the logical grouping of related data within each table.

Appendix C provides a sequential explanation of stages through which the database schema was modified in order to reach Boyce-Codd Normal Form and includes an illustrated view of the final normalised schema.

6.4.4 Query Response Time

Query response time was also addressed at both the hardware level and the database structure level through the selection of appropriate hardware and the use of indexation and table partitioning. From a hardware perspective, the topics discussed in the coursework for this degree of Unit 6 are of relevance [UCT03c]. The importance of disk access time, data transfer rate and double-buffering are emphasised, all of which improve the query response time. As such, the hardware selected, to maximise these recommendations within the limited budget were:

- 10,000rpm hard disk with a data access time of 4ms and data transfer rate of 120MB/s was the fastest hard drive within budget

- 8 gigabytes of DDR2 RAM – at the time of writing, DDR2 RAM was the de facto standard for standard desktop machines

- 64-bit processor and 64-bit Windows OS to be able to utilise this amount of RAM

Apart from the hardware, several database level decisions were also taken to increase query response time. Indexing of both the primary metadata table and its corresponding data tables along the primary key/foreign key constraint significantly improved the query response time. Additional indices were also created for frequently used fields such as Stock Price and Stock Volume. To facilitate indexation of the data table, a composite primary key was used, as each row of the data table was unique to a specific stock on a specific date. As the amount of daily stock data that was stored increased in terms of new factors, the database structure had to be revised. Indexing every column in the data tables was not a sustainable solution for two reasons, one being that the database physical storage requirements grew exponentially, and two being the indexation time which, in test trials took approximately two days to complete. As a result, Table Partitioning was the next concept investigated and later implemented. As per the coursework for this degree [UCT03e], there are two methods of table partitioning, horizontal and vertical partitioning. By virtue of partitioning, the destination tables will always be smaller than the source table, and as such, the benefit of partitioning is derived from the decreased query time due to reduced table scanning. The design of the simulator required that the data to be extracted for a simulation encompass all rows for all relevant stocks, but only specific columns relevant for the test. As such, it made more sense to vertically partition the table, leaving the frequently-accessed fields for any simulation in one table and segmenting the other fields into a second table with the link between the two tables remaining the composite key. This partitioning had a direct consequence by significantly improving indexation and subsequently reducing query response time. Cached execution plans, a feature of Microsoft SQL Server were also used in reducing query response time. In essence, the execution plan offers the developer the ability to estimate the bottlenecks in the SQL queries/stored

procedures. It also allowed SQL Server to cache the optimal path for a query and reuse the cached plan rather than recompiling a path for each query request.

6.5 Parsing Vendor Data

The simulation engine was architected against the database schema as introduced in section 6.4. This schema was customised for this application and differed from the database schema of the data as provided by the data vendor. The vendor schema had different data layouts and naming conventions for their data which had to be mapped to the custom database schema designed for this application. As such, and bearing in mind the quantity of data, it was necessary to develop software to parse the vendor data, transform the data into a format that could be imported into this schema and perform the necessary pre and post data validation checks. Pre-validation, for the purposes of this research, refers to the validation of data during the data transformation process before being inserted into the database whilst post-validation refers to the data validation process once the data had been successfully inserted into the database.

Importing the volume of data alluded to in section 6.3 of this chapter proved to be a significant challenge. As a result of the schema differences, the data had to be reformatted. However, whilst it would have been preferable to be able to open and modify the data and column order in a tool such as Excel, most tools could not accommodate the quantity of data. For this research, Thomson Financials DataStream database was provided in CSV format for the 6000 stocks described in section 6.3. Both Reuters and Bloomberg confirmed that, if requested, they would also provide the data in CSV format. DataStream segmented the data into ten CSV files each approximately 100 megabytes in size yielding a total text file size of one gigabyte. This file size proved unmanageable for most text editors to open for viewing, including Excel 2007, Word 2007 or Notepad. Whilst Visual Studio 2005's editor was able to open the file for viewing, the quantity of data in raw text format made the exercise futile. As explained, a utility had to be developed to parse the CSV files, first to check for inconsistencies and secondly to reformat the file for importing.

6.5.1 Vendor-provided CSV file format – Issues and Design Decisions

Figure 6-1 is an example of the original file format:

```
1   Header    Date,Anglo American(Price), Anglo American (Dividend Yield)
2   Header    Date, R:AGLJ(P), R:AGLJ(DY)
3   Data      1995/01/01,23.23,10,
4   Data      1995/01/02,23.22,10.2
5   Data      ...
6   Data      ...
7   Data      ...
8   Header    Date,SNOM(Volume), SNOM(Price-To-Earnings) ), SNOM(Price), SNOM,(AvgMV)
9   Header    Date, R:SNOM (VOL), R:SNOM (PE), R:SNOM (P), R:SNOM (MV(AVG,$P))
10  Data      1995/01/01,32,7,12,23000
11  Data      1995/01/02,39,7.1,12.32,23121
12  Data      1995/01/03,322,7.1,12.34,23154
```

Fig 6-1: Sample of original vendor CSV file

The file format is explained as follows: Each stock is grouped by having two Header rows (line 1 and line 2) and multiple Data rows (line 3-7) before the next stock's Header rows begins (line 8 and line 9). The first of the two Header rows (line 1) indicates the full name of the stock and the various columns of data provided for that stock, implicitly implying the order of the columns for that stock. In figure 6-1, line 1 indicates that the Data rows 3-7 refer to Anglo American with the first piece of data following the date being Price and then Dividend Yield for this particular stock. The second Header row (line 2) provides the unique vendor-assigned stock symbol for the stock as well as the abbreviation for the data factor being provided. In this example, line 2 indicates that the unique code for Anglo American is R:AGLJ and price and dividend yield are abbreviated to P and DY respectively.

Figure 6-1 illustrates some of the errors detected during the pre-validation parsing process:

1. The column headings in line 1 and line 8 are inconsistent. Line 8 contains more columns of data than line 1 (Volume, Price-to-Earnings and AvgMV being the additional columns). The Header on line 1 also has data for Dividend Yield which is not in the Header on line 8

2. The order of columns between the two segments is different. Line 2 shows column order as P / DY, whilst line 9 shows column order as VOL / PE / P / MV(AVG,$P)

3. The spacing following commas are not consistent

4. The data rows do not include a stock identifier symbol. In order to determine which stock the data row refers to, the utility needs to extract the name from the header row

5. Line 9 has a legitimate comma within the data type definition `(MV(AVG,$P))`, which, in CSV terms, may be regarded as two separate elements

6. The second data point in lines 10-12 provides the VOL for the stock. The value for VOL in line 12 is inconsistent with those in lines 10 and 11; 322 compared to 32 and 39 respectively. There is possibly a data capturing error.

7. Each row in a database table would have to have identical schema, therefore the header information would have to be incorporated into the data rows, making it impossible to import the vendor file in its original form

These are a subset of the errors that were extracted from the CSV files and logic for the correction of each error had to be built into the parser. One important feature of the parser was the ability to supply it with the expected column order of the reformatted output CSV file. The importance of this feature lies in the fact that the CSV file can only be imported if the column ordering matches that of the database table into which it is being imported. For each

stock header in the original CSV file, it would rearrange columns to achieve the expected column output.

One issue that did arise was that of missing columns. In the example, the first header contains two columns (P and DY) whilst the second header contains four columns (VOL, PE, P and MV). In order to solve this problem, the parser made two passes through the file. The first pass generated a distinct collection of all the column names within the file. In this example, it would have collected five columns, even though there are six, since one of the six was repeated (P). With this unique collection, the second parsing of the file would read in each header row, determine how to shift the columns around and pad missing columns with blank values (,,). The result is a CSV file that has been formatted to the same column order as the destination SQL table.

In developing the parser for this phase of the research, several choices were available. The parser could have been built using SQL stored procedures; however stored procedures are best served once the data resides within the database. Whilst a stored procedure could in essence read in the data from a CSV file, implementing some of the logic required for the manipulation of the file format and resaving the file back to CSV were outside the functionality of stored procedures. The decision to develop the parser as a windows-based application allowed the logic for both parsing, reformatting and data validation to be housed in a single application. This allowed easier debugging without losing any performance. Performance was maintained through the use of SQL-CLR, the functionality in SQL Server 2005 which allows the developer to develop stored procedures using the C# language and debugger.

6.5.2 Importing Data into the database

There are essentially two methods of importing data into the database, via insert statements or via native bulk insert capabilities inherent in most commercial RDBMSs. The latter option was selected because the former option proved sub-optimal for the importing of a large dataset. This will be explained later in this chapter. In order to import data into SQL Server

using its built-in bulk insert task, the data must be formatted to match the table schema into which it is being imported.

For the DAILYDATA table as illustrated in figure 6-2, the basic table schema may be described as:

uStockCode	DailyDate	CLO_Dom	Vol	MV_Dom	PE	DY

Fig 6-2: Column order of the DailyData table

Figure 6-2 represents the column ordering of the DailyData table that is input to the parser. The columns of the CSV file will be reordered to match the column order of figure 6-2.

If one were to reformat the example CSV file of figure 6-1, the expected output in tabular format would be:

SQL Table	***uStockCode***	***DailyDate***	***CLO_Dom***	***Vol***	***MV_Dom***	***PE***	***DY***
CSV file	Stock	Date	P	VOL	MV(AVG,$P)	PE	DY
	R:AGLJ	1995/01/01	23.23				10
	R:AGLJ	1995/01/02	23.22				10.2
	.						
	.						
	.						
	R:SNOJ	1995/01/01	12	32	23000	7	
	R:SNOJ	1995/01/02	12.32	39	23121	7.1	
	R:SNOJ	1995/01/03	12.34	32	23154	7.1	

Fig 6-3: Expected tabular output for a properly formatted file

The expected format of the CSV file for the above example was:

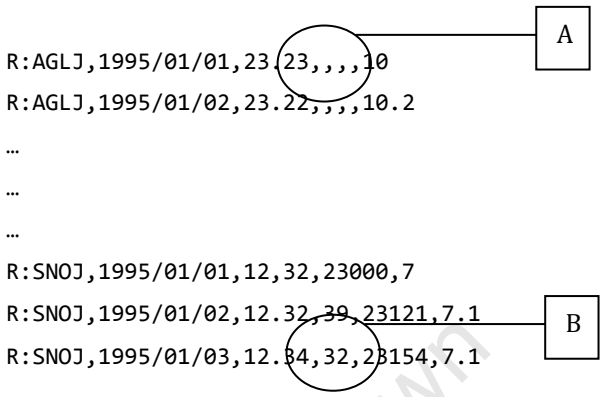


Fig 6-4: Expected CSV output for a properly formatted file

In the expected format of the CSV file, the file headers for each stock were removed and inserted into the data rows themselves. The columns were reordered to be consistent. Data errors such as the volume error were corrected (B). Where no data was provided or columns were omitted, the data was padded with blank values (A). This provided properly-formatted CSV files which could then be bulk inserted directly into the database using SQL Server Integration Services.

6.5.3 Testing the Parser

Before utilising the parser to generate the final database import files, the parser had to be tested. An incremental approach was used in the testing of the parser as described below.

The following methodology was implemented to confirm the parser's logic:

1. For each type of error that the parser was meant to correct, a separate test input file was created with that particular error in it and the output file was checked to determine if the error was identified and corrected

2. Once each individual correction was verified, multiple input files were generated with multiple permutations of the errors tested for in (1)

3. For each output file, the errors that were meant to corrected were verified

4. Each test output file of (3), a sample subset of data, was then imported into the database to ensure that even though the parser had fixed the errors and corrected the column ordering, the database would accept the re-formatted file. This was a second validation of the parser since a successful import via the RDBMS data validation meant that the data met the schema and data type specifications.

Once all the data had been inserted into the database, the post-validation checks were run and this provided further confirmation as to the success of the parser.

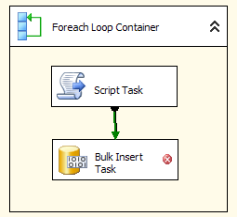
Each of these steps was not concluded without error on the first attempt, and several iterations of each had to be performed before sample data files were successfully inserted into the database. Reasons for these iterations ranged from the parser not implementing the corrections or not being able to handle multiple errors on a single line to simple formatting mistakes such as a missing comma, which in a CSV file, is essential to the demarcation of fields. A successful step 4 above, with sample data being imported into the database, did not positively rid the parser of all errors. Upon post-validation of the data, several more errors were identified, such as incorrect date formatting, and again the parser was duly modified and the entire exercise repeated until all inserted data successfully passed post-validation.

6.6 Database Bulk Insert

As explained earlier, there were two options for importing data into the database; the first was via Insert statements; the second via the bulk insert task offered by SQL Server Integration Services (SSIS), a module of Microsoft SQL Server designed to optimise such tasks. The first option posed three problems, one being that single insert statements had be generated from the formatted CSV file, one for every data row; secondly, the size of the

transaction log became an implementation consideration as it grew exponentially with each insert; and thirdly, the execution of individual insert statements was significantly slower than the built-in bulk insert tool. The bulk insert task is not disadvantaged by any of these issues and was developed specifically for this type of data handling.

SSIS in SQL Server 2005 introduced the ForEach Loop Container within which one can insert a workflow that can be repeated for several items in the container. In the case of this research, the ForEach Loop iterated through the ten parser reformatted CSV files in turn and bulk inserted each into the database. When one considers that the time to insert a single file is approximately thirty minutes, the total exercise takes 300 minutes/5 hours, and hence automation is preferred over manually importing each file. This is also a more scalable solution should the investment universe be expanded to more sectors, thereby generating even more CSV import files. With this approach, the ForEach module was pointed to a folder of CSV import files and could be left to run overnight without user intervention. The claim to scalability of this approach is based on the limit in the number of files that can be held in a folder within Windows. Microsoft documentation [MS01] indicates a *Files Per Volume* limit of 4,294,967,295 files which is well in excess of the expectations of the simulator requirement. Further, the ForEach Loop does not have a maximum file counter limit. Figure 6-5 is an illustration of the SSIS package required for the Bulk Insert task. The script task is simply required to aid the ForEach loop by pointing it to the next file in sequence.




```
Public Class ScriptMain
  Public Sub Main()
        Dim connmgr As ConnectionManager = Dts.Connections(CONNECTIONNAME)
        connmgr.ConnectionString=Dts.Variables("User::FileName").Value.ToString()
        connmgr.AcquireConnection(Nothing)
        Dts.TaskResult = Dts.Results.Success
  End Sub
End Class
```

Fig 6-5: SSIS package (top) and Script Task code (bottom)

Some of the issues that arose from this exercise were:

1. Bulk Insert halted with an error if it encountered a column value that did not match the destination column data type

2. If an error occurred, one cannot rollback the transaction for that particular bulk insert, thereby tainting the entire table and requiring a complete restart of the exercise

3. If an error occurred, it did not provide an indication of which row of the formatted CSV file was responsible for the error, making the debug process difficult

Several approaches were tested in an attempt to reduce the bulk insert time:

1. One solution was to import the entire CSV file into a destination table which had all fields set to varchar as the field data type. This ensured that data type mismatch errors were avoided during import. The problem that arose in this exercise was the conversion time to cast the columns to the correct data type after importing was multiples of the bulk insert time and failures that occurred still did not provide any indication of the exact row with the error.

2. Removing automatic indexing on the destination table reduced the bulk insert time by more than 60%, however the subsequent indexing time post-insertion also took multiples of the bulk insert time.

3. An attempt was also made to have the parser generate a file of insert statements rather a CSV output. This proved to be too slow and the size of the transaction log was unmanageable.

The most successful method was the use of the bulk insert tool directly into an index-enabled table with correct data types.

In order to aid the debugging process, the parser was modified to add an additional FILENO marker into each row of the CSV file as a unique identifier of a file. When an error occurred, rollback could be attained through simply deleting all inserted rows with the current FILENO at the time of error. Once all the data was successfully inserted into the database, the FILENO field was dropped.

6.7 Summary

The information provided in this chapter, whilst seemingly obvious in hindsight, took several months to distil from several failed attempts using alternate methods of importing and cleaning the data. The result of implementing the suggestions of this chapter was a partially clean database against which the simulator could be executed.

Chapter 7

Parameter Scripting and Configuration

"Only two things are infinite, the universe and human stupidity,
and I'm not sure about the former"
Albert Einstein

7.1 Introduction

Having constructed an input historical stock database (chapter six) that the simulator can be executed against, the next logical step would be the detailed explanation of the simulation process itself. However, the database data is not the only input for a successful simulation. If one adopts the view that the simulator is an environment engine, three critical considerations for a simulation are:

1. If a simulation is meant to operate on a select basket of stocks in the database, how does the engine get provided a definition for selecting this basket of stocks

2. If the environment is a function of the state of numerous variables, what are the initial values for each state variable that ultimately create a custom environment for this particular simulation

3. In what format/location are the input and output values of the simulator stored

In answer to these three questions, two input parameter definitions, a universe definition and a simulation parameters definition, were created. The universe definition indicates to the simulator, the pool of stocks from which it may construct a portfolio, for example, using only companies that mine Gold as opposed to all 6000 stocks in the database. The simulation parameters definition fully describes the simulation to be executed, from the investment methodology to the holding period of the simulation to the amount of money to be spent. It essentially provides the initialisation values for all environment variables within the simulator, without which the results of a simulation would be completely random and meaningless. This chapter deals with how each of these two inputs is generated for the simulator.

7.2 Files and Folders

Before discussing each of these input files in detail, the first decision that needed to be taken was how the input parameters and output results would be stored as this would determine how the input parameters would have to be generated – either created in a text editor and stored in files or created via an additional custom-designed utility and stored in a database.

One of the design objectives expressed in section 5.2 was that the software agent engine should be capable of executing from the command line without the need to invoke the user interface. Within the Microsoft Windows programming environment, the user interface component exerts additional overhead to an application through the additional use of memory and processor time for refreshing interface components. For example, in a single thread application, for a processor intensive operation, the user interface can "hang", and the operating system has to periodically redirect processor time to refreshing the interface. This

redirection of processing time reduces the overall efficiency of the simulator by reducing simulation time, which is a key design objective that had to be minimized. Early design specifications identified that the simulator would more often be executing batch simulations which did not require any user interface interaction, so a decision had to be made as to whether a user interface could be included without sacrificing performance, or whether it should be completely excluded.

Two options were considered in answering this question:

1. Build an engine **without** a user interface, essentially a console-driven application, and find a method of passing the input parameters to it

2. Build an application **with** a user interface, but have the engine operate in a separate thread/process, with its inputs passed in from the user interface layer

The final decision had to take into consideration flexibility and extensibility. As such, option one, unlike option two, was selected because it resulted in a fully abstracted engine. As a consequence, a user interface could be developed at a later stage, and also in any other programming language or medium; so a C#-based engine could have an ASP.NET-based user interface developed at a later stage without any impact to the engine. This achieved both flexibility and extensibility and was in contrast to the multi-threaded option. Extensibility, in this context, refers to the ability of the developer to utilise the API to develop a user interface in a language of his/her choice without having to take the simulator into consideration. In practice, user interfaces were developed for Windows Forms and Windows Mobile devices and could be extended for devices such as the Apple iPhone or RIM Blackberry. Whilst a multi-threaded application would allow the inclusion of an interface without necessarily sacrificing performance, it was decided against because of the increased complexity in development of multi-threaded applications and more so in the debugging of the engine. It also inextricably tied the engine and the user interface to the same programming language, restricting the flexibility of the design.

The outcome of this decision to select option one was a console-based engine which needed a method of parameter input. Two options were available in terms of passing data into the engine:

1. The engine could read the input parameters from a database

2. The engine could read the input parameters from parameter files on disk

The latter option was selected because it afforded the end user the ability to manually generate any number of input files and modify such files at will without any database knowledge. Being text files, it also avoided the need to develop and debug an additional user interface for the user to interact with a database, if that path had been chosen. Before taking this decision to use files to store input parameter data, sample code was developed that read input data from the database and wrote the results back to the database. This was then compared to the time it would take to read/write CSV files and then compared to the time to read/write Excel files. CSV read/write operations significantly outperformed the Excel or database operations and was therefore selected as it reduced simulation time. In an attempt to further reduce simulation time, where possible, data was batched in memory during simulation execution and written in bulk at the end of simulation cycles to minimise the number of hard drive access operations as each drive access forced the processor to direct resources away from the simulator.

As explained in earlier chapters, the simulation engine may be represented as an input/output block. The two files alluded to in 7.1 above, serve as inputs to the engine, whose data is read in at the start of a simulation and used to initialise the environment variables within the simulator. In so doing, the simulator runs a customised simulation based on these inputs. If the user wished to simulate an alternate scenario, he/she would either modify these two files and re-run the simulation engine or create an alternate set of files to which the engine would be pointed. The architectural benefits of this approach are:

1. It is in keeping with the design objective in chapter five of stateless operation. The engine can operate as a stand-alone application. It has no dependence on any other application or database for inputs.

2. The second benefit would be the ability to reuse the same files in multiple simulations without having to recreate the input parameters for the engine. This improves the extensibility and flexibility of the system. For example, the user may create ten simulation parameter files and only one universe definition file, and have the engine run the ten scenarios against this single universe. The converse is equally applicable. Further, in a real-world implementation, the user would generate several versions of each file and thereafter simulate all possible permutations.

3. A third benefit applies to the architecture of the simulator engine. The architectural approach is that of the plug-in pattern [MHH+08] which consists of two parts, the plug-in and the plug-in host (the engine). The input plug-in to the engine, in this particular case, requires these two input files. Assume the specification of the design were to change to require live data feed as input to the simulator. The developer would design an alternate input plug-in which incorporated the live feed as part of its input and translate this to a format that the engine expected. This would then be "*plugged-in*" to the input interface of the engine. This serves to increase the flexibility and extensibility of the engine because it does not require modification of the engine interface.

With this approach of storing the various discrete inputs and outputs of the system in separate files, the result was a plethora of files for which a methodology had to be established to ensure orderly control of the data. Since the data was all stored in files, the logical and simple approach was to arrange the files into various folders and order the folder hierarchy in a manner that would be easily understandable for the end user. This representation of the data is in keeping with the Windows metaphor of hierarchal data storage as seen in Windows Explorer, where folders represent collections of files of similar functionality/relationship.

In recommending a folder structure that would be intuitive to the user, the simulation process was first decomposed into its various functional roles, and these functional roles were then further decomposed to represent the various granular forms of data that was being stored.

The recommended folder structure for the software was as follows:

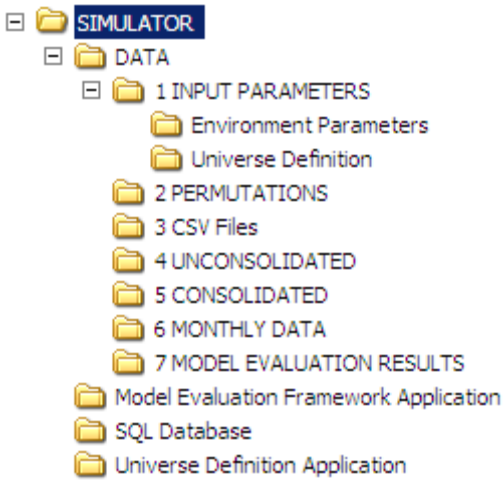


Fig 7-1: Proposed folder structure

Under the root folder, SIMULATOR, there exist four child folders, DATA, Model Evaluation Framework Application, SQL Database and Universe Definition Application. With the exception of the DATA folder, the remaining folders each house programs / files that support the simulator, whilst the DATA folder contains the simulator engine and all simulation input parameter and output result files.

Within the DATA folder, each subfolder is prefixed with a number to represent its stage of a simulation lifecycle. For example, the first stage of the lifecycle requires the two input parameter files introduced in 7.1 above. The universe definition file resides in the Universe Definition folder, whilst the simulation parameters file resides in the Environment Parameters folder. Each of these two folders may contain multiple versions of these files

with different data representing different simulation scenarios. These two input files are then passed through a permutation generator to generate various simulation permutations, which are stored in the PERMUTATIONS folder. The simulation engine selects the appropriate files from here when executing a simulation.

The results of a simulation are two files, a CSV file which provides the detailed daily performance of the portfolio, and an Excel spreadsheet which contains the relevant cumulative statistics for all iterations of that particular simulation from all the CSV files generated for that simulation. These two outputs reside in the CSV Files and UNCONSOLIDATED folders respectively. The rationale for the two file formats lies in the data that each contains. For example, suppose the user were to run a ten year simulation on a monthly basis, in effect 120 simulations. Each simulation represents one month and the daily portfolio performance is stored in a single CSV file. The end result of the process therefore yields 120 CSV files. However, in order to provide a high-level view of the performance of this simulation across the ten years, a single Excel file is associated to this simulation. Each column in the Excel file would represent one month whilst the various rows represent key criteria. Hence the output would contain 120 columns of data and a manual scan of this file can provide the user with an overview of the performance of this strategy over ten years.

A real-world implementation of the simulation engine would probably have the simulator executing several thousand ten year monthly tests. This would result in multiple Excel files, one for each ten year test, for example. The user may wish to gain a high-level view of the performance across the tests. As discussed later in the research, a tool was developed to parse the various Excel files in the UNCONSOLIDATED folder to generate a consolidated view of specific key statistics from each file of the unconsolidated files into a single Excel file. This single Excel file then resides in the Consolidated folder.

Whilst one could continue this explanation through the lifecycle of the simulation, the objective of this explanation was meant to illustrate the following architectural considerations/benefits:

1. The various folders house files generated at various stages of the lifecycle and hence need to be segregated appropriately. This logical separation, allows the user the ability to find the appropriate data files quickly. The hierarchical nature of the folder structure is in keeping with the Windows Explorer metaphor.

2. A significant advantage of this architecture is the modularity of functionality. For example, the tool that reads the unconsolidated Excel files to generate the consolidated Excel file need not have knowledge of the rest of the system. The tool to create the Universe Definition file (discussed later in the research) can operate in isolation without knowledge of the simulator, and the simulator can utilise the resulting file without knowledge of the creation tool. For as long as the design implemented flexible API interfaces, each of these tools can therefore be developed by independent development teams and tested without the incorporation of the simulation engine. Modularity also aids the debugging process.

3. In investigating a particular simulation, this design allows the user to view the simulations at either a high-level, via the consolidated Excel summary files; and then drill down to the most granular level to the actual raw data results of the simulator for a specific date of simulation with the CSV files. This is particularly useful in the debugging process for software development, and equally in understanding the results of the simulator in production. Further, it allows the user complete control of the results which would not be possible had the results been stored in the database. For such a case, the user would either have to be fluent in SQL to query the data themselves, or a utility would have to be built for this purpose, which could not be justified for this design.

The remaining folders in the folder structure are used in the post-processing of the results and each will be discussed in detail in the chapters to follow. In summary, the decision to create text files which an end user could easily read and modify combined with a lifecycle-ordered folder structure that the end user could navigate like any other Windows folder

allowed for the two input parameter files to be specified both in terms of content as well as method of generation. These will be discussed in the sections to follow.

7.3 Universe Definition Parameter File

Before considering the various options available for the generation of this file, it is important to reiterate the rationale for this file which is motivated by the design objective of millisecond response time; very simply, the smaller the universe being simulated, the faster the simulation. Hence the need to constrain the universe being simulated to the most relevant stocks through filter criteria.

This section details the rationale for, and design of the tools to, generate the input parameter file for the universe definition. The questions being asked at this stage of the design were:

1. how to develop a tool that afforded the end user maximum flexibility and granularity in dissecting the universe

2. how to develop an interface between this tool and the database layer which would be flexible enough to allow for highly complex definitions to evolve over time without constantly having to modify the interface between these two layers

A stock may be considered to be an entity with both static and dynamic properties. Static properties refer to items such as its name, the major and minor industry sector to which it belongs, the country in which it trades; essentially metadata for the stock. A stock also has daily, weekly, monthly and annual data associated with it, such as daily volume traded and annual earnings figures. The benefit of numerous properties, static and dynamic, is that the data may be spliced along various dimensions that represent the properties. The objective of such splicing is the filtering of the universe. For example, one could combine a static and a dynamic factor by filtering the universe for only stocks that are categorised in the vehicle sector (static) with a daily volume in excess of 2,000,000 shares (dynamic) being traded. One could further complicate the filtering by defining additional factors which do not exist within

the raw data, but is rather derived as a calculation from existing values; for example, one may wish to create a filter that selects all stocks with a positive ratio of earnings/sales, having only the separate earnings and sales figures in the raw data.
In order to arrive at a design solution, one first needs to understand the types of data that this tool will be handling, and as well as the malleability and relationship of one data item to another; for example, one cannot filter on certain data items without other data items being considered; and one cannot filter on certain data items because the resulting groups of stocks are non-sensical – example, filtering stocks based solely on share price.

The first logical level of filtering a stock universe would be to determine whether all stocks or a particular subsector of stocks in the universe are applicable to a simulation. Further filters can then be applied to this reduced subset of stocks. This is important where the processing time for additional filters is significantly reduced for a reduced number of stocks. In order to offer a user interface to accommodate this type of segmentation, one needs to understand how stocks are classified.

In 1999, the Standards and Poor's along with MSCI/Barra embarked on a project to reclassify all equities traded on all exchanges globally such that equities could then be comparable across sector, industry, country and region amongst others. The result was GICS, the Global Industry Classification Standard [SP99]. The standard provides a four-level hierarchy of classification, such that each classified stock, based on its primary business activity, is allocated **one of each** of the:

- 10 Sectors (Major Sector),
    - 24 Industry groups (Minor Sector),
        - 67 Industries (Sector) and
            - 147 Sub-industries (Sub-sector)

In selecting the stock universe, the user has the choice of selecting ALL stocks at one of these four levels of classification or by selecting specific stocks at one of these four levels; for

example, one of the 67 Industries listed above is Chemicals and the user may wish to select all Chemicals stocks or specific stocks within this sector.

In designing the tool to empower the user with such flexibility, the combination of hierarchial data and common Windows metaphors for such data representation, resulted in the user interface illustrated in figure 7-2 where the user is presented with a visual representation of the four-level GICS classification.

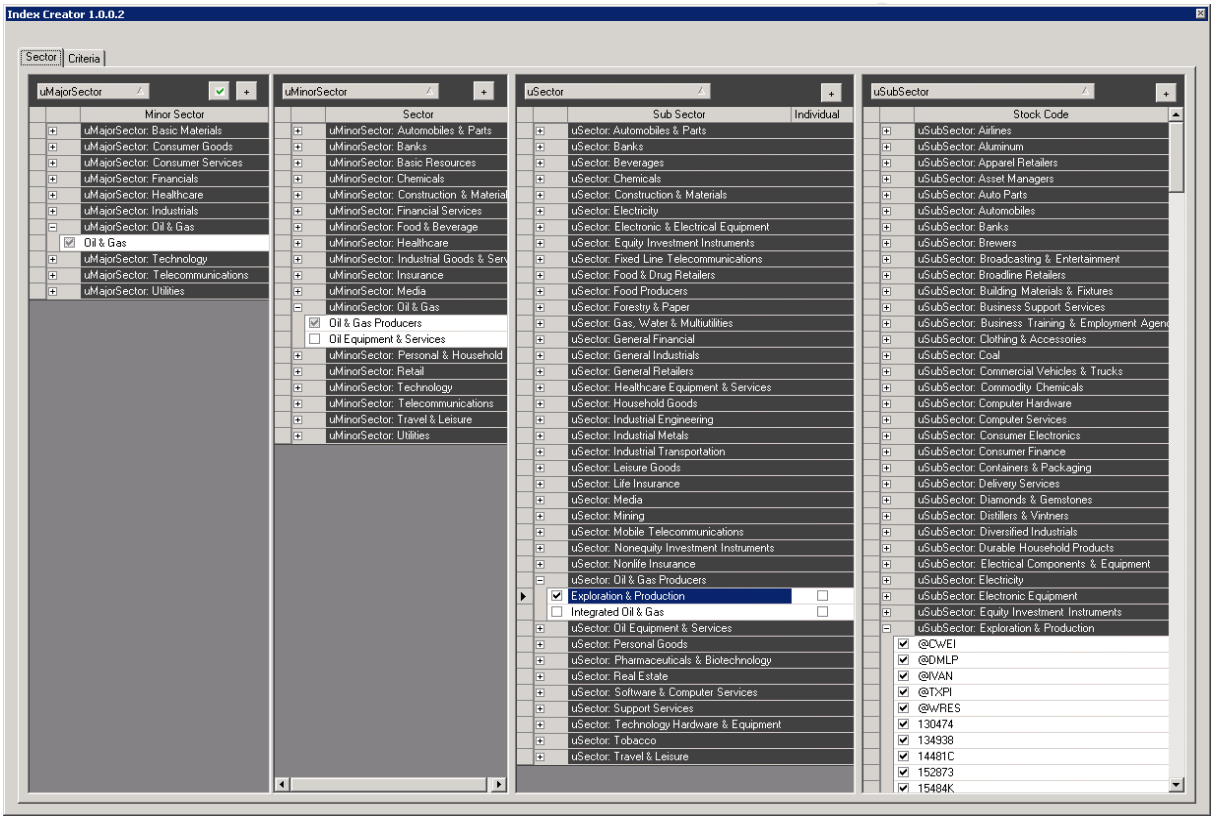


Fig 7-2: Granular selection of stocks by sector hierarchy

Hierarchical tree structures and cascading selection mechanisms are common user interface metaphors used in the Microsoft Windows operating system, the target platform for the simulator. Sector classification, by its very nature and implementation, is hierarchically

organised. The interface design illustrated above follows common practice by using both the tri-state checkbox and the tree structure. The user of Windows is familiar with expanding a tree and selecting a checkbox. By selecting a checkbox, cascading branches in trees further down the hierarchy are expanded and automatically selected. As used in most software installation programs, the user is also familiar with the tri-state checkbox which implies partial/full selection of criteria further down the hierarchy based on whether the checkbox background colour is solid white or partially grey. The final selection level indicates to the user all the stocks within the selected sub-sectors and offers the user the granularity to select/deselect individual stocks within the selected subsector. This method of hierarchical representation was selected predominantly because of its resemblance to other common application interfaces within the operating system as well as third-party software applications such as the Microsoft Office application suite.

As outlined earlier, in designing the solution, the data was analysed to determine how best to represent it, as well as how many degrees of freedom need to be afforded to the user in further filtering the universe. Having reduced the potential universe of stocks based on GICS sector classifications, the next step was to analyse the types of data that the user may use to further divide up this universe. An understanding of this data would enable the second phase of the tool to be specified. The results of the data evaluation indicated that the data may be classified into two groups:

1. Metadata versus actual stock data (for example: the currency the stock trades in which is metadata versus stock price which changes constantly)

2. Data that will be used based on specific values versus data that will be based upon a range equality (for example: a list of all stock exchanges in the United States which is a fixed list versus a list of all stocks whose market capitalisation is within or outside a particular range)

To test this classification system, each of the 150 fields that could potentially be used as filter criterion was tested against the criteria listed above. All the fields evaluated did in fact pass this test by fitting into one of the two classifications.

The next question was whether the user was able to differentiate between the four types of data that Points 1 and 2 above have alluded to. It became evident that point 1 which alluded to metadata versus stock data actually determines which table within the database gets queried for a list of possible options from which the user would select. The user should not have a need to differentiate the data in this manner in their filtering process, but rather be presented with a consolidated list of potential fields and the tool should be intelligent enough to determine which tables to query. If this were the case, then the focus moves to the second point. Here again, the question was asked as to whether the user should be able to differentiate between static lists of options or fields that allow them to specify ranges. Again, the answer was driven by the potential ability of the tool to be able to make this differentiation and present the user with an appropriate interface based on the type of field they selected to manipulate.

The result of these evaluations was an interface as illustrated in figure 7-3. The interface is divided into three columns with the first two columns having two rows. The first column differentiates between static and dynamic fields and the second column offers the appropriate type of interface for those types of fields.

As per the coursework for this degree [UCT03f], several techniques are available in gathering user feedback. For the purposes of this research, and in consideration of the small user base, the techniques of unstructured interviews and system usage monitoring were utilised. At the completion of each phase of the prototyping/feature addition, users were offered an opportunity to use the system and their usage patterns and feedback were recorded for triage purposes. At each session, users were informed of any new features added, as well as briefed on any modifications/improvements to issues that had been raised in previous sessions. Users were encouraged to re-test the improved features to confirm the

success or failure of the improvement. Two examples of user-driven improvements are discussed.

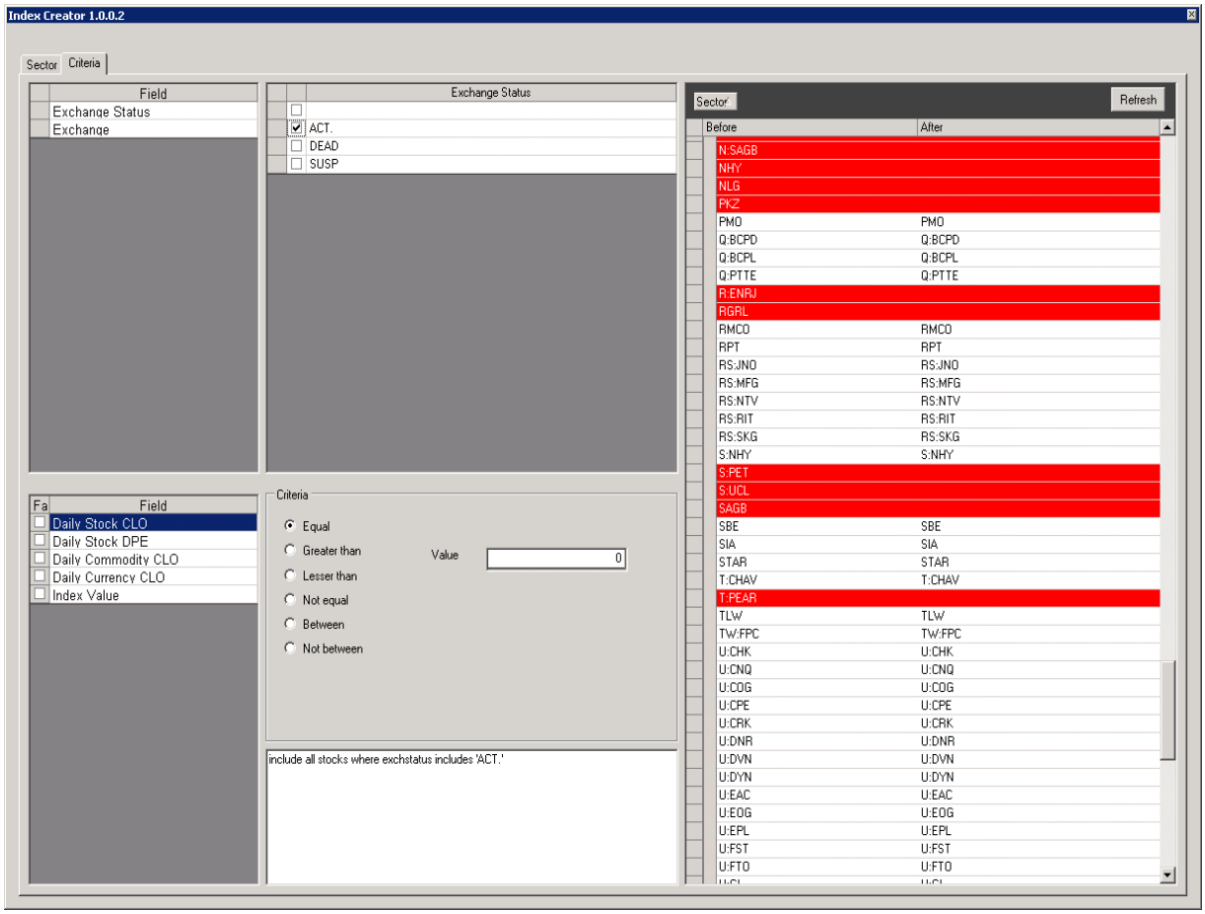


Fig 7-3: Selection of static (top) and dynamic (bottom) filter criteria

The third column in figure 7-3 was added in post-design specifications and was the result of this user feedback. Users of the initial version of the tool had to select their criteria and then

run a simulation through the simulation engine before being able to tell what universe was actually being produced by their selections. This was unfeasible on a time basis because an exercise that required constant tweaking of the universe definition parameters was substantially increased because of the additional simulation tests that had to be run for each tweak. Most of the users requested a faster alternative to this problem. The solution that was proposed and accepted by the users during feature testing, presents the user with a Before/After result snapshot of their filter selections. The Before column presents the user with the resulting universe of stocks post GICS sector filtering, but pre any other filters being applied; and the After column naturally indicates the stocks that pass the filter criteria. Those stocks that are eliminated through the filter selection are highlighted in red to the user.

In figure 7-3, as a result of selecting ACT (abbreviation for ACTively traded stocks) in the Exchange Status parameter (top middle), all delisted stocks and suspended stocks are removed from the list, for example S:PET, a delisted stock, has been highlighted. This gives the user an at-a-glance representation of the impact of his/her parameter values.

The second example of user feedback evolving the design was the inclusion of a textbox at the bottom of the second column. As a result of the potential number of dimensions that this tool allows the universe to be filtered across, many users lost track of exactly what filters they were applying. This feature provided the user with a non-technical explanation of how the universe was being filtered based on their multiple selections. An example of this was "include ALL stocks in the STEEL sector whose EXCHSTATUS=ACT and DAILY STOCK DPE between 1 AND 3 and (EXCHANGE=New York OR COUNTRY=United States)". Users requested this field as it consolidated their multiple selections and gave them an ability to understand how the 'and-ing' and 'or-ing' of criteria was being applied.

One of the questions that arose in the design of the tool and the assessment of design objectives was that of stateless design. Again, the tool had to be built to minimise dependencies. The question that arose was how the tool would receive its list of options that could be offered to the user. Two options were considered:

1. Stored the fields and associated table name in a file on disk and mark each field as either static or dynamic. Store the complete GICS sector breakdowns and the stocks which fall into each sector on file and load this on start-up as well.

2. Store these options in a table in the database and read it from there

Interestingly, after several iterations of the design, the final solution proved to be neither of these. Option 1 was flawed in that it had to remain synchronised to the database schema in the event that fields got added/removed. The potential for mismatch between the file schema description and the actual schema was not considered negligible and this idea was dismissed. Option 2 potentially suffered from the same problem of schema synchronicity as Option 1, except that SQL scripts and triggers on tables could be used to build a system whereby any schema change in a table would trigger the description table to amend itself. However, development and testing of this intricate solution and its complicated implementation proved unnecessary.

The solution that was arrived at was simply to create field names whereby the name itself had a marker which the tool could use to differentiate the types. This avoided both options 1 and options 2 and ensured that since the tool always read the in-production database schema on execution, it would get the latest fields in the database and apportion them appropriately based on the marker in the field name. The decision was taken that all fields in the appropriate tables that represented static data would have their field names prefixed by "ST" and all fields that represented dynamic fields which needed to offer ranges, were prefixed by "DYN". The tool thus had no dependency on metadata tables. Upon start-up, it would determine the GICS sectors by querying the database for distinct sector lists. It would then query the appropriate tables for all their field names and apportion them to the appropriate parts of the interface. For each field, it would retrieve the unique range of static/dynamic values that currently exist in the database and base its offering to the user on these results. For example, it would read in the STExchange field as being a static field. It would represent the field to the user as Exchange, dropping the ST, and then query the database for a unique list of all the Exchanges that could be chosen from and present this to

the user for selection. In summary, the addition or deletion of a database field would, *without code modification*, be incorporated into the user interface of the Universe Definition tool, and since this tool, *without code modification*, had the ability to generate a dynamic SQL statement to pass into the database layer, the end-to-end process passed the test of both extensibility and flexibility. This will be discussed in more detail later in this chapter.

Ultimately, the objective of this tool was the generation of a file which contained the various user defined parameter values which would be used to filter the universe. The tool merely provided a graphical representation for the user, in an attempt to ease the process of setting parameter values as well as adding a layer of data validation to ensure that parameters entered are valid for the particular field. That aside, the user still has direct access to the underlying file to manually edit the file, in its native text format. The rationale for saving the result of the selections into a text file for manual modification applies for two reasons. One reason is the need to satisfy the stateless design objective; the second being the ability of a user to simply copy-and-rename an existing text file and merely change a few parameters manually rather than work through the user interface which may often be the less efficient solution for a power user. It was also for these reasons that a binary format as an alternative was decided against.

The output of this tool, the universe definition file, would serve as input to the engine. More specifically, the data in the file would be passed to the database stored procedure which would use the parameter values in its SQL query to return the filtered universe. The problem that arose is that of interfacing these parameter values with that of the database stored procedure. The traditional stored procedure required one input variable defined for each type of input to be received. As a result, if this tool offered 150 fields that could be manipulated, the stored procedure would need 150 input parameters to map the values of each of these field selections. Each time the parameters in the tool changed, the stored procedure interface would need to be similarly modified; neither a flexible nor extensible solution. Further, the more complex the input parameters became, the more difficult it was to replicate the stored procedure interface. For example, if a parameter was an object array, this could not be easily transferred to an appropriate data type in the stored procedure.

Additionally, the building of a complex argument with and'ing and or'ing of fields could not easily be communicated to the traditional stored procedure from an external application.

In order to be extensible, it was identified that the solution needed to satisfy the design criteria whereby a change to the tool would not require a change to the stored procedure interface. What this meant was that the stored procedure would need a method of accepting this data regardless of whether the number of parameters changed. It would also need to be able to accept, without need for change, any type of logic that needed to be applied to the filtering of the universe.

The first solution to be tested was to create an XML structure and pass this XML file to the database stored procedure which can accept data of type XML. It could then deconstruct the XML file and read the data using the built-in functions for XML data handling. This solution failed for the following reasons:

1. It was found to be cumbersome when parameters needed to be changed because of the need for changes to the XML schema, the universe selection tool which generated the XML file and the stored procedure interface

2. Creating a flexible XML schema resulted in an unnecessarily complex schema

3. To the non-technical user, the XML schema was significantly more complex to understand and modify than a simple text file or Excel spreadsheet

The solution to the challenge of flexibility and extensibility proved to be simpler than expected. Rather than creating data structures such as XML to transfer data between these two layers, it was simpler to pass the SQL expression being used to generate the Before/After window in the tool, directly to the database stored procedure. Several benefits arose from this solution:

1. The stored procedure would require a single input parameter of type string to accept the complete universe description as a single SQL string statement.

2. The solution affords extensibility and flexibility for the stored procedure because it could accept a SQL query of any complexity representing any number of table/joins/where groupings. To further emphasise the flexibility of this approach, one could also pass in a complete SQL stored procedure listing that would be executed as a sub-query to return a list of stocks, where this listing could have embedded sub-queries or stored procedures.

3. Since the tool was built to not require user modification if the database schema changed, which meant that any new fields in the database would be provided to the user without code modification; and since the treatment of these fields was already built into the logic of the tool, the SQL creation portion of the code required no modification for schema changes.

4. The solution abstracted the database stored procedure from the tool such that one need not utilise the tool to provide the database layer with a universe layer. If the future of the simulator, for example, required a neural network to decide the universe selection, it would simply have to convert its results to a SQL query and the database stored procedure would be able to process this without any knowledge of the neural network

In summary, after several design iterations and considerations, a tool was developed that operated directly off the schema of the database, allowed the user flexibility in grouping multiple parameters together using a range of logic and then communicate with the database via a resulting SQL query that fully described the parameters as set by the user.

Section 8.6, Database-Simulator Engine Interface, investigates the interface issues and solutions in more detail.

7.4 Scripting Language

The parameters that define a simulation environment are stored within a file which is referred to as the simulation **script** and resides in the SCRIPTS folder within the simulator folder structure.

Several options were available when deciding the type of format in which this data would be saved to file. These options included XML, TEXT (CSV/Tabbed) and EXCEL formats as these represent common file types used by users of the Windows environment. The objectives in determining the final format were the:

- size of the file;
- openness of the format; and
- ease with which a novice could read and modify the raw file data.

The results of the testing of these file types were:

1. XML, being one of the standards for interoperable data storage, was the file format initially selected for the storage of the input parameters. Whilst it satisfied the criteria for file size and openness of format, the XML format proved complicated for the non-technical user wishing to modify the file by hand.

2. The Excel format was decided against because of the proprietary format and the inability to modify the raw data without using the Excel interface, which was not necessarily available to the user.

This narrowed the options to either tab-delimited or comma-delimited text files. The decision to use tab-delimited was taken for three reasons:

1. The most important reason was that the Tab-delimited file allows one to layout a tree hierarchy within the file. This is useful for the user as it conforms to the way

hierarchical data is generally represented within Windows and allows the criteria to be logically grouped and ordered within the text file.

2. The second reason for this selection was as a result of the user interface software component that was used to graphically represent a tree structure. This component could natively illustrate a tree structure if the underlying file structure was tab-delimited. This saved several hours of software development as the component could both read a tab-delimited file and represent a tree as well as write back the modified tree structure as a tab-delimited file.

3. Finally, a tab-delimited file could be modified by a novice Windows user more easily than a Excel file or an XML file

User interface components were designed to aid the user in generating and editing a script file. For the purposes of this chapter, it is worthwhile summarising that a script is essentially a self-describing file detailing the simulation parameters to be run. An example of a script file, as copied out of its text file, is illustrated in figure 7-4. Note the hierarchical tree structure inherent in the layout of the text file; created through the use of tabbed spacing.

```
TEST G E&P HP1_00001
            SubTest  SA All L
                        Factors
                                    Factor 1
                                                Name = EBITMarginRelative_on_StockPickDate
                                                Rolling Days = 30
                                                Weight = 40
                                                ExclNegSQLData = No
                                                ExclNegPerChgData = No
                                    Factor 2
                                                Name = PEMarginRelative_on_StockPickDate
                                                Rolling Days = 130
                                                Weight = 60
                                                ExclNegSQLData = Yes
                                                ExclNegPerChgData = No
                        Days
                                    Holding Period = 30
                                    Use Calendar Months = TRUE
                                    Monthly Holding Period = 1
                        Correlation
                                    Use Correlated Groups = FALSE
                                    Correlate to INDEX
                                    Code = J203
                                    Rolling Back Days 30
                                    Minimum Trading Days 30
                                    From1 0
                                    To1 10
                                    From2 0
                                    To2 0
                                    From3 0
                                    To3 0
                        Fund Cash
                                    InvestorsCash = 1,000,000.00
                                    Opening Short = 0.00
                                    Debt = 0.00
                        Ranking Table
                                    Short BOTTOM
                                    SHORT 0
                                    Long TOP
                                    LONG 20
                                    Ascending = True
                        Auto Exits
                                    Sell=FALSE
                                    DynamicFloor=FALSE
                                    Risk Capital Rate = 0
                                    Short Loss Limit = 0
                                    Long Loss Limit = 0
                                    RelativePositioning=FALSE
                                    RelativeCalendarDays = 20
                                    CoverShortBelow = 60
                                    SellLongAbove = 60
                        Rates
                                    Cash Allocation Rate = 0
                                    Cash Collateral Rate = 0
                                    Equity Margin Rate = 0
                                    Avg Borrow Rate = 0.5
                                    Purchase Brokerage = 0.25
                                    Sale Brokerage = 0.25
                                    Performance Fee Rate = 0
                                    Mgmt Fee Rate = 0
                        Long/Short Liquidity Constraints
                                    Long % Daily VT = 25
                                    Long Days to enter = 10
                                    Short % Daily VT = 25
                                    Short Days to enter = 10
                                    %NAV Shorts Limit = 100
                                    %NAV Longs Limit = 100
                        Universe Filter
                                    Universe Filter Filename = G_E&P.sql
                                    Filter by = Liquidity
                                    Filter Method = NA
                                    Position Size = 1000000
                                    % of Daily VT = 50
                                    Days for Avg  = 50
                                    Days to enter = 5
                        Perf Indices/Rates/Rel Index Category
                                    Benchmark Index = OILEPWD
                                    Perf Index 1 = M2DWM2$
                                    Perf Index 2 = S&PCOMP
                                    Perf Index 3 = MSWRLD$
                                    BA Rate = US_CASH
                                    PRIME Rate = US_PRIME
                                    Rel Index Category = 1
                        General
                                    Subtest Description = G Oil L
                                    Subtest Currency = USD
```

Fig 7-4: Example of a script file

Within the software user interface, the hierarchical tree structure was chosen to display this data to the user. This method of data representation is understood by users of the Windows operating system due to its prevalence throughout the operating system and within most desktop applications. Through this interface, the user can clearly understand the parameters that make up the test to be run in the simulator, but also has the ability to change these parameters and re-save the data to file. The individual parameters have been grouped to represent their functional role within the simulator.

The illustration on the right shows the simulation parameters for a single test, TEST MastersDemo. Some of the branches of the tree have been expanded for illustrative purposes. This is essentially a text file and as such, can be edited using any text editor.

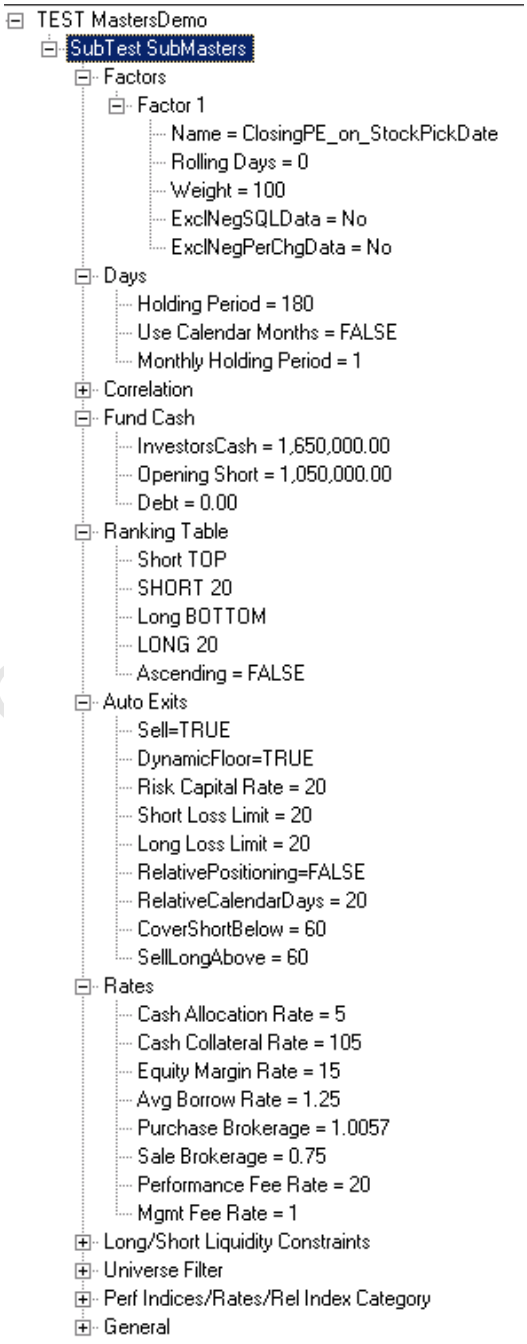


Fig 7-5: TreeView UI component

An architectural benefit of creating a parameter file which in essence is a configuration file, is that the file represents the sum total of user interaction with the system; since the user does not interact with the system post simulator execution, other than to review the results. As such, these parameters can be parsed for configuration issues before execution to ensure simulator runtime stability. Prior to a simulation executing, the script was parsed for two reasons; one being to initialise the variables of the simulator to the configured values, the other to sanity check the values and datatypes. For example, the boolean SELL parameter accepts TRUE or FALSE. If the user has entered another datatype, for example an integer value, it would halt the execution and request a script modification. Apart from datatype checking, value range checking was also applied at this stage to reduce simulation errors later on. An example of a value range error would be one where the user cannot enter a Cash Allocation Rate of 300 as this is an out-of-range number for this context.

The tree structure aids batch simulations by collapsing simulation parameters at the TEST level. Figure 7-6 is an example of what the user would see in a batch script which contains several tests. One of the tests has been expanded to illustrate the same parameters as illustrated in figure 7-5. This ensured that the simulator was extensible to cater for multiple tests.

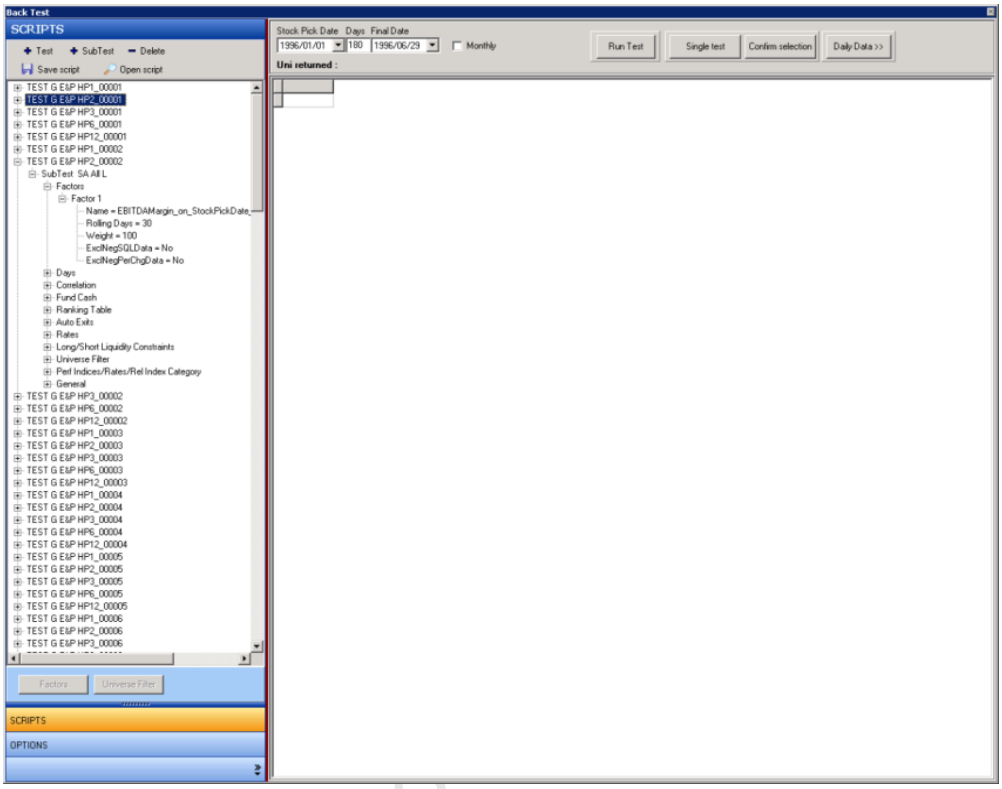


Fig 7-6: Batch script

Figure 7-6 highlights part of the user interface that was developed to aid in the visualisation, creation and editing of script files.

7.5 Summary

In summary, by creating a flexible universe-definition tool that auto-populated itself based on the underlying database schema; and that allowed the user complete flexibility in partitioning the universe across multiple dimensions, an extensible and flexible environment was created. In combination with a flexible, SQL based interaction between the tool and the database layer, extensibility was again realised.

The creation of files/folders that were consistent with the Windows metaphor, and logically grouped by functionality, afforded the user granular detail and easy, systematic access to the underlying simulation data without the need for database knowledge.

The creation of data at various stages of the lifecycle supported the stateless design objective by removing dependencies between systems/tools/processes, but relying rather, on the existence of data files as an input.

Finally a flexible, hierarchical parameters file, grouped by functionality, allowed the user the ability to manually create/edit files, allowed the system to error check parameters before execution and accommodated extensibility by allowing various scenarios to be batched before simulation.

Chapter 8

Database Layer

"The only source of knowledge is experience"

Albert Einstein

8.1 Introduction

At this stage in the research, chapter six introduced the complexities of constructing a database of historic data against which the simulator will execute. Chapter seven introduced the scriptable nature of the design which allowed the simulation environment and stock universe to be described. These two chapters provide foundations for the simulator logic. As described in chapter five, the database layer is the first segment of business logic to be invoked by the simulation engine via the business logic layer, which passes the data from the two input files, universe selection and simulation parameters, to the database layer. The objective of this layer, which occurs within the database engine, is to extract the universe defined by the universe selection parameters, and

- Step 1 : reduce this universe by removing illiquid stocks,

- Step 2 : reduce it further by removing stocks that do not pass the investment model criteria being tested

- Step 3 : normalise and rank the reduced portfolio of stocks which would then be passed back to the simulation engine for historic backtesting

This chapter investigates how each of these objectives was achieved with specific emphasis on how the final solutions evolved to meet the design objectives set out in chapter five.

8.2 Database Layer Overview

Figure 8-1 provides a high-level view of the architecture of the database layer. In understanding the end-to-end processes of the simulator, one understands that a simulation can be decomposed into two phases. The first phase begins with taking the complete database of stocks and reducing this to only the relevant stocks and then filtering these stocks against investment criteria to determine whether they are worthy of being a part of a portfolio or not. Those stocks that pass this test are then simulated in phase two to determine how they would have performed.

Looking at phase two first, the simulation engine was built to house the intelligence / business logic of the simulator; the simulation of stocks that have passed the universe reduction and criteria testing phases. Due to the complexity of the simulation logic, this engine was built in a high-level programming language. Once it had received the list of stocks to be simulated, it would retrieve the necessary data from the database for these specific stocks and perform said simulation.

The question that arose in the early design phases of the project was the location of business logic for the first phase of the simulation process; the universe reduction and criteria testing. One option was to build an application in a high level programming language and have the application perform the tasks of universe reduction and criteria testing. The disadvantage of this solution lies in the fact that these tasks of universe reduction and criteria testing require

knowledge of the complete dataset within the database. For example, in order to reduce the complete universe to only chemical stocks in Russia with a market capitalisation of two billion dollars, either the software would need to retrieve every stock in the database and all data relevant to the filtering process (Sector/Country/Market Cap) or it would have to invoke a stored procedure which would do the same. Likewise, in testing the reduced universe of stocks against investment criteria for example Average Price-to-Sales, either the software would need to read in the complete time series for each stock from the database, process the average and then rank the results; or it would invoke a stored procedure to do the same.

As explained in chapter six, one of the benefits of choosing the relational model over the object-oriented model for the database layer, was the existence of the advanced query language as well as tools to optimise data queries. As such, data-intensive logic that spans the data within the database should reside within optimised stored procedures which negate the need for data transfer to external applications. RDBMS's are optimised to perform such tasks and negate the need for software development of additional interfaces for the user.

Having made the decision to locate this business logic at the database stored procedure layer, the following objectives had to be addressed:

1. Is the database architecture of stored procedures extensible enough to enable the variations of steps 1, 2 and 3 as indicated in figure 8-1; that is, could the structure of the stored procedures offer multiple alternate methods of reducing the universe without significant software modifications?

2. In keeping with the design objective of reduced simulation time, was the architecture minimising the overall simulation time? Was the execution plan optimised to achieve results faster than had we written an external application to perform the same tasks?

3. If the database stored procedure layer was to accommodate three major steps of the simulation process along with catering for variants of each step, how would this orchestration take place?

This chapter details the considerations taken in developing the solutions to answer these questions.

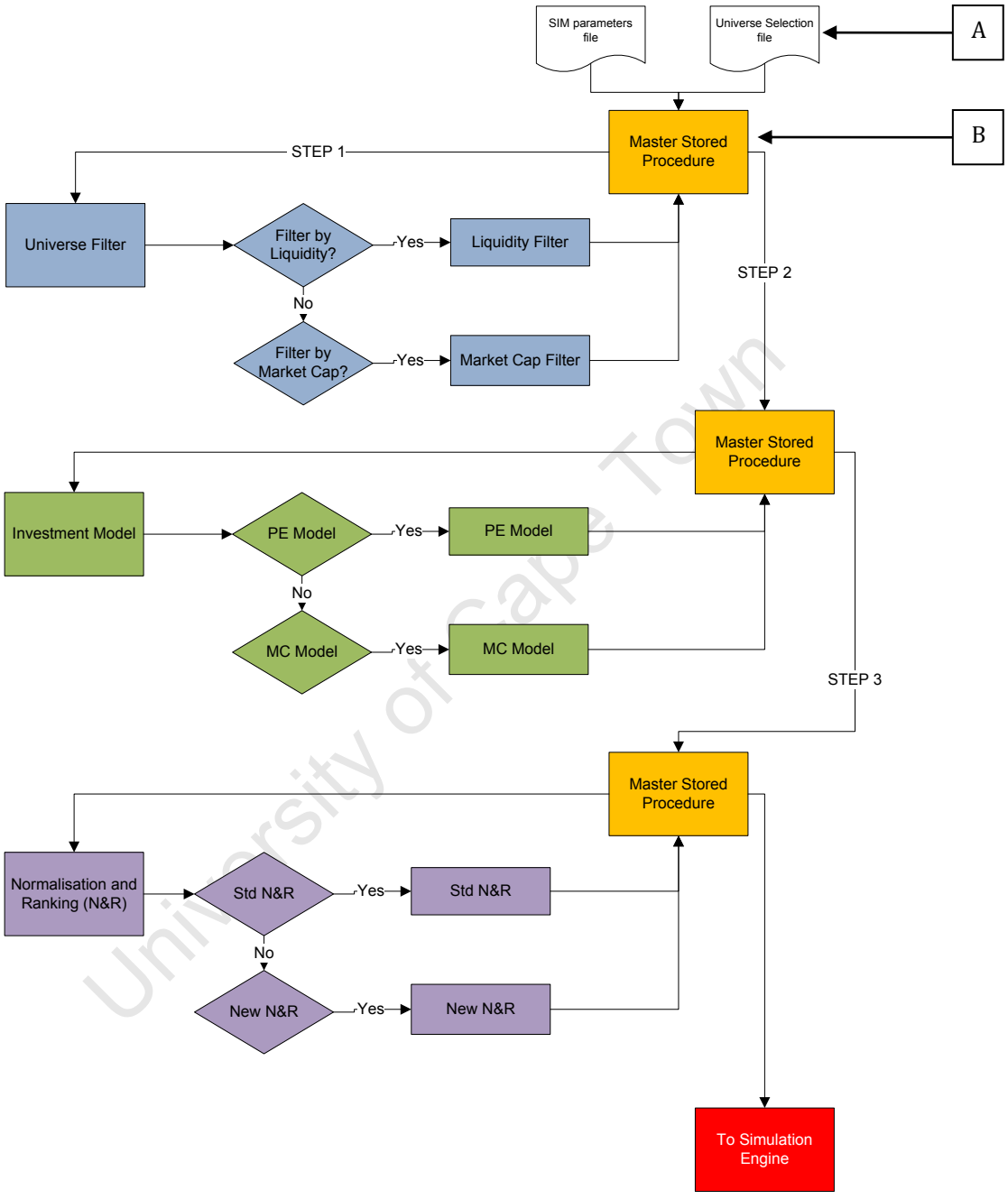


Fig 8-1: Flowchart of database stored procedure logic

Whilst figure 8-1 is the evolutionary result of several iterations of design, the initial design was a single stored procedure approach, which contained all the logic for these different processes. This initially seemed feasible because there existed only one methodology for each of the three steps. However, as the design evolved and alternate versions of each step needed to be offered, several major flaws were identified with this approach:

1. First was the exponential complexity of the stored procedure code as more sub-routines were added. Each step required multiple IF branches to determine and execute the requested methodology for each step. This affected the manageability of the code as well as the speed of the stored procedure.

2. Second, the SQL Server execution plan was far less optimised as a result of the excessive branching within the stored procedure. This resulted in reduced performance and simulation time which was in violation of the design objective.

3. The overall design of housing all the business logic for all three steps in a single stored procedure violated the extensibility objectives of the design because an increase in complexity resulted in a decrease in performance. An extensible solution should allow for growth of functionality without material performance sacrifice.

To solve the issues of extensibility and code manageability, one had to recognise that the first step was modularising the single stored procedure approach along the lines of functionality; that is, partitioning the single stored procedure into three stored procedures, one for each step. However, whilst this solved the short-term problem of code manageability, if each of these stored procedures had to then house several tens of versions of its primary function, the result would again be three unmanageable stored procedures. Further, in the transformation to three stored procedures, one would have to ensure that the completion of one stored procedure then called the next appropriate stored procedure in sequence. This raised an extensibility issue should a new step be introduced at a later stage, for example, in between steps 2 and 3. This would require a modification of step 2 to redirect its output to

the new step rather than to the original step 3, which created room for error should this update be omitted.

The solution to these design problems was derived from classic object-oriented design. If one considers each version of a particular step to be an instance of that step, and then considers that it is actually polymorphism at work, the solution would be to create a separate stored procedure for each version of that step, and then create a base stored procedure that would resemble that of an abstract class in object-oriented design. This base stored procedure would then call the appropriate "instance" stored procedure for the version of its functionality that needs to be run. The advantage of this design is its extensibility whilst solving the issue of code manageability by having separate versions reside in separate stored procedures. This same method could apply to steps two and three. In figure 8-1, the Universe Filter block is akin to the base stored procedure, whilst the Liquidity Filter and Market Cap Filter blocks represent the "instances" of the base stored procedure.

In order to solve the issue of step one having the logic to call the appropriate next step in the process flow, the solution to further abstract the design and create a Master stored procedure was elected as it extended the concept already being used for each step/sub-step. This stored procedure would be the only visible or publicly-accessible conduit to the underlying database layer and would essentially orchestrate the calls to the various stored procedures representing each of the steps. As such, each step need not have knowledge of any other step. If a new step were to be added, it would simply be created stand-alone and the orchestration stored procedure would be modified to determine when it would be called. Further, by having an orchestrating Master stored procedure, one could easily rearrange the order of steps.

The database layer flowchart in figure 8-1 can be explained as follows: The Master stored procedure (B) orchestrates all activities at the database layer. The simulation engine invokes the publicly-accessible Master stored procedure and passes the two input files (A) to it. The Master stored procedure consequently invokes the Universe Filter stored procedure which

in turn invokes the appropriate filter sub-procedure as indicated by the parameter file. The resulting dataset is returned to the Master which passes it to the Investment Model stored procedure which in turn calls the appropriate Investment Model sub-procedure. Again the result is passed back to the Master. It is then passed to the Normalisation and Ranking stored procedure which calls its appropriate sub-procedure and returns the result to the Master which then returns the overall resulting portfolio back to the simulation engine.

Whilst the final solution as illustrated in figure 8-1 may appear convoluted, several advantages must be highlighted.

- Each sub-procedure is essentially a separate stored procedure housing very specific business logic to be applied to the universe. Extensibility is achieved by this model as new sub-procedures can be added at will and the only change that will have to occur is at the parameter file level which directs the Master stored procedure to the appropriate sub-procedure.

- The second point to emphasise is that the entire process, whilst orchestrated by the Master, is driven by the parameter file which dictates which sub-procedure to run for each of the three phases of this process.

- A third advantage is the encapsulation of logic at multiple layers which significantly aid the testing and debugging process. Suppose one wished to create a new sub-procedure for step two, the investment model. One could create the new stored procedure in isolation and then be able to test this procedure by simulating inputs and validating the output without the need to modify or run any other part of the simulation engine or any other part of the database process. Further, one can debug problems by intercepting the inputs during an actual simulation, using tools such as the SQL Server Profiler, and then stepping through the code later to determine and correct the cause of the problem.

- A fourth advantage is that the three-step process can be easily expanded with the addition of new steps by adding an additional branch at the Master stored procedure level and modifying the input parameter file to direct the Master stored procedure to this step. This affords the simulator architecture extensibility at this layer of the design.

One of the problems that arose from this solution was the fact that using temporary tables, which are essentially in-memory datasets, was much less efficient than using actual indexed tables. As Rickhard and Rossberg [RR05] point out, temporary tables, by their nature, force a stored procedure to be recompiled for each execution, rather than utilising a cached execution plan. This recompilation overhead significantly impacts the performance of the query. The constant back-and-forth passing of the result set from the Master to each stored procedure was then revisited. Further performance improvements were achieved by having the user-defined universe reside within an actual indexed database table which each stored procedure had direct access to and could modify at will and which negated any need for passing datasets in memory through this inefficient temporary table mechanism. The results of the modification was a reduction in simulation time from nine seconds to six seconds, which as illustrated in chapter five, has very significant effects on bulk simulation time.

8.3 Step 1: Universe Filtering

To recap, the original database used for this research consisted of six thousand stocks within the global Resources sector. Chapter seven discussed the creation and use of the Universe Definition file which further reduced the universe based on criteria such as "*All Gold Stocks*". Step 1 of the database layer activities extends universe filtering by further reducing the generalised universe definition in applying broad criteria relevant to **any** investment universe. An example of such a criterion is that of liquidity as described in section 2.7 which defined liquid and illiquid stocks. Depending on the type of simulation being executed, a potential method of universe filtering could be the elimination of illiquid stocks. For example, if one were simulating a portfolio which intended on deploying ten billion dollars into large capitalisation stocks, illiquid small capitalisation stocks, whilst still available for

purchase, would be avoided due to the fact that the investment amount would have a direct impact on the share price.

The choice of which universe filtering method (sub-procedure) to execute is dictated to the Master stored procedure by the input parameters file which consequently makes the process extensible because the only change required for addition/deletion of methods would be in the parameter file. Should one choose to develop an alternate filtering technique, for example, filter all stocks which start with the letter B rather than their liquidity, a new stored procedure which houses this logic would be created and the parameter file would instruct the Master stored procedure to pass the user defined universe SQL string, to the newly defined filter rather than to the liquidity-based filter.

For the purposes of this research, liquidity filtering and market capitalisation filtering were the only filtering techniques implemented. By implementing these methods, the extensible infrastructure was tested and the increased performance of cached execution plans was realised through the reduction of simulation time. The primary input to either of these filters was the user-defined SQL statement generated from the index definition tool described in chapter seven. This SQL statement, when run, returned a reduced universe of stocks against which the liquidity or market cap filter was executed. Liquidity filtering essentially removed any stocks within this reduced universe that were not sufficiently liquid; whereas Market cap filtering simply removed all stocks whose current market capitalisation at the time of the simulation were below a specified level as defined by the parameter file. Having reduced the universe through filtering, the emergent stocks were potential candidates for trading. The next step would be to determine, from this reduced successful universe of stocks, how many of these stocks then met the investment criteria being tested.

8.4 Step 2: Investment Model

At this stage, the investment model, as explained in section 2.12, is introduced to the process. The investment model dictates the criteria that define the attractiveness of the stock, for example, rank stocks in descending order by their dividend yield, with the top 10 stocks

being the best **buy** candidates and the bottom 10 stocks being the best **sell** candidates – simply because one may wish to invest in stocks that yield high dividends.

In order to achieve this, the stored procedure uses the filtered universe as described in section 8.3 to extract the corresponding financial data for the criteria being tested, in this case the dividend yield for each stock. More complicated criteria may require the stored procedure to calculate a value for a formula which uses underlying stock data, for example, the division of the price by the previous year's earnings (PE ratio). The input parameter to the Master stored procedure would be used to indicate the investment model to be applied to the filtered universe. An important point to emphasise is that the architecture allows extensibility as each new model resides in an independent stored procedure and the only change would be the input parameter file to instruct the call of the procedure. Further, by allowing each model to reside in an independent stored procedure, the encapsulation allows for valid SQL data processing to occur without conformance to a particular methodology enforced by a framework; that is, the internal workings of any model need not be similar in design to any other model. The only consideration for the developer of an investment model is that the input to the procedure is of type dataset and the output must be a dataset of the same schema – essentially making the model a black box to the rest of the system.

One of the unintended architectural benefits that was realised through this design of an "abstract" stored procedure orchestrating independent stored procedures, was that of multi-factor modelling. If the user wished to combine any two models, the orchestration stored procedure could execute the filtered universe against each of the independent models and then combine the results using section 8.5 as discussed below. The same architectural benefit could be afforded to the other steps as well, where, for example, the user may wish to combine more than one type of filtering in step 1 before moving to step 2.

8.5 Step 3: Normalisation and Ranking

The final process at the database layer is the normalisation and ranking of the universe that emerges from the application of the investment criteria in the investment model. As

explained earlier, the need for this layer is to ensure that stock comparisons that are based on the investment criterion are comparable. For example, if the investment criterion was the ranking of a combination of market value and price/earnings ratio, one cannot compare two companies that are vastly different in size, for example Microsoft ($170bn market capitalisation) and CNN ($31bn market capitalisation), without first standardising the data. In order to achieve such comparison, one must ensure that the values to be compared are first normalised to the same base.

Several statistical techniques are available to "flatten" numbers to enable the comparison of data. Examples of these include:

- Mean shifting
- Percentile matching
- Applying a nonlinear smoothing curve
- Z-score
- Log transform
- Tail preservation

Each of these techniques has specific application domains within which they perform best. Whilst an in-depth analysis of each technique is outside the scope of this research, each of these techniques was considered and in some cases tested, before the final technique was selected. The reasons for discarding techniques ranged from:

- inaccuracy due to an incorrect problem domain for the technique,
- difficulty in implementing the technique using SQL queries (smoothing curves for example), or
- unacceptably long processing times

For this research, the statistical method of z-score normalisation was selected, due to its simplicity in being implemented in SQL, combined with the speed of calculation since it only

depended on average and standard deviation functions, both being optimised native functions of the RDBMS.

As defined by Urdan [Urd05], Z-Score normalisation, based on the normal probability curve, requires that each member in the population should have the average of the population subtracted from it and the result divided by the standard deviation of the population.

$$x(normalised) = \frac{x-\bar{x}}{\sigma(x)}$$

Consider the following example using three stocks and two investment criteria: (data provided from Factset as at 12 February 2009)

Stock	PE	Market Value
Apple Inc	18	$86bn
Google	26.5	$112bn
Microsoft	10.3	$170bn
Average	18.27	122.67
Std Deviation	6.62	35.11

Table 8-1: Sample stock data

As evident in the data, these stocks are vastly different in terms of their company size. As such, it would be difficult to compare these companies. Suppose one wished to test the hypothesis that the combination of PE and Market Value would be a leading indicator of stock profitability. In such a case, one could not simply rank the sum of these two numbers because the quantum of each is so vastly different.

Applying the z-score on the PE and market Value columns to generate their z-scores yields:

Stock	PE	zPE	Market Value	zMV
Apple Inc	18	-0.04	$86bn	-1.04
Google	26.5	1.24	$112bn	-0.3
Microsoft	10.3	-1.2	$170bn	1.35
Average	18.27		122.67	
Std Deviation	6.62		35.11	

Table 8-2: Sample stock data with z-score value

Once the individual factors have been normalised, the sum of the normalised fields yields a normalised score which allows for stocks to be compared in a multivariate model as indicated in table 8-3. One could not simply have added the original values of the PE and Market Value columns to create a cumulative view of each stock for comparison to its peers.

Stock	PE	zPE	Market Value	zMV	ZScore
Apple Inc	18	-0.04	$86bn	-1.04	-1.08
Google	26.5	1.24	$112bn	-0.3	0.94
Microsoft	10.3	-1.2	$170bn	1.35	0.14
Average	18.27		122.67		
Std Deviation	6.62		35.11		

Table 8-3: Multifactor scored value

Table 8-3 indicated a multi-factor model with the score being calculated as

$$zScore = zPE + zBookValue$$

In keeping with the discussion of extensible architecture, it was decided to use a "sum-of-the-parts" approach to developing the investment models. What is meant by this is that each individual investment factor was built as an independent module, and the results of several modules could be aggregated at the "abstract" orchestration stored procedure layer

thereafter. As a result, one could perform single factor or multi-factor tests with no modification to the code.

One may also wish to have individually weighted the components that make up the score based on varying levels of conviction with regards to the value add of each factor to the overall score. For example, one may believe that PE is twice as powerful an indicator of stock attractiveness than Book Value and hence the score would be calculated as

$$Score = 2zPE + zBookValue$$

The scores are thereafter ranked numerically and the top and bottom decile/quintile will represent the best buy/sell opportunities respectively. In keeping with the theme of extensibility, the input parameter file specifies which factor(s) need to be evaluated, and the weight for each of these factors. The scores of a single investment model are multiplied by the user-specified weight before being returned to the orchestrating stored procedure for investment models. As such, the orchestrating procedure simply adds the scores of multiple tests to arrive at an overall score. If the user wished to have a score which was the subtraction of one factor from another, the weight would be specified as a negative.

Figure 8-2 is a screenshot of a normalised and ranked universe resulting from a database query. The reader's attention is drawn to the last three columns. The last column, _valEBITDAMarg1 indicates the actual value of this factor (raw data) for each stock in the filtered universe. As explained earlier, stocks cannot be compared to each other on this basis, and this column is therefore normalised with the values in _EBITDAMarg1. As can be seen in this column, the numbers all range between -1 and 1 and can therefore be compared to each other and also ranked. In this particular example, there is only one factor, therefore the _RankingFactor (zScore) column is simply the sum of one number, however, had the example included several factors, the _RankingFactor column would be the sum of each of the normalised numbers to generate a score for each stock. The entire table is thereafter sorted by this _RankingFactor in descending order. This is the method used by the system to cater for multi-factor models.

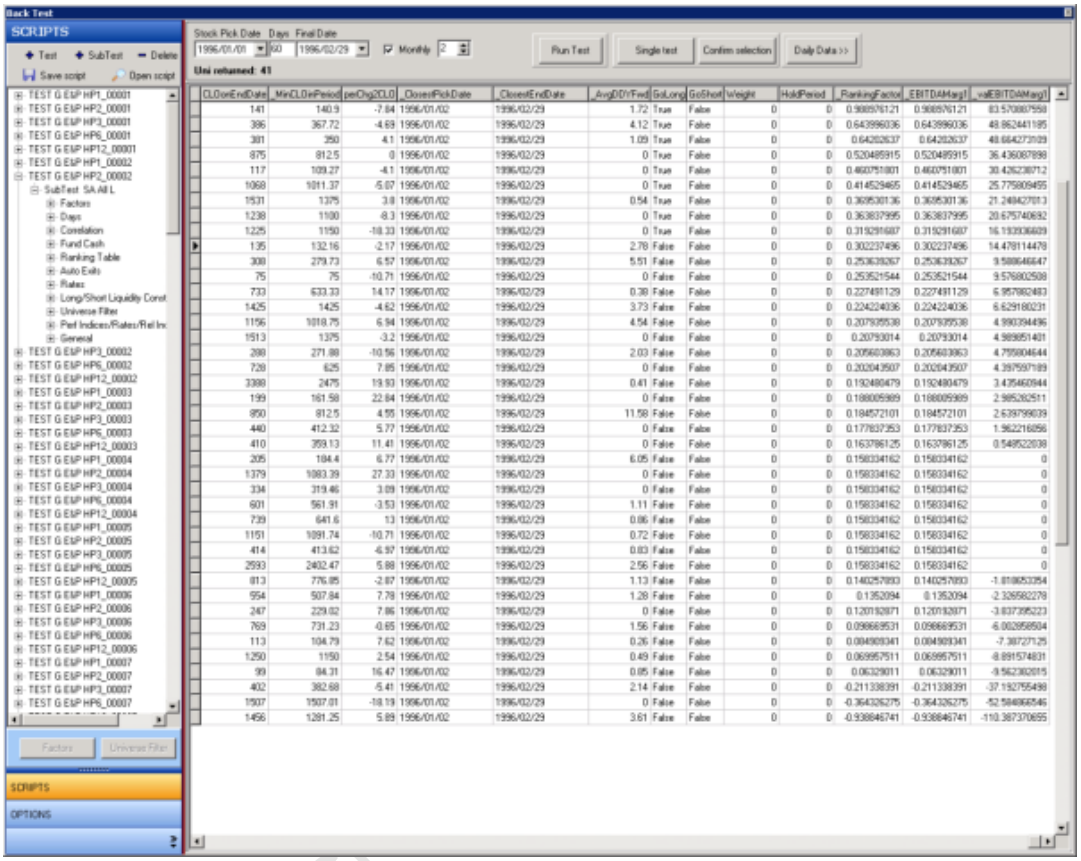


Fig 8-2: Example of a normalised and ranked universe

8.6 Database-Simulator Engine Interface

As explained in section 7.2, interaction with the database is the first step of the simulation process. The database layer requires several inputs and generates a filtered-and-scored dataset of stocks to be simulated by the simulation engine. As such, this interface between these two pieces of software had to be as efficient, yet as generic as possible, to reduce query time, afford extensibility and reduce the quantity of data transfer. A further key

consideration in designing this interface was to ensure that the underlying extensibility designed into the database layer was also maintained.

Key observations in the design of the interface were:

1. The use of a single parameter at the database layer which received the universe definition as a SQL statement, afforded extensibility and flexibility regardless of how complex the definition became. In essence, one could pass the entire content of a stored procedure to this variable as a string and it would execute this to generate the reduced universe

2. The parameters used by the Master stored procedure were only a subset of the overall parameters of the simulation engine, and were selected only if they were required by the database layer to reduce the quantity of data transfer to the Master stored procedure

3. Where possible, generic fields were created, such as *IntParam1*, *IntParam2*, *NumericParam1*, which would hold integer or numeric values independent of the process they applied to, which would be dictated by another parameter. For example, if in figure 8-3, the @UniFilterBy parameter indicated filtering by liquidity, then IntParam1 would hold the investment amount and IntParam2 would hold the minimum value traded amount. However, if the @UniFilterBy parameter indicated filtering by market cap, these parameters would hold values indicating the upper and lower percentage range to filter by.

4. Since the RDBMS could not cater for table structures being input as parameters, tables were passed to the interface as comma-separated strings, with a function being developed to decompose this string back into a table structure (figure 8.5)

```
ALTER   PROCEDURE [dbo].[BTC_EQUITY_xFACTORs_TEST_V2]
@StockPickDate datetime,
@EndDate datetime,
@UniFilterBy varchar(255),
@UniFilterStringParam1 varchar(255),
@UniFilterStringParam2 nvarchar(max),
@UniFilterIntParam1 int,
@UniFilterIntParam2 int,
@UniFilterNumericParam1 numeric(24,2),
@UniFilterNumericParam2 numeric(24,2),
@Ascending bit,
@InclTheseStocksOnly varchar(8000),
@TestNamesCSV varchar(8000),
@WeightsCSV varchar(8000),
@RollingDaysCSV varchar(8000),
@ExclNegSQLDataCSV varchar(8000),
@ExclNegPerChgDataCSV varchar(8000)
AS
```

Fig 8-3: Interface to the Master stored procedure

```
ALTER   PROCEDURE [dbo].[BTC_UNIVERSE_FILTER_ON_LIQUIDITY]
@StockPickDate datetime,  /*drop all companies listed after this date */
@EndDate datetime,
@Days2Enter int, /*no of days to go back from stockpickdate below*/  /*@UniFilterIntParam1*/
@ExitDays int, /*@UniFilterIntParam2*/
@KnightTrade numeric(24,2), /* % of AvgVT */  /*@UniFilterNumericParam1*/
@PositionHeld numeric(24,2), /*@UniFilterNumericParam2*/
@InclTheseStocksOnly varchar(8000),
@MaxRollingDaysAcrossTests int
AS
```

Fig 8-4: Interface to the Universe Filter stored procedure

One of the challenges of the interface was the ability to transfer variables that contain multiple selections. For example, the variable @TestNamesCSV is the variable that holds the investment model that is to be applied. The architecture needed to be flexible enough to allow for multiple tests to be merged at execution. Since the number of tests that could be merged is decided by the user, one has to create an array construct which will allow several values to be passed into one variable. This was achieved through the use of comma separated strings. Once in the Master stored procedure, these strings were passed through a conversion process which decomposed the comma-separated string and returned the individual values in a single column table format. This technique allowed for significant extensibility and was used throughout the database layer development.

This is illustrated in the code segment of figure 8-5.

```
ALTER FUNCTION [dbo].[CSV2TempTable]
(
        @Item varchar(8000)
)
RETURNS
@ParsedList table
(
        Item varchar(500)
)
AS
BEGIN
        DECLARE @OrderID varchar(500), @Pos int

        SET @Item = LTRIM(RTRIM(@Item))+ ','
        SET @Pos = CHARINDEX(',', @Item, 1)

        IF REPLACE(@Item, ',', '') <> ''
        BEGIN
                WHILE @Pos > 0
                BEGIN
                        SET @OrderID = LTRIM(RTRIM(LEFT(@Item, @Pos - 1)))
                        IF @OrderID <> ''
                        BEGIN
                                INSERT INTO @ParsedList (Item)
                                VALUES (CAST(@OrderID AS varchar)) --Use Appropriate
conversion
                        END
                        SET @Item = RIGHT(@Item, LEN(@Item) - @Pos)
                        SET @Pos = CHARINDEX(',', @Item, 1)

                END
        END
        RETURN
END
```

Fig 8-5: User-defined SQL function – CSV string to Table conversion

8.7 Summary

In summary, the core logic to generate a basket of stocks for simulation lies within the structured stored procedures within the database. Through modularising the functionality and creating a "polymorphic" database architecture, the design afforded extensibility to add more steps to the process as well as more sub-processes to each process, without modification of code at any level other than the parameter file.

Chapter 9

Business Logic Layer

"You can never solve a problem
at the level on which it was created"
Albert Einstein

9.1 Introduction

Chapters six, seven and eight introduced the reader to the design considerations in constructing a database, building tools to setup parameters and define investment universes and developing database-layer logic to perform tasks such as liquidity filtering and modelling. The resulting dataset from the research described in chapter eight is a list of stocks which potentially represent an investment portfolio that the Portfolio Manager may wish to invest in. This portfolio of stocks would have satisfied the liquidity constraints as well as the investment criteria being tested. However, the Portfolio Manager has not yet been able to ascertain whether this portfolio, based on this particular investment criteria, has acceptable historic performance; a potential indication of the portfolio's future performance.

The next step in the process is to backtest the performance of this portfolio in line with the financial mechanics outlined in chapter two, whilst applying the constraints of backtesting as detailed in chapter four. The Business Logic Layer is essentially the simulation engine or software agent, as introduced in appendix D and referred to throughout this research paper. This simulation engine has the responsibility of providing the user with an accurate assessment of the performance of this selection of stocks during a historic investment window.

This chapter introduces the reader to the high-level logic employed in this simulation engine.

9.2 Fund Management Overview

In order to architect an object-oriented solution as mandated by the design objectives of chapter five, one needs to first decompose the problem being solved into its various functional entities. Once these entities have been identified, a mapping of their relationships to each other can be established in order to determine, if any, the hierarchies of the objects relative to each other.

Whitten [WBD01] describes the *Requirements Discovery Method* as a systems analysis technique which focuses on fact-finding techniques to determine a description of the problem to be solved. As per Whitten, the expected outcome of a successful analysis process is both an understanding of the problem to be solved as well as an ability to identify the entities, stakeholders, properties and methods for the system design.

This discovery process was conducted through continuous informal interviews with the stakeholders and observations of the business process as well as a detailed code-review of the existing simulator in a production environment. It was further substantiated through a thorough analysis of the fund accounting documentation for funds which had been invested in the stock market. The outcome of the discovery process, as expected, correlated to the mechanics as described in chapter two. Below are key findings of the discovery process

which were used later in the chapter to define the objects for the design (words in bold refer to key facts):

- *Pivotal to the management of money is the **Portfolio Manager**, the investment professional responsible for deploying the invested capital.*

- *It is the fund manager who assimilates multiple sources of information, in the continuous monitoring of the portfolio and through an understanding of this data, is able to **buy** and **sell** stocks in the portfolio as well as manage the **risk** of the portfolio.*

- *Types of the information assimilated by the Portfolio Manager include **external market data** such as economic global interest rate data / index data as well as **individual stock data** such as corporate news and stock price movement.*

- *The investment process begins with an investment of capital into a **bank account**. Some or all of this capital is then used in the purchase of stocks. During an investment lifecycle, money resides either in **Long Equities**, **Short Equities** or as Cash in the bank with each returning different rates of return on a daily basis.*

- *The Manager may buy or sell stocks throughout the investment window, whilst also **earning interest** at the current rate of the time, on all monies in the bank. The decision to buy or sell stocks is based on the assimilation of the data as mentioned above.*

- *On a daily basis, the portfolio is monitored in terms of the various **liquidity** and **risk** constraints as per the portfolio mandate. **Breaches** or potential breaches, once identified, have to be resolved using various investment methods, each appropriate to the type of breach.*

- *The success/failure of an investment strategy is ultimately the **return profile** of the portfolio; however, along with this single metric of performance, the Manager generates a plethora of **statistics** regarding the portfolio's investment profile which aid in the daily management and decision-making processes of the portfolio.*

- *The results of the portfolio simulation should be presented via a **GUI** which accommodated interrogation of the results*

9.3 Software Agents

Whilst a detailed discussion of software agents and artificial intelligence are outside the scope of the research, this section shall summarise key elements of agent design that were used in this research. Bradshaw [Bra97] references Shoham in defining an agent as "*an entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes*". Bradshaw further acknowledges that an agent operating in an environment should ideally learn from its experience as well as communicate with other agents in the environment. The proposed design in this chapter is a derivative of these descriptions, and whilst it does not follow the strict definition of an artificial intelligence software agent, the principles used have been derived from the AI subject matter described in Appendix D as proposed by Norvig [RN03].

As per Norvig [RN03], the simulation environment for the agent meets the definition of being fully observable, deterministic, sequential, dynamic, continuous and multi-agent. The "agents" to be developed were both goal-based and utility-based.

Each of the objects to be described in section 9.4 was developed to operate as an agent. As will be explained, a trading environment ("the environment") was created, within which these agents exist. An example of such an agent would be the Bank agent, to be described in more detail later. This agent is aware of its goal, the interest rate at the time, which it derives from the environment and any calls to it which depicts either withdrawals or deposits. It remains autonomous and calculates interest on cash daily (goal-based), regardless of the

rest of the system. It also has the logic to accept or reject a request for cash from a stock object depending on the state of the portfolio and environment at the time of the request. This is a result of the inherent utility function that has to be satisfied. Each of the objects to be described in section 9.4 has been developed with this framework as a guiding principle.

9.4 The Object-Oriented Design Process

The result of chapter eight was a ranked universe of liquidity filtered stocks that had been normalised and scored on the basis of an investment model. This result serves as the input to the simulator which will simulate the performance of this portfolio of stocks over a given historic time period using the simulation engine.

Section 9.2 represents a high-level business analyst viewpoint and serves as a prelude to the first phase in the object-oriented (OO) design process. As explained by Bahrami [Bah99], this business analyst viewpoint must be converted to a functional specification of the project. This functional specification then allows the designer to identify the classes to be modelled within the system, through the differentiation of actors and actions. Each class will in turn be designed with properties and methods appropriate to their role/representation in the functional specification – bearing in mind the need for differentiated objects within a class.

From the descriptions of 9.2, the following classes were identified:

- The *Portfolio Manager* (PM) serves as an actor or stakeholder in the system. In terms of this research, this entity represents the Agent as described in appendix D; the primary responsibility of the entity being to orchestrate the various functional roles of portfolio management.

- A *Stock* entity represents the tradable entity in the portfolio which the PM can buy or sell during the lifetime of the portfolio

- The *Bank Account* entity which keeps track of capital that has not been invested and therefore accrues interest on debit balances whilst tracking payments of interest for debts on borrowed stock

- The *External Market* entity which houses the external macroeconomic data which the PM assimilates in the portfolio management decision making process. This entity also provides the *Bank Account* entity with external market data such as interest rates on debit/credit balances at a particular point in history.

- The *Portfolio* entity which maintains the daily historical performance information and statistics of the portfolio throughout the simulation period. Portfolio management and risk management rules are based on this entity.

- The *User Interface* entity which provides feedback to the user

In terms of OO, the second phase of the design was that of class hierarchy. Class hierarchy requires that the general classes as described in the previous paragraphs, be segregated as subclasses and superclasses. As defined by Bahrami, the distinction between the subclass and the superclass is that the subclass inherits all the properties and methods of the superclass, but not vice versa.

From the entities defined thus far, one understands that for all entities, with the exception of the stock entity, there shall only be one instantiation of the class. The primary class in the design, referred to as *uPortfolioManager*, serves as the orchestration agent and instantiates each of the *uPortfolioData, uBank, uMarketData* and *uUserInterface* classes. Since the design only required a single bank account, or a single user interface, there was no need to create multiple instantiations of these classes.

The *Stock* entity however, represented a class which could be further decomposed into stocks that were bought ("*going Long"* as described in chapter two) and stocks that were

borrowed ("*going Short"* as described in chapter two). Both these distinctions were derivations of the *Stock* object, and therefore created the class hierarchy. This class hierarchy has the *Stock* class as the superclass and the *uLong* class and *uShort* class, as the subclasses. Bahrami refers to this derivation of the subclasses from the superclass as inheritance.

Whilst the *uLong* class and *uShort* class have been described as two separate classes in this research, they are in essence identical. As a result, polymorphism was used to allow for generic code to be written for a single class which would behave differently based on the two different types of input.

The functional specification of the simulator for this research yielded the properties and methods for each class. As an example, the functional specification indicated that the PM buys and sells stocks in the portfolio as well as performs risk management. To encapsulate these actions, the verbs *DoBuying, DoSelling and DoRiskMgmt* represent the methods within the *uPortfolioManager* class that performs these actions. It is important to note that the functional specification evolved during the research process, and consequentially, so too did the class properties and methods.

Figure 9-1 is an illustration of the class diagram for the simulator. The class diagram highlights the relevant methods and properties of each class.

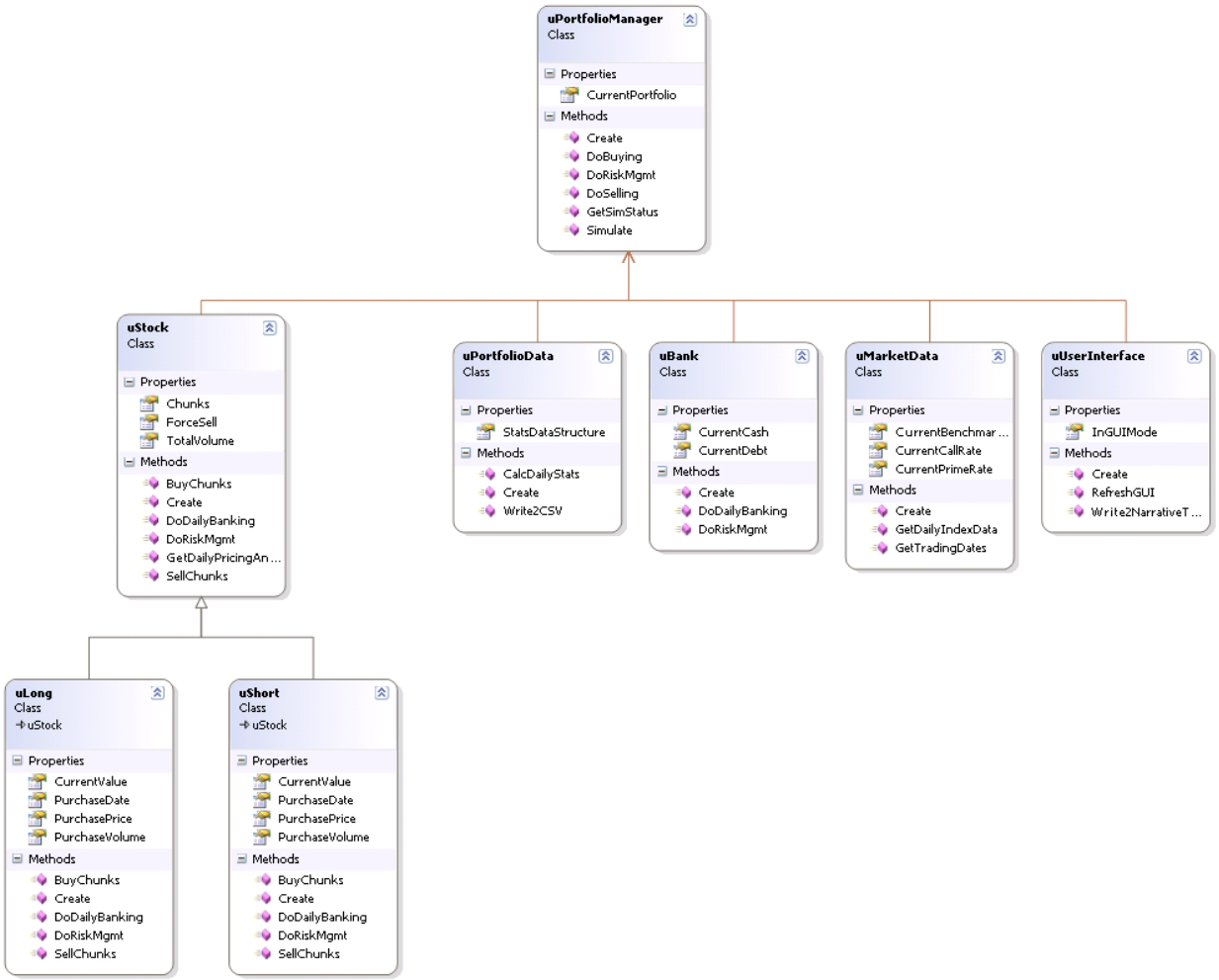


Fig 9-1: Class diagram of Business Logic Layer

The *uPortfolioManager* class is responsible for creating each of the associated classes. *uStock* is the only class with multiple instances of the class created, one instance for each of the stocks in the input portfolio. The *uStock* class instances are stored within the *CurrentPortfolio* arraylist of *uPortfolioManager*. Since stock is purchased in lots at varying periods during the simulation, the concept of *chunks* was created. In hindsight, it should have been referred to as *lots*. A *uStock* object is determined to be either a long or a short

investment purchase as per the ranked portfolio returned from the database layer. Once this determination has been made, each chunk that is purchased/sold is represented by a new instance of either the *uShort* or *uLong* class for the stock in question. All instances of *uShort*/*uLong* object for a particular *uStock* object are stored in the Chunks arraylist of the *uStock* object.

As the *uPortfolioManager* object iterates through each day of the simulation period, it iterates through each stock within the *CurrentPortfolio* arraylist and invokes the *BuyChunks()* and *SellChunks()* methods of the stock object. The stock object in turn iterates through its *Chunks* arraylist and for each *Chunk*, invokes the underlying methods. These methods use the state variables of the *Chunk* to determine the appropriate action to execute.

In a similar manner, the *uPortfolioManager* object iterates through the single-instance classes and invokes methods which in turn determine / modify the state of the environment. Examples of this are the changing of the bank balances, the change of current date, and the state of portfolio risk amongst others. The effect of these classes on the environment will be used as input to the *uStock* objects in their next iteration. For example, the end of day banking calculation may result in a risk breach such that a particular stock may be over-exposing the portfolio to a particular risk parameter; for example excess exposure to South America. In the next iteration of *uStock* objects, the defaulting South American object will have read in the state variable and determined that it would need to reduce its position size, and invoke its internal sell method.

The *uPortfolioData* class is a publicly accessible class that calculates the various performance statistics of the portfolio on a daily basis throughout the simulation period. This functionality has been encapsulated as it may be considered a reporting function and its constant evolution with newer statistics is made simpler through segmenting it. This class represents the “backoffice” in a standard fund management business.

The *uMarketData* class serves as the interface between the *uPortfolioManager* and the market environment. This class contains the non-stock specific, macroeconomic data that

may be used in the simulation. Such examples include the interest rate, the call rate, the various world indices as well as political information that cause regime shifts in the market structure.

The *uUserInterface* class is the interface between the Presentation layer and the Business Logic layer (BLL). As per the three-tier design pattern, the BLL cannot access the Presentation layer directly. One of the input simulation parameters indicates whether the simulator is being executed in GUI mode or command-line mode. In the event that the latter is true, the *uUserInterface* object becomes redundant as its methods are not invoked.

Figure 9-2 provides a high-level flowchart of the logic within the *uPortfolioManager* object of the simulator.

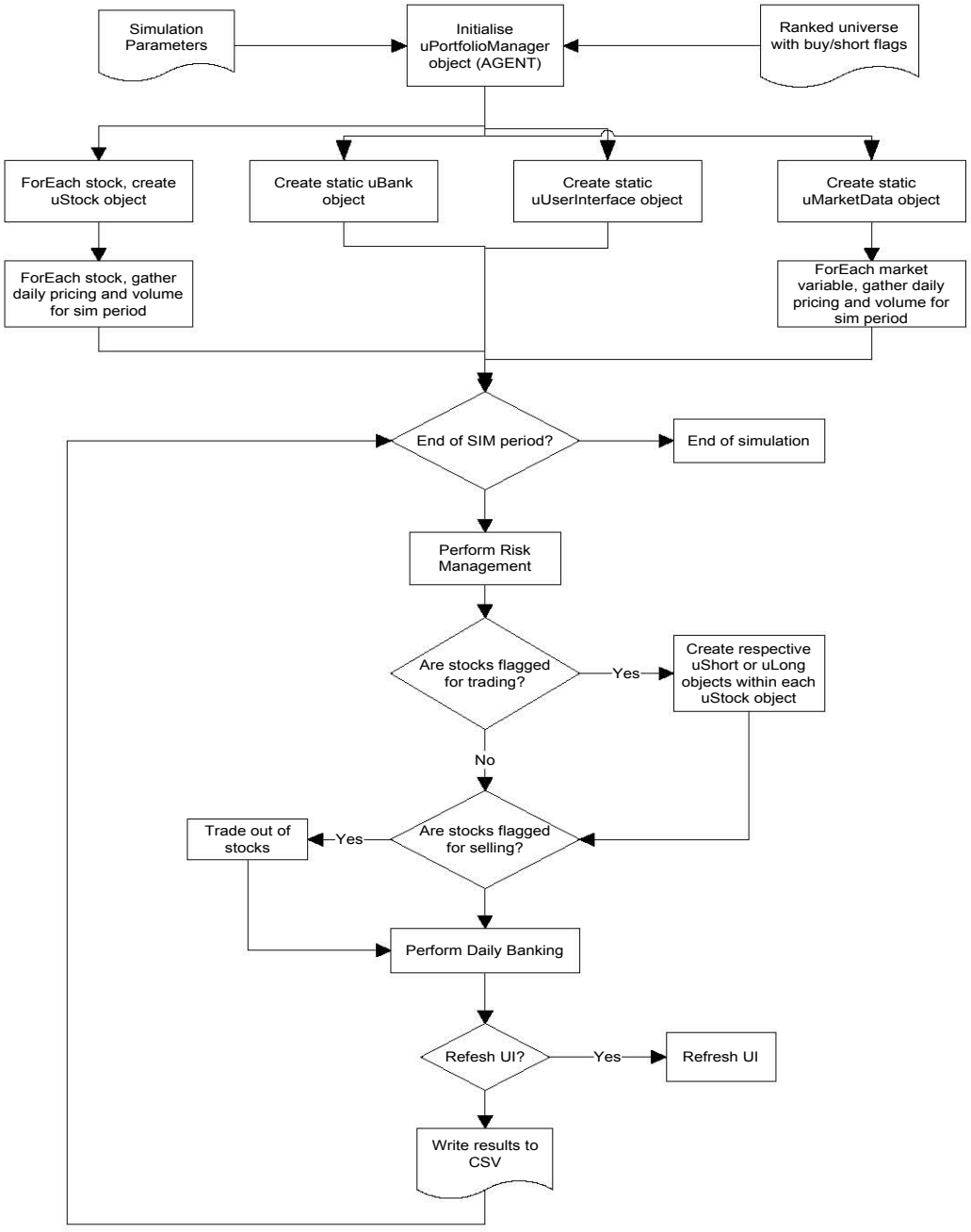


Fig 9-2: High-level flow chart of *uPortfolioManager* logic

For each stock in the input ranking table, *uPortfolioManager* creates a stock object which will later be populated with *uShort* or *uLong* objects. Other initialised objects are single instance classes such as *uBank* and *uUserInterface*. Once the initial objects (excluding *uShort*/*uLong* objects) have been created, the loop through each day of the simulation period begins.

As illustrated in figure 9-2, for each day, *uPortfolioManager* orchestrates several processes, as would a fund manager. For each stock within the portfolio, several determinations are made:

1. Perform the necessary risk management checks to determine if any mandate violations force a stock to be removed/reduced in exposure

2. Is the stock in a buy/sell phase? If so, attempt to buy/sell and increase / reduce the number of *uShort*/*uLong* instances for the stock in question

3. Recheck the risk management to ensure that step (2) had the desired effect to resolve issues raised in step (1)

4. Determine if there is availability to increase stock exposure in the portfolio, and if so, determine which stocks to buy and proceed to step (2)

5. Value the stock's contribution to the portfolio after step (2) has completed

6. Perform the general banking functions of the portfolio such as interest and mark-to-market calculations

7. If running in non-command line mode, update GUI, otherwise write results to file only

9.5 Summary

Having derived a functional specification through an understanding of the mechanics described in chapter two combined with the analysis techniques referred to in section 9.2, the various entities and relationships were defined for the business logic layer of this design. Through the use of OO techniques such as inheritance and polymorphism, entities were created to represent the various roles, with methods and properties representing the real-world mechanics of the entity. Similar to the modular design adopted for the database layer as described in chapter eight, this chapter also introduced an orchestration agent which coordinated the various workflows of the entities created. The orchestration agent is the construct as described in appendix D, AI Agents. The agent was designed to operate in a partially-observable, sequential, discrete environment with goal-seeking, utility-based logic.

Chapter 10

Evaluation

No amount of experimentation can ever prove me right;
a single experiment can prove me wrong
Albert Einstein

10.1 Introduction

The objective of this research paper was the investigation into the design considerations and development of a simulator to backtest investment portfolios. As discussed in section 1.3, the determination of simulator success lies in the accuracy of the simulated result. A simulated result, in this context, refers to the historic performance profile of a portfolio of stocks, constructed through the combination of historic market data and the application of backtesting logic as discussed in this thesis. One method of comparison in the determination of simulation success is the comparison of the simulated result to the performance profile of a market-invested portfolio. An alternate method of comparison would be against the result of alternate simulator(s). This chapter shall detail a comparison of the simulated result to those of an existing simulator as well as to an actual portfolio that was invested in the market. Through these comparisons, an explicit measure of success will be determined.

The chapter shall also investigate the ability of the simulator to generate a portfolio whose return outperforms that of the industry benchmark. To facilitate this test, this chapter details a proprietary methodology to generate an investable portfolio, indirectly .

Sections 10.3, 10.4 and 10.5 describe the results of the evaluation criteria.

10.2 Alternate simulators

Before embarking on the evaluation of the first evaluation criteria, the comparison of the simulator results to those of alternate simulators and market-invested portfolios, it is first prudent to discuss the investigation and decisions regarding alternate simulators.

In testing the simulator accuracy, the scientific process would require a comparison of the simulator to related products. Whilst several products are available on the market, they fall into one of two categories; those offered for free which are limited in functionality and those offered for a fee, some of which offer the features discussed in this research.

As highlighted by O'Shaugnessy [OSha98], the FactSet Alpha product is the "*gold standard*" for portfolio backtesting. It's direct competitor would be Capital IQ ClariFI. With only product brochures available to compare feature sets, both these products manage the nuances of backtesting discussed in this research such as look-ahead bias, survivorship bias, multifactor modelling. What these products do not offer, being a non-standard request, is the ability to model hedge fund portfolios, which comprise of multiple strategies. This is a feature that the simulator detailed in this research does cater for. Further, the research simulator allows for the development of investment models based on any theory that can be programmed as a stored procedure. Both Alpha and ClariFI offer a limited ability to create investment models based on a template provided.

Comprehensive comparison to the fee-based simulators was not possible due to:

1. the cost of rental/purchase of these systems prohibited their use

2. freeware products did not accommodate backtesting beyond one year due to data collection, licensing and storage constraints

3. enterprise-capable hedge fund specific simulators are not freely available

4. the majority of the available simulators only satisfy the simulation feature of the simulator and not the stock-suggestion features

As such, the simulator was compared to its two most relevant competitors. The first competitor was the legacy simulator currently in use within the investment firm of the researcher (referred to as PreSim for the purposes of this research); the same system this research simulator was intended to replace should its results prove more accurate. The second competitor was the return series of an actual market-invested portfolio.

PreSim may be described as:

1. PreSim is an Excel spreadsheet-based simulator without the use of VBA – any business logic is incorporated within the spreadsheet with the use of conditional operators

2. PreSim does not enjoy the benefits of OO or the constructs of a high-level programming language or database engine

3. PreSim, without a back-end database, relies on retrieving data at runtime from a data vendor using an Excel Add-In

4. PreSim can only simulate a two-year historic backtest due to limited capacity for data storage

5. PreSim cannot avoid many of the nuances of backtesting such as survivorship bias or time-lag issues due to the complexities of implementing this logic and the lack of required data for these features

6. Presim has been in use for several years and has had its results verified on several occasions. Since it represents one acceptable solution for the investment firm, it inherently provides one benchmark of comparison for the research simulator to match or exceed in accuracy.

10.3 Resemblances to both an actual portfolio and to PreSim

In comparing the output of the simulator to that of an actual portfolio, one is essentially testing the realism and accuracy of the simulator logic. If the general mechanics of investing combined with the nuances of backtesting were accurately accounted for in the business logic of the simulator, one would expect that the simulated portfolio would bear significant resemblance to a portfolio of identical stocks that was actually invested on the stock market during the same time horizon.

The comparison of simulator results to PreSim results only become relevant if the alternate simulator has proven to reliably and accurately simulate a given portfolio of stocks. In section 10.2, the reader was made aware that PreSim has been in use and had its results verified by the investment firm, hence capable of serving as a benchmark for comparison. Emphasis was also placed on comparing the results of the simulator in this research to those of an actual market-invested portfolio for which accurate and detailed records exist.

In order to test the accuracy of the simulator, historically invested portfolios, as invested by the investment firm employing the researcher, were used to benchmark the results. The primary reason for using these particular portfolios was the accessibility to detailed daily transaction records for the fund. This is generally proprietary information for all investment firms and therefore inaccessible for public research. These daily transaction records of actual funds would serve to determine whether the simulator was accurately reflecting the

various line items involved in determining a portfolio's daily valuation and proved invaluable during the debugging phase in determining the discrepancies between the simulated and actual portfolios.

Whilst the scientific approach to testing would be the testing of the simulator against a statistically significant number of real-world portfolios, this was not an option available for the research. The reason lies in the fact that access to real-world portfolios was restricted to those of the investment firm of the researcher, a consequence of the proprietary nature of this data to an investment firm. The investment firm of the researcher, in its history, had a total of four historic portfolios. The fund managers, having full knowledge of this fact, agreed that approximately fifty successful tests had to be run between PreSim and the research simulator, the in-sample data, with the four real-world portfolios representing the out-of-sample data. As shall be explained in this section, at the end of the testing phase, in excess of one hundred simulator-simulator tests were performed successfully, which exceeded the mandate from the fund managers. Further, all testing against the real-world portfolios proved successful. The reader is made aware of the fact that a real-world portfolio, represents the most accurate, audited representation of a portfolio. The more correlated the simulator results are to the real-world portfolios, the more accurate one can deem the simulator to be. The fund managers agreed to the use of the real-world portfolios as out-of-sample data as each portfolio represented a sufficiently lengthy time horizon in excess of six months, which provided ample datapoints and characteristics which would test the simulator's implementation of fund accounting logic.

In order to perform the testing, the simulator was passed the identical portfolio of stocks on the same start date as the actual portfolio and allowed to simulate the portfolio during the exact holding period of the actual portfolio. PreSim was also passed the identical input information in order to generate its historic portfolio time series of returns.

In selecting an actual portfolio against which the simulators would be compared, it was important that the portfolio:

a. contain multiple currencies, and hence country exposures

b. multiple equity sectors

c. both equity and swap positions

These criteria were selected as they:

1. represent the types of portfolios used by the investment firm

2. are the scenarios for which this simulator will be used

3. records exist for portfolios which have these characteristics

4. the legacy simulator, PreSim, is capable of simulating portfolios of these characteristics

In understanding that a portfolio return profile is essentially a time-series of data, several statistical tools are available to test for the similarity when comparing two time-series. Examples of these include the Sharpe Ratio and Treynor Ratio and Information Ratio. However, as explained by Markellos [MM08] in *The Econometric Modelling of Financial Time Series*, covariance, and more specifically correlation, can be used to determine portfolio similarity. The former indicators such as the Sharpe Ratio, are not reflective of the time-series similarity in movement, but rather the distribution of data. For the purposes of this research, it was important to prove that the time series of the research portfolio and that of the market portfolio were of acceptable similarity. Further, the final return of the portfolio from its inception point, should also be of acceptable similarity.

In saying this, one needs to define acceptable similarity. The investment firm for which the research simulator was built, comprises of 12 fund managers. As a firm, all portfolio simulation tools have had a historic acceptable minimum of 0.7 correlation to an actual

portfolio. As such, Presim would also have adhered to the same restriction during its development and testing. In section 1.3, it was explained that an acceptable correlation differs from fund manager to fund manager and across investment firms and the industry in general. This is based on the emphasis the firm or manager places on the accuracy of the result in their portfolio decision making process. For the investment firm in question, the output of the simulator serves as one piece of guidance data amongst many that are used to make an investment decision. As such, the 0.7 correlation level was agreed for all products across the firm.

Comparing the research simulator results to the actual portfolio returns, the simplest measures of accuracy of business logic were:

a. correlation of daily portfolio returns as produced by the simulator, to those of the actual implemented portfolio and to PreSim

b. accuracy of the portfolio return between the start and end of the holding period, between the portfolios

c. Daily transaction audit

The initial results of the testing revealed an average correlation of 0.3 between the daily portfolio returns of the simulated portfolio and that of the actual portfolio with PreSim attaining a 0.78 correlation to the same portfolio. This correlation level was considered unacceptably low by the fund managers, as it provided minimal confidence in the explanatory statistics derived from the simulated time series. The primary stakeholder thereafter defined a minimum correlation level of 0.7 as acceptable for this research. Further investigation into the low correlation of 0.3 revealed differences between the portfolio returns where the business logic did not accurately implement several of the mechanics of chapters two and three such as:

a. the interest income methodology generated from the stock on loan

b. the ability of the fund manager to use long stock positions as collateral against short stock positions, rather than using cash

c. the difference in pricing when entering/exiting positions; the simulator only had access to closing prices for each stock and assumed these prices to be the entry and exit prices for the day

d. Swap cash pricing did not match those of the fund on the day of transaction due to the delay in prime broker calculations – the time lag rule was not applied appropriately

e. New forms of incorrect data within the database that had not been detected and cleaned; for which new correction methods had to be developed

In order to enhance the evaluation as well as verify the simulation accuracy, multiple portfolios of varying characteristics had to be simulated. In the absence of a scientifically designated number of portfolios, the portfolio managers of the research firm were asked to gauge, based on their experience, an acceptable number of simulations to verify the accuracy of the simulator. A decision was take to test a minimum of fifty portfolios. By the end of testing, in excess of 100 portfolio profiles were generated in both the research simulator and PreSim. With each evaluation and subsequent correlation test, investigation was done to determine differences in the simulations and changes were made accordingly, examples of which were described above. Testing multiple portfolios allowed iterative modifications to subsequently be made to the business logic of the simulator; eventually yielding an average correlation of 0.76 between the portfolio returns of the simulator to those of the actual portfolios invested in the market. Figure 10-1 illustrates the return profile of the NAV series for an actual portfolio versus that of the simulator. Exceeding the minimum number of fifty portfolios provided both peace of mind as well as enhanced the accuracy of the simulator through iterative improvements after each simulation.

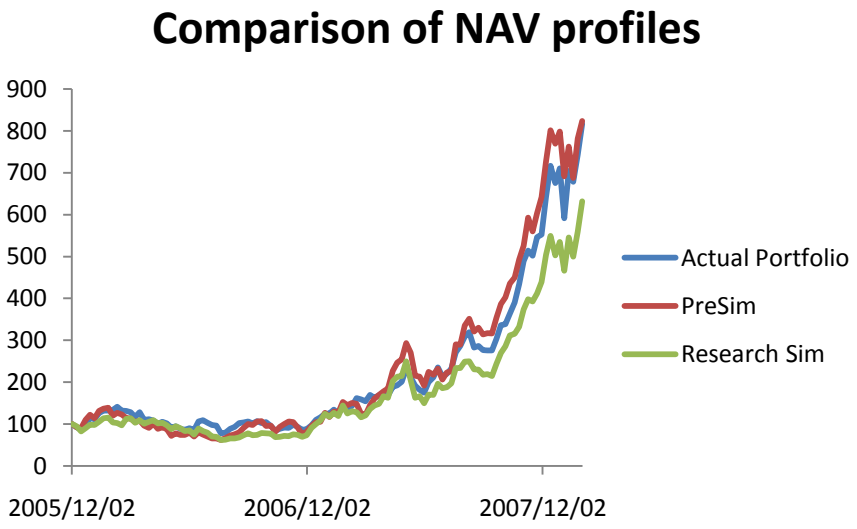


Fig 10-1: Performance profile of Actual NAV series vs Simulated NAV series

The correlation level of 0.76 was accepted by the fund managers, who also accepted reasons for the differences in the portfolio returns; differences which prevented higher correlations. The primary reasons for the difference in returns were attributed to:

a. the pricing data for the simulator being provided from a different data vendor to the one used for the actual portfolio pricing;

b. the simulator not identically resembling the entry/exit pricing which was based on intra-day pricing for the actual portfolios. The simulator only had access to closing prices as provided by the data vendor

c. Whilst the overall volumes transacted per stock matched those of the actual portfolio, the proportions purchased per day in order to achieve the total volume differed between the simulator and the actual portfolio. This was a result of different volume data due to different vendors.

d. The frequency and timing of portfolio hedging differed resulting in dissimilar outcomes

e. Buying and selling did occur in the actual portfolio during the period and since some of these actions were decided by the portfolio manager's interpretation of the stock at the time, the simulator, relying on trigger levels did not execute the same transactions

It must be noted that the correlation of PreSim was superior to that of the research simulator for one defining reason. PreSim was able to accommodate the exact portfolio rebalancing and trading activities, whilst the simulator rebalanced and traded the portfolio based on the agent logic. In simulating a portfolio against which one did not have an actual portfolio to compare with, PreSim would require the user to arbitrarily determine rebalancing and trading dates whereas the simulator would rely on its logic to achieve a return series.

The achievement of acceptable correlation between the simulated results and those of both an existing simulator as well as a market-invested portfolio have validated the success of the simulator. This success is enhanced through the ability to accurately explain the variance in these results as well as the ability to rule out stock volatility as a factor to increased correlation. In the case of this portfolio series, stock volatility exceeded 70%, thereby ensuring that the delta in daily profit and loss was large enough over time to not artificially increase the correlation statistic. In summary, research question one of section 1.3 "*Design and implement portfolio backtesting software which is able to accurately model a stock market portfolio*" has been successfully accomplished.

10.4 Simulator-recommended Portfolios – Case Study

The methodology described in this section is proprietary to the investment firm. The implementation of this methodology in the simulator was requested by the portfolio manager as mandatory for its use within the organisation.

Whilst the development of a simulator to accurately recreate a historic time series of returns for a given portfolio of stocks was a primary objective of this research, the distinguishing feature of this research simulator from alternate simulators is the capability of the design to recommend portfolios for consideration by a portfolio manager. This was achieved through the plug-in interface design at the stock selection level. Whilst one could provide a given portfolio to the simulator, one could also have built, for example, a random stock pick generator which would plug-in to the same interface and results in portfolio simulations of the random picks.

Random stock pick generators, however, are not the panacea for successful stock portfolios. Rather, as discussed in chapter eight, the design allowed the user to create varied investment models, liquidity filters and ranking/normalisation modules whose responsibility, given an investment universe, was to filter the universe, retain those stocks which met the investment criteria and select a subset of the retained stocks after being ranked based on a ranking heuristic.

Since the methodology to derive a portfolio differs from investor to investor, a direct comparison to alternate products is not possible. In fact, direct comparison to the PreSim tool within the same company was irrelevant, since each tool used different methodologies. The simplest method of determining the success of this portion of the research was to simulate the final recommended portfolio to determine whether it met the criteria of an investible portfolio as outlined by the portfolio manager; that is, high return with low volatility.

The objective of this section is to describe a methodology based on the premise that an investment model can be determined whose history of outperforming the industry benchmark should probabilistically outperform in the future. Given such a model, the simulator would be able to generate an investable portfolio as at the current date. This portfolio would be tested against the index performance at the end of the investment horizon in order to determine if a simulator-recommended portfolio can outperform the industry benchmark. Section 10.5 details the results of this test.

It must be noted that what is to be described is proprietary and as such, cannot be referenced to a particular academic citing. The techniques used in the methodology such as z-factor scoring and testing for multicollinearity and heteroscedasticity are statistical techniques which form the building blocks for the methodology and as such are not discussed in detail. The combination of the building blocks in this particular methodology is what makes the model proprietary. As explained in chapter five, an important consideration in the development of the simulator was the creation of a generic framework within which one could model an investment strategy without having to modify the actual simulator source code. This was achieved through the addition of a new stored procedure which housed the logic to represent this particular methodology, whilst implementing the standard input and output interfaces as defined by the simulator architecture. As a consequence, the logic for the model is encapsulated within a stored procedure and the simulator remains unaltered.

The example to be discussed in this section was based on the universe of 288 steel-related securities within the Resources sector.

For this example:

1. A single universe definition was created such that all tests were subject to the same set of stocks

2. For consistency, a single liquidity filter method and a single ranking/normalisation method were selected

3. Permutations of different investment models were created

The output of the testing would be a variety of tests representing variations of the different investment models developed by the user. These tests would then be ranked based on specific portfolio characteristics/statistics such as portfolio risk and estimated return.

780 permutations were generated off the variables in table 10-1; 390 tests going LONG only and 390 tests going SHORT only. Each of the 780 permutations tested a single investment factor. The reason for running separate long and short tests was to isolate the best performing individual factors for long and short tests. Once the individual factors were identified, they could then be combined in a multi-factor test scenario to determine if the combined factors enhanced the performance of the portfolio.

Test	Rolling Days	ExclNegSQLData	Monthly HP	Ascending
ClosingPE_on_StockPickDate	0	Yes	1	TRUE
P2CF_on_StockPickDate	30	No	3	FALSE
P2BV_on_StockPickDate	90		6	
MarketCAP_on_StockPickDate	180			
DebtOnCommonEquity_on_StockPickDate	356			
ClosingDY_on_StockPickDate	730			
ROA_on_StockPickDate	1100			
ROETotal_on_StockPickDate				
CurrentRatio_on_StockPickDate				
NetMargin_on_StockPickDate				
ClosingPE_on_StockPickDate_perChgFrom_Rolling_MIN				
ClosingPE_on_StockPickDate_perChgFrom_Rolling_AVG				
ClosingPE_on_StockPickDate_perChgFrom_Rolling_MAX				
P2CF_on_StockPickDate_perChgFrom_Rolling_MIN				
P2CF_on_StockPickDate_perChgFrom_Rolling_AVG				
P2CF_on_StockPickDate_perChgFrom_Rolling_MAX				
P2BV_on_StockPickDate_perChgFrom_Rolling_MIN				
P2BV_on_StockPickDate_perChgFrom_Rolling_AVG				
P2BV_on_StockPickDate_perChgFrom_Rolling_MAX				
ClosingDY_on_StockPickDate_perChgFrom_Rolling_AVG				
ClosingDY_on_StockPickDate_perChgFrom_Rolling_MIN				
ClosingDY_on_StockPickDate_perChgFrom_Rolling_MAX				
ROA_perChgFrom_Rolling_ROA				
ROETotal_perChgFrom_Rolling_ROETotal				
NetMargin_perChgFrom_Rolling_NetMargin				
CLO_perChgFrom_Rolling_AVG				
CLO_perChgFrom_Rolling_MIN				
CLO_perChgFrom_Rolling_MAX				
CLO_perChgFrom_Rolling_CLO				
CLO_Stdev_of_perChgFrom_Rolling_CLO				

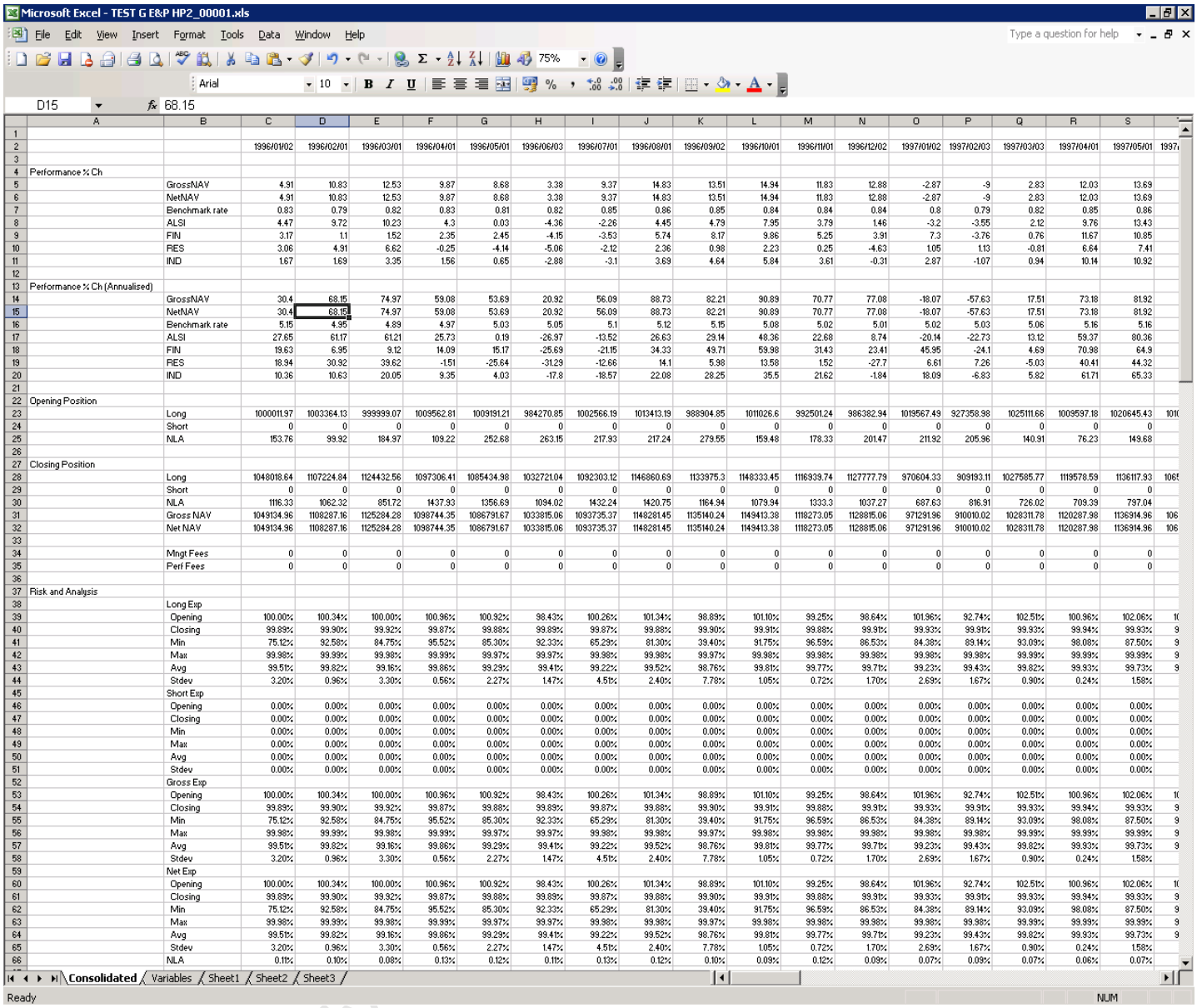
Table 10-1: Variables for test permutations

The creation of permutations and the generation of unique script files to describe each possible permutation of the variables in table 10-1 were achieved through the development of automation tools described in Appendix B. Several automation tools were created for different purposes; to generate permutations of script files, to collate results, to evaluate model performance and finally to consolidate best-performing portfolios to generate the final portfolio. These tools were essential due to the quantity of data generated.

For the purposes of this exercise, the portfolio manager had requested a 10-year backtest of the various permutations. As a result each permutation was tested with the start date of the portfolio being the start of each month for the 10 year period; resulting in 120 portfolios per permutation. Brown [Bro00] recommends this approach to backtesting with the creation of

portfolios with a moving monthly average in order to remove seasonality (regular peaks and troughs) within the data series. The result of 120 iterations or each of the 780 permutations resulted in 9360 portfolio return series, each with unique characteristics / statistics.

For each of the 780 permutations, a single consolidated file was created which listed various portfolio statistics for each of the 120 iterations, as illustrated in figure 10-4.



Microsoft Excel - TEST G E&P HP2_00001.xls

File Edit View Insert Format Tools Data Window Help

D15 68.15

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
2			1996/01/02	1996/02/01	1996/03/01	1996/04/01	1996/05/01	1996/06/03	1996/07/01	1996/08/01	1996/09/02	1996/10/01	1996/11/01	1996/12/02	1997/01/02	1997/02/03	1997/03/03	1997/04/01	1997/05/01
4	Performance % Ch																		
5		GrossNAV	4.91	10.83	12.53	9.87	8.68	3.38	9.37	14.83	13.51	14.94	11.83	12.88	-2.87	-9	2.83	12.03	13.69
6		NetNAV	4.91	10.83	12.53	9.87	8.68	3.38	9.37	14.83	13.51	14.94	11.83	12.88	-2.87	-9	2.83	12.03	13.69
7		Benchmark rate	0.83	0.79	0.82	0.83	0.81	0.82	0.85	0.86	0.85	0.84	0.84	0.84	0.8	0.79	0.82	0.85	0.86
8		ALSI	4.47	9.72	10.23	4.3	0.03	-4.36	-2.26	4.45	4.79	7.95	3.79	1.46	-3.2	-3.55	2.12	9.76	13.43
9		FIN	3.17	1.1	1.52	2.35	2.45	-4.15	-3.53	5.74	8.17	9.86	5.25	3.91	7.3	-3.76	0.76	11.67	10.85
10		RES	3.06	4.91	6.62	-0.25	-4.14	-5.06	-2.12	2.36	0.98	2.23	0.25	-4.63	1.05	1.13	-0.81	6.64	7.41
11		IND	1.67	1.69	3.35	1.56	0.65	-2.88	-3.1	3.69	4.64	5.84	3.61	-0.31	2.87	-1.07	0.94	10.14	10.92
13	Performance % Ch (Annualised)																		
14		GrossNAV	30.4	68.15	74.97	59.08	53.69	20.92	56.09	88.73	82.21	90.89	70.77	77.08	-18.07	-57.63	17.51	73.18	81.92
15		NetNAV	30.4	68.15	74.97	59.08	53.69	20.92	56.09	88.73	82.21	90.89	70.77	77.08	-18.07	-57.63	17.51	73.18	81.92
16		Benchmark rate	5.15	4.95	4.89	4.97	5.03	5.05	5.1	5.12	5.15	5.08	5.02	5.01	5.02	5.03	5.06	5.16	5.16
17		ALSI	27.65	61.17	61.21	25.73	0.19	-26.97	-13.52	26.63	29.14	48.36	22.68	8.74	-20.14	-22.73	13.12	59.37	80.36
18		FIN	19.63	6.95	9.12	14.09	15.17	-25.69	-21.15	34.33	49.71	59.98	31.43	23.41	45.95	-24.1	4.69	70.98	64.9
19		RES	18.94	30.92	39.62	-1.51	-25.64	-31.29	-12.66	14.1	5.98	13.58	1.52	-27.7	6.61	7.26	-5.03	40.41	44.32
20		IND	10.36	10.63	20.05	9.35	4.03	-17.8	-18.57	22.08	28.25	35.5	21.62	-1.84	18.09	-6.83	5.82	61.71	65.33
22	Opening Position																		
23		Long	1000011.97	1003364.13	999999.07	1009562.81	1009191.21	984270.85	1002566.19	1013413.19	988904.85	1011026.6	992501.24	986382.94	1019567.49	927358.98	1025111.66	1009597.18	1020645.43
24		Short	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25		NLA	153.76	99.92	184.97	109.22	252.68	263.15	217.93	217.24	279.55	159.48	178.33	201.47	211.92	205.96	140.91	76.23	149.68
27	Closing Position																		
28		Long	1048018.64	1107224.84	1124432.56	1097306.41	1085434.98	1032721.04	1092303.12	1146860.69	1133975.3	1148333.45	1116939.74	1127777.79	970604.33	909193.11	1027585.77	1119578.59	1136117.93
29		Short	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30		NLA	1116.33	1062.32	851.72	1437.93	1356.69	1094.02	1432.24	1420.75	1164.94	1079.94	1333.3	1037.27	687.63	816.91	726.02	709.39	797.04
31		Gross NAV	1049134.96	1108287.16	1125284.28	1098744.35	1086791.67	1033815.06	1093735.37	1148281.45	1135140.24	1149413.38	1118273.05	1128815.06	971291.96	910010.02	1028311.78	1120287.98	1136914.96
32		Net NAV	1049134.96	1108287.16	1125284.28	1098744.35	1086791.67	1033815.06	1093735.37	1148281.45	1135140.24	1149413.38	1118273.05	1128815.06	971291.96	910010.02	1028311.78	1120287.98	1136914.96
34		Mngt Fees	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35		Perf Fees	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	Risk and Analysis																		
38		Long Exp																	
39		Opening	100.00%	100.34%	100.00%	100.96%	100.92%	98.43%	100.26%	101.34%	98.89%	101.10%	99.25%	98.64%	101.96%	92.74%	102.51%	100.96%	102.06%
40		Closing	99.89%	99.90%	99.92%	99.87%	99.88%	99.89%	99.87%	99.88%	99.90%	99.91%	99.88%	99.91%	99.93%	99.91%	99.93%	99.94%	99.93%
41		Min	75.12%	92.58%	84.75%	95.52%	85.30%	92.33%	65.29%	81.30%	39.40%	91.75%	96.59%	86.53%	84.38%	89.14%	93.09%	98.08%	87.50%
42		Max	99.98%	99.99%	99.98%	99.99%	99.97%	99.97%	99.98%	99.98%	99.97%	99.98%	99.98%	99.98%	99.98%	99.98%	99.99%	99.99%	99.99%
43		Avg	99.51%	99.82%	99.16%	99.86%	99.29%	99.41%	99.22%	99.52%	98.76%	99.81%	99.77%	99.71%	99.23%	99.43%	99.82%	99.93%	99.73%
44		Stdev	3.20%	0.96%	3.30%	0.56%	2.27%	1.47%	4.51%	2.40%	7.78%	1.05%	0.72%	1.70%	2.69%	1.67%	0.90%	0.24%	1.58%
45		Short Exp																	
46		Opening	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
47		Closing	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
48		Min	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
49		Max	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
50		Avg	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
51		Stdev	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
52		Gross Exp																	
53		Opening	100.00%	100.34%	100.00%	100.96%	100.92%	98.43%	100.26%	101.34%	98.89%	101.10%	99.25%	98.64%	101.96%	92.74%	102.51%	100.96%	102.06%
54		Closing	99.89%	99.90%	99.92%	99.87%	99.88%	99.89%	99.87%	99.88%	99.90%	99.91%	99.88%	99.91%	99.93%	99.91%	99.93%	99.94%	99.93%
55		Min	75.12%	92.58%	84.75%	95.52%	85.30%	92.33%	65.29%	81.30%	39.40%	91.75%	96.59%	86.53%	84.38%	89.14%	93.09%	98.08%	87.50%
56		Max	99.98%	99.99%	99.98%	99.99%	99.97%	99.97%	99.98%	99.98%	99.97%	99.98%	99.98%	99.98%	99.98%	99.98%	99.99%	99.99%	99.99%
57		Avg	99.51%	99.82%	99.16%	99.86%	99.29%	99.41%	99.22%	99.52%	98.76%	99.81%	99.77%	99.71%	99.23%	99.43%	99.82%	99.93%	99.73%
58		Stdev	3.20%	0.96%	3.30%	0.56%	2.27%	1.47%	4.51%	2.40%	7.78%	1.05%	0.72%	1.70%	2.69%	1.67%	0.90%	0.24%	1.58%
59		Net Exp																	
60		Opening	100.00%	100.34%	100.00%	100.96%	100.92%	98.43%	100.26%	101.34%	98.89%	101.10%	99.25%	98.64%	101.96%	92.74%	102.51%	100.96%	102.06%
61		Closing	99.89%	99.90%	99.92%	99.87%	99.88%	99.89%	99.87%	99.88%	99.90%	99.91%	99.88%	99.91%	99.93%	99.91%	99.93%	99.94%	99.93%
62		Min	75.12%	92.58%	84.75%	95.52%	85.30%	92.33%	65.29%	81.30%	39.40%	91.75%	96.59%	86.53%	84.38%	89.14%	93.09%	98.08%	87.50%
63		Max	99.98%	99.99%	99.98%	99.99%	99.97%	99.97%	99.98%	99.98%	99.97%	99.98%	99.98%	99.98%	99.98%	99.98%	99.99%	99.99%	99.99%
64		Avg	99.51%	99.82%	99.16%	99.86%	99.29%	99.41%	99.22%	99.52%	98.76%	99.81%	99.77%	99.71%	99.23%	99.43%	99.82%	99.93%	99.73%
65		Stdev	3.20%	0.96%	3.30%	0.56%	2.27%	1.47%	4.51%	2.40%	7.78%	1.05%	0.72%	1.70%	2.69%	1.67%	0.90%	0.24%	1.58%
66		NLA	0.11%	0.10%	0.08%	0.13%	0.12%	0.11%	0.13%	0.12%	0.10%	0.09%	0.12%	0.09%	0.07%	0.09%	0.07%	0.06%	0.07%

Consolidated | Variables | Sheet1 | Sheet2 | Sheet3

Ready NUM

Fig 10-2: Consolidated portfolio statistics for 120 iterations of a single permutation

As per the description of the folder structure of the simulator in chapter seven, these consolidated files were stored in a single folder. A second automation tool, the Model Evaluation Framework (detailed in Appendix B), was developed to sweep through the plethora of resulting consolidated files and score each of the 120 iterations based on the average statistics housed in each of the consolidated files. Table 10-2 illustrates the weightings of each statistic in the overall scoring of the model. Negative weights are used to

penalise a model, for example, the higher the volatility of a model, the more it is penalised. These weightings were explicitly selected by the portfolio manager.

Statistic	Weight
Arithmetic mean return (monthly)	5.00
Geometric mean return (annualised)	-12.00
Arithmetic mean return in DOWN months	10.00
Stdev of differential to Index	15.00
Avg of differential to Index	30.00
Prob of Return > Index	1.00
Stdev of differential to Positive	-10.00
Avg of differential to Positive	15.00
Prob of Return > 0	5.00
+5% and above	20.00

Table 10-2: Weightings for model scoring

The result of the scoring of models is the ability to rank the models based on their investable status. At this point, the portfolio manager is presented with 390 ranked models for long portfolios and 390 ranked models for short portfolios, each representing the outcome of a single investment factor. The objective at this stage was the selection of the optimal, uncorrelated long and short factors and the combination of these factors into a multifactor model. As such, one of the features developed in the automation tool was the determination of uncorrelated tests from a ranked list using correlation matrixes as illustrated in table 10-3. In the example of table 10-3, factors in the *Factors* column of the table were ranked according to their score. Through definition by the portfolio manager, any factor which had a correlation of less than 0.7 to all other factors was to be considered as an uncorrelated, but highly ranked factor. As such, two long factors and five short factors were selected to create a single multi-factor portfolio.

Unique Test Name	Factor	ClosingPE_on_StockPickDate	ClosingPE_on_StockPickDate	P2BV_on_StockPickDate_perChgFrom_Rolling_MAX	P2BV_on_StockPickDate_perChgFrom_Rolling_MAX	CLO_perChgFrom_Rolling_AVG	CLO_perChgFrom_Rolling_MIN	CLO_perChgFrom_Rolling_CLO	CLO_Stdev_of_perChgFrom_Rolling_CLO	CLO_Stdev_of_perChgFrom_Rolling_CLO	CLO_Stdev_of_perChgFrom_Rolling_CLO
TEST Long_OSPD_Asc_ExclNeg_00002	ClosingPE_on_StockPickDate	1.00									
TEST Long_OSPD_Asc_ExclNeg_00003	ClosingPE_on_StockPickDate	0.98	1.00								
TEST Long_ROLLING_Asc_ExclNeg_00149	P2BV_on_StockPickDate_perChgFrom_Rolling_MAX	0.66	0.66	1.00							
TEST Long_ROLLING_Asc_ExclNeg_00155	P2BV_on_StockPickDate_perChgFrom_Rolling_MAX	0.63	0.64	0.94	1.00						
TEST Long_ROLLING_Desc_00112	CLO_perChgFrom_Rolling_AVG	0.66	0.66	0.79	0.81	1.00					
TEST Long_ROLLING_Desc_00130	CLO_perChgFrom_Rolling_MIN	0.70	0.72	0.77	0.79	0.90	1.00				
TEST Long_ROLLING_Desc_00166	CLO_perChgFrom_Rolling_CLO	0.72	0.71	0.79	0.79	0.92	0.93	1.00			
TEST Long_ROLLING_Desc_00186	CLO_Stdev_of_perChgFrom_Rolling_CLO	0.66	0.68	0.76	0.78	0.78	0.82	0.80	1.00		
TEST Long_ROLLING_Desc_00192	CLO_Stdev_of_perChgFrom_Rolling_CLO	0.69	0.71	0.77	0.81	0.81	0.85	0.83	0.95	1.00	
TEST Long_ROLLING_Desc_00198	CLO_Stdev_of_perChgFrom_Rolling_CLO	0.69	0.72	0.76	0.79	0.79	0.82	0.81	0.94	0.97	1.00

Unique Test Name	Factor	ClosingDY_on_StockPickDate	ClosingDY_on_StockPickDate	P2CF_on_StockPickDate_perChgFrom_Rolling_MAX	P2CF_on_StockPickDate_perChgFrom_Rolling_MAX	P2CF_on_StockPickDate_perChgFrom_Rolling_MAX	P2BV_on_StockPickDate_perChgFrom_Rolling_MIN	CLO_perChgFrom_Rolling_AVG	CLO_perChgFrom_Rolling_MAX	CLO_perChgFrom_Rolling_MAX	CLO_perChgFrom_Rolling_MAX
TEST Short_OSPD_Desc_00001	ClosingDY_on_StockPickDate	1.00									
TEST Short_OSPD_Desc_00002	ClosingDY_on_StockPickDate	0.92	1.00								
TEST Short_ROLLING_Asc_ExclNeg_00091	P2CF_on_StockPickDate_perChgFrom_Rolling_MAX	0.64	0.55	1.00							
TEST Short_ROLLING_Asc_ExclNeg_00103	P2CF_on_StockPickDate_perChgFrom_Rolling_MAX	0.64	0.54	0.87	1.00						
TEST Short_ROLLING_Asc_ExclNeg_00105	P2CF_on_StockPickDate_perChgFrom_Rolling_MAX	0.70	0.61	0.69	0.72	1.00					
TEST Short_ROLLING_Asc_ExclNeg_00109	P2BV_on_StockPickDate_perChgFrom_Rolling_MIN	0.66	0.59	0.93	0.87	0.70	1.00				
TEST Short_ROLLING_Desc_00125	CLO_perChgFrom_Rolling_AVG	0.65	0.62	0.64	0.73	0.60	0.65	1.00			
TEST Short_ROLLING_Desc_00147	CLO_perChgFrom_Rolling_MAX	0.73	0.69	0.73	0.66	0.68	0.73	0.67	1.00		
TEST Short_ROLLING_Desc_00148	CLO_perChgFrom_Rolling_MAX	0.69	0.64	0.72	0.66	0.71	0.73	0.67	0.93	1.00	
TEST Short_ROLLING_Desc_00152	CLO_perChgFrom_Rolling_MAX	0.67	0.67	0.71	0.66	0.70	0.74	0.64	0.86	0.85	1.00

Table 10-3: Long and Short factor correlation matrixes

The methodology was successfully implemented within the generic stored procedure

framework as described in chapter eight and further enhanced in Appendix B, Automation Tools. Using the simulator in this manner allowed for approximately 800 portfolios to be generated and profiled against each other using the criteria in Table 10.2. These criteria were determined by the fund managers in order to standardise the comparison and select the best performing portfolio. Section 10.5 represents the results of a portfolio which was then invested in the stock market against the South African All Share Index as the benchmark. As shall be explained in section 10.5, the recommended portfolio outperformed the index, thereby successfully accomplishing the second evaluation criteria.

10.5 Performance of a simulator-recommended portfolio

Whilst the constituents of the portfolio to be described in this section cannot be made public, the simulator was used successfully in 2005/2006 to recommend, backtest and optimise a portfolio of stocks which was then invested on the JSE. The criteria for the determination of the investable portfolio were mandated by the fund manager, the user of the simulator.

The methodology described in section 10.4 was used to create this portfolio of stocks. The premise of the methodology was to backtest, using the simulator, several years of investment portfolios, based on varying investment models, as selected by the investment manager. The objective (as defined by the fund manager) was to identify an investment model which outperformed the industry benchmark at least six out of the last ten years of simulation (60% probability of success). Using the methodology of 10.4, an investment model was eventually developed which achieved this metric. This model was then applied to stocks on the JSE at the start of 2004.

Figure 10-5 illustrates the return profile of the simulator-recommended portfolio as well as that of the industry benchmark (the All Share Index – ALSI) over a one year holding period. Table 10-4 reflects the portfolio statistics that both the goal-seeking agent and the model evaluation framework targeted and achieved, with the actual portfolio generating an 11.46%

return with an average monthly volatility of 1.32%. This is in contrast to the outperforming industry index which produced a 22.05% return during the same investment horizon.

This result, whilst validating the ability of the simulator and the methodology to select a portfolio whose return profile exceeds that of the cash rate, invalidated the theory that a simulator-recommended portfolio with a history of outperforming the industry benchmark, could be used to generate a portfolio that would exhibit the same characteristics when actually invested in the market. Such a result is not necessarily a failure of the simulator, but may also be attributed as a function of the market regime at the time. For example, a possible explanation may have been that the 60% outperformance years were bull market years and the year in which this test was performed may have been a bear market, so investment timing may have been a factor. Any number of possibilities exist to validate the notion that past performance is not necessarily an indication of future performance.

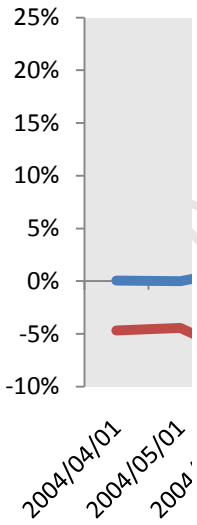


Fig 10-3: Fund return versus ALSI index return

Key Statistics			
Current month return	-1.28%	Percentage of positive months	69%
Year-to-date return	-1.05%	Average positive month return	1.45%
Return since inception (Mar 2004)	11.46%	Percentage of negative months	31%
Rolling 3 month return	-1.18%	Average negative month return	-0.51%
Rolling 6 month return	6.83%	Maximum monthly drawdown	-1.28%
Rolling 12 month return	11.02%	Maximum cumulative monthly drawdown	-1.28%
Number of months	13	Correlation in positive ALSI months	0.50
Average monthly return	0.85%	Correlation in negative ALSI months	-0.22
Volatility of Knight monthly returns	1.32%	Overall correlation	0.56

Table 10-4: Key Statistics of Invested Portfolio

In summary, research question two of section 1.3, "*Show that the software is able to recommend an investment portfolio that performs as well as, or better than portfolios generated from currently available techniques*" has been refuted on the basis of the results achieved. The reader should note that whilst the research question has been refuted based on the results achieved, these results are the consequence of implementing one particular methodology and does not categorically imply that no methodology exists which may result in a portfolio that does outperform the industry benchmark.

Chapter 11

Conclusion

Increased computational power over the past decades has unleashed a new generation of stock-picking algorithms. Investors that utilise these algorithms have the choice of assuming their reliability or backtesting the algorithms using historic data to investigate if it would have performed as profitably historically. The underlying thesis for such an investigation is that past performance may provide guidance on future performance.

The objective of this research was an investigation into the challenges of backtesting the investment portfolios generated by these algorithms. The result of the investigation suggested a software architecture to enable backtesting of historic investment portfolios, in particular, emphasizing the nuances and business logic variations in profiling a currently-invested portfolio to that of a historic portfolio. Chapter three described the additional considerations of backtesting such as look-ahead bias, survivorship bias and data timing which are not required when profiling a currently-invested portfolio.

The evaluation of the research was two-fold: first was the requirement to simulate a portfolio as realistically as possible to one that would be implemented on the stock market; second was to determine whether a simulator-recommended portfolio that has historically, consistently outperformed the industry benchmark, could achieve the same success into the future.

The research began with an investigation into the mechanics of investing in the stock market and an understanding of the various risk-reward measures that need to be considered when

managing an investment portfolio. These considerations would dictate the general logic of the simulator to be developed. These considerations were further enhanced through a thorough understanding of the nuances of backtesting as a methodology and the various techniques to counter these nuances. These techniques were incorporated into the simulator in order to satisfy its requirement to backtest historic portfolios.

In evaluating the success of the simulator, the simulated results were compared to that of a market-invested portfolio as well as an existing portfolio simulator. Following iterative improvements to the simulator, a correlation of 0.76 was achieved to a market-invested portfolio. In the absence of an industry-accepted simulation correlation level, and in understanding that each fund manager/investment firm determines correlation criteria based on their use of the simulations, this was determined to be an acceptable correlation level by the fund managers within the investment firm (the stakeholders) who deemed the simulation successful. Further, the variance in the correlation between these portfolios was quantified and the explanation accepted by the fund managers.

In answering the second research question relating to whether a simulator-recommended portfolio could outperform the industry benchmark, a methodology had to first be developed to construct such a portfolio. Section 10.6 described how this was achieved. The reason for the objective was to further enhance the value proposition of the simulator by extending the standard definition of a simulator to one which could also be used to generate a portfolio for simulation rather than simply requiring a portfolio as an input. The benefit of this approach is the breadth of flexibility it affords the fund manager in testing new investment ideas. The example methodology, whose criteria were set by the fund manager, required that the simulator be used to backtest a series of investment models over several historic periods in order to identify a model which outperformed the industry benchmark a minimum of 60% of the years in which it was implemented. That same model was then used by the simulator to determine a portfolio to be invested in the current stock market. This test occurred over a period of one-year. The result of the test was a simulator-recommended portfolio which, whilst historically outperforming the index, underperformed the index during the test period. The portfolio generated a return of 11.46% versus an index return of 22.05%. As

such, the hypothesis that a simulator-generated portfolio would outperform the industry benchmark failed. Whilst this particular portfolio may have failed to exceed the benchmark, the reasons for the failure lie not within the simulator, but rather with the investment model as explained in 10.5.

Both the research into backtesting and the suggested architecture for an agent-based simulator are presented in this research at their most elementary levels. As a consequence, there is significant future work that can be identified which may potentially improve the quality, accuracy and realism of simulations as well as its predictive power. This section highlights some of the key potential bodies of work which may be implemented. Simulation time has been highlighted throughout the research as a key design object for minimisation. A potential improvement of the current architecture would be the implementation of threading to accommodate multi-path processing. At present, the linear design insists that one round-trip simulation completes before another begins. In a multi-threaded environment, for example, the database logic for a second simulation could be invoked whilst the first simulation has completed this phase and is in the simulation engine phase. The current architecture is based off user-input in terms of model selection and uses brute force to determine optimal models through the simulation of all possible permutations of the selected models. Future work should concentrate on appending a neural network (NN) or genetic algorithms (GA) to the input of the simulation engine with the NN or GA determining the factors and weights based on the underlying data. Coupled with this development should be greater emphasis on positive feedback loops such that the system has more ability to improve its simulations through a better understanding of previous simulation results for similar factors. The use of intra-day data should also be considered a method of improving the predictive power of the engine. Coupled with this, greater emphasis should be placed on improving the stock buying and selling modules of the engine during the simulation window. Currently, the system relies on static price targets or threshold profits/losses in order to signal entry/exit scenarios. The current simulation process requires financial model evaluation at the start of the process in order to select the initial portfolio of stocks. This portfolio is then static through the investment period. An improvement would be the re-evaluation of the investment model periodically. This could determine if stocks in the

portfolio no longer meet the criteria and should be removed, or it could highlight new stocks that now meet the criteria and should be added to the portfolio. At the database layer, more time should be afforded to improving indexing, partitioning and execution plans as these ultimately affect simulation time.

In conclusion, this research has provided an understanding of the challenges and nuances to the backtesting of investment portfolios. It has proposed a simulator architecture, the prototype of which was used to generate results that were acceptably correlated to actual market-invested portfolios. The research further concluded that the simulator cannot be used to model future performance based on the consistency in simulation results of investment models during historic investment horizons.

References

"Standing on the shoulders of giants" - Anon

[MMV03a] Johan Marx, Raphael Mpofu and Gerhard van de Venter. Investment Management. Van Schaik Publishers. 2003. ISBN 0 627 02535 8 . Page 8.

[MMV03b] Johan Marx, Raphael Mpofu and Gerhard van de Venter. Investment Management. Van Schaik Publishers. 2003. ISBN 0 627 02535 8 . Page 8.

[Sta87] Meir Statman. How many stocks make up a diversified portfolio? Journal of Financial and Quantitative Analysis, Vol. 22, No.3, 353-363, September 1987

[KT91] Daniel Kahneman and Amos Tversky. Loss Aversion in Riskless Choice: A reference-dependent model. Quarterly Journal of Economics. 1991.

[Shi01] Robert J Shiller. Irrational Exuberance. Broadway Books. 2001.ISBN 0767907183

[OSha98] James P. O'Shaughnessy. What Works on Wall Street: A Guide to the Best Performing Investment Strategies of All Time. McGraw-Hills. 1998. ISBN 0070482462

[OSha98a] James P. O'Shaughnessy. What Works on Wall Street: A Guide to the Best Performing Investment Strategies of All Time. McGraw-Hills. 1998. Page 38. ISBN 0070482462

[JSE96] http://www.jse.co.za/history_trading.jsp

[Jol02] I.T. Jollifer. Principal Component Analysis. Springer. 2002. ISBN 0387954422

[Bla93] http://findarticles.com/p/articles/mi_m0SMG/is_n16_v13/ai_14635715

[Mor02] Geoff Morgan. Cleaning Financial Data. 2002. Originally published in the June/July 2002 issue of Financial Engineering News. www.fenews.com/fen26/morgan.html

[RN03] Stuart Russell, Peter Norvig. Artificial Intelligence, A modern approach 2nd Ed. Prentice Hall. 2003. ISBN 0130803022.

[SO02] Sarbanes-Oxley Act of 2002 - Public Company Accounting Reform and Investor Protection Act of 2002. http://www.sarbanes-oxley.com/

[Wil99] Jarrod W. Wilcox. Investing by the Numbers. John Wiley and Sons. 1999. ISBN 1883249546. Page 226.

[KB06] Miroslav Karny, Josef Bohm. Optimised Bayesian Dynamic Advising. Birkhauser. 2006. ISBN 1852339284. Page 104.

[Rit08] Colin Ritchie. Database Principles and Design. Cengage Lrng Business Press. 3rd Edition. 2008. ISBN 1844805402.

[MHH+08] J.D Meier, Alex Homer, David Hill *et al*. Application Architecture Guide 2.0 – Patterns and Practices. Microsoft. 2008

[SP99] GICS – Global Industry Classification Standard Online:http://www2.standardandpoors.com/spf/pdf/products/GICS_DIRM 2006.pdf

[RR05] Rickard Redler, Joachim Rossberg. Pro Scalable .NET 2.0 Application Designs. Apress. 2005. ISBN 1590595416. Page 382.

[Urd05] Timothy C. Urdan. Statistics in Plain English. Lawrence Erlbaum. 2005. ISBN 0805852417. Page 33.

[Bah99] Ali Bahrami. Object Oriented Systems Development using the Unified Modelling Language. McGraw-Hill. 1999. ISBN 0071160906. Page 6.

[Bro00] Jerry Brown. Seasonal Adjustment Methods. Minnesota Employment Review. January 2000.

[UCT03] Dept. Of Computer Science, University of Cape Town, Masters Coursework. Database Systems, Capturing Semantics. Unit 19.

[UCT03a] Dept. Of Computer Science, University of Cape Town, Masters Coursework. Database Systems, Problems with the OO Model. Unit 19.

[UCT03b] Dept. Of Computer Science, University of Cape Town, Masters Coursework. Database Systems, Fixed-length records and variable-length records in files. Unit 6.

[UCT03c] Dept. Of Computer Science, University of Cape Town, Masters Coursework. Database Systems, Performance measures of disks. Unit 6.

[UCT03d] Dept. Of Computer Science, University of Cape Town, Masters Coursework. Database Systems, Data Normalization. Unit 12.

[UCT03e] Dept. Of Computer Science, University of Cape Town, Masters Coursework. Database Systems, Over-Normalization. Unit 13.

[UCT03f] Dept. Of Computer Science, University of Cape Town, Masters Coursework. HCI, User Feedback. Unit 9.

[WBD01] Jeffrey L. Whitten, Lonnie D. Bentley, Kevin C. Dittman. Systems Analysis and Design Methods 5th Edition. McGraw-Hill. 2001. Page 173

[KT08] Alex Kriegel, Boris Trukhnov. SQL Bible 2nd Edition. John Wiley and Sons. 2008. Page 6. ISBN 0470229063.

[Mad02] Ananth Madhavan. Implementation of Hedge Fund Strategies. Atrader.com. 2002. Page 74. http://www.atrader.com/Implementation-of-Hedge-Fund-Strategies.pdf

[MS01] Microsoft Technet. How NTFS Works. 2003. http://technet.microsoft.com/en-us/library/cc781134(WS.10).aspx.

[For09] Gary Ford. Systems Trading for Spread Betting : An End-to-end Guid to Developing Spread Betting Systems. Harriman House. 2009. Page 121. ISBN 1905641737.

[CF98] T. Daniel Coggin, Frank J. Fabozzi. Applied Equity Valuation. Wiley. 1998. Page 64. ISBN 1883249511.

[Nar09] Rishi K. Narang. Inside the Black Box : The Simple Truth About Quantitative Trading. Wiley Finance. 1998. Page 120. ISBN 0470432063.

[GK04] Commodity Trading Advisors: Risk, Performance Analysis and Selection. Wiley Finance. 2004. Page 56. ISBN 0471681946.

[CK06] Quantitative Equity Portfolio Management: An Active Approach to Portfolio Construction and Management. McGraw-Hill. 2006. Page 543. ISBN 0071459391.

[BD98] Peter L. Bernstein, Aswath Damodaran. Investment management. Wiley Frontiers. 1998. Page 348. ISBN 0471197157.

[Dom02] Harry Domash. Fire Your Stock Analyst: Analyzing Stocks On Your Own. FT Press. 2002. Page 79. ISBN 0130353329.

[Sey09] Rudiger U. Seydel. Tools for Computational Finance. Springer. 2009. Page 96. ISBN 3540929282.

[Gla03] Paul Glasserman. Monte Carlo Methods in Financial Engineering (Stochastic Modelling and Applied Probability). Springer. 2003. Page 4. ISBN 0387004513.

[Ste02] Leigh Stevens. Essential Technical Analysis: Tools and Techniques to Spot Market Trends. Wiley. 2002. Page 4. ISBN 047115279X.

[Ach00] Steven Achelis. Technical Analysis from A to Z. McGraw Hill. 2000. Page 2. ISBN 0071363483.

[Sad96] Fahim Imam-Sadeque. Quantitiative Investment Techniques. 1996. http://www.actuaries.org.uk/__data/assets/pdf_file/0019/27154/investment.pdf

[Kar01] Orhan Karaali. An Overview of Investment Engineering Techniques. 2001. http://www.cfapubs.org/doi/pdf/10.2469/cp.v2001.n7.3154

[Vid04] Ganapathy Vidyamurthy. Pairs Trading: Quantitative Methods and Analysis. Wiley. 2004. Page 5. ISBN 0471460672.

[Bra97] Jeffrey M. Bradshaw. Software Agents. AAAI Press. 1997. ISBN 0262522349.

[Jae02] Robert Jaeger. All about Hedge Funds : The Easy Way to Get Started. McGraw-Hill. 2002. Page 81. ISBN 0071393935.

[RA91] F. James Rutherford, Andrew Ahlgren. Science for All Americans. Oxford University Press. 1991. Page 6. ISBN 0195067711.

[FFK06] Frank J. Fabozzi, Sergio M. Focardi, Petter N. Kolm. Financial Modeling of the Equity Market: From CAPM to Cointegration. Wiley. 2006. Page 426. ISBN 0471699004.

[Tor08] Richard Tortoriello. Quantitative Strategies for Achieving Alpha. McGraw-Hill. 2008. Page 19. ISBN 0071549846.

[KV08] Andrew Kumiega, Benjamin Van Vliet. Quality Money Management: Process Engineering and Best Practices for Systematic Trading and Investment. Academic Press. 2008. Page 149. ISBN 0123725496.

[KV08a] Andrew Kumiega, Benjamin Van Vliet. Quality Money Management: Process Engineering and Best Practices for Systematic Trading and Investment. Academic Press. 2008. Page 98. ISBN 0123725496.

[MM08] Terence Mills, Raphael N. Markellos. The Econometric Modelling of Financial Time Series. Cambridge University Press. 2008. ISBN 0521883814.

[BF98] Peter L. Bernstein, Frank J. Fabozzi. Streetwise. Princeton University Press. 1998. Page 21. ISBN 0691011281.

Appendix A

User Interface Layer

"A window into the soul" - Anon

A.1 Introduction

For the non-technical user of the software, the complexity and direct usage of the architecture at the database or business logic layer is daunting and unfeasible. For this reason, a user interface had to be developed, which embodied all the functionality offered by the simulator in a manner that was easy for the user to use without requiring training. This chapter illustrates and documents the rationale behind the various graphical interfaces developed for each of the functional areas of the simulator.

A.2 Primary Graphical Interfaces

The user interface is comprised of three primary graphical interfaces with secondary interfaces for additional functionality. These three interfaces map directly to the underlying simulator processes, that is, the collection of simulation parameters, the database query to receive the ranked universe and the daily simulation. Each of the three primary interfaces can be decomposed further, with the decomposition revealing functionality as well as performance-enhancing architectural decisions in the design and implementation.

Each of the three primary interfaces that will be decomposed in this chapter are:

1. Main Screen – setup market variables and view results of simulation
2. Backtesting Screen – setup backtesting variables and run investment model
3. Daily Simulation Screen – run daily simulation and view simulator logic and results

1. Main Screen

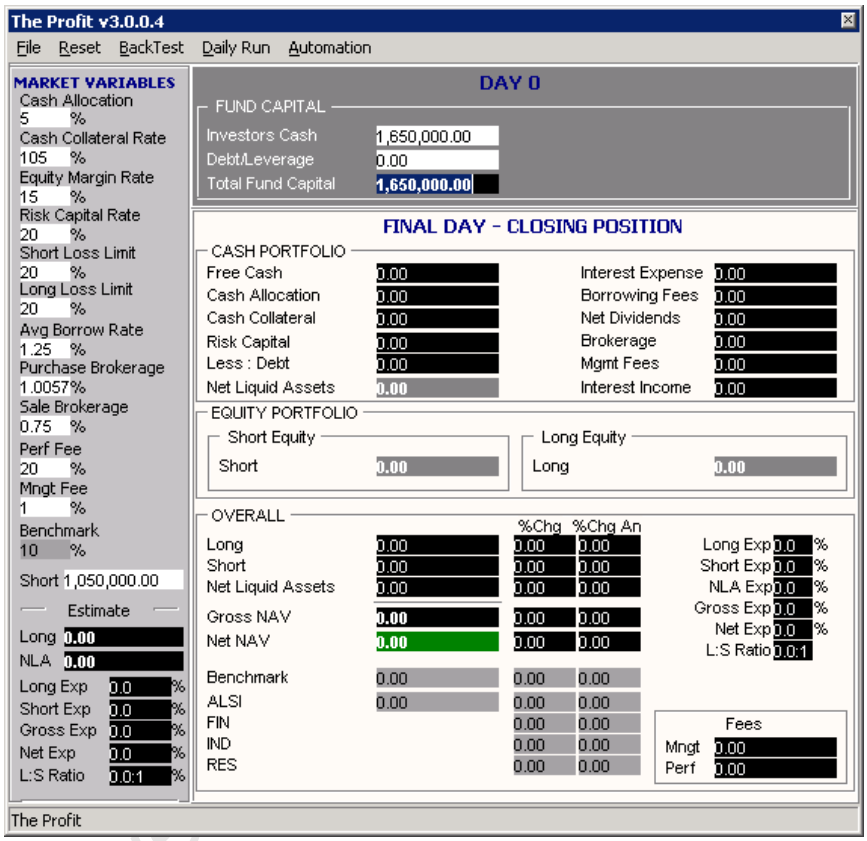


Fig A-1: Main UI

Upon executing the simulator, the user is presented with the central interface, or Main Screen as illustrated in figure A-1. The interface is divided into three frames, the Market Variables on the left, the Day 0 on the top and the Final Day – Closing Position at the bottom. This was the only interface in the first version of the simulator. The subsequent interfaces to be developed in this chapter evolved as a result of the user needing the ability to configure the parameters of the simulation and view the results within a windows interface.

On this interface, there are two types of text boxes, those in white and those in black. White textboxes indicate user-definable parameters whilst black textboxes indicate results of an action that has been performed. The Market Variables textboxes contain some of the parameters that reside within the Simulation Parameters script file as discussed in chapter seven. Whilst the simulator is configured with default values for all simulation parameters, upon loading an existing script file, all user interface components are refreshed with values that correspond to the parameters file.

Day 0 parameters indicate the starting investment value for the simulation. Final Day – Closing Position indicate the results of a single simulation. From this screen, the user had access to all the key performance indicators to determine the success of the simulation. In the event of a command-line simulation, this screen would not be refreshed with the output data. This was done to reduce the simulation time, which whilst initially thought immaterial, proved to be significant.

2. Backtesting Screen

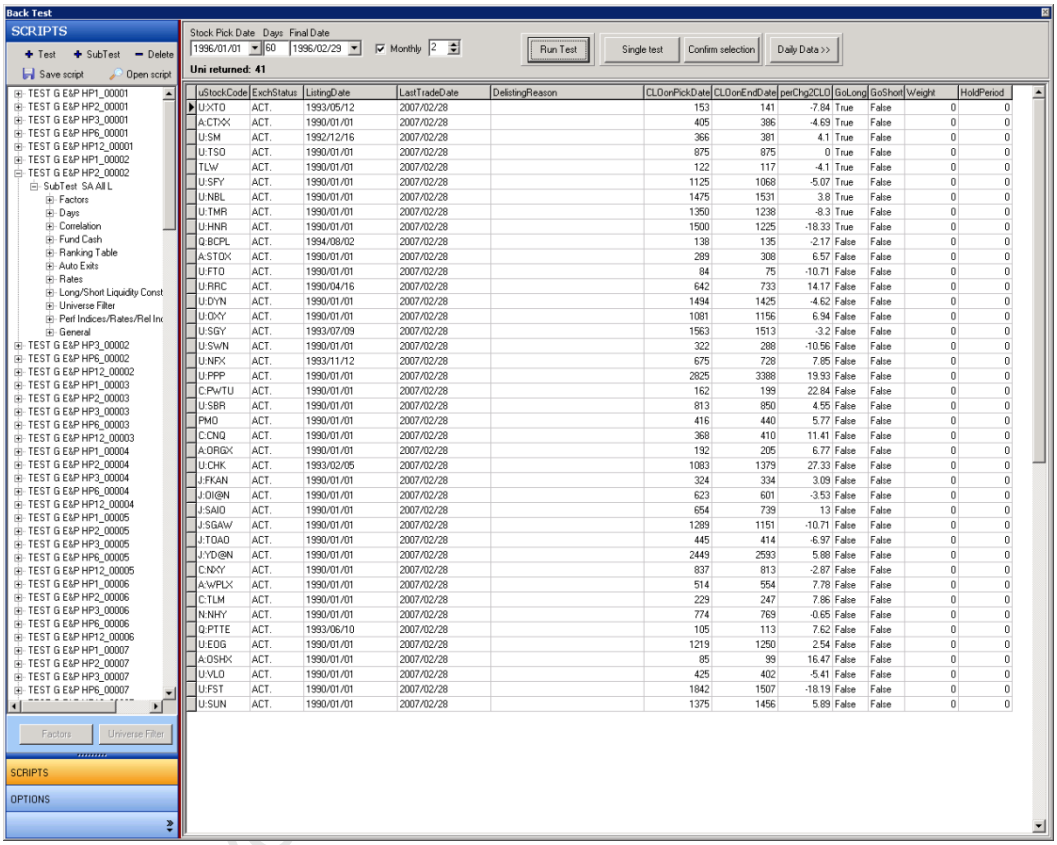


Fig A-2: Backtesting UI

The Backtesting screen, as illustrated in figure A-1 is divided into three frames. The left frame contains the framework for the parameter script (as described in chapter seven). It also houses the simulation parameter options that can be used in the script. The top frame contains the historic date range that the test will simulate. The central frame illustrates the results of the normalised and ranked universe that is returned from the database. In normal use, the user would generate the script file using the GUI, select the time period and click the RUN TEST button. This would result in the parameters and universe selection file being

transferred to the Master stored procedure in the database layer which runs the various stored procedures and returns the result set shown above. This is the universe that will then be simulated when the user clicks the DAILY DATA button. Note that the GoLong and GoShort columns indicate whether to buy or short the particular stock. A stock cannot have both columns set to true or both to false. Note also that only ACTive stocks have been returned which was the universe selection parameter selected in the chapter seven. As per the historic date range, this is a sixty day holding period with the stock universe indicator below the date range indicating 41 stocks that will be simulated.

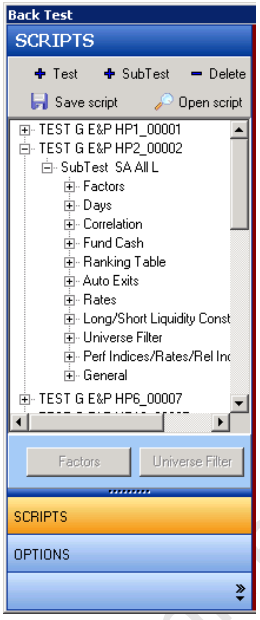


Fig A-3: Parameter and Scripts pane

This chapter shall focus on the left frame, illustrated in figure A-3. This frame, using the Outlook 2003 Navigation interface is multi-functional. The Scripts tab of the frame allows the user to create, edit and delete a script, whilst selecting the Options tab at the bottom reveals a range of configurable simulation parameters, all of which are saved to the simulation script.

Beginning with the SCRIPTS tab, the user is provided several options, creating a new test, a new subtest, deleting a script, saving a script or opening an existing script. In the example of figure A-3, an existing script has been opened. As can be seen, this script contains multiple tests, each of which will be iterated several times depending on the historic period being tested.

Upon selecting the + TEST button to create a new test, the user is presented with the dialog box illustrated in figure A-4:

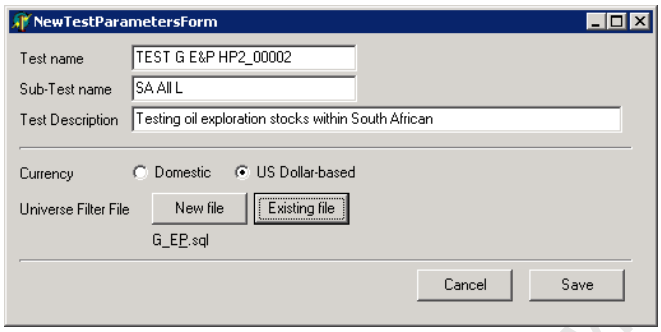


Fig A-4: Create a new test

Each test created must first be assigned a unique Test and SubTest name as well as the Universe Selection file to be used for this particular simulation. This Universe File is SQL Server compatible file containing the sql statement used to generate the desired universe, as explained in chapter seven. Upon successfully completing this dialog box, a test is created in the tree structure and pre-populated with the simulation parameter values as currently configured throughout the interface. As the user changes values on the Main Screen's Market Variables or the Options tab of the Outlook Navigation Bar, the script file will be refreshed to reflect these changes. The user also has the ability to change values directly within the tree structure by simply double-clicking the parameter to be reconfigured.

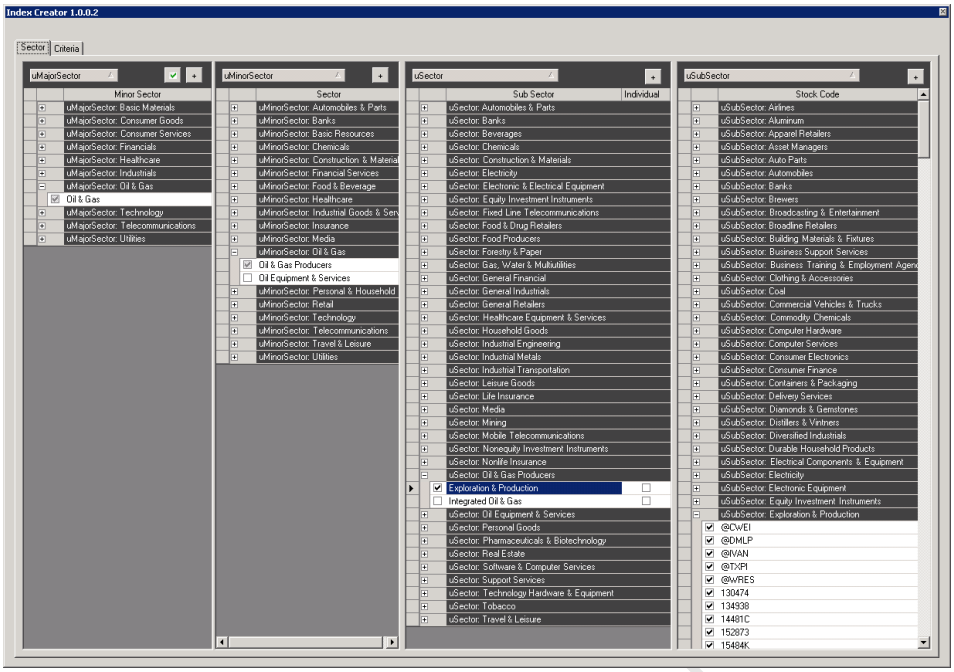


Fig A-5: Universe Selection Tool

The two disabled buttons on this tab of the interface, FACTORS and UNIVERSE FILTER are only enabled when the user specifically selects a SUBTEST of a TEST within the tree. Selecting UNIVERSE FILTER launches the IndexCreator utility which was discussed in chapter seven. Below is a reminder of the interface that is presented to the user. As explained, the result of this utility is a sql statement that is stored in a .sql file in the IndexDefinition folder. This is the file that was requested and pointed to in the CREATE NEW TEST dialog box.

Upon selecting the FACTORS button, the user is presented with the following dialog box:

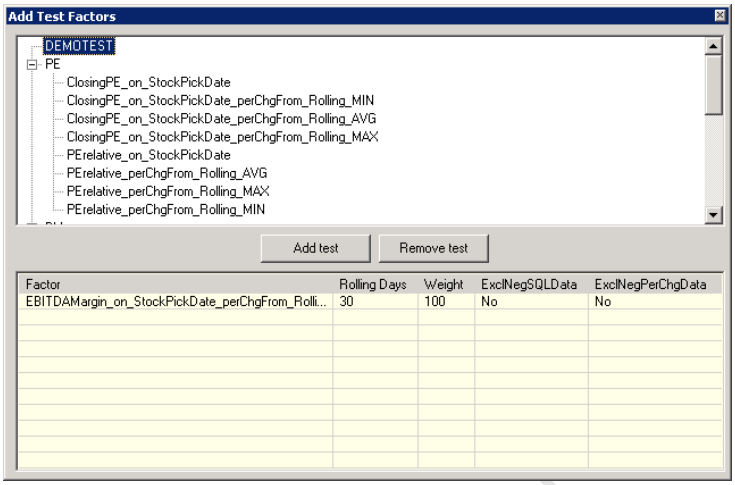


Fig A-6: Add Test Factors Dialog

The tree hierarchy groups a multitude of investment models according to the investment being tested. In this example, eight different investment models are available should the user choose to test the PE ratio as a stock selection methodology. Each investment model listed in the tree has a corresponding stored procedure that is invoked by the Master stored procedure. The name of the Investment model selected is saved as a parameter in the script that is passed to the database layer. The names of the Invesment Models displayed in the tree are read from a text file in the root of the SIM folder. The impact of adding a new model is reduced to creating a new stored procedure and adding the test name into this text file. This emphasises the extensibility of the system since none of the existing source code requires modification upon addition of a new test. Selling ADD TEST displays the following dialog box to configure parameters for this Investment Model:

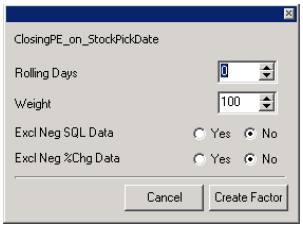


Fig A-7: Factor Parameters dialog

This interface allows the user to combine several investment models within a simulation, with each of the investment models having their own set of values for the factor parameters. As explained in chapter seven, the simplest method to transfer this data to the database layer is to create one variable for each of these parameters and populate it with comma-separated strings that can be decomposed at the database layer. In this example, the stored procedure would have the following highlighted variables, one for each of the parameters relating to the Investment Model.

```
ALTER   PROCEDURE [dbo].[BTC_EQUITY_xFACTORs_TEST_V2]
@StockPickDate datetime,
@EndDate datetime,
@UniFilterBy varchar(255),
@UniFilterStringParam1 varchar(255),
@UniFilterStringParam2 nvarchar(max),
@UniFilterIntParam1 int,
@UniFilterIntParam2 int,
@UniFilterNumericParam1 numeric(24,2),
@UniFilterNumericParam2 numeric(24,2),
@Ascending bit,
@InclTheseStocksOnly varchar(8000),
@TestNamesCSV varchar(8000),
@WeightsCSV varchar(8000),
@RollingDaysCSV varchar(8000),
@ExclNegSQLDataCSV varchar(8000),
@ExclNegPerChgDataCSV varchar(8000)
AS
```

Fig A-8: Orchestration Stored Procedure interface

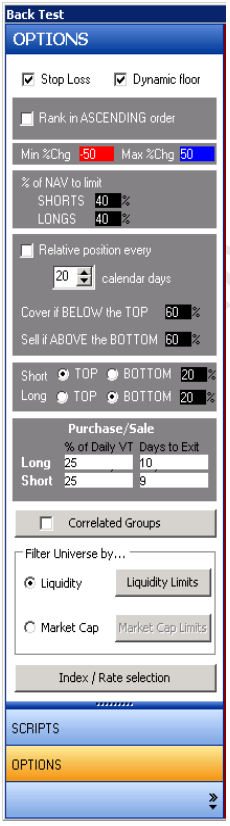


Fig A-9: Test Options

In selecting the Options tab of the navigation bar, the user is presented with the interface as illustrated on the left. This interface is segmented into various categories of options for varying functionality.

Whilst most of the interface options are self-describing and explanatory to the finance-oriented user, attention is drawn to the Liquidity and Index/Rate Selection functionality which offer additional configuration options.

A typical example of one function would be the *Filter Universe by...* group box. As explained earlier in this chapter, one of the primary database layer stored procedures is that of universe filtering with a separate stored procedure of each type of filter. Essentially, the user selects the

appropriate filter in this interface and configures the appropriate values for that filter by selecting the button associated with the filter. As new filters are developed the option is added at this interface layer and the appropriate stored procedure is created.

In clicking the Liquidity Filter button, the user is presented with the following dialog box:

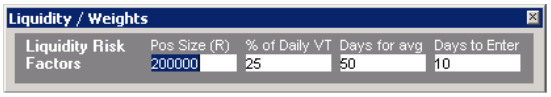


Fig A-10: Liquidity Filter parameters

Another interface from this Options tab is that of the Index/Rate Selection button as illustrated below:

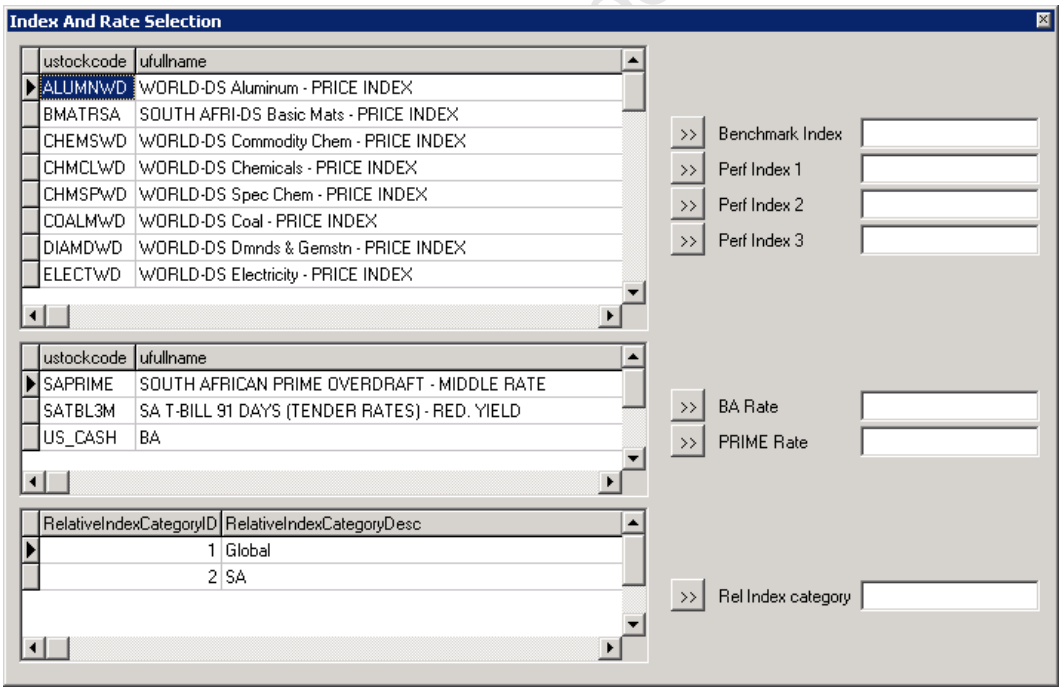


Fig A-11: Benchmarks selection dialog

The database contains global Rates and Indices. An example in the South African context would be the PRIME rate as offered by the banks and the ALL SHARE INDEX or ALSI on the JSE. This dialog box extends the extensibility of the software by allowing the user to select which RATE and INDEX to simulate with. Should new indices or rates be added to the database, there is no modification of software code at the interface layer as the above interface is database-driven.

3. Daily Simulation Screen

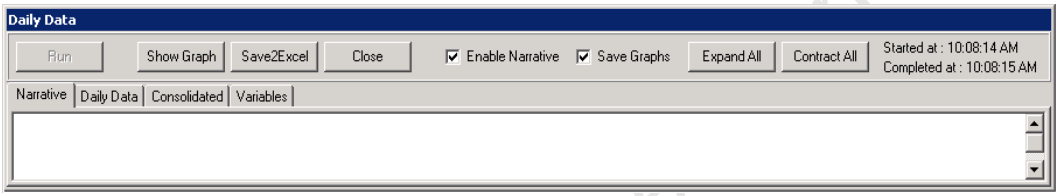


Fig A-12: Daily Simulation Screen

The Simulation screen is divided into two frames. The top frame has several buttons, the most important of which is the RUN button to initiate the simulation. At the top right is an indication of the time taken to complete the simulation, which in this case took one second to simulate 41 stocks over 60 days - as selected earlier in this chapter. The other buttons on the top frame allow the user to enable/disable and view certain features like the graphing component or the narrative component. These are variable and by default unchecked in order to further reduce simulation time.

The central frame is a tabbed environment with various simulation outputs. The Narrative tab provides a detailed breakdown of 95% of all logic decisions made by the simulator for each stock for each day. This option is essential in the post-mortem of the results in order to sanity check the outcome. The Daily Data tab provides a transaction-level breakdown of the portfolio on a daily basis as well as detailed statistics of the overall portfolio performance once the simulation has completed. This tab is saved generate the CSV file for the simulation. The Consolidated tab extracts the required statistics from the Daily Data tab into a new column for this iteration of the simulation. Whilst the Narrative and DailyData tabs are

cleared at the start of each simulation, the Consolidated tab retains the previous information to eventually contain a consolidated view of all the statistics for all the months tested in this particular simulation. The Variables tab is a data dump of the script file for reference purposes. Each of these tabs shall be illustrated in this section.

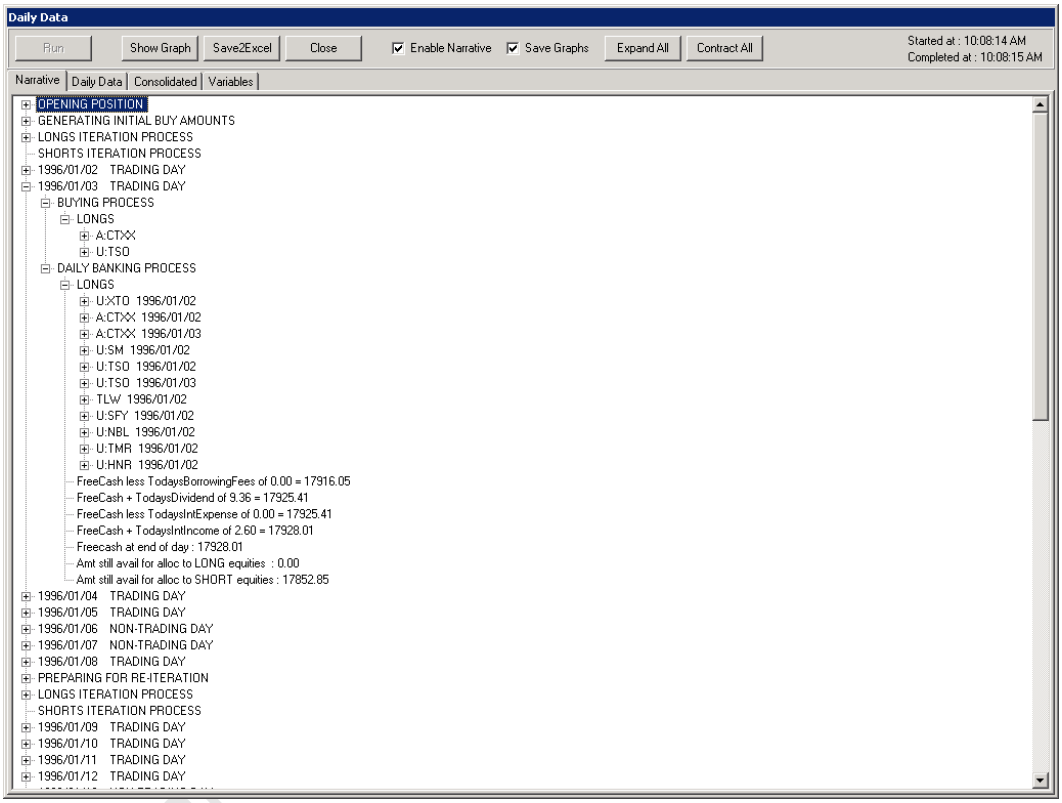


Fig A-13: Narrative tab

The Narrative tab, as illustrated above provides the detailed logic flow of the simulator on a per stock per day basis. Tracing this data provides a useful method to test the logic of the simulator as well as to query the results of a simulation. Every decision made by the simulator is documented in this narrative which, if printed for a 30-day simulation is several thousand pages long as it details every *if..then* programming block and every variable change throughout the simulation environment.

The DailyData tab is illustrated below. This tab, which is saved to CSV at the end of the simulation, groups the daily transaction data and the statistics separately.

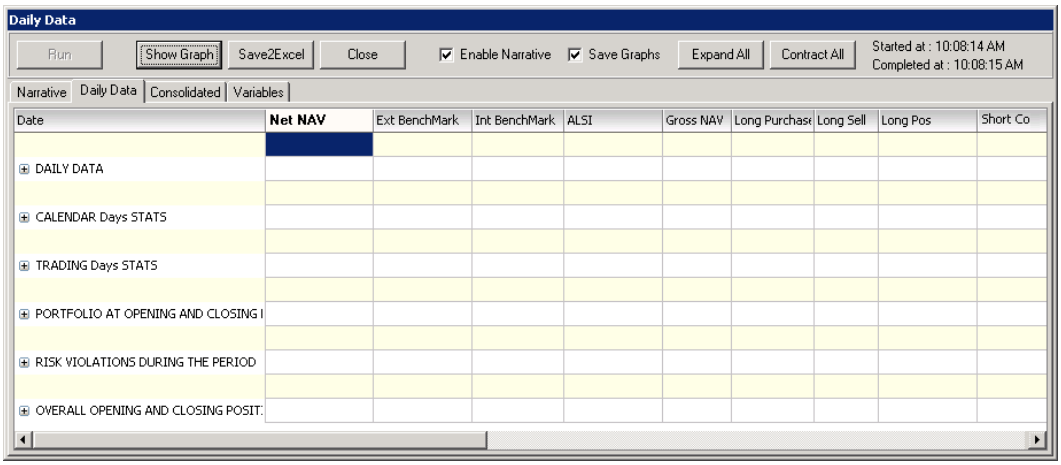


Fig A-14: Daily Data tab

Expanding the DAILY DATA group reveals the following transactional history of the portfolio:

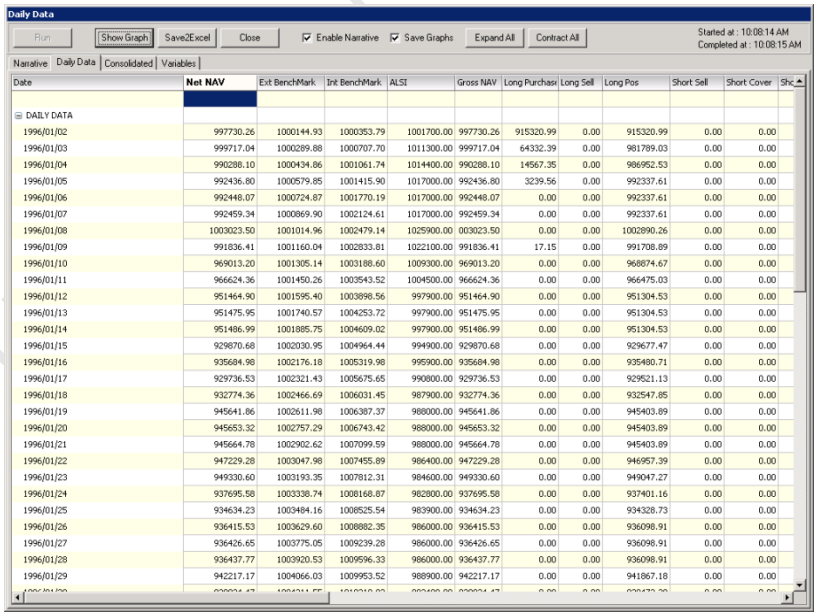
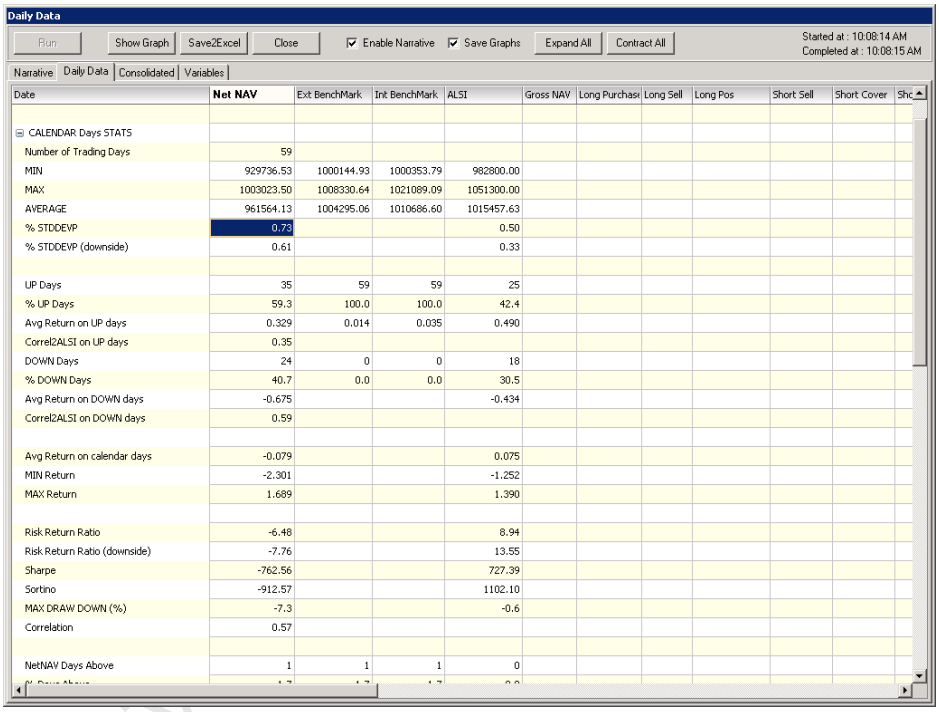


Fig A-15: Daily Data group expanded

Expanding the Calendar Days Stats group reveals statistics that have been calculated from the Daily Data transaction history. Calendar Day Stats indicate that the statistics have been calculated from start day to end day including weekends and non-trading days, essentially every row of the transactional table. Trading Day Stats indicate statistics that have been calculated only on row data that represent valid trading days, and excludes weekends and non-trading days.



Date	Net NAV	Ext BenchMark	Int BenchMark	ALSI	Gross NAV	Long Purchas	Long Sell	Long Pos	Short Sell	Short Cover	Sho
CALENDAR Days STATS											
Number of Trading Days	59										
MIN	929736.53	1000144.93	1000353.79	982800.00							
MAX	1003023.50	1008330.64	1021089.09	1051300.00							
AVERAGE	961564.13	1004295.06	1010686.60	1015457.63							
% STDDEVP	0.73			0.50							
% STDDEVP (downside)	0.61			0.33							
UP Days	35	59	59	25							
% UP Days	59.3	100.0	100.0	42.4							
Avg Return on UP days	0.329	0.014	0.035	0.490							
Correl2ALSI on UP days	0.35										
DOWN Days	24	0	0	18							
% DOWN Days	40.7	0.0	0.0	30.5							
Avg Return on DOWN days	-0.675			-0.434							
Correl2ALSI on DOWN days	0.59										
Avg Return on calendar days	-0.079			0.075							
MIN Return	-2.301			-1.252							
MAX Return	1.689			1.390							
Risk Return Ratio	-6.48			8.94							
Risk Return Ratio (downside)	-7.76			13.55							
Sharpe	-762.56			727.39							
Sortino	-912.57			1102.10							
MAX DRAW DOWN (%)	-7.3			-0.6							
Correlation	0.57										
NetNAV Days Above	1	1	1	0							

Fig A-16: Daily Data Statistics

Below is an example of the graphing that is generated by the simulator to reflect the performance of the portfolio (black line) to that of the market (blue line) and the bank (grey line). In this example, the portfolio simulated proved unsuccessful when compared to the market.

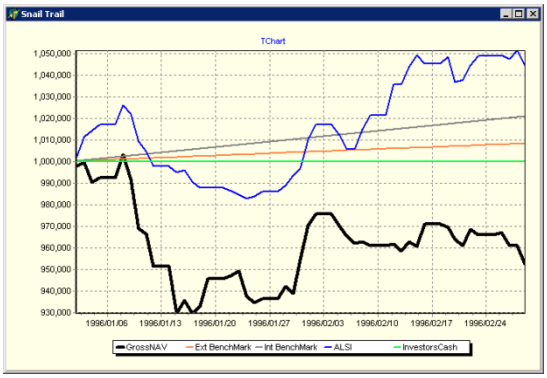


Fig A-17: Graph of portfolio and benchmark return profiles

Appendix B

Automation

"Automation applied to an efficient process will magnify the efficiency…" - Bill Gates

B.1 Introduction

As explained in chapter five, one of the design objectives of the simulator was speed of simulation. The section, Simulator Timing, reiterated the focus on response time. The problem that arises however, is that if one were to run 1000 iterations of the same simulation script, then one cannot do so manually. As a consequence, and in order to test the development, an automation tool had to be developed. This chapter details the various automation tools that aid the simulation process.

B.2 Automation Tools

Below is the interface of the Test Automater. It's purpose is to iterate through each script file in the SCRIPTS folder and for each script file, execute the script on a monthly basis for 11 years (130 iterations). This particular automation had 195 scripts, each running 130 times – a total running time of 35 hours at 5 seconds per test.

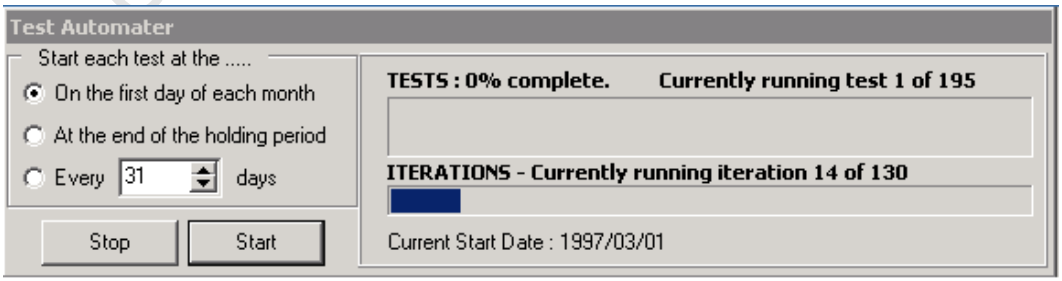


Fig B-1: Automation progress UI

Via the interface, the user has the ability to specify whether each iteration of a test should start at the beginning of the next month in the sequence, every *x* days or at the end of the holding period of the last iteration.

Automating the simulation process was the first step towards several other automation initiatives. The next was the automatic generation of the script files. The premise behind this utility was that every combination of a particular group of variables should be tested. The user was required to create a script marking each of the variables that needed to be permutated with a flag XXX. For each of these flags, the user was also required to generate an Excel spreadsheet indicating the various values that this flag could be configured to.

Consider the Excel implementation below which results in 12 possible permutations.

Test	Days	Sector
T1	30	Gold
T2	60	Diamonds
	90	

Table B-1: Potential values per variable

The script automation tool, (fig B-2), would compute the number of permutations of this file, and generate, in-memory the various combinations. For this example, it would be:

Test	Days	Sector
T1	30	Gold
T1	30	Diamonds
T1	60	Gold
T1	60	Diamonds
T1	90	Gold
T1	90	Diamonds
T2	30	Gold
T2	30	Diamonds
T2	60	Gold
T2	60	Diamonds
T2	90	Gold
T2	90	Diamonds

Table B-2: Resulting permutations

For each of these combinations, the generic script file is duplicated and the TEST, DAYS and SECTOR variables within the parameter script are modified. The benefit of this is that the fund manager need only specific a single Excel file with the various values per parameter and not have to spend several hours manually iterating through scripts to "Find And Replace" values. The following interface was developed for this purpose:

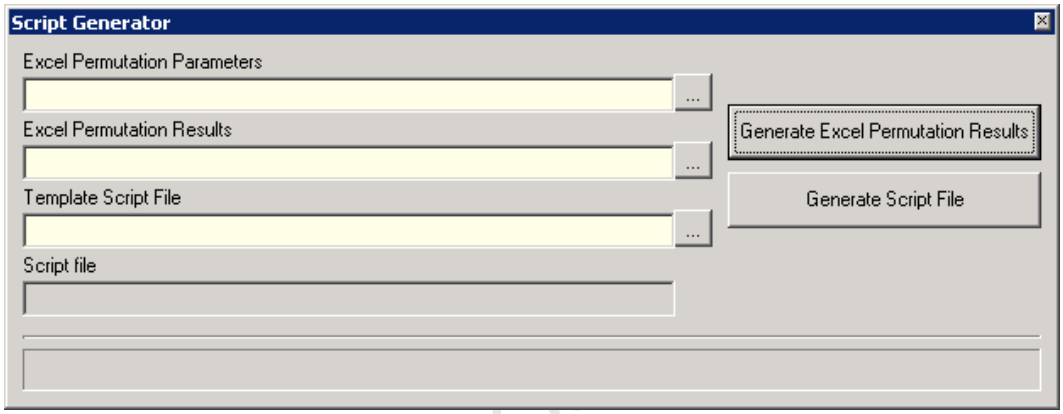


Fig B-2: Permutation generator UI

The user is required to indicate where the Excel file with the various permutations resides. By then clicking the Generate Excel Permutation Results button, the second Excel file is generated with each row representing a separate permutation. The location of this file is inserted into the Excel Permutation Results textbox. The reason this textbox is editable, is in the event that the file already exists and the user wishes to skip this first step. The third text box requires the user to input the location of the generic script file with the various parameters flagged. By clicking the Generate Script File button, the script file is duplicated as many times as there are permutation rows in the Excel spreadsheet and the variables are accordingly adjusted in each.

At the stage, the following can be summarised, the user need only create a single script and then automate the process for several scripts to be generated. The user then utilises the Test Automator to simulate each of these scripts *x* number of times with the results being written into CSV and Excel format in the CSV FILES and CONSOLIDATED folders. Consider now the

quantity of result files. For the example above of 195 scripts or permutations each doing 130 iterations, the number of CSV files is approximately 25,350 that get generated – each CSV file representing the simulation if started at a different period in time. One now needs to create a time series of this data. Again, whilst it is possible to manually open each CSV file and extract the necessary data to compile an overview of the simulation, it was decided that the process was suited to automation. As such, the following interface was developed to automate this process:

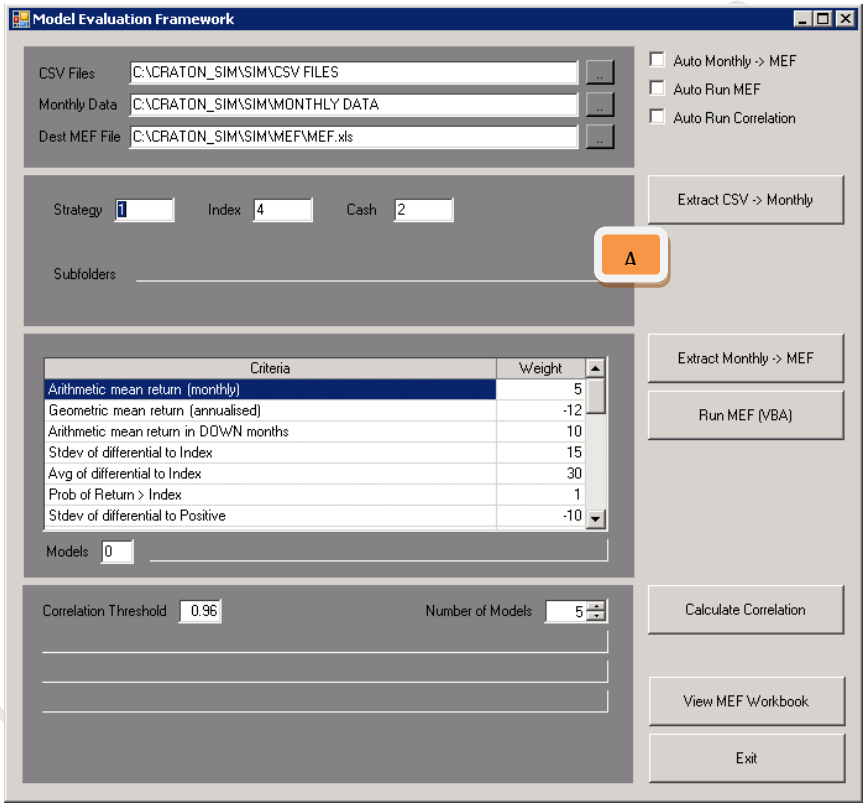


Fig B-3: Model Evaluation Framework UI

Having pointed the interface to the location of the CSV files, the Extract CSV → Monthly button opens each CSV file in turn and extracts the relevant data to generate a time series of

the performance of the simulator. These times series are saved in the MONTHLY DATA folder as indicated by the second textbox on the interface.

The process of extracting this data from 25350 CSV files takes approximately 5 seconds to complete, making the automation invaluable when compared to the time taken to previously manually generate the same result. The rest of the interface deals with the automation of the MONTHLY files to the proprietary Model Evaluation Framework which determines which of the simulation permutations performed the best. This is based on a weighted scoring system. Unfortunately this cannot be described further in this thesis as it remains proprietary.

Appendix C

Database Normalisation

"Everything should be made as simple as possible, but not simpler" – Albert Einstein

C.1 The path to BCNF

As explained earlier, varying degrees of normalisation may be applied to data. Each degree of normalisation is an extension of its predecessor and can only be effected if the normalisation predecessor has been executed without exception. Database theory, as explained in the coursework for this degree [UCT03d] and illustrated through example by Richie [Rit08] defines the various forms of normalisation as First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF) and an enhanced version of 3NF called Boyce-Codd Normal Form (BCNF). Whilst there does exist three additional Normal Forms, this research shall be restricted to BCNF. These various normalisation forms shall be explained by way of examples using the data relevant to this research.

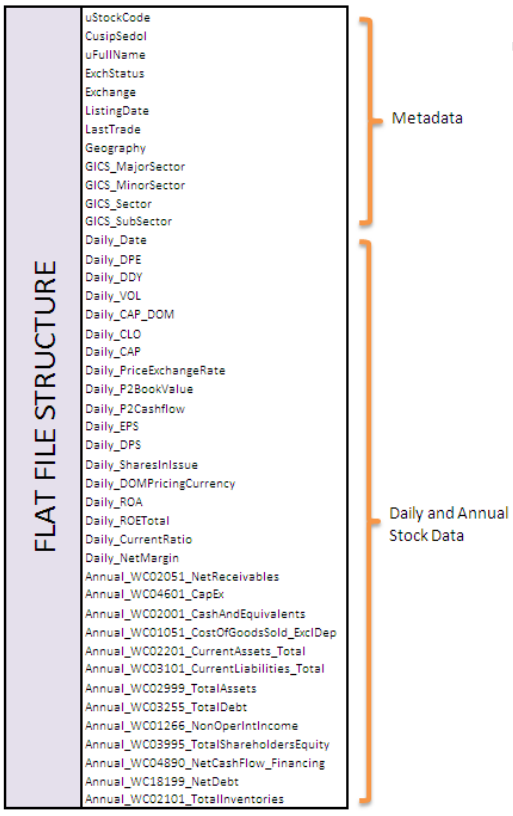


Fig C-1: Original "Flat file" representation of the data prior to normalisation

The starting point for the normalisation process begins with the raw form of the

data, as illustrated in figure C-1; which provides all the column headings for which multiple rows of data exist. For the purposes of illustration, the fields have been grouped together to represent the frequency of data stored in the respective fields (static, daily, annual). This grouping of data is the first step towards normalisation.

Figure C-1 represents a "flat file" where the data for each row is filled in, regardless of whether it has occurred previously. For example, the field *Daily_Date* indicates that a specific row represents data for a specific date. This means that each day's price for a single stock is stored in a new row. However, whilst each row may have a new unique price, each row contains the same metadata; that is; each row repeats the name of the stock, its country of listing and its sectors amongst others. The annual data fields are populated on rows that represent the annual dates on which the data became available. All rows between these release dates remain empty.

The two most evident issues with the data are:

- Data redundancy

- Inefficient allocation of space by storing rows of empty data for which space has been allocated through the use of fixed-length data types

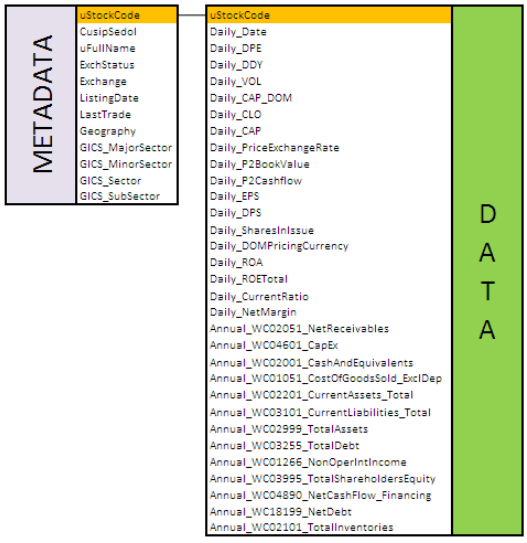


Fig C-2: First Normal Form

First Normal Form calls for the elimination of repeating groups of data through the creation of separate tables of related data; essentially the removal of **columns** of related, but repeating data. From figure C-1, it is evident that the repeating group of data are those fields within the METADATA category. In order to achieve 1NF, the fields within the METADATA category will be moved to a separate table and linked to

figure C-1 through a primary key representing the stock.

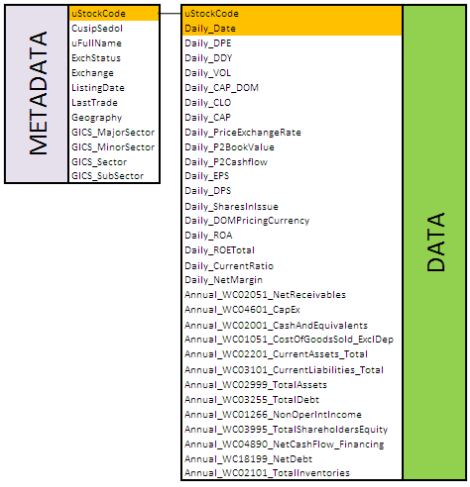
Whilst splitting the data into two tables was necessary to achieve 1NF, joining the table on a single primary key exposed an unintended side-effect. *uStockCode*, the field representing the primary key, would be repeated for every stock for every row of data. This repetition of the primary key violates the premise of primary keys being unique row descriptors.

Two solutions were available to solve this problem:

1. Create a unique incremental row identifier and link this to the METADATA table; an example of which may be an auto-generated incrementing integer; or

2. Determine whether the combination of existing fields could be used as a composite primary key

The disadvantage of option 1 is that is does not enforce data integrity constraints on the data in the table. For example, whilst two rows may have unique auto-generated primary key identifiers, both rows could now legitimately contain the same stock code on the same date with different prices; one would not be able to determine the correct price on that date as the two rows contained conflicting information.



In considering option 2, it was evident that a combination of stock code and date would uniquely identify each row of data such that there should never be another row containing the same combination of values.

Fig C-3: Defining a composite key from existing fields

In so doing, three benefits should be noted:

1. a composite primary key was created out of the existing fields, rather than increasing the quantity of the data by inserting a new auto-generated field

2. data integrity can be monitored/maintained by the database engine

3. primary keys, by default, are indexed fields, which means that querying this table as well as joining this table to the METADATA table during querying, will ultimately be more responsive and contributes to the design objective of millisecond response time

As explained in the introduction of normalisation in this chapter, one cannot proceed to a high form of normalisation until the current form of normalisation is complete. In this example, 1NF has not yet been achieved because of the existence of the annual data. Again, the database structure is better served by splitting the data table into DAILYDATA and ANNUALDATA tables that are both linked to the METADATA table. In similar fashion to the DAILYDATA table, the ANNUALDATA table will also be indexed on the composite primary keys of stock code and date which uniquely describe each row. The result is illustrated in figure C-4.

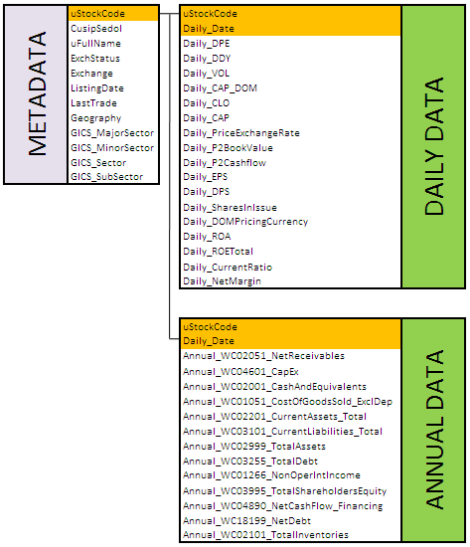


Fig C-4: Splitting Daily and Annual data into separate tables

One major advantage of splitting apart the ANNUALDATA fields, is that they were only populated on every 1/365 rows per stock; that is, one row per year. The rest of the rows remained empty and this impacted the size of the database as the fields were of fixed length.

In keeping with the theme of removing repetitive data, Second Normal Form goes one step further, by identifying and removing repeating **rows** of data, which are then separated out into new tables and joined through foreign keys. This is similar to the First Normal Form which removed **columns** of repeating data.

From figure C-4, it is evident that the only table which now comprises rows of repeating data would be the METADATA table. Each row of the DAILYDATA and ANNUALDATA tables contains specific data pertinent to a particular stock on a particular date. The METADATA table however, contains fields such as *Exchange*, for example the *New York Stock Exchange*, and this value is repeated in all rows for stocks listed on this exchange. A more concise description of the data would be to create a separate table EXCHANGE, which contains one row for each unique exchange name, and associate this table to the METADATA table via a

foreign key. The exchange name, geography names and GICS sector classifications are the three types of repeating data in the METADATA table. Figure C-5 illustrates how Second Normal Form was implemented to remove these repeating rows from the table, into separate tables.

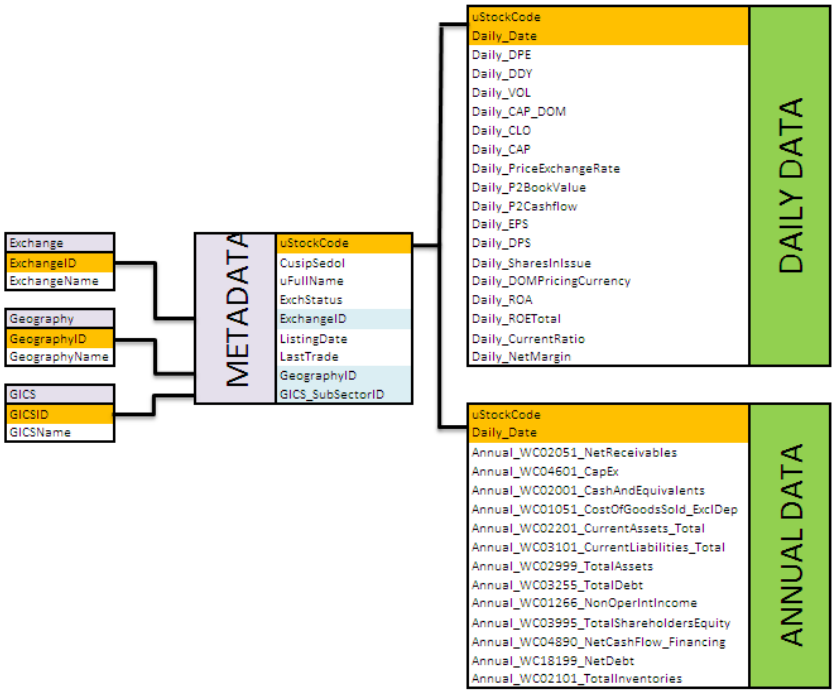


Fig C-5: Application of Second Normal Form

Before proceeding to Third Normal Form, it is worth noting, that an additional identifier was created in each of the EXCHANGE, GEOGRAPHY and GICS tables which was an auto-generated integer. One could also have simply used the ExchangeName or GeographyName field as primary key as these were unique fields. The benefit of this approach of using an auto-generated number as the primary key, is that the METADATA table will essentially be storing integer values in place of character values, which results in significant table storage reductions.

Also noteworthy, was the decision to store only the lowest level GICS identifier rather than all four GICS levels for a particular stock. This decision was taken since the GICS classification, which will be explained in chapter seven, is a hierarchical structure, and all preceding layers can be determined if one has knowledge of the lowest level in the structure. This resulted in further reducing data storage.

Third Normal Form, which can only be implemented once the schema is in 2NF, requires that all fields in a table be dependent on the primary key and all transitive dependencies must be eliminated. By way of example, every field in the DAILYDATA table of figure C-5 represents an attribute of a particular stock on a particular date. Hence all the fields are dependent on the primary key. The same can be said for the other tables in figure C-5. As such, the tables, by being in 2NF, are also in 3NF.

An evolved version of third normal form, also referred to as Boyce-Codd Normal Form (BCNF), requires that the schema be in 3NF and that there exist no dependencies within the primary keys themselves. An example of a dependency amongst keys would be the case of two primary keys, *StudentID* and *StudentName*. In this case, *StudentName* can be derived from *StudentID*, and can therefore be eliminated, without sacrificing functionality, to achieve BCNF. The schema, as illustrated in figure C-5 above, is in BCNF.

Note that figure C-5 also represents the final schema diagram for the database. The DailyData and AnnualData tables show a subset of the fields in the tables, both of which exceed 150 finance-specific fields.