Experimental Computational Simulation Environments for Algorithmic Trading

Michal Galas

The Thesis submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University College London.

Department of Computer Science

University College London

January 18, 2014

Abstract

This thesis investigates experimental Computational Simulation Environments for Computational Finance that for the purpose of this study focused on Algorithmic Trading (AT) models and their risk. Within Computational Finance, AT combines different analytical techniques from statistics, machine learning and economics to create algorithms capable of taking, executing and administering investment decisions with optimal levels of profit and risk. Computational Simulation Environments are crucial for Big Data Analytics, and are increasingly being used by major financial institutions for researching algorithm models, evaluation of their stability, estimation of their optimal parameters and their expected risk and performance profiles. These large-scale Environments are predominantly designed for testing, optimisation and monitoring of algorithms running in virtual or real trading mode. The state-of-the-art Computational Simulation Environment described in this thesis is believed to be the first available for academic research in Computational Finance; specifically Financial Economics and AT. Consequently, the aim of the thesis was: 1) to set the operational expectations of the environment, and 2) to holistically evaluate the prototype software architecture of the system by providing access to it to the academic community via a series of trading competitions. Three key studies have been conducted as part of this thesis: a) an experiment investigating the design of Electronic Market Simulation Models; b) an experiment investigating the design of a Computational Simulation Environment for researching Algorithmic Trading; c) an experiment investigating algorithms and the design of a Portfolio Selection System, a key component of AT systems.

Electronic Market Simulation Models (Experiment 1): this study investigates methods of simulating Electronic Markets (EMs) to enable computational finance experiments in trading. EMs are central hubs for bilateral exchange of securities in a well-defined, contracted and controlled manner. Such modern markets rely on electronic networks and are designed to replace Open Outcry Exchanges for the advantage of increased speed, reduced costs of transaction, and programmatic access. Study of simulation models of EMs is important from the point of view of testing trading paradigms, as it allows users to tailor the simulation to the needs of particular trading paradigms. This is a common practice amongst investment institutions to use EMs to fine-tune their algorithms before allowing the algorithms to trade with real funds. Simulations of EMs provide users with the ability to investigate the market micro-structure and to participate in a market, receive live data feeds and monitor their behaviour without bearing any of the risks associated with real-time market trading. Simulated EMs are used by risk managers to test risk characteristics and by quant developers to build and test quantitative financial systems against market behaviour.

Computational Simulation Environments (Experiment 2): this study investigates the design, implementation and testing of an experimental Environment for Algorithmic Trading able to support a variety of AT strategies. The Envi-

ronment consists of a set of distributed, multi-threaded, event-driven, real-time, Linux services communicating with each other via an asynchronous messaging system. The Environment allows multi-user real and virtual trading. It provides a proprietary application programming interface (API) to support research into algorithmic trading models and strategies. It supports advanced trading-signal generation and analysis in near real-time, with use of statistical and technical analysis as well as data mining methods. It provides data aggregation functionalities to process and store market data feeds.

Portfolio Selection System (Experiment 3): this study investigates a key component of Computational Finance systems to discover exploitable relationships between financial time-series applicable amongst others to algorithmic trading; where the challenge lays in identification of similarities/dissimilarities in behaviour of elements within variable-size portfolios of tradable and non-tradable securities. Recognition of sets of securities characterized by a very similar/dissimilar behaviour over time, is beneficial from the perspective of risk management, recognition of statistical arbitrage and hedge opportunities, and can be also beneficial from the point of view of portfolio diversification. Consequently, a large-scale search algorithm enabling discovery of sets of securities with AT domain-specific similarity characteristics can be utilized in creation of better portfolio-based strategies, pairs-trading strategies, statistical arbitrage strategies, hedging and mean-reversion strategies.

This thesis has the following contributions to science:

Electronic Markets Simulation - identifies key features, modes of operation and software architecture of an electronic financial exchange for simulated (virtual) trading. It also identifies key exchange simulation models. These simulation models are crucial in the process of evaluation of trading algorithms and systemic risk. Majority of the proposed models are believed to be unique in the academia.

Computational Simulation Environment - design, implementation and testing of a prototype experimental Computational Simulation Environment for Computational Finance research, currently supporting the design of trading algorithms and their associated risk. This is believed to be unique in the academia.

Portfolio Selection System - defines what is believed to be a unique software system for portfolio selection containing a combinatorial framework for discovery of subsets of internally cointegrated time-series of financial securities and a graph-guided search algorithm for combinatorial selection of such time-series subsets.

Declaration

I, Michal Galas, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis. Some of the work presented in this thesis has previously been published and submitted by the author in the following papers:

1. Galas, M., Brown, D., & Treleaven, P. **"A Computational Social Science Environment for Financial/Economic Experiments"**, CSSSA2012.

2. Treleaven, P., Galas, M. **"Algorithmic Trading, Flash Crashes & IT Risk"**, Risk Management in Financial Institutions, Euromoney Books, 2013.

3. Treleaven, P., Galas, M., & Lalchand, V. **"Algorithmic Trading: review"**, Communications of the ACM 56.11 (2013): 76-85.

4. Galas, M., Grochmalicki, J., & Treleaven, P. **"Algorithmic Trading Development Process"**, Manuscript Submitted.

Acknowledgements

I believe that Experimental Computational Simulation Environments will have significant impact on the way research is conducted in Computational Social Science. The EPSRC and ESRC are each interested in funding data infrastructure projects: including Big Data Analytics, Systemic Risk Modelling and scientific approaches to Socio-Economic-Financial Simulation. Investments in Big Data Analytics infrastructure will, in authors opinion, naturally evolve into experimental Computational Simulation Environments (cf. Financial Wind Tunnels).

The Computational Simulation Environment investigated in the thesis was successfully deployed over a network of servers in UCL. The environment is used for financial economics research and for the organization of algorithmic and manual trading competitions on a UK, European and global scale. Collaborators included Euronex (Deutsche Borse), Barclays, Knight, City, LMAX, Microsoft. Various components of the experimental system were utilised in commercial projects with LMAX, Tower Trading and Microsoft.

All the core elements of the three experiments were designed and implemented by the author with support of a group of approximately 60 BSc, MSc and PhD students over the period of 4 years. The author of the thesis was responsible for the research and design of the software architecture of the elements, for implementation of the key algorithms, for the design of experiments and for supervision of multiple groups of students that were testing the created systems.

The author would like to express his sincere gratitude to Prof Philip Treleaven and Dr Dan Brown for supervision of the thesis. The author would also like to thank Dr Jan Grochmalicki, Vidhi Lalchand, Akash Parvat Goswami, Kacper Chwialkowski and numerous groups of BSc, MSc and PhD students for their help.

Contents

List of Figures

Chapter 1

INTRODUCTION

This chapter introduces the concept of Computational Simulation Environments for financial experiments and describes three applications of such environments to the problem of algorithmic trading. The chapter gives a brief summary of the objectives set for the thesis, the study of the problems and the contribution of the covered topics to science. Finally, it provides an overview of all the other chapters in the thesis.

1.1 Motivation and Context

This thesis investigates experimental computational environments for simulation of trading algorithms and their associated risk. Algorithmic Trading (AT) is a relatively new discipline of computational finance that investigates applications of computers to the automation of one or more stages of investment strategies, including automation of risk management and asset allocation [Nuti et al., 2011]. Automation of investment strategies implies the existence of exploitable, repetitive patterns of behaviour of the traded securities. Discovery of such patterns of behaviour requires significant amount of research and computational power, and therefore justifies the existence of computational environments for algorithmic experiments.

To conduct a rigorous investigation of algorithms and risk associated with various potential investment scenarios, one requires a software system able to evaluate the quality ('utility') of strategies being exposed to different 'temporal' circumstances. One also requires a management and reporting system able to execute different hypotheses. This scientific rigour can be imposed by the simulation environments and the way they allow users to operate. The environments may enable repetition and replay of experiments, monitoring of the behaviour of experiments and most importantly direct utilisation of successful experiments without a need of recoding. Given the above one can argue that the field of computational finance focus on the computational investigation of financial systems, models of behaviour of financial algorithms, and finally on the ways to measure/summarize such algorithms. One important consequence

of this state of affairs is the fact that experimental simulation environments may shape evolution of algorithmic trading and general social sciences for the coming decades.

1.2 Problem Statement

In the light of the above, three different experiments were conducted to create computational simulation systems suitable for experimentation in the design of trading algorithms, their performance and risk. The relationships between the experiments are summarized by Figure 1.1.

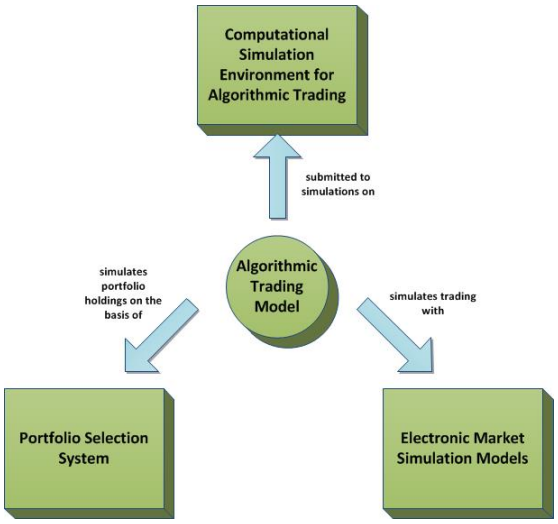


Figure 1.1: Experiments Relationship.

An ability to investigate and evaluate behaviour of models is central to computational finance. The author identified seven general simulation types that can be conducted to experiment with such models: 1) a Free-Run, 2) a Back-Test (normal circumstances), 3) a Stress-Test (abnormal circumstances), 4) a Forward (Monte-Carlo)-Test, 5) a Sensitivity-Test, 6) an Optimisation, and 7) a Multi-Agent-Simulation version of the described tests.

Free-Run is a process of simulation on-the-fly, with results being generated on the basis of newly provided input parameters, as soon as they appear.

Back-Test is a process of simulation on the basis of historical records of event occurrences.

Stress-Test is a process of simulation on the basis of 'abnormal (highly unexpected) data circumstances', while i.e. the back-test can be considered a run over historic 'expected data circumstances'.

Forward (Monte-Carlo) -Test is a process of simulation on the basis of synthetic data generated for multiple, alternative, scenarios most commonly with use of a form of linear drift and an element enforcing stochastic process.

Sensitivity-Test is a process of model simulation on the basis of synthetic data of different time and values granularity that allows us to estimate the models' sensitivity characteristics.

Optimisation is a process of running multiple disconnected simulations on the same model but on the basis of different model parameters, with a goal to maximize or minimize an outcome of a simulation measured by the utility function on the model.

Multi-Agent Simulation is a process of model simulation with more than one active, decision taking elements. All the previously mentioned types of simulations and optimisation can be performed on a single-agent model or a multi-agent model. Typically, a single-agent model implies that the model itself is the only active element in the simulation that reacts to the stimuli generated by the simulation process.

For the purpose of the thesis the first two listed simulation types will be applied to experiment with the algorithmic trading models, in a form of a software platform.

A crucial option while experimenting with algorithmic trading is a possibility of executing order instructions with no involvement of real funds. For this reason two models of electronic market exchanges were researched and experimented with. The first model enables the simulation of execution of order instructions against an end-of-day price being streamed to the model from an actual exchange. The second model enables the simulation of execution of order instructions against real-time prices being streamed from external real-life exchanges. This type of simulation allows testing of trading capabilities in real-time, but is limited to testing 'market taking' (passive) capabilities due to the fact that it is impossible to influence prices and volumes with instructions issued to the exchange simulator. Design of exchange simulators in this way enables fine-grained experiments customized for:

1. high-frequency traders interested in immediate reactions to real-time prices with positions in traded securities maintained over short periods of time with bursts of order instructions;
2. experiments for intra-day traders interested in identifying daily market regimes and benefiting from maintaining a few hour to one-day positions, with use of a few order instructions a day;
3. portfolio managers to perform experiments with long-term positions adjusted at the end or beginning of a day.

As previously stated, automation of investment strategy execution implies existence of exploitable, repetitive patterns in information describing behaviour of tradable securities and more generally in analytical data used to trade securities. Two approaches can be taken in the process of discovery of tradable patterns. One can either look for behavioural patterns of individual securities or for patterns in relationships between groups of securities. The former is well explored in academia (e.g., [Cañete et al., 2008],

[Gabrielsson et al., 2012], [Huang et al., 2011]) and therefore will not be a subject of investigation in this thesis. The latter fits into the general framework of Markowitz's Modern Portfolio Theory (going back to [Markowitz, 1952]) and Lo's Adaptive Markets Hypothesis [Lo, 2004], and due to challenges related to the discovery of stable relationships between securities is far more interesting from the perspective of this thesis. Consequently, the third described experiment involves research of a computational simulation system capable of combinatorial searches over a decision space defined by financial time-series to deliver sets (portfolios) of securities with the strongest stable relationships and the largest amounts of securities in the discovered sets.

1.3 Research Methodology

This work focuses on computational finance rather than performance problems encountered in software engineering research. As a result the scientific effort is directed at holistic, operational results of the experimental systems and the applicability of proposed software architectures to a given financial economic problem of algorithmic trading. Consequently, the methodology steps assumed in this thesis can be considered as follows:

1. To build the knowledge of experimental computational simulation environments the author will summarize existing background knowledge and research related to academic literature.

2. The author will then focus on the design of individual experiments and on implementation of experimental environments to support such work.

3. This will be followed by integration of all the experimental systems, to support access for academic community; that will allow further evaluation.

4. When all the experimental components of the environment are in order, the author will organize a few trading competitions with varying degrees of complexity and trading circumstances; to capture the cross-section of interested academic community (or at least a representative subset) over a longer periods of time.

5. During the organized competitions the author will observe and evaluate the behaviour of participants through the behaviour of their trading models in the created environment. This will also allow observation of the experimental components of the environment.

6. Finally, based on such qualitative observations of participants and quantitative evaluation of models, a summary can be drawn. This will include the overall holistic, operational behaviour of the experimental computational simulation environment, as well as a discussion of suitability of individual experimental components of the environments.

A software engineering approach was assumed to be the most suitable to study the applications of computational simulation environments in algorithmic trading. Consequently, for every designed experiment a software system was created to enable simulation. Such systems were constructed in a modular way to connect individual modules into a larger architecture, with the architecture constituting an algorithmic trading environment. All the designed experiments involved utilisation of created software modules to investigate behaviour of the functionality. This approach was found advantageous due to the following: a) it enables experimentation with a possibility of having reusable components on the basis of which more complex systems and consequently more sophisticated experiments can be conducted; b) each experiment enforces high test coverage in the component, which has a positive influence in the stability of the environment.

Given the large-scale on which the experiments were conducted as well as a need for sufficiently tested software capable of supporting academic community (competition participants), parts of implementation and testing of the computational simulation systems involved groups of students working on less critical software components. The following software development practices were utilised: a) an agile development practices (Scrum, [Wikipedia, 2013b]) were used to manage execution of tasks & Sprints (a period of time during which specific set of tasks has to be completed) and for organisation of daily-, weekly- & sprint- handover meetings; b) continuous integration methodologies were used to cleanly integrate new additions to the already existing source & binary repositories; c) a test-driven development was used to ensure that necessary quality of the code and to drive development of functionalities; d) a pairs-review and self-documentation methodologies were used to improve quality & readability of committed code; e) to support the above development practices the following technologies were used: Git (http://git-scm.com/), Maven (http://maven.apache.org/), YouTrack (http://www.jetbrains.com/youtrack/), TeamCity (http://www.jetbrains.com/teamcity/), Sonar (http://www.sonarsource.org/) and Nexus (http://www.sonatype.com/Products/Nexus-Professional).

1.4 Thesis Structure

The research methodology described above is reflected in the structure of the thesis. Given the uniqueness and large-scale of the designed experiments, case studies of individual environments are described in separate chapters, containing performance results compared to assumed expectations. Furthermore, an overall operational experiment that utilise all the designed environments is described in a separate chapter. The experiment is provided in a form of behavioural results of various trading competitions that utilised the experimented environments.

The evaluation criteria for individual experiments focus on justification of designed software architecture. Limited quantitative evaluation of software performance is also provided. To quantify the overall

applicability of utilised architectures, a set of experiments in a form of trading competitions quantify the behaviour of models in the environment, and consequently, indirectly quantify the quality and stability of software architectures.

Chapter 2: BACKGROUND AND LITERATURE REVIEW discusses existing research, software architecture designs and technologies applicable to experimental computational simulation environments. To provide the reader with a sufficient understanding o financial economics modelling, a background knowledge of the on-the-fly and historical simulation of algorithmic trading is provided. Furthermore, since the main focus of the thesis is on the software architecture designs of simulation environments (e.g., market simulations) a variety of separate, but nevertheless related building blocks of such environments is presented. The sources can be grouped into the following categories: a) Data Streaming & Information Services, b) Analytical Tools & Business Intelligence, c) Experimental & Simulation Environments, d) Trading Platforms for Manual and Algorithmic/Automated Traders, e) Big Data Facilities & Data Warehousing, and f) Complex Event Processing Engines.

Chapter 3: ELECTRONIC MARKET SIMULATION MODELS investigates the simulation of Electronic Trading Exchanges, used to trade virtually without the risk of losing real money. Chapter 3 describes two simulation models that embody the key features of an electronic market exchange. The models incorporate functionalities for order book, matching engine, account management, profit & loss (P&L) clearing, event-based price streaming and order instruction/confirmation messaging to harness the most primitive features of a modern electronic market exchange.

Chapter 4: COMPUTATIONAL SIMULATION ENVIRONMENT FOR AT describes the research and design of an experimental Computational Simulation Environment able to support a variety of AT strategies. The environment consists of a set of distributed, multi-threaded, event-driven, on-the-fly, Linux services communicating with each other via an asynchronous messaging system. The Environment allows multi-user real and virtual trading. It provides a proprietary API to support development of algorithmic trading models and strategies. It allows advanced trading-signal generation and analysis in near real-time, with use of statistical and technical analysis as well as data mining methods. It also provides data aggregation functionalities to process and store market data feeds.

Chapter 5: PORTFOLIO SELECTION SYSTEM describes a large-scale search algorithm for discovery of sets of securities with an AT domain-specific similarity characteristics that can be utilized in the creation of improved portfolio-based strategies, pairs-trading strategies, statistical arbitrage strategies, hedging and mean-reversion strategies. Discovery of exploitable relationships between financial time-series is important in AT. Recognition of sets of securities characterized by a very similar/dissimilar behaviour over time, can be beneficial from the perspective of risk management, recognition of statistical arbitrage and hedge opportunities, and can be also beneficial from the point of view of portfolio

diversification.

Chapter 6: EVALUATION AND ASSESSMENT presents evaluation of the outcomes of the three performed experiments by investigating the quality, performance and stability of the created experimental systems. The evaluation is performed in three stages: stage one presents results of software testing through continuous integration, stage two presents performance tests of the Environment, and stage three describes a holistic approach to testing stability of the systems through utilisation in algorithmic and manual trading competitions.

Chapter 7: CONCLUSIONS AND FUTURE WORK discusses the overall results obtained from the three experiments. The results are examined with respect to the initial objectives set in the Introduction chapter and are related to the literature and publications review in Chapter Two. A discussion on applicability of the experimental computational simulation environments in a variety of other problems is presented. Relevant conclusions are drawn, accompanied by a list of contributions to the field of Computational Finance and Algorithmic Trading. Finally, a discussion, on what was learned during the course of the case studies, is presented with conclusions on possibilities for further research.

1.5 Contributions

This thesis investigated the experimental Computation Simulation Environments for the study of trading algorithms and their risk. The principal scientific contributions can be considered as follows.

Electronic Markets Simulation - identifies key features, modes of operation and software architecture of an electronic financial exchange for simulated (virtual) trading. It also identifies key exchange simulation models. The models of Electronic Markets are frequently used as part of Computational Simulation Environments either by quantitative developers interested in simulating their trading models or by financial economists interested in modelling e.g., the stability of banking systems. Consequently, an academic study of such simulations is important in future research of this field.

Computational Simulation Environment - design, implementation and testing of a prototype experimental Computational Simulation Environment for Computational Finance research, currently supporting the design of trading algorithms and their associated risk. Computational Simulation Environments are at the heart of the Big Data Analytics and consequently are an important topic that will shape the future of Computational Social Science. A goal is to make the experimental Environment available to the academic community.

Portfolio Selection System - defines what is believed to be a unique software system for portfolio selection containing a combinatorial framework for discovery of subsets of internally cointegrated time-series of financial securities and a graph-guided search algorithm for combinatorial selection of such time-series subsets.

Low-Level AT Classification - the thesis also classifies key AT strategies and their low-level features and presents how such classification can be used in evaluating similarity of trading models (this is an important subject from the point of view of diversification of trading strategies). This knowledge is then uniquely incorporated into the Computational Simulation Environment via a software framework supporting event-driven and service-oriented paradigms for building such AT strategies.

Key Computational Finance Simulation Types - finally the thesis identifies the key simulation processes in experimental computational simulation environments, that can be performed on analytic models, including: a free-run, a back-test, a stress-test, a sensitivity-test, an optimisation and a multi-agent versions of the listed simulations.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

The chapter presents existing background information related to the concept of Computational Simulation Environments for financial experiments, including research, technologies, architectural designs and implementation techniques.

The thesis focuses on the experimental Computational Simulation Environments for Computational Finance, specifically on its application to Algorithmic Trading. A variety of separate, however, related work exists both in academia and the industry. To provide sufficient background to the reader, this review describes the following subjects: a) Computational Simulations for Finance, b) Components of Computational Simulation Environments, c) Algorithmic Trading, and d) System Architectures for AT.

The description of computational simulations in finance, presents the variety of available techniques and consequently the complexity of algorithms being subjected to simulations. To introduce the reader to the environment within which proposed experimental algorithms may work, a high-level overview of Computational Simulation Environments is provided. The background to algorithmic trading is necessary to understand the intended application of this study. Information about the software architecture trends in the AT is required to give the reader a description of possible software technologies applicable to the study. Finally, the description of an infrastructure for financial systems gives an insight on the underlying hardware, software and network infrastructure.

2.1 Computational Simulations for Financial Economics

To effectively conduct a large-scale research in finance and economics, the researchers need access to terabytes/petabytes of real-time and historic data [Cukier and Mayer-Schoenberger, 2013] (e.g., financial, economic, social/news, retail, healthcare, etc.) stored in the large-scale, distributed data banks e.g., Hadoop/MapReduce-like [Abouzeid et al., 2009]. The main components of such an environment are: a)

big databanks; b) real-time streamed data feeds, such as trading data and social media data; c) high-performance computing; d) analytics using data mining, simulation modelling and stream processing; e) computational science using complex systems, computational statistics and machine learning techniques; and f) a user-interface for controlling and visualising the computations, including the programmatic control and machine readable reporting. On the basis of the listed components it is possible to run various types of simulations, either in a historic or live mode.

One of the key original contributions of this thesis to financial economics is a list of possible types of simulations considered from the point of view of experimental computational simulation environments (rather than i.e. statistics) that the author believes is unique and has never been described before. The list was introduced in Chapter 1. However, for completeness, the following set of paragraphs describes again in more detail the most common types of simulations.

Free-Run is a process of simulation on-the-fly, with results being generated on the basis of newly provided input parameters, as soon as they appear. A key property of the free-run process is the fact that the investigated model can not affect the input data and consequently the environment in which it resides (the sandbox effect, a lack of the feedback loop to the environment). This property is the main differentiator between the free-run and the actual execution of the model.

Back-Test is a process of simulation on the basis of historical records of event occurrences. The goal of the process is an observation of a potential behaviour of a model in the past, to eliminate any unexpected and disadvantageous behaviours, and to maximize the impact of advantageous behaviours in the future. Furthermore, the back-test processes are used for a model-tuning and for estimation of model behaviours.

Stress-Test while the back-test can be considered a walk over the historic 'expected data circumstances', the stress-test is a process of simulation on the basis of 'abnormal (highly unexpected) data circumstances'. The goal of such process is to estimate and improve the model's susceptibility to such data circumstances.

Forward (Monte-Carlo) -Test is a process of simulation on the basis of a synthetic data generated for multiple, alternative, scenarios most commonly with use of a form of a linear drift and an element enforcing stochastic process. Often the scenarios are generated with parameters for the drift and stochastic process estimated from the historic data to enforce continuity between the history and the potential future scenario. This approach is used to estimate the most probable statistical distribution of future scenarios and consequently the most probable statistical distribution of model behaviours. This approach can also be used to bind the model within the most advantageous thresholds of future scenarios.

Sensitivity-Test is a process of model simulation on the basis of a synthetic data of different time and values granularity that allows estimation of models' sensitivity characteristics: susceptibility to the minimal changes in such data granularities. This approach is used to evaluate the most optimal range of the

data granularity for the model, as well as the boundaries within which the model exhibits the expected behaviour.

Optimisation is a process of running multiple disconnected simulations with use of the same model but on the basis of different model parameters, with a goal to maximize or minimize an outcome of a simulation measured by the utility function of the model. The key feature of optimisation is a possibility of repetitiveness of an achieved outcome. Given that the free-run, the back-test, the stress-test, the forward-test and the sensitivity-test are all forms of simulation, they can be used as an underlying process for optimisation (with an exception that different goals of an outcome for each type of simulation are expected).

Multi-Agent Simulation all the mentioned types of simulations and optimisation can be performed on a single-agent model or a multi-agent model. Typically, a single-agent model implies that the model itself is the only active element in the simulation that reacts to the stimuli generated by the simulation process. In a multi-agent simulation the model is an aggregate of the active elements being active & reactive to the simulation process and the rest of the active elements in the model.

Given the already large-scale of all investigated environments, this thesis focuses on experiments with the free-run and the back-test simulations, rather than all types of mentioned simulations.

2.2 Components of Computational Simulation Environments

Experimental computational simulation environments, although unique in financial economics, share some characteristics with existing algorithms and systems. For example, such environments need to support the on-the-fly data provision to models; this provides relationship to the existing financial data streaming and market information services. The environments need to also support historic data provision on a large scale, this provides relationship to the existing big data and data warehousing facilities. If a given model processes complex information on-the-fly, the supporting environment will most likely draw from complex event processing engines. All models simulated in the designed environments may be considered analytic models and consequently this provides relationship to various analytic tools & libraries as well as other simulation environments. Finally, particular application of the investigated experimental environments is algorithmic trading, this provides relationship to trading platforms and social media platforms that feed analytic information to trading models. The following literature review aims to familiarize the reader with concepts that may become features of the investigated environments.

Trading Environments are complex, enterprise-size systems that allow their users to trade, by providing the entire supporting infrastructure and manage the variety of risk factors on a micro and macro scale. Financial institutions interested in trading design their systems with respect to the speed of execution, the ability to handle large volumes of data in a rapid manner, and the possibilities of advanced data analytics.

Moreover, the physical location of trading and risk models is often chosen to be co-located with trade execution venues, while the location of modelling and simulation functionalities is often co-located with sources of the historical data.

According to [IBM, 2008] the *'high processor performance, ultra-fast communications and very low latency messaging systems are collectively approaching theoretical limits that accelerate the critical systems supporting trade matching. The physical distance is quickly becoming the boundary for any message to travel so new achievements in speed will be more of a commodity rather than game-changing. As the speed attribute becomes a commodity the focus of many execution venues will refocus on their total system capabilities'*.

The increase of the amount of derived analytic data goes hand in hand with the growth of the amount of streamed and processed data. All such information needs to typically be handled and processed in real time. To add to the complexity of the problem, the appearance and disappearance of financial instruments from the markets needs to be dynamically accommodated by the trading and risk systems (as a consequence of the fact that the trading advantage is achieved from the data analysis and risk estimation, and not only from the speed). As a consequence, all the mentioned problems force the designers of financial environments to make it highly scalable, with the possibility of deployment and removal of financial instruments (and consequently supported streaming and processing functionalities) in a rapid fashion.

Yet another trend that can be observed in the industry is an architectural shift towards the Service-Oriented Architectures (SOA), e.g., [Erl, 2006] and Event-Driven Architectures (EDA), e.g., [Michelson, 2006]. These approaches significantly reduce the complexity and the cost of integration, and allow virtualization of the entire environments with an increase of the speed of trading through a suitable collocation.

While investigating software architecture trends in IT, a variety of related work concerning financial data and environments was identified both in academia and the industry. They can be grouped into: a) Financial Data Streaming and Market Information Services; b) Analytical Tools and Business Intelligence; c) Experimental and Simulation Environments; d) Trading Platforms for Manual and Algorithmic/Automated Traders; e) Big Data Facilities and Data Warehousing; and f) Complex Event Processing Engines. These are reviewed below.

2.2.1 Financial Data Streaming and Market Information Services

A number of papers survey data sharing services, such as the market information services (e.g., [Shepherd, 1997], [Ainsworth, 2009]), or the messaging systems and the large scale live data streaming of financial data (e.g., [Andrade et al., 2009], [Fusco et al., 2010] and [Turaga et al., 2010]).

The major academic platforms for financial and economic data are the Wharton Research Data Services (https://wrds-web.wharton.upenn.edu/wrds), which provides web-based, terminal-based and programmatic access to its resources for data mining.

The major commercial platforms for financial data are the Bloomberg (www.bloomberg.com) and the Thomson Reuters (www.thomsonreuters.com) platforms. Both are capable of streaming large quantities of data in either live or historical mode, and both architectures provide big data storage on the server-side and buffering capabilities on the client-side. Apart from the data streaming, both platforms provide basic data analytics on the client-side and charting functionalities for data visualization.

2.2.2 Big Data Facilities and Data Warehousing

The term 'big data' describes an extremely large, distributed dataset in a scale beyond the capabilities of the commonly used software (e.g., petabytes); while the term 'data warehousing' describes the dataset used for reporting and analysis. Research on these subjects is covered by [Agrawal et al., 2011] and [Abouzeid et al., 2009].

2.2.3 Complex Event Processing Engines

CEP Engines are capable of processing events organized in an SQL-style type of queries, constructed on the basis of table-like buffers created and maintained in near-real time. CEP technologies can be considered general analytical tools best utilized for the purpose of the on-the-fly monitoring and analytical calculations (see [Barga et al., 2006], [Wu et al., 2006], [Luckham, 2008], [Chandramouli et al., 2010]).

Prominent commercial CEP Engines include: StreamBase (www.streambase.com) and StreamInsight (http://msdn.microsoft.com/en-us/library/ee362541.aspx). Both systems offer SQL-like languages to query events on-the-fly. Open source systems also exist with Esper (http://esper.codehaus.org) being well regarded.

2.2.4 Analytical Tools and Libraries

Analytical tools are the mainstay of scientific and engineering, spanning statistical libraries, data mining frameworks and machine learning toolkits ([Gangadharan and Swami, 2004], [Khan and Quadri, 2012]).

Well-known general-purpose analytical toolkits are: Matlab (www.mathworks.com), R (www.r-project.org), Mathematica (www.wolfram.com/mathematica), SPSS (www.spss.com) and SAS (www.sas.com). Their design supports access to historical and live sources, and quantitative data manipulations on the processed data. Matlab, R and Mathematica can be considered lower-level platforms, providing rich environments consisting of libraries of application-specific analytical functions where users can code in the platform-specific languages. R and Mathematica focus mainly on historical data analytics, although limited event analytics is also possible. SPSS and SAS can be considered higher-level analytical platforms with a strong support for machine learning and data mining functionalities.

Well-known analytical libraries include: the Numeric Algorithms Group (www.nag.co.uk) which provides a variety of numerical libraries, compilers and visualization tools for engineering, scientific and research applications. Likewise, machine learning frameworks, usually available on an open-source basis, include: Mahout (http://mahout.apache.org), Weka (www.cs.waikato.ac.nz/ml/weka) and Encog (www.heatonresearch.com/encog).

2.2.5 Experimental and Simulation Environments

Computing clouds, grids and clusters are often utilized as experimental environments for running simulations, to model data and to calibrate models, especially for large scale financial analytics (see [Bullock, 2011], [Blog, 2012], [Cloudscaling, 2012]). Various commercial environments exist, with the most prominent being Google Cloud Services (www.google.com/enterprise/cloud/index.html), Amazon Web Services (http://aws.amazon.com/) and Microsoft Azure Cloud (www.windowsazure.com/en-us).

All major universities provide experimental environments of these types. For example, UCL currently hosts three research computing facilities: the Legion Cluster, the Unity SMP, and the Condor Pool (www.ucl.ac.uk/isd/common/research-computing/services).

2.2.6 Trading Platforms for Manual and Algorithmic Traders

Trading platforms are systems capable of supporting manual and algorithmic traders. Such systems typically support data provision, visualization and data analytics, and the forwarding of trades to exchanges. More sophisticated trading platforms also support risk management and post-trade analysis, and provide an API for implementation of trading models.

Major banks and hedge funds typically have proprietary trading platforms. There are also vendor platforms, such as MetaTrader, TradeStation and X_Trader amongst others, and even open source systems e.g., Marketcetera.

The majority of academic research on algorithmic trading is devoted to data analytics and algorithms for forecasting, pricing and risk, etc. Significantly less academic work investigates the trading platforms and the trade-support infrastructure (see [Batten et al., 2012] and [Risca, 2008]).

2.2.7 Social Media Platforms for Sentiment Analysis

Social media platforms are analytic systems designed to aggregate, process and analyse large quantities of publicly available social data including i.e. RSS feeds and web blogs. This types of platforms are capable of supporting analysis of data, data modelling & monitoring as well as model simulation and evaluation. This includes social pattern recognition, data mining and predictive analytics.

Platforms and services for social analytics are being provided by Thompson Reuters (www.thomsonreuters.com), Google (www.google.com/analytics/features/social.html) and SAP (www.sap.com/solutions/solutionextens

ions/social-media-analytics/index.epx) amongst other vendors. Various smaller service providers are also available i.e. Lexalytics (www.lexalytics.com/industries/social-media-monitoring), Social-mention* (http://socialmention.com/). The academic and open source communities currently focus their efforts on storage & search facilities for unstructured data (e.g., http://lucene.apache.org/, http://glaros.dtc.umn.edu/gkhome/software, http://uima.apache.org/) and on natural language processing and text analytics (e.g., http://alias-i.com/lingpipe/, http://gate.ac.uk/, http://nlp.stanford.edu/software/).

Academic research investigates numerous areas of social analytics. While the early research focused on the basic relationships between social features e.g., brand recognition and its relationship with financial performance of companies (i.e. [Yoon et al., 1993]) later research focuses on sentiment detection and analysis (i.e. [Vovsha and Passonneau, 2011]), technologies that support sentiment analysis (i.e. [Chua et al., 2009]), and predictive influence that social data may have, for example, on trading (i.e. [Vincent and Armstrong, 2010]).

To summarize the above section, all the mentioned systems and technologies are capable of providing partial, very limited functionality of computational simulation environment. The experimental computational simulation environment combine the mentioned systems as features of a large-scale environment for algorithmic trading. Such simulation environment is believed to be unique in academia for Financial Economics modelling.

Having the above 'big picture' in mind, we should now focus on the algorithmic trading requirements related to modelling. This is to help with investigation of the key model types, modes of operation and data flows that the experimental environment should support.

2.3 Algorithmic Trading (AT)

In electronic financial markets, algorithmic trading [Treleaven et al., 2013] is a field of computational finance that combines different analytical techniques, amongst others (from statistics, machine learning, physics and economics) to create algorithms capable of taking, executing and administering investment decisions with optimal levels of profit and risk. AT aims to automate one or more stages of the trading process, where the stages can be defined as: a) pre-trade analysis (data analysis, state of the world analysis); b) signal generation (decision taking process, policy formation); c) trade execution (execution of actions, policy execution) and d) post-trade analysis (evaluation of results, utility analysis). Trade execution [Nuti et al., 2011] is subdivided into two broad categories: agency/broker execution, when a system optimizes the execution of a trade on behalf of a client, and principal/proprietary trading, where an institution is trading on its own account. Historically, AT draws from cybernetics [Ashby, 1956], [Maruyama, 1963] and control systems by introducing one or more feedback loops into the process; typically between post-trade analysis and pre-trade analysis. Two stages that are typically part of such

feedback loops are money management (asset allocation) and risk management.

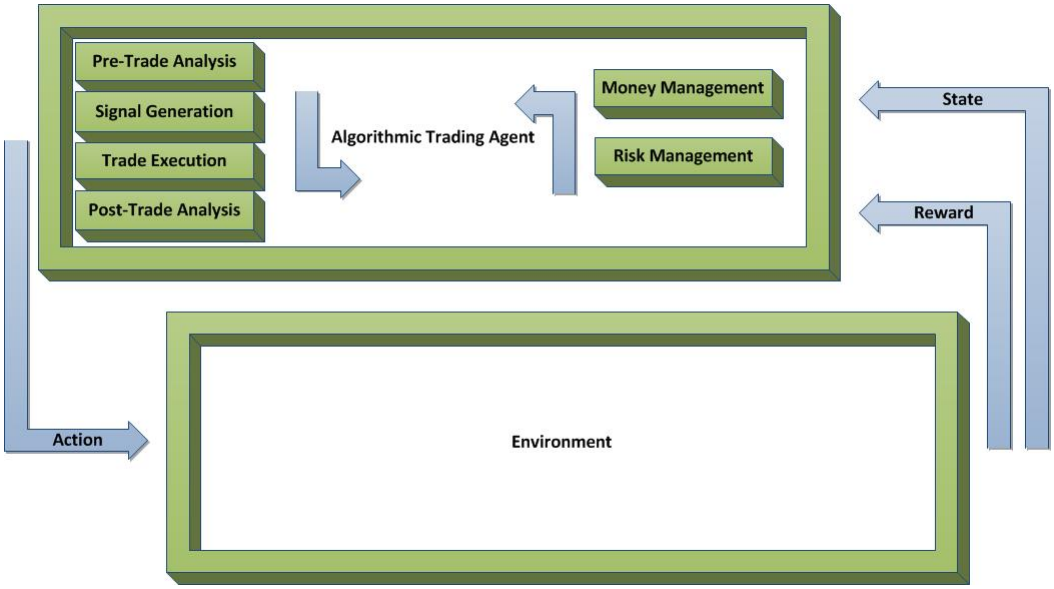


Figure 2.1: Key stages of Algorithmic Trading process.

A typical algorithmic trading process may execute as follows: 1) during the pre-trade analysis stage, the trading model acquires and abstracts the information about the current state of the financial markets and retrieves the most indicative (with respect to a pre-defined utility function) set of features of analysed data, 2) such indicators are then utilised in the signal generation step, where the features are filtered by a set of rules that define circumstances when a trading signal should occur, 3) every time new market information arrives, it is streamed through the pre-trade analysis process and signal generation process, 4) in an instance the signal generation process indicates that tradable opportunities exist in the model, a signal is sent to the trade execution element responsible for generation of a set of instructions to trade particular securities defined in the strategy, 5) to be able to evaluate the utility of the executed transaction, the algorithm performs a post trade analysis.

Apart from the describe stages, the most advanced strategies also utilise the money management and the risk management elements, where: 6) the money management element allocates a specific amount of assets to each transaction with an aim to maximise a utility of the model, 7) while the risk management is responsible for minimising risk of a loss and can also be expressed in a form of a utility function.

The Pre-Trade Analysis [Madhavan, 2002] aims to help trading strategies to take the most optimal action at a specific time, given all the available information. Such analysis forms a first step in the creation of trading decisions in a model. It is a process of analysis of input data properties that leads to identification and quantification of a set of data features on the basis of which strategies can take their trading decisions. Such data features can be used to predict the future behaviour of financial instruments (that the algorithmic system is intended to trade), can be used to evaluate the levels of exposure/risk

associated with the financial instruments, and can be also used to calculate potential costs associated with trading the financial instruments.

During this phase, the strategy may identify exploitable relationships between various assets (rather than just features of one of the assets alone) on the basis of which it may generate profitable transactions. Identification of co-movement patterns, in securities the strategy trades, can be considered one type of data relationship that can be exploited to generate profit in AT.

The pre-trade analysis is intended to improve indicativeness of the input data in AT. Raw input data is assumed to seldom carry a desired degree of information to support the decision-taking process. To improve the quality of data, the pre-trade analysis stage may be further divided into pre-processing, feature selection, feature extraction and feature cross-validation. The pre-processing stage ensures that the raw data is de-noised, de-trended, normalized etc., before being considered further. The feature selection stage ensures selection of a set of relevant statistical features of the pre-processed data. The feature extraction stage ensures calculation and retrieval of the selected statistical features. Finally, the feature cross-validation stage ensures that only the most indicative features in a set are being used. Given such a list of strong indicators a trading algorithm may start to form patterns out of the selected data features. The formed patterns may then be utilized in signal generation and other decision-taking processes. Formation of trade-exploitable patterns is often based on recognition of co-relations between different features of the input data.

Data Mining is a method of discovery of useful information in data, and is particularly applicable to this building block of an algorithmic trading strategy. Finding non-obvious patterns in datasets is comparable to identification of indicative features relevant to trading strategies. The processes used for feature selection and cross-validation are often similar in both fields.

The Signal Generation stage [Chootong and Sornil, 2012] is another building block of algorithmic trading strategy. It is responsible for generation of signals that define the behaviour of a trading model, on the basis of the most indicative features identified during the pre-trade analysis process. Trading strategies can be defined in terms of tasks (actions) they are to perform. Such tasks are constrained by the given information, or more precisely by the sets of features of the given information identified as influential during the pre-trade analysis process. When a given set of constraints is met, it defines a signal for the execution of particular task of the strategy process, e.g., signal for issuing orders or maintaining, closing and rebalancing market positions. Furthermore, the stage may also be responsible for the generation of support signals for e.g., risk management and money management.

There are two approaches to the process of the definition of behaviour of models that drive execution of risk management, money management and trade execution blocks of algorithmic trading strategies, in

the next time horizon. These are the static and dynamic approaches. In the static approach, the behaviour of a model is defined once, typically in a form of a set of logic rules with thresholds; where the values of the thresholds can be optimised when the model no longer passes the fitness test of its utility function. In the dynamic approach, the definition of the behaviour of a strategy is defined in a continuous manner during a learning process.

From the perspective of the signal generation stage Artificial Neural Networks and Support Vector Machines are able to model complex, non-linear relationships in data, and therefore are particularly suitable for mapping data inputs (in the form of the most indicative features identified during the pre-trade analysis process) to data outputs (in the form of signals that define the trading, the risk and money management behavioural policy). Furthermore, Evolutionary Algorithms are particularly suitable to the process of optimisation of parameters of static strategies. In case of a dynamic control system the machine learning approaches, i.e. the Reinforcement Learning methodologies are particularly applicable.

The Trade Execution stage [Cesari et al., 2012] can be viewed as a process that defines optimal execution of order instructions and transactions. After the trading signal has been generated, the risk and transaction costs estimated and accepted, and the assets allocated for a specific investment horizon the execution functionality issues a set of order instructions (of a specific type and with required parameters), and manages the execution of orders (until successful or unsuccessful completion of transactions). For best execution and to minimize the market impact, advanced trading platforms provide a smart order routing [Foucault and Menkveld, 2008] functionality that allows the division of large instructions into small, more optimal orders.

The Post-Trade Analysis stage [Kissell and Malamut, 2005] is responsible for generation and evaluation of results of the trading activity, such as the final execution price, the account profit & loss (P&L) and the remaining margin. The results of such analysis are then fed back to the pre-trade analysis process and can be used to improve the future actions of the strategy.

The Money Management (asset allocation) stage [Tsagaris et al., 2012] is a decision process responsible for the division and assignment of available funds. The process can be applied to a portfolio of orders, a portfolio of trading models managing the orders, a portfolio of accounts managing funds of the trading models, a portfolio of institutions managing the accounts etc. Consequently, the process is applicable to multiple levels of the AT abstraction.

The Risk Management stage ([McNeil et al., 2005], [Clark, 2010], [Berkowitz, 2001]) in some respects is similar to the asset allocation approaches, where multiple levels of risk are identified and the risk features are monitored to influence behaviour of trading algorithms according to the generated risk signals.

2.3.1 AT Classification

The most natural way of classifying AT [Domowitz, 1993] is on the basis of standard strategy classes and major elements of the strategies. The major classes of strategies that one can differentiate are: a) Arbitrage-based strategies, b) Hedging strategies, c) Directional strategies and d) Portfolio-based strategies.

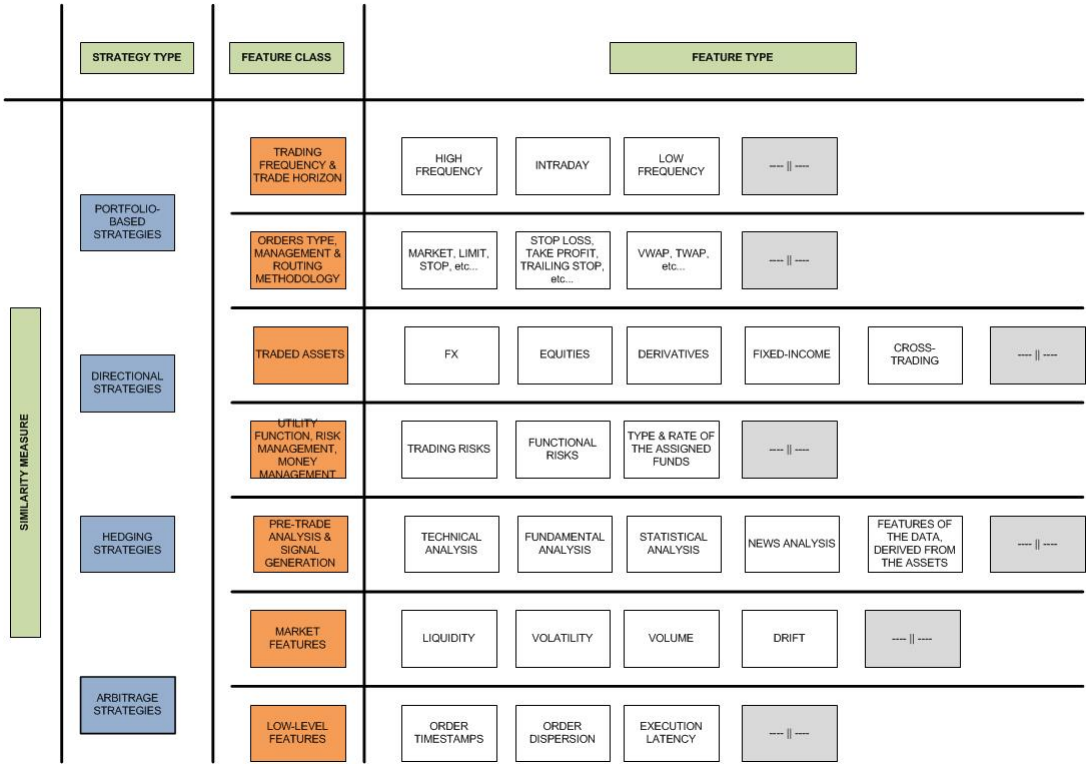


Figure 2.2: Low-level Algorithmic Trading Taxonomy.

The Arbitrage-based strategies: [Ross, 1973] are the class of strategies that attempt to benefit from the difference in the price of simultaneously purchased and sold assets. A variety of possibilities may be explored in this group, such as the classical transactions on the same or similar financial instruments traded on different markets or the statistical arbitrage.

The Hedging strategies: [StuIz, 1984] are a class of strategies that attempts to hedge the risk of a transaction by purchasing opposite positions in the market. This class of transactions benefits from the fact that more aggressive and hence riskier strategies can be utilized in trading since the risk of losses is minimised. It is often possible to mix the hedging and portfolio-based strategy classes (the major difference being the amount of assets traded by a strategy).

The Directional strategies: [Fiorenzani et al., 2012] are a class of strategies based on the assumption of a correct prediction of a market movement. A majority of the breakout, mean-reversal, trend following and statistical prediction strategies falls into this category.

The Portfolio-based strategies: that originate from the [Markowitz, 1959] are a class of strategies based on the idea of a diversified portfolio of asset classes with different risk exposure characteristics. The main driving forces of these types of strategies are pricing, risk management, replication, neutralisation and rebalancing.

The above major classes of strategies can also be characterized by a set of features that describe the strategies' key characteristics, e.g., trading frequency, trading horizon and similar. The trading frequency is a measure of generated trades in a specified period of time, while the trade horizon is the amount of time a particular position was held on the market. If one was aiming at comparing trading strategies, the trading frequency and the trading horizon features would be measured, and, if the obtained results were similar, it could indicate that the strategies may share a similar underlying functionality.

Algorithmic trading strategies can also be classified in terms of the order types they issue, the functionalities used to manage the already issued orders, and also the methods employed to route orders. If the compared strategies use similar order sequences, if they manage orders with similar mechanisms and they route orders to brokers using similar policies, then the compared strategies would share similar trading mechanisms.

To enable an algorithmic trading strategy to take successful trading decisions, every strategy needs to be able to handle a variety of asset classes and analytical information derived from the assets the strategy is trading. The comparison of assets that a given strategy trades (and the assets it takes into account during the pre-trade analysis process) gives a strong indication of the similarity of the strategies. An analogy can be made to an analysis of a black box, where we analyse inputs (data feeds) and outputs (orders) of the black box to evaluate how the black box performs.

Advanced algorithmic trading strategies are designed to be able to maximise or minimise some predefined utility function. This function can be a simple, single-factor function i.e. maximisation of profit, or it may be defined as a multi factor function (with variety of different features) that needs to be maximised or minimised. Quite often a part of the utility function is related to the risk and money management features of the strategy. An attempt can be made to recognize such general factors and use them in a similarity measure.

The markets have the power to affect the execution of strategies, especially when the trading strategies respond to the market events or try to predict the events. It should be possible to capture some of the features on the basis of which the strategies trade (by monitoring the state of the markets at or around the time of trades).

Finally, a set of low-level strategy features may also help to identify similarities between different strategies (the concept of strategy similarity measure is important from the point of view of strategy diversi-

fication). The timestamps, the sizes and the dispersion of orders should confirm/negate the fact that the compared strategies act on the same or similar underlying market patterns. At the same time, a similar execution latency, measured at the same location and the same machine, should at least strengthen the assumption of similarity of the underlying functionalities.

2.4 System Architectures for AT

Algorithmic Trading and risk platforms can be considered three-tier systems, where the first tier consists of external communication elements, the second tier is responsible for internal communication, and finally the third layer provides the actual business logic. Cisco's Trading Floor [Risca, 2008] presents a good representation of a typical architecture of a trading & risk system.

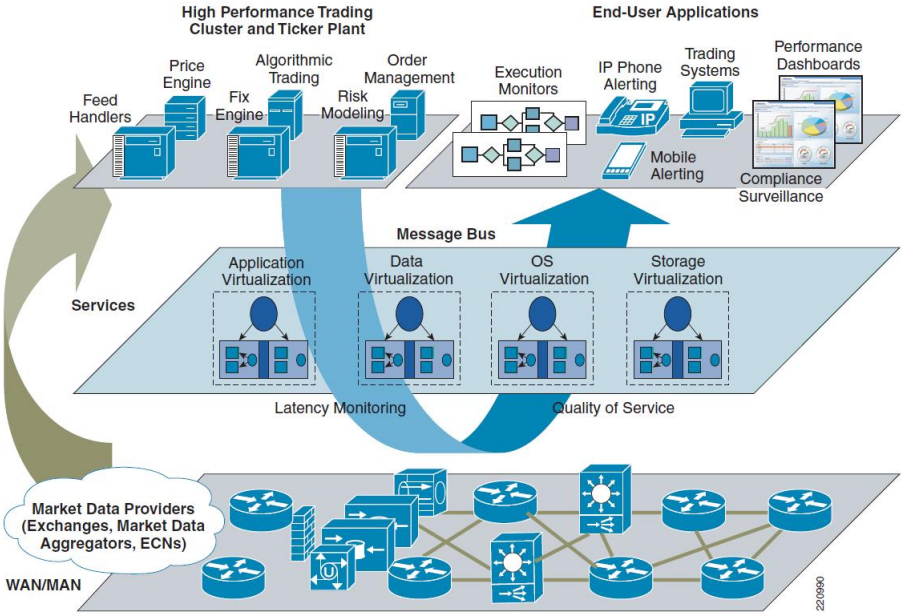


Figure 2.3: Cisco Trading Floor Architecture [Risca, 2008], where ECN is an acronym for Electronic Communication Network.

2.4.1 External Communication Layer

The bottom layer of the Cisco's Trading Floor represents a wider, external communication layer and a low-level hardware & network architecture of the trading floor. This also comprises the external systems with which the trading floor exchanges information. The performance of the platform depends on various interrelated factors with the most important one being the speed of the underlying hardware and networks. Consequently, when considering the architecture of this type of financial systems, the emphasis needs to be put on the hardware and connectivity. Apart from the hardware and network, the operating systems on which the platform relies must also be chosen with care. In majority of cases the platform is distributed amongst many servers in a cloud (the operating system on the servers needs to be able to deliver such distributed functionality). When speed is of essence (and in majority of cases it is),

the customization of the underlying hardware, software and network architecture is paramount. One can consider dedicated hardware for specific needs of either a fast execution, storage or multi-processing. Kernels of utilized operating systems can be tailored to specific needs of the underlying hardware as well as the platform. All the unnecessary drivers may be removed, and the kernel may be optimized and recompiled to support a particular need. The network architecture may also be optimized by reorganization of its structure, collocation of services with markets and exchanges, and also by utilization of dedicated connection lines. In addition to the architecture and organization of the network, the communication protocols also play a major role; they need to be able to transport information - quite often a substantial volume of it - in the quickest possible way, with a guaranteed delivery.

The Cisco's Trading Floor also defines the external communication layer from the data-streams perspective, and it specifies the 'market data' stream and the 'trading orders' stream. The 'market data' stream [Risca, 2008] *"carries pricing information for financial instruments, news, and other value-added information such as analytics. It is unidirectional and very latency sensitive, typically delivered over UDP multicast. It is measured in updates/sec. and in Mbps. Market data flows from one or multiple external feeds, coming from market data providers like stock exchanges, data aggregators, and ECNs. Each provider has their own market data format. The data is received by feed handlers, specialized applications which normalize and clean the data and then send it to data consumers, such as pricing engines, algorithmic trading applications, or human traders"*. The 'trading orders' stream [Risca, 2008] *"is a type of traffic that carries the actual trades. It is bi-directional and very latency sensitive. It is measured in messages/sec. and Mbps. The orders originate from a buy side or sell side firm and are sent to trading venues like an Exchange or ECN for execution"*.

2.4.2 Internal Communication Layer

The middle layer of the Ciscos Trading Floor is a service layer that allows internal communication within the platform. The internal communication is a backbone of every enterprise-size platform. It needs to be reliable and fast, it also needs to be flexible and modular. The internal communication is necessary as a result of the distribution of elements of the platform. A modern approach to development of trading & risk platforms incorporates both the service and event-oriented methodologies.

From the functional point of view, the internal communication layer of an enterprise trading & risk environment can be divided into three types of application components: the components that produce information, the components that consume information and the components that both publish and subscribe. Every middleware communication bus provides multiple data streams (queues or memory-less topics) that allow communication of publishers with subscribers. Such data streams ensure that groups of subscribers receive only the relevant information provided by the groups of publishers connected to a particular data stream.

2.4.3 Business Logic Layer

The top layer of the Ciscos Trading Floor describes the logic behind all the modules of the platform. Typically, this includes: the connectivity engines, data aggregation units, data processing elements, order management & routing, execution elements, trading strategy elements, graphical and programmatic interfaces, monitoring tools and many more, depending on the requirements and specification. These are described below.

The FIX (Financial Information eXchange) Engine module: The most common format for the order and market data transport is the FIX protocol ([Cameron, 2009]). The applications which handle FIX messages are called FIX Engines. They are designed to send/receive high volumes of FIX messages, translate the messages to relevant internal formats of the architecture and interface it to all the relevant modules of the platform. The FIX Engine must be able to handle multiple streams at once, and be able to deal with reconnections, requotes and message synchronization issues. The FIX Engine also needs to be capable of handling different versions of the FIX protocol and non-standard FIX messages customized by the information providers.

The Feed Handlers: When the market data is handled by the FIX Engine and passed to the rest of the elements of the system, it needs to be cleansed, processed and aggregated. The Feed Handler module provides functionalities to clean the data from outliers, gaps and other statistically significant problems to make sure that the data is of high quality. The Feed Handler module is also responsible for processing the data to provide data analytics. It also provides aggregation facilities to store the cleansed and processed data in the database, for a future use by other elements.

The Price Engine: According to ([Consulting, 2008]), the goal of the engine is to allow an automatic pricing of securities, using inputs from different market sources and adopting financially consistent pricing models. The Price Engine needs to be designed to support concurrent, real-time calculations for hundreds of assets at the same time. The engine leverages the robust financial models and they need to allow only a minimal number of market operations to efficiently control large amount of securities. The engine processes prices from different sources, and performs data compression and filtering to obtain an internal best price upon which all subsequent calculations are based. When the reference securities change their market prices, the engine handles the logical dependencies, sending recalculation signals to the correct models and instruments.

The pricing models utilized in the engine generate prices for the target assets using one or more benchmark securities. Usually, benchmark instruments are chosen for their high liquidity and because they consistently represent market variations. The Price Engine models map the benchmark variations into price variations, for the target security, by applying a pricing parameter. The financial meaning of the pricing parameter depends on the model. All pricing models in the Price Engine should be automatically

calibrated to the market prices. This automatically defines the pricing parameter in order to reproduce the market best prices.

The Risk Modelling: The risk engine ([Kumar, 2009]) is a management system designed to identify all significant risks associated with a given set of assets. The engine provides a set of comprehensive measurement tools to assess, control and communicate risk. It incorporates the advanced statistical analyses, the proprietary valuation models provided by the Price Engine, and the advanced data aggregation provided by the Feed Handler module.

One of the goals of the engine is to identify and manage a risk exposure within a given portfolio of assets and calculate the full Profit and Loss (P&L) distribution at every level of exposure. Understanding financial risk requires more than one single number. Therefore, the engine needs to be able to provide a robust analytical framework to calculate and communicate a variety of statistics and measures. The engine often contains an analytical suite including a library of historical market-stress scenarios, as well as a set of functionalities, to evaluate trading asset sensitivities to a variety of selected risk factors. The Risk Engine uses a robust analytical framework that includes a Monte Carlo simulation-based Value-at-Risk (VaR) analysis as well as other risk measures available, such as exposures and sensitivities to 'the Greeks' (delta, duration, gamma, etc.).

The Order Management: The Engine (sometimes called Order Management System, OMS) is used for a rapid order entry and processing. It facilitates and manages the order execution, typically through the FIX Engine. The Order Management systems often aggregate the orders from different channels to provide a uniform view.

The engine allows input of single and multi-asset orders to the system for routing to the pre-established destinations (often through a Smart Order Routing module). Apart from issuing the orders, the engine also allows to change, cancel and update orders. Additional functionality of the engine is an ability to handle execution reports and an ability to access information on orders entered into the system, including detail on all open orders and on previously completed orders.

The Algorithmic Trading: The module supports the use of trading strategies that issue and manage orders with use of an embedded logic coded within the algorithm of the strategy. This allows the logic of the algorithm to decide on aspects of the order such as the optimal timing, price, or quantity of the order. The module supports a variety of strategies, including market making, inter-market spreading, arbitrage, directional or pure speculation. The investment decision and implementation may be augmented at any stage and is usually supported by the analysis of data provided by the Feed Handlers, the Price Engine, the Risk Modelling and the Order Management. The module plays a role of a manager of all the handled strategies, providing functionality to enable/disable selected strategies and modify factors of the strategies that are being executed.

The Trading Systems and The Alerting modules: Provide a means of manual trading by allowing their users a visual access to a set of GUIs and tools. The two visualize the states of the markets and selected assets, and also provide a graphical representation of analytical data. The trading systems also provide the interfaces to execute orders and maintain positions on the market. The module needs to be characterized by a strong charting capabilities and functionalities that allow users a quick response to different states of the markets. The Alerting modules are an extra means of communicating with clients and are usually not part of the architecture, but need to be considered as a part of a trading floor.

The Execution Monitors, The Performance Dashboards and The Compliance Surveillance: These modules allow monitoring of the architecture's performance. The modules also allow the compliance of the performance and the behaviour with the predefined policies. This is to reduce the risk of malfunction of modules of the architecture and to allow a quick response to the threats and the issues within the architecture.

2.5 Conclusions & Summary

The intention of the chapter was to introduce the reader to the background knowledge of experimental computational simulation environments that will pave the way for the chapters describing experiments. A literature review of related subjects was also performed to provide points of reference to the current state of affairs in computational finance and financial economics (algorithmic trading being part of the fields).

The introduced background knowledge (mainly the information about classification and modelling of algorithmic trading, as well as information about simulation types for experimental computational simulation environments) is based on authors own experience of work as a quantitative developer and quantitative analyst in financial industry.

The reviewed pieces of scientific literature were selected to familiarize the reader with various technologies, architectures and components that can form part of a unique software architecture of experimental computational simulation environment for financial economics.

The key objective of the thesis is to holistically study the behaviour of experimental environments by allowing access to it to academic community via a series of trading competitions. The review of possible simulation types, features of such environments as well as the modelling domain was aimed at allowing the reader to find relationships between the proposed research methodology (Introduction Chapter), the scientific questions (experiment Chapters) and the obtained results (Evaluation Chapter).

Chapter 3

ELECTRONIC MARKET SIMULATION MODELS

This chapter describes the important concept of a model of an electronic trading exchange in the context of simulation of trading, systemic risk and economic stability. The chapter then presents results of the implementation of two simulation models, namely the end-of-day model and the live model utilized in UCL's algorithmic trading competitions and manual trading competitions. The models contain key features of electronic exchanges, including functionalities for: the order book, the matching engine, the account management, the P&L clearing, the event-based price streaming and order instruction/confirmation messaging.

3.1 Electronic Markets (EMs)

The concept of an electronic market is based on the exchange of financial assets via electronic communication systems. The electronic markets are central hubs for bilateral exchange of securities in a well-defined, contracted and controlled manner. Modern markets are based on electronic networks and replace Open Outcry Exchanges with the advantage of an increased speed, reduced costs of transaction, and programmatic access. This allows the simulation of work of the electronic markets, to model the processes of trade, systemic risk and economic stability.

Financial institutions use simulated trading to 'fine tune' their algorithms prior to deploying the algorithms for real trading [Katz and McCORMICK, 2000]. The exchange simulators provide their users with a possibility to participate in a market, receive live feeds and monitor their profit or loss without bearing the risks associated with the real-time market trading. These simulated environments are used not only by the participants working to gain market experience but also by the risk managers to test the risk applications and by the software developers to build and test quantitative financial products.

To enable trading and transaction control the participants of electronic trading systems are required

[Madura, 2012] to hold asset accounts with the exchanges, brokers or clearing houses (where the three are interconnected to provide the Accountancy, Clearing and P&L calculation services to the users). The participants are then provided with a user interface that can be utilised to communicate trading instructions from the participants to the exchange and the state of the account and the transaction confirmations from the exchange to the participants. The participants can deposit assets to the provided account and the value of such deposited assets is represented electronically in the system. Each time a participant issues an order instruction, a matching and exchange process is triggered by matching the best bid with the best ask offer for a given asset, with the obtained assets being transferred to the participant's account in a form of a position in the traded asset. Each transaction is considered a contract and if required, the actual physical assets are exchanged 'post factum' on the basis of such contract being agreed upon transaction in the system.

The purpose of the financial exchanges is to enable asset trading. Asset classes are groups of securities with varying degrees of risk and different investment characteristics (including but not limited to the potential for delivering returns and performance in different market conditions with a specific level of risk). The main traded asset classes include: a) foreign exchange b) equities, c) fixed income, d) derivatives, and e) commodities. For completeness:

Foreign exchange (currencies): is an asset class that involves the exchange of one national currency for another, or the conversion of one currency into another currency.

Equities (shares): are issued by a public limited company. Equities have the potential to generate income in two ways: a) the capital growth can be received through increases in the share price, or b) the income can be received in the form of dividends. Neither of these is guaranteed and there is always a risk that the share price will fall below the level at which it was obtained.

Fixed income (bonds): are issued by companies and governments as a way of raising money. They are considered to offer stable returns (normally a regular stream of a fixed amount of income over a specified period of time, and a promise to return the capital to the investors on a set date in the future), and to be of lower risk (hence deliver lower returns) than equities. Once a bond has been issued, it is a subject of transactions between the investors without the involvement of the issuer.

Derivatives (e.g., futures, options): are contracts between two parties that define conditions (the dates, the resulting values, the notional amount) under which payments are to be made between the involved parties. The values of such financial instruments are derived from one or more of the underlying assets. The derivatives may broadly be categorized as option and obligation contracts. Option contracts provide the buyer with the right, but not the obligation to enter the contract under the terms specified. Obligation contracts obligate the contractual parties to the terms over the duration of the contract.

Commodities: can be considered as any bulk of goods that is a physical substance (e.g., coffee, oil, gas and many metals) and is traded on an exchange. Commodities are typically bought and sold on the futures market, where the producers combine with the manufacturers and investors to form a market. The main attraction of investing in commodities is that their performance has a limited correlation with the other asset classes and that, consequently, they offer a clear diversification.

3.1.1 Electronic Market infrastructure

Large-scale software systems, electronic markets being one of them, are typically built in a modular fashion. The technical prerequisites for electronic market [Hu et al., 2012], [Su and Iyengar, 2012], [Lybäck and Boman, 2004] can be characterised as follows (Figure 3.1): a) a trade communication protocol - to define ways of communicating with EMs, b) a matching & exchange engine - to define transaction rules and enable trading, c) an accountancy and clearing functionality - to define who can trade and what is the state of their holdings, and d) a risk management functionality - to manage exposure to loss.

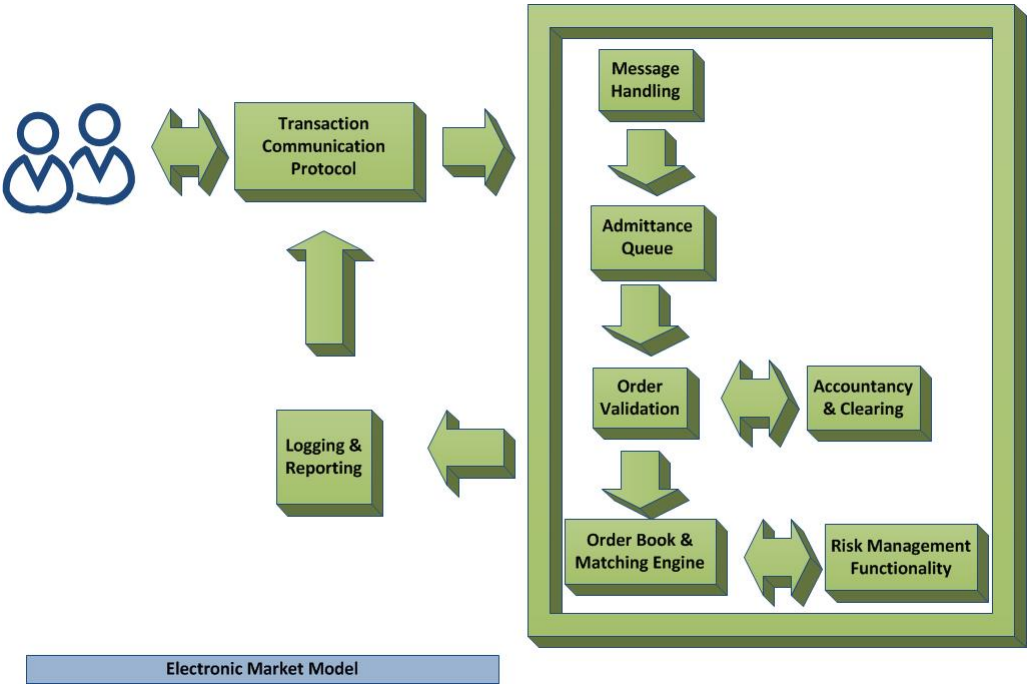


Figure 3.1: Key elements of an Electronic Market.

Each of these model components are now described:

The trade communication protocol: is a specification of the information instructions that can be used to communicate orders and requests issued by the market participant to the exchange as well as events generated by the exchange to inform the participant of the state of the exchange. One example of such protocol is FIX, the Financial Information eXchange protocol [Fixprotocol, 2013] for the computer communication of information.

FIX can be roughly divided into the session level messages and the application level messages. The session level messages enable the exchange and its users to establish and maintain communication. The application level messages enable communication of the pre-trade, trade and post-trade information including:

- **Pre-Trade information includes:** tradable security definition, market data
- **Trade information includes:** various types of orders and order execution reports, order status requests and responses
- **Post-Trade information includes:** account and position requests and responses

Matching engine: is a functionality enabling representation of the order book of the exchange, the execution of the order matching, the assets exchange and the market data generation. A centralized order book contains a listing of the buy and sell orders organized on the basis of the tradable securities, the order type, the anticipated price (if specified), the order quantity, and the time of arrival. The execution and order matching functionality operates on the basis of the order book, where it evaluates the best bid/ask price at a time. The matching process occurs in accordance with the exchange-defined algorithm. Generally, if at least one order is available on the other side of the order book, the subjected order is matched with, and executed against, the best and the oldest offer on the other side. Such a process continues until the full size of the order is executed or until there are no more suitable orders.

On the basis of these standards, the electronic exchange system calculates the opening prices, the prices during continuous trading and the closing prices of all securities, with the auction process being taken when the system is outside of the continuous trading hours.

Accountancy & clearing functionality: are two related functionalities responsible for storing and managing information about users, providing authentication and authorization facilities as well as keeping the state of users' accounts, positions, leverage, available margin and other managed asset statistics up to date. Such information is required to maintain participant sessions and can be either requested by the participants or can be streamed to the subscribed users.

Risk management functionality: is an element of the exchange that ensures the participant can not lose more than anticipated on their transactions. This is typically organized on the basis of the available margin. When an account's available margin falls below a certain predefined threshold the functionality prevents the participants from a further increase of margin and closes their active positions to prevent further loss. In more advanced exchange systems the functionality is also constructed to prevent algorithm risk.

Apart from the typical software systems prerequisites there are also infrastructure considerations that need to be taken into account in construction of the electronic market, these can be identified as follows:

a) the hardware, b) the electronic network and service distribution, c) the operating systems and computer cluster set-up.

Hardware prerequisite is an important consideration due to the fact that the EMs are particularly demanding in terms of the resources they use. This applies both to the computing power (often millions of computational tasks executed concurrently at all times) and to the memory requirements (due to the increasing volume of data generated by the exchanges). Utilisation of suitable hardware is vital to the performance and efficiency of an exchange. The actual matching engine functionalities of an exchange are often deployed as a dual-computer hot-backup mechanism characterized by a high degree of fault tolerance and high availability, to eliminate the single point of failure [Su and Iyengar, 2013].

Network and Services: considerations for the EMs are related to the high throughput, the low latency and scalability. The underlying architecture should be characterized by a predictable performance as well as reliability in the information delivery. [Ciliendo, 2007] suggests tracing the low-level network patterns and monitoring the actual speed of the network interface as ways to improve network performance. Furthermore, utilisation of the publish/subscribe patterns as well as the request/reply patterns can improve the scalability of services.

Cluster: consisting of the distributed storage and the compute services utilised by the EMs needs to be characterized by the reliability, the availability, the performance and the security. Consequently, the EMs often run on customised versions of the OSs. To support the services of the cluster the OSs may be tailored to the specific needs of both the hardware on which the EMs runs and the network. [Harris, 2004] suggests that the enterprise operating system implementations found in the massively parallel scientific applications are suitable for financial applications. [Ciliendo, 2007] suggests that a system designed for the optimal performance should have a minimum of unnecessary tasks and subsystems running, and should be tailored towards a specific workload (such cluster will perform better under the intended load characteristics but is likely to perform worse for different workload patterns).

3.1.2 Standard Order Instructions

One part of the previously mentioned trade communication protocol is a set of Standard Order Instructions that define 'permissible actions' the participant can issue to the exchange for it to trigger the exchange of assets. A standard set of order instructions accepted in all modern electronic markets includes: a) a market order instruction, b) a limit order instruction, and c) a stop order instruction.

Market Order: is an order instruction to buy or sell a selected tradable security immediately at the best available price. The order is likely to be executed as it does not contain restrictions on the price or the time of execution and it often imposes lower commissions than the other order types. The only time this order may not be executed is when no sufficient liquidity is available on the market.

Limit Order: is an order instruction placed to buy or sell a set number of a selected tradable security at a specified price or better. The limit orders are useful when applied to a low-volume market or highly volatile stocks. This is because when the trade gets executed, traders get the specified, or better, bid or ask price.

Stop Order: is an order instruction to buy or sell a selected tradable security when its price surpasses a particular point. Once the predefined entry/exit price point is crossed, the stop order becomes a market order, consequently providing a greater probability of achieving the predetermined entry or exit price. The stop orders are often used in situations where the traders want to specify an investment price horizon on the traded position, limiting the loss or locking on the profit. With the stop order being turned into a market order, the instruction is not guaranteed to be getting the desired entry/exit price.

3.1.3 Order Book and Matching Process

When considering the construction of an electronic market model it is crucial to comprehend the way the order book works and to understand the matching process. For consistency the following section will present such process from two perspectives: a) the perspective of an algorithmic trading strategy, and b) the perspective of an electronic market.

3.1.3.1 AT Process Flow

AT systems are driven by data, as a consequence of this it is crucial for such systems to have persistent access to data at all times. There are two standard data access paradigms for AT systems: a) the 'event-driven', and b) the 'service-oriented' paradigms. The flow of the particular processes in AT for both of the paradigms are described below:

The Event-Driven Paradigm: such systems work on the basis of events being generated by occurrences of some pre-defined circumstances. For AT systems such circumstances can be: the arrival of a new price quote from an electronic market, the generation of a new data feature, the update of an order instruction executed on an exchange, the update of an account status and many more. The advantage of the asynchronous event-driven paradigm in AT is the minimal latency of an event notification right after the actual event happens. Given that the AT systems need to be characterized by high responsiveness this is obviously advantageous.

The Service-Oriented Paradigm: when the speed of execution is not a major drive, the service-oriented paradigm is a practical solution (where information, quite often large amounts of it, can be 'served' on a per-request basis). In this instance, the AT systems often run on one or more scheduled threads and pool large amounts of data every time the schedule triggers execution of a particular component of the AT system.

The two paradigms are utilized in the key action flows of AT systems in a following fashion:

In the event-driven approach to AT: the common action flows always start with a new event (in a form of a data feed, an order execution report or an account state update) being received by a component of a strategy. The following may be considered a typical flow of actions:

[A] The component receives a new tick: In this scenario the active element is handling newly generated tick updates:

[a1] Pre-trade analysis: Such information is then passed to the pre-trade analysis elements to perform analysis of the current state of the environment of the AT system.

[a2] Signal generation: The pre-trade analysis elements generate their own view of the multi-dimensional state of the world. On the basis of the values of such dimensions the AT system is able to generate a set of signals; instructions defining actions of the system with respect to its understanding of the environment and existing investment opportunities.

[a3] Trade execution: The generated instructions are then translated to orders and are executed on electronic exchanges by the trade execution element of the AT system.

[B] The component receives a new order execution report: In this scenario one or more order instructions are already issued by the trade execution element of the AT and one of the components of the AT receives a new event indicating that the state of one of the issued instructions has changed:

[b1] Post-trade analysis: Such execution element is then passed to the post-trade analysis component of the AT system that updates its measures of risk with respect to all the currently held positions.

[b2] Signal generation: The values of the features defining risk levels are fed-back to the signal generation element. This updates the current view of the multi-dimensional state of the world held by the strategy and if the thresholds for signal generation are passed then the element generates a new set of actions that are to be executed by the strategy.

[b3] Trade execution: If the actions are turned into order instructions then the trade execution element translates them into orders and executes them with the electronic exchange.

[C] The component receives a new account update information: In this scenario an arbitrary amount of funds is deposited into the account used to trade with the electronic exchange and one or more positions is held in the account. When the state of the account changes (e.g., due to a change of value of a held position) the exchange generates an account update event that is received by one or more of the components of a strategy:

[c1] Pre-trade analysis: Such an account-update event is passed to the pre-trade analysis component of a strategy (often the money management element) and a set of performance measures is updated and recalculated with the information the event carried.

[c2] Signal generation: The newly obtained strategy-performance values are used to update the multi-dimensional view of the world the strategy holds and if performance thresholds are passed, a set of actions is generated by the signal generation component of the strategy.

[c3] Trade execution: As in the previous cases, if the actions can be translated into the order instructions, the trade execution element performs the task and executes orders with the electronic exchange.

In the service-oriented approach to AT: the common action flows start with one or more scheduled threads triggering the retrieval of an information from the data buffers. The following may be considered a typical flow of actions:

[A] Execution of a higher frequency schedule: In this scenario the active element is triggering retrieval of a buffer of the active and historic order execution reports. It is assumed that at this stage one or more order instructions is already issued and are being actively executed by the electronic exchange. The higher frequency schedule is to ensure control of the execution of orders before any other actions are undertaken. The buffers are used to track changes in states of orders and consequently to manage associated risks.

[a1] Post-trade analysis: The retrieved buffers are then passed to the post-trade analysis element of the strategy to update various utilised risk measures and to construct a new view of the multi-dimensional representation of the world in the strategy.

[a2] Signal generation: Based on the updated representation of the world the strategy then generates a set of actions, in the instance the thresholds on the features of the model are passed. Given the nature of the signals (generated on the basis of the risk measures) the most likely outcome is in the form of the order instructions, to change the size of held positions, to revaluate portfolio, or to close positions altogether.

[a3] Trade execution: If applicable, the actions are then translated into the order instructions by the trade execution element and the orders are executed with the electronic market exchange with which the system trades.

[B] Execution of a medium frequency schedule: In this scenario we assume that we want to control the state of the account more often than to perform trading actions, but less often than to perform control of the execution and involved risks. This enables money management and fund allocation.

For this purpose we want the AT system to retrieve the most up to date information about the state of the account (on which the system trades) and the active positions on this account:

[b1] Pre-trade analysis: The retrieved account information is provided to a set of performance measures working as part of the money management element in the pre-trade analysis component of the strategy. The values received from the updated measures are then provided to the multi-dimensional representation of the state of the world the AT system formed.

[b2] Signal generation: Based on the representation of the world, held by the strategy, the signal generation element evaluates whether the thresholds that indicate triggering of actions were passed and, if that is the case, the element generates signals and passes them to the trade execution element.

[b3] Trade execution: If applicable the element then translates the actions into order instructions and executes the instruction with the electronic market exchange.

[C] Execution of a low frequency schedule: The utilisation of a service-oriented paradigm in the AT system usually implies that the system requires larger amounts of data and the execution of order instructions is relatively low frequency. Therefore, in this scenario we assume that the AT system is trading on the basis of a portfolio of securities and revaluates the portfolio when it diverges too far from optimality. In this scenario the executed schedule retrieves buffers of price information either for all available securities or for a selected list of securities pre-defined for the portfolio:

[c1] Pre-trade analysis: The retrieved time-series are passed to the pre-trade analysis element that performs the analysis of the data. If necessary, it performs the optimal portfolio selection and generally forms a multi-dimensional representation of the state of the world at the time.

[c2] Signal generation: This model of the world is passed to the signal generation element that evaluates every dimension in the model and generates actions in the instance the thresholds for particular dimensions are passed.

[c3] Trade execution: The generated actions are passed to the trade execution element which, if applicable, translates the actions into order instructions and executes the orders with the electronic markets.

3.1.3.2 Exchange Process Flow

Based on the described order instructions, as well as the sources treating about the subject, e.g., [Hughes Jr et al., 2012] a trading process of an electronic market can be described as a set of the following steps:

[A] Admit order instruction to the order book: This is a process of receiving the order instructions issued by the participants via one or more UIs and processing it up to the point of putting it to the

order book or rejecting, either due to the problems with the order definition parameters or due to the lack of available funds.

[a1] Receive order instruction from a participant: The participant would utilise the previously specified communication protocol to a set of parameters defining the order instruction and would the send it either with a graphic or programmatic user interface to the electronic market. Such instruction, together with other instructions would be placed in an admittance queue for the orders that await processing.

[a2] Validate the order against faulty parameters: When one of the processing threads becomes available, it retrieves the oldest order from the queue and submits it to the verification and clearing process. The order verification process involves checking individual values of the provided parameters against a specified protocol. In the instance the values are incorrect, a rejection report is sent back to the participant that issued the instruction and such order is being disregarded. The eligible order instructions are all the types of new orders, modification orders and cancellation orders.

[a3] Clear the order against available margin: The process of order clearing involves checking the availability of funds that are necessary to cover the order. During the clearing process the algorithm would investigate the leverage, the available funds, the current price and size of the ordered security and, if the participant has sufficient margin available on their account, the order would be admitted further with a confirmation report being sent to the participant. In case the order is not cleared, a rejection report would be sent instead.

[a4] Admit the order to the order book: After the order was validated and cleared, it is ready to be admitted to the order book structure. The order book is typically organized on the basis of the security, followed by the bid/ask price, then the price value and finally a queue of orders organized on the first-in-first-out (FIFO)-basis.

[a5] Send the participant a confirmation that the order was accepted: Each time a state of the order changes an execution report, with an update of the state of the order, is sent back to the participant that issued the order. A typical flow of updates includes the update sent when the order is validated and cleared, the update sent when the order is partially filled, and finally, the update sent when the order is fully filled.

[B] Match orders in the order book: The process of matching orders in the order book is performed by a set of securities-processor-threads that stores and processes transaction information in the order book (that is typically stored in-memory), and later in the execution log that records the executed orders. During the operations on the order book for particular security, the order book is

exclusively accessed by assigned thread to prevent other processes from accessing the functionality. This is to increase the speed of the matching process. The incoming transactions are processed in a strictly first-in-first-out fashion.

[b1] Establish the current best (marker) price: The key step in order execution is to establish the current best price, which becomes a temporal marker for matching of the orders. This is true in both the case when the exchange in is the auction mode, and the case when it is in the continuous matching mode. In the auction mode the exchange would typically accept a batch of initial orders and would set the marker price as an average price of the batch. In the continuous matching process the marker price is set on the basis of: a) the last best price, b) the price of the remaining (partially filled) orders from the last batch, and c) the amount/density of orders in the surrounding price levels (above and below the current marker price). In the last case the key factors are the sizes of orders in the surrounding queues and their priorities in the queues.

[b2] Retrieve next batch of the bid/ask orders: Once the temporal marker for a security price is established, the security-processor-thread can calculate the next optimal size of a batch of orders which it aims to match and execute it in the new cycle. This is followed by a set of operations of retrieval, processing and matching. The operations are executed on information that is queued in the marker price level order queue. The concept of the optimal size of a batch differs from one exchange to another, and is typically selected to fit particular goals of an exchange and its matching priorities. The selected batch sizes can exceed current marker price-level.

[b3] Match and fill orders: The batch of orders retrieved from the order book contains both the bid and the ask side. Such a batch is then pre-processed with respect to the priority of instructions on each of the side. After that, the matching process starts to match the bid-side orders with the ask-side orders, saving the information about each of the partial and full order fills, on each of the sides, to the log. When the process exhausts all the available orders (on the current marker price-level) it moves the marker to the next price-level (provided that the marker is still crossed). This continues until the process exhausts the entire bath of orders to zero on one of the sides. The remaining orders are returned to the order book and the next cycle is initialized. For a received bid-side order the order is traded if its bid price is higher than the current marker ask price. For a received ask-side order, the order is traded if the ask order price is lower than the current marker bid price. On occasion the marker prices can shift during batch execution (e.g., if the exchange is obliged to accept the best national price for a given security, streamed from external sources). Finally, the executed batch can

contain supervisory instructions (apart from the new orders and the best national prices) in the form of the position closing, order cancellation or order modification instructions. The supervisory instructions are not part of the matching or marker price changing process but are responsible for amending the state of the order instructions that already reside in the order book.

[b4] Send the owners of the orders a filling process information: Once the transaction information (in the form of the partial and full order-fills) is stored in the order book's activity log, the activities may be dispatched to the interested parties. Such approach is selected due to the relative latency-cost of information distribution to the external functionalities.

[C] Generate a price quote: A temporal summary of the best (and a few close to the best) price-levels, of the traded securities, is generated by the electronic market after every matching cycle. Such summary is then sent back to each interested party in a form of the price quotes. The price quotes are based on the log of the executed orders, containing the transaction information that includes the prices, the trading volumes and the basic information about the quoted securities. Thanks to the price quotes, all the interested parties are always up-to-date with the state of the traded securities and can take further trading decisions when necessary. The following are typical steps taken by the exchanges to generate the price quotes:

[c1] Evaluate the order book queues and set a threshold for issuing the next price quote: The orders, queued at different price-levels in the order book, are matched in batches by the security-processor-threads and the information about such filled transactions is dispatched to the log (together with the surrounding levels of prices and order sizes on the price-levels). The exchanges, typically, have their own algorithms for establishing the current best price (which affects the next price level that is to be matched) and the order bath size for the processing. The best-price marker is typically affected by the remaining partially filled orders, from the previous execution batches, as well as the distribution of the sizes of queued orders at the closest price-levels.

[c2] Track the order filling process for the cycle: To generate a price quote, after every batch is admitted to the log, the quote-generation-functionality establishes the average best price for the batch. The log also records the state of the order queues at the surrounding price levels, including the order volumes. On the basis of such information the functionality generates full book price quotes, where the full book is typically represented by 7 to 10 price levels with the transaction volumes on each level.

[c3] Issue a next price quote: When the next full book quote for particular security is dispatched, it is streamed to all the subscribed sessions interested in this stream of quotes. Mod-

ern exchanges utilise the producer/consumer messaging patterns to stream the price quotes. Due to the amount of quotes and the lack of need for the message persistence of delivery, the exchanges also utilise User Datagram Protocol (UDP), a multicast protocols for the quote delivery.

3.2 Software Engineering principles and drivers

Various interesting aspects of software engineering can be taken into account while considering the functionalities of electronic markets. These can be the principles that can inform the creation of such systems, the typical action flows in such systems as well as the typical participants, their requirements and their key drives. The combination of such factors affect the way the electronic markets work and provide their services. For completeness, a description of the Algorithmic Trading development process is also included to give the reader an overview of the effects of Algorithmic Trading on the modern electronic exchange design.

3.2.1 Market Participants

The typical electronic market participants can be divided into the following six categories: a) the quantitative developers, b) the quantitative analysts, c) the traders, d) the portfolio managers, e) the trade-floor managers and f) the risk managers.

The Quantitative Analysts: are the creators of the different AT models and measures of AT risk & performance characteristics. Their work focuses mainly on the applications of the complex numerical methods (statistics, financial maths, econophysics, machine learning) in finance. Quantitative Developers (Programmers) are a particular type of 'quants' specializing in low-level implementations of numerical methods.

The Traders: are the active market participants specializing in various investment methodologies involving tradable securities. The AT traders utilize work of the 'quants' to automate one or more stages of their decision making, trading and evaluation process.

The Trading Floor Managers: are responsible for managing larger groups of traders and their strategies. The focus of this participant is on the organization and coordination of work of the trading floor and on ensuring the optimal performance of the floor with respect to the predefined goals and quantitative measures evaluating the goals.

The Portfolio Managers: are responsible for management of larger portfolios of tradable securities on the basis of longer investment horizons. While the Traders typically engage in the arbitrage, speculation, high-frequency or intra-day trading, the Portfolio Managers work on the basis of stable, long term returns with a limited appetite for risk.

The Risk Managers: are not actively engaged in the investment process but are rather the assessors of the different types of risks for the Traders, Portfolio Managers and Trading Floor Managers. The Risk Managers engage with the 'quants' to build various risk measures, while the Traders and Portfolio Managers often work with the 'quants' on AT models.

3.2.2 Trading Approaches

The market participants can be characterised and classified by their approaches to trading. The number of active participants on the market and their approaches to trading can affect the running of the elements of an exchange. Consequently, the understanding of trading approaches of typical classes of participants is important in the design and maintenance of an exchange.

The 'intraday trading' is the most common trading approach, with a few dozen of transactions being executed to adjust and maintain positions through the day. Such positions are then being closed at the end of every day. In this approach the Traders usually look for the daily trends and the news-announcements they can benefit from.

Another popular approach is the high-frequency approach, where the Traders attempt to benefit from the extremely short market inefficiencies by generating bursts of high frequency transactions. In this approach the Traders look for the inefficiencies in pricing, volumes and liquidity and the positions are not being held for longer than few seconds.

Finally, a third popular approach across the Traders is the long-term portfolio asset allocation, where the Traders maintain positions on the basis of long-term trends, social sentiments and fundamental economic prognosis.

3.2.3 Trading Stimulus

As previously stated, while considering functionalities of the electronic market models one should investigate the market behaviours that can inform the creation of such systems. This may include an attempt to identify the typical stimulus for traders and their models. It may also include an attempt to infer a typical set of actions such strategies may exhibit with respect to the stimulus. Thanks to the data collected during the course of the UCL AT Competitions (details about the competitions organized with use of the electronic market simulators described in this Chapter can be found in the Evaluation Chapter) the author had a unique chance to identify the typical trading stimuli exhibited by the participants. These can be summarized as follows:

Stimulus based on time, traded volumes and available liquidity: the timing, margin, traded volumes and available liquidity should be considered the most basic stimulus that influence the behaviour of every Trader and, consequently, are reflected in the Traders trading strategies.

Stimulus based on P&L, risk and performance information: are a feedback stimulus on the basis of the already held positions in the traded securities. The traders may apply performance and risk measures to their positions and perform the analysis on the types of actions that can be beneficial at a given threshold of a measure that is being taken into account.

Stimulus based on fundamental analysis: the Traders may perform the fundamental analysis of a state of particular security and may evaluate whether the security is undervalued or overvalued. On this basis they may create a strategy that will attempt to exploit the security's intrinsic value. The typical behaviour of the Trader may involve the fundamental analysis and selection of the most promising securities on such basis.

Stimulus based on technical/quantitative analysis: the Traders may perform the statistical analysis of historical data on a set of chosen technical indicators or quantitative measures to attempt to spot the exploitable patterns in the data. On the basis of such patterns the traders may then attempt to create a strategy.

Stimulus based on news and social analysis: the Traders may perform analysis of news announcements or sentiment analysis of the social media to try to find exploitable patterns with relation to which they can trade. This can be then incorporated into a strategy.

3.2.4 Algorithmic Trading Development Process

Algorithmic Trading simulation is not viable without the electronic market models due to its programmatic access requirement. Consequently, both the AT and the EM concepts are interdependent as they strongly influence each other's software architectures and the process flows.

An Algorithmic Trading model describes the behaviour of a system of variables in time. The variables are linked with the trade execution process through a set of trading signals. The expression 'algorithmic trading model' is often used interchangeably with the 'algorithmic trading strategy' (e.g., [Katz and McCORMICK, 2000]). This is due to the fact that the model can be considered a description of a set of steps defining a trading strategy. The idea of the strategy comes from the pre-'financial computing' times, when computers were not directly used in the trading process and the Traders were often describing the sets of repetitive analytic and decision-taking actions (aimed at generating profitable trades), as strategies. The idea of the trading model, on the other hand, comes from the world of mathematics and physics, where the scientists that were used to creation of parametrized models of abstract ideas started taking interest in computational finance.

The following section will describe the AT development process to provide the reader a broader understanding of the 'electronic exchange'-related concepts.

3.2.4.1 Key stages of the development process

As discussed in the Literature Review Chapter of the thesis, there are six standard development components that can be recognized in construction of the algorithmic trading strategies. These are: a) the pre-trade analysis, b) the signal generation, c) the trade execution, d) the money management, e) the risk management and f) the post-trade analysis.

- During the pre-trade analysis process the model acquires and abstracts the information about the current state of the financial markets and retrieves the most indicative set of features of the data with respect to a pre-defined utility function.

- Such indicators are then utilised in the signal generation step where a set of rules is used to define the circumstances of occurrence of a trading signal. Every time a new market information occurs, it is being streamed through the pre-trade analysis process and the signal generation process.

- In the instance the signal generation process indicates that the tradable opportunity exists in the model, a signal is sent to the trade execution element, that manages the generation of a set of instructions to trade particular securities defined in the strategy.

- To be able to evaluate the executed transaction and a new value of the utility function of a strategy at a given time, a model performs the post-trade analysis.

- Apart from the first four stages, the most successful strategies also utilise the money management and the risk management elements. The money management element assigns a specific amount of funds to each transaction that maximise the utility of the model. The risk management element is responsible for the process of minimising (or managing, when a certain level or risk is appreciated) the risk of a loss. Both the elements are often expressed in a form of a multivariate utility function and are contained as part of the pre-trade and the post-trade analysis (or both in the instance of a feedback loop).

3.2.4.2 AT model and its infrastructure

When describing the interactions between the electronic markets and the algorithmic trading strategies, it is worth taking into account all the communication streams of a typical trading model with the rest of the financial systems infrastructure, to understand the way they exchange information. The majority of the financial environments provide either a two or three-stream communication approach [Risca, 2008], where the model needs to be authenticated to each one of the streams to be able to use it. The streams can be defined as: a) the market-data feed stream, b) the trade stream and c) the account-management stream.

- The market-data feed stream provides the methods of obtaining information about the state of the supported financial market instruments.
- The trade-stream provides the methods of issuing trading instructions and controlling execution of the instructions.
- The account stream provides the methods of controlling the state of the managed funds, the market positions and P&L.

3.2.4.3 AT model and data

Apart from the infrastructure, trading strategies rely heavily on data (as they are driven by information). This may be the information about a temporal price of a security, a fundamental quantity describing a company or a sector, a news or a social opinion. Therefore, the key feature of every strategy is an ability to access necessary information, to analyse it, to abstract it and to take a profitable decision on such basis.

3.2.4.4 Typical stages of life of an AT model

During its lifespan the AT model is taken through a set of stages [Katz and McCORMICK, 2000]: 1) the R&D stage, 2) the testing & optimisation stage, 3) the trading & re-optimisation stage, and 4) the re-factoring stage. These are described below:

The R&D stage: involves one or more quants performing research on available data to identify potentially exploitable patterns. On the basis of such patterns the quants, often with the traders, perform a paper-trading exercise concluded with a formation of a potential model. The model is then implemented and tested for the existence of logical and semantic bugs.

The testing & optimisation stage: at this stage the formed AT system is submitted to a series of tests, simulations and optimisations to evaluate its risk and performance characteristics and later to fine tune the parameters of the algorithm. This exercise is often performed in collaboration with the quants, the traders and the risk managers.

The trading & re-optimisation stage: when the risk and performance characteristics of an algorithm are finally approved by the traders, the risk managers, the portfolio managers and the trading floor managers the algorithm is taken to the production environment and is the allowed to actively trade with one or more of the electronic market exchanges. Periodically, when indicated by the risk managers, the algorithm is submitted to a process of re-optimisation, to preserve/improve its risk and performance characteristics.

The re-factoring stage: finally, if the risk and performance characteristics of the AT system cannot be preserved any further, the risk manager moves the strategy out of the production environment and sends the AT system to re-factoring. One or more of the quants evaluate the applicability of such algorithm for

reuse and either takes the components of the algorithm to stage 1 or closes the life management cycle of the AT system.

3.3 Simulation with Electronic Market Models

Within Computational Simulation Environments an electronic market simulator can be defined as a software system that provide a selection of functionalities of a physical exchange for the purpose of transaction simulation. The selection of functionalities depends on the purpose of simulation.

The users of such systems are interested in the simulation of individual transactions or groups of transactions, for the modelling of action flows of such systems, or for the modelling of behaviour of the entire agent-trading ecosystems. They may also be interested in improving the software architectures of trading exchanges or in investigating the theoretical basis of the exchange theory (e.g., investigating game theory that may improve the process of the exchange of goods or improve the management of systemic risk). Such systems allowed the aspiring traders (e.g., students), to trade virtually and gain experience, without losing real funds. Algorithmic trading institutions are using the simulated trading exchanges to 'fine tune' their algorithms before enabling them for real trading [Katz and McCORMICK, 2000]. Simulations of exchanges provide users the chance to participate in a market, receive live feeds and monitor their profit or loss without the burden of bearing risks associated with the real market trading.

The typical users of the exchange simulators can be divided into four groups:

- The Quants and the Quantitative Developers would typically use such systems while developing their new trading strategies. The exchange simulators are especially used in the processes of back-testing and the optimisation of the strategies. While observing an outcome of the utility function utilised for evaluation of the developed strategy, they would typically be able to test and improve the obtained strategy.

- The Developers of the computational finance infrastructure use the exchange simulators to test their systems. The exchange simulators are used to test the performance of such systems, the validity of information flow and the quality of integration of their infrastructure with the exchange communication protocol.

- The National Banks, the Regulators and the Government institutions would utilise the exchange simulators to model and manage systematic risk of the entire economies.

- Finally, the Researchers would utilise the exchange simulators to model the multi-agent behaviour of the market participants. As described, the simulated environments are used not only by the users (willing to gain some market experience), but also by the risk managers (to test their risk systems), and by the software developers (to build and test their quantitative financial systems).

The remaining part of the chapter will focus on the design and implementation of an electronic market simulator architecture and a software service capable of providing such functionality. It will introduce a simulation model that embodies the key features of the electronic market exchange.

3.3.1 Simulation Types and Applications

Given all the various scenarios and applications in which the EM simulators can be utilised we can differentiate four key models of the simulators:

LIVE EM for multi-agent transaction simulation: the live, on-the-fly exchange simulator that enables matching and execution of the participant-issued order-instructions against the real or synthetic price quotes. The simulator typically respects a major part of the standard communication protocol, including the key types of the order instructions, the price quotes, and the execution reports. Such a simulator is characterized by utilisation of the simplified version of the order validation and clearing functionalities, the lack of logging functionality, the lack of auction functionality and the simplified version of accountancy and the P&L clearing modules. The order book and the price marker functionalities are also simplified. The key focus in the design of this type of the EM simulator is shifted to the exchange protocol.

EOD EM for end-of-day multi-agent transaction simulation: the end-of-day model of the exchange simulator is similar to the live model in that it is able to receive and send the on-the-fly events, indicating an intent for transaction against the real or synthetic price quotes. The execution model, however, is significantly different in that the process of clearing, matching and execution of all the orders, accepted in a given day, is scheduled to happen only at the end of the day. The only processing that happens live, in the on-the-fly mode, is the process of order-parameter validation. At the end of the execution process the exchange generates the execution reports, indicating the state of transaction. One significant difference here is the fact that only the full-order fills are possible. Another significant difference is the fact that the orders are matched against the one day candle representation of an average price, rather than against the price quotes (ticks).

While considering the case of the EOD EM, a simplification in the execution protocol of that type of simulator is to enable the approximation of daily execution. This is particularly useful for portfolio managers who seldom trade intraday but are instead interested on a longer investment horizons. Furthermore, this simulator is expected to not contain the auction functionality at all.

BACK-TEST EM for single-agent historic transaction simulation: the exchange simulator for back-tests differs significantly from the two previous models in that it is designed to run on the historical time-series of price quotes (ticks or higher granularity data e.g., candlestick - open/high/low/close data pattern) and is typically designed for a single participant/agent only. During the back-test process, the back-test manager retrieves (from a database) the time-series of price quotes for required securities and

feeds it synchronously to both the participant's API and the exchange simulator. The participant's API contains an algorithmic trading strategy that is submitted to the back-test. The back-test manager feeds individual quotes to both the functionalities and waits, after every quote, for notification in case the strategy logic is to issue any orders. In case orders are issued the order instructions are entered to the exchange simulator for immediate execution (during the execution it is assumed that all the steps in the exchange communication protocol are respected). After the matching process is finished and the strategy receives order execution confirmation in a form of an execution report, the back-test manager moves to the next feed cycle, providing subsequent quote to the strategy and the simulator. This process continues up until the point where the entire anticipated history of quotes is exhausted.

The key simplification in this type of market simulator is the fact that the feed cycles are executed on single threads and the threads are responsible for running all the calculations in the simulator and the API (each simulation cycle starts after the orders and the prices are provided for processing in the cycle). The accountancy functionality requires only one account for the current user and no authentication is necessary. The utilisation of any messaging functionality is also minimised and the strategy is coupled directly with the exchange simulator.

LIVE EM for multi-agent financial system simulation: is an order vs. order simulator, which focuses on modelling and measuring responses of trading agents to changing market circumstances (rather than focusing on strictly following exchange protocol, as in case of the previously described simulators). Consequently, the order book and the matching engine of the simulator contain additional functionalities allowing introduction of e.g., market shocks [Subrahmanyam and Titman, 2012], regime changes [Ang and Timmermann, 2011] or influence of changes of trading parameters on an agent behaviour. This enabled the multi-agent exchange simulations to support modelling of a large-scale group behaviour of market participants.

Since the number of actively trading agents can be as high as a few hundred thousands, the simulator needs to be able to handle large amounts of highly concurrent connections at the same time. The authentication functionality however, as well as the range or handled securities and order types, is significantly simplified. As latency is not considered a key issue, the P&L calculation and tracing functionalities can be extended to provide more sophisticated methods of evaluating participants.

BACK-TEST EM for multi-agent financial system simulation: this order vs. order type of simulator is designed to model the entire ecosystem of agents and exchanges over time. Rather than using one exchange simulator, the back-test system utilises a few exchanges with different execution regimes, different execution parameters and with a possibility of introducing e.g., endogenous and exogenous shocks [Johansen and Sornette, 2010] to the system. The entire system is submitted to either a back-test (over historic datasets of price quotes stored in the database), or to a stress-test (over a multi-set of

synthetic time-series of price quotes), all to observe evolution of the system.

The accountancy functionality of the utilized simulators is significantly simplified to hold basic trading statistics. The clearing or authentication functionalities are also minimalistic. The order book and the matching engine functionalities support only the most basic order instructions. In most cases, the utilized simulators support only the end-of-day mode since trading cycles may be simplified not to incur overcomplicated functionalities. On the other hand, such simulators can contain sophisticated stop functionalities or global risk and market stability measures to support evaluation of the state of the ecosystem.

3.3.2 Experimenting with EM Simulators

As mentioned, the second experiment of the thesis involves the simulation of an electronic market. The results of the implementation of two-out-of-the-five listed simulation models, namely the end-of-day (EOD) model and the on-the-fly (LIVE) model (utilized in UCL's algorithmic trading competitions and manual trading competitions) are presented. The models contain the key features of the electronic exchanges, including functionalities for: the order book, the matching engine, the account management, the P&L clearing, the event-based price streaming and the order instruction/confirmation messaging. The market price quotes are streamed on-the-fly from the real exchanges (HotSpotFX, CME, CBOT and EUREX). The developed simulators allow participants to trade algorithmically or in a manual fashion in order to support evaluation of their performance. Furthermore, the simulations allow participants to issue, change, or cancel the orders, consequently, providing close to real trading conditions.

The developed models of electronic markets were designed for three classes of experiments: a) to simulate the trading environment for strategies submitted for stress-tests: this is to observe the behaviour of strategies on live data, before they are being allowed to trade with real money, b) to organize algorithmic and manual trading competitions on the basis of the models: this allowed to experiment with different architectural designs and their abilities to cope with larger amounts of users behaving in an unexpected way, and c) to experiment with exchange configurations and their influence on the trading characteristics of software agent traders.

The research on simulating electronic markets focuses on answering questions on the choice of software architectures suitable for replicating such an environment, methods of minimising the difference between trading on both simulated and real exchange-methods of reflecting real-time market behaviour (such that the performance of a trader on a simulation becomes an indicator of performance on the real market). Formally the objectives of the experiment were set to: 1)understand the way electronic markets work, 2) identify a list of the key electronic market simulation types, 3) identify a list of technologies suitable for electronic market simulators, 4) develop a software architecture suitable for representation of one or more of the simulators, and 5) test the developed simulators. The objectives related to quality and

stability of the environment were then evaluated against a set of trading competitions while quantitative evaluation of performance of the environment was performed during the initial unit, integration and performance tests.

The key questions and experiments in this part of the thesis focused on identification and development of the essential elements of the exchange models. This is to support the execution of orders and the communication of information with the participants. The developed models incorporated functionalities for: a) bookkeeping of admitted orders, b) for matching orders with prices, c) for user account management, d) for clearing of orders and consequently for P&L calculation. Moreover, three facilities were developed to support the typical exchange communication protocol to enable the models to communicate with users. These facilities are: the event-based price streaming, the order instruction/confirmation messaging and the account state updating.

The key concerns taken into account during the creation of the models were the accuracy in calculation of user holdings, the realistic cost estimation and speed of execution, the transaction security, the information persistence in the underlying distributed connectivity, the error handling and the fault tolerance capabilities.

3.4 EM System

The following section provides a detailed description of the design, implementation and testing processes of the EM in both the EOD and the LIVE modes, with a particular focus on the features of such systems, their software architecture and their key functionalities. Justification of the design and technological choices is also provided.

3.4.1 EM Functionality

Support for a standardised, internal communication protocol: Both of the EM models support a standardised, common set of messages defined to support the exchange of information with the simulators. The protocol defines the information representation for the session, the pre-trade, trade and post-trade messages. The protocol is a generalisation of the FIX protocol, as well as the proprietary LMAX, Chi-X/BATS and EUREX protocols.

Support for a market limit and stop order instructions: Both of the EM models support transactions on the basis of the most common types of order instructions, namely the market, the limit and the stop orders, with a typical combination of parameters that can be encountered in the real exchanges.

Support for an external communication via JMS: Both of the EM models provide the support for a scalable synchronous and asynchronous JMS communication on the basis of the pro-

ducer/consumer pattern, enabling participants to trade and monitor price fluctuations as well as the state of their positions. The participants can chose to either utilise a set of Java interfaces that support the EM protocol, or can implement their own functionalities on the basis of the protocol specification.

Management of users and trading accounts: Both of the EM models enable administration of users and their trading accounts, including a programmatic access to the EM database (to create, modify and delete information about users and the state of their accounts). The parameters of the trading accounts can be amended with respect to the available funds, leverage, and margin-call-threshold, amongst others.

Support for subscription and consumption of the external price quotes: Both of the EM models support the subscription and consumption of the external price quotes, enabling integration with the real exchanges, the synthetic sources of price quotes or the historic sources of price quotes. Furthermore, the price quotes sources, to which EM models are subscribed, are handled on-the-fly.

Configurable sets of tradable securities: Administrators of both of thr EM models can configure a list of tradable securities and parameters of the securities, including e.g., a min. lots size, a min. tick change and others.

Support for the order matching: Both of the EM models are capable of storing orders in the order book and processing large amounts of orders in a parallel fashion, as well as matching the orders against price quotes in either the continuous-matching or burst-a-day mode, depending on a model.

Support for P&L tracing of user accounts: While executing transactions the EM models are capable of supporting the profit and loss calculations of positions held in the accounts and supporting the tracing of changes in the account state.

3.4.2 EM Software Architecture

The described EM features of both simulation models are supported by the following elements of the software architecture:

Order Book: The book-keeping facility is a necessary element of an exchange and allows it to keep track of admitted order instructions. The representation of data in the order book enables a quick lookup of traded securities, price levels, and volumes traded on each price level. The order book needs to allow a fast insert, update and removal of the order instructions and needs to allow an accurate representation of quantitative data to avoid approximation errors. The typical representation of the order book is based on a tree-like data structure and orders that arrive to the exchange and initially reside in the admittance queue are allocated to a suitable node in the tree after clearing.

Matching Engine: Apart from keeping track of the order instructions in the book-keeping facility, the simulations consist of a matching engine that utilises the order book to perform searches over potential counter-party orders and to execute matching instructions on the first-in-first-out, best-price basis. The EM matching engine is capable of executing a large amount of search & execution procedures every second. The quick lookup functionality of the order book is the key to a maximisation of the speed of search.

Account Management Facility: The EM models, together with the ability to match and execute the order instructions, are able to recognize owners of order instructions, authenticate the users, and are able to keep track of users holdings and available funding. The Account Management facilities in the EM provide the representation of information for the users, their accounts and the positions they hold, and are basis for the order instruction clearing. The Account Management facilities provide programmatic access to the functionalities for the user registration & maintenance and for the lookup and update of the state of user accounts. It is also used to generate the account reports.

P&L Clearing Facility: For the P&L Clearing Facility, every time a user issues a new order instruction to the EM, the simulator needs to be able to evaluate the users capability to afford such transaction and to clear it with respect to the available funds. Consequently, the exchange simulator keeps track of the current value of all the holdings of a user. Based on such continuous P&L calculations the user holdings are monitored, and his/her capabilities to afford costs of maintenance of opened positions are cleared until the user reaches a margin call (a level of exposure that the clearing functionality can no longer accept). The margin thresholds are defined by an administrator in the accountancy functionality. Real-time calculations of P&L are particularly troublesome due to the rate of the price changes and the potential amount of positions in securities the users can hold.

Event-based Price Streaming: The EM exchange simulators are able to communicate changes in prices of tradable securities to the registered users for further analysis, and to trigger revaluation of the already held positions. This is performed in three ways: a) with use of an aggregated view of the top-of-the-book, best-available bid/offer prices of securities, b) with use of a full-book, aggregated view of the bid/offer prices of securities on different depths of prices, and, c) with use of the exchange events issued after clearing of each order instruction. Such information for every exchange-supported security is streamed to users in a form of price-events called ticks.

Messaging System for Order Instructions and Execution Confirmations: In order to control the execution of order instructions in the exchanges, the users are provided with a capability to establish a communication channel with the exchange simulator and with use of such session are able to issue orders and can monitor the execution of such orders. The EM exchange simulators provide a set of standardised order types that they are able to process, together with a list of required parameters. For every issued

order instruction, the exchange simulator will respond with one (or more) execution report, that typically contains information on the progress of the execution of the instruction or contains information on why the instruction was rejected so that the user can correct the problem.

Messaging System for Account and Position Updates: Apart from communicating the price updates and the order execution reports, the exchange simulators require means of communicating changes in the states of their user accounts, including the changes to the currently held security positions. There are two methodologies utilized by the exchange suppliers for the purpose of provision of such information to the users. This is either with use of a request-based methodology (where users can request the information when it is required) or with use of an event-based asynchronous information streaming (where the updates of the state of the account and its positions are being periodically sent to the current session holder of a registered account). Both methodologies are supported by the EM.

3.4.3 EM Dynamic Data Flows

The following paragraphs describe the flow of information in and out of the designed simulation engines, as well as the internal data flow. Since the described EMs are dynamic (data-driven, on-the-fly, event systems) the key information types can be identified as: ticks, orders, execution reports and account updates.

Models of Matching and Execution: Many variations of the matching and execution models can be encountered while attempting to simulate an electronic market. Two such models were experimented with, namely: a) the order-to-price live matching, and b) the order-to-price end-of-day matching.

In the real electronic markets, the submitted orders are processed and recorded in a central order book for matching. The price and quantity specified in the orders are matched against counter-orders in the order book. For a partial or full match to occur, each order needs to be matched with an order providing the opposite side of the trade. The price and the type of order play a role in whether or not an order can be part of a batch of orders that are submitted to a matching process with a corresponding counter-order in a particular cycle of order execution. This model of matching and execution is called 'order-to-order matching' and is the protocol adopted by most of the electronic exchanges all over the world in the continuous matching process.

The described EM architecture, however, focuses on another matching convention that attempts to simulate an exchange environment in the order-to-price matching mode. In the order-to-order matching convention the demand-supply dynamics of a security could result in an order not being fully filled for some time, or, in some cases, not being filled at all, either due to a lack of liquidity or due to surpassing the maturity time of an order. In the order-to-price convention, the matching against price typically doesn't take the order size into account and consequently majority of orders end up being fully filled. The two sections below detail two modes of the order-to-price matching convention experimented with

as part of this thesis.

The end-of-day Order Matching and Execution: This process can be considered as a two part process. The first stage of the matching process is the order validation and accumulation phase. The received orders are time stamped, the orders are allotted a unique order ID (which acts as the primary key to distinguish the order from other orders in the order book) and the specification of the order is persisted in the processing queue (in the sequence in which they arrived at the exchange). The trading hours are treated as the order accumulation phase. The key feature of the stage is the pre-specified end-of-day timestamp at which the second stage of matching & execution commences.

During an occurrence of the end-of-day timestamp, which can be treated as the second stage, the orders accumulated during the trading hours are processed (triggering a rapid sequential call to process all the orders). The processing of an individual order involves the evaluation of the account balances of a user who issued the order, the evaluation of price stated in the order to the end-of-day price. If there is a match with a daily price, the order is filled for the quantity stated in the order.

The data flow of matching of the market orders is the simplest to execute. All market orders are filled at the price prevailing at the end-of-day timestamp. The limit orders specify a limit price. The limit price in a buy order states the maximum price the user intends to pay for the quantity specified. The limit price in a sell order specifies the minimum price the user intends to get for the quantity specified. The end-of-day engine matches all the limit orders by checking if the limit price levels in the buy or sell orders were met during the trading hours before the end-of-day timestamp. The stop orders specify a stop price, executed in a manner similar to the limit orders but with respect to the transaction's stop-conditions.

The Live Order Matching and Execution: such data flow involves processing orders as soon as they arrive in the simulator. The market orders are executed at the best available price. The limit orders are executed as soon as a price equal to or better than the limit price becomes available. The stop orders in the live execution mode differ from the limit orders. A stop order is an order to buy (sell) a specific security once the price reaches a specific level. In the case of a buy (or sell)-stop order, the stop price needs to be above (or below) the prevailing price. The live execution is also summarized by live account updates to user accounts. As the order execution leads to changes in the value of the accounts' positions, the accounts are updated whenever an order gets executed. Unlike at the end-of-day mode when the P&L calculator (responsible for computing updated values for users' accounts) is triggered just once a day at the closing timestamp of the system, in the live mode it is triggered whenever an order is executed.

Representation of Orders in the Order Book: Orders from various users are persisted in the order book and indexed with a unique order ID. The key specifics of the order, e.g., the price, the quantity, the type of order and the order expiration, are persisted in the database (in a process similar to logging of the events in the real exchanges) in a manner such that the order can be recreated from the key information

being stored in the database.

The simplest way of searching and retrieving orders from the order book is to search for a unique order ID that is allotted when orders arrive in the exchange during the trading hours. The simulation takes care of allotting the same order ID for all amendments issued for a specific order. The updated price & quantity overwrite the previous order details in the database.

Clearing, Accountancy & PnL Calculation Process: are three interrelated functionalities that ensure validity of the performed transactions. All three rely on the availability of information about the users, their accounts, the orders they issue and the current prices of particular securities. The typical data flow that can be considered with relation to the functionalities involves clearing of the available account margin where, in the instance a user issues a new order instruction, an account associated with the user needs to meet the minimum available margin criteria. Consequently, each order queued for clearing is validated against an account it is associated with. If such account can not be found, the order is rejected. If the account can be found the information on the available margin is retrieved. Such information should be available at the point when it is required, consequently, all the accounts are submitted to the continuous P&L calculation process evaluating the state of the account with respect to the recent changes in value of all the holdings. On such basis the available margin is estimated. After the information about the available margin is retrieved, the clearing process checks if there is enough margin available to execute the order instruction. In case the order is not cleared it is rejected. In case it is cleared, it is submitted to the order book for execution.

3.5 Summary Discussion

As mentioned in the Introduction Chapter of the thesis, this work focuses on computational finance rather than performance problems encountered in software engineering research. As a result the scientific effort is directed at holistic, operational results of the experimental systems and the applicability of proposed software architectures to a given financial economic problem. This is reflected in the structure of the thesis. An overall operational experiment that utilise all the designed environments is described in the Evaluation Chapter of the thesis. The discussion of results related to this particular experiment is nevertheless provided below. This is also followed by comparison of performance results against set expectations.

The objectives of the EM experiment were set to: 1) model the way the electronic markets work, 2) identify a list of the key electronic market simulation types, 3) identify a list of technologies suitable for the electronic market simulators, 4) develop a software architecture suitable for the representation of one or more of the simulators, and 5) test the developed simulators.

The first point was addressed by presenting: the standard steps utilised by a typical electronic market, the

steps utilised by participants of the exchange, and the identification of the participants themselves (the way they trade and their needs while developing their algorithms). This step helped to unify a standard communication protocol developed to communicate events between the simulators and the participants.

The second point was addressed by identifying the LIVE and EOD simulation types, for on-the-fly and historic electronic market simulators designed for individual users or for a multi-agent modelling. Further experiments concentrated on the LIVE and EOD price-to-order simulators (for multi-user purposes, where for both of the approaches simulators were developed).

The third point was addressed by various experimentations with asynchronous messaging systems, in-memory databases, relational databases, column-oriented databases and concurrent processing techniques. This was concluded with a unique architecture of the exchange simulators.

The fourth point was addressed through development and implementation of two exchange simulators. During that time a list of typical-usage and alternative-usage scenarios was developed, then transformed into an object-oriented class and interface representation. Finally, the developed architecture was implemented in the Java programming language as a set of distributed services.

The fifth point was addressed through utilisation of a test-driven approach during the development process as well as a set of services for continuous integration, project control, source control, source quality control, team management and deployment automation. Consequently, the simulators were submitted to continuous testing during the implementation stage.

Furthermore, the utilisation of the developed simulators in the algorithmic and manual trading competitions organized by UCL in 2010, 2011 and 2012 allowed testing and evaluation of the designed software systems.

3.5.1 Performance Tests

To address the above questions and limitations quantitatively, Figure 3.2 presents the key statistical experiments related to the performance evaluation of the LIVE EM for a multi-agent transaction simulation, utilised in majority of the UCL's Algorithmic Trading Competitions (for more details please refer to Chapter 6). The experimental simulation environment is believed to be unique in academia for financial economics modelling. To work around a lack of comparable environments, a set of common sense thresholds was selected to evaluate the achieved performance. The thresholds define the maximum amount of processing time that is acceptable for various functionalities of the environment.

The amount of time it takes to handle a new order request and to insert it to the admittance queue is important from the perspective of a user who needs the system to be able to accept orders for execution as soon as possible. This functionality is susceptible to the amount of users and the amount of orders generated by the users.

Electronic Market Simulator				
The description of a PSS experiment	Expected Threshold Levels	Obtained Results		
	The maximal expected value	The minimal registered value	The average registered value	The maximal registered value
The amount of time it takes to handle a new order request and insert it to the admittance queue.	0.5sec	0.2sec	0.2sec	0.3sec
The amount of time necessary to validate an order.	1sec	0.3sec	0.7sec	0.9sec
The amount of time necessary to match a new market order in an order book.	0.5sec	0.05sec	0.05sec	0.06sec
The overall amount of time necessary to send a new market order and receive an order fill report.	5sec	0.94sec	1.3sec	2.1sec

Figure 3.2: Electronic Market Simulator performance statistics.

Order validation and verification is the most time-consuming process of the market simulator and, consequently, the architect of the exchange should design the system to reduce the amount of time (necessary to validate an order) to a minimum.

The amount of time necessary to match a new order in the order book is another functionality which is susceptible to the overall amount of users and the amount of orders they generate. Furthermore, it is also susceptible to the traded volume and the order type, and, consequently, for the purpose of the experiment, the market order type was selected to eliminate the delays related to the actual specifics of an order.

The overall amount of time necessary to send a new market order and receive the order-fill-report is interesting from the point of view of the entire simulation environment as it is an ultimate indicator of the overall performance of such environment.

Chapter 4

COMPUTATIONAL SIMULATION ENVIRONMENT FOR AT (ATRADE Platform)

This chapter describes the design, development and implementation of a Computational Simulation Environment for experiments with algorithmic trading and risk. The environment is a large-scale, distributed, event-driven system that allows multi-user, real and virtual trading and provides a proprietary remote SDK with AT strategy-templates supporting a variety of AT strategy types. The Environment provides unique capabilities not only to manage algorithm risk but also to evaluate and rank AT strategies according to their performance and risk characteristics.

4.1 Experimental Computational Environments

The use of computational techniques in social science (sociology, finance, economy) to carry out research and inform policy has only emerged recently, but is expected to have a major impact in academia, industry and government agencies. Computational Simulation Environments are central to this research. Furthermore, analytical tools are the mainstay of science and engineering, spanning statistical libraries, data mining frameworks and machine learning toolkits. One may be interested in coupling analytical tools, streaming, sharing and warehousing services as a part of a larger computational cluster, for simulation, validation, optimisation and monitoring of analytical algorithms.

4.1.1 Research Questions

The research questions addressed through work on the Computational Simulation Environments and their software architectures are as follows:

- What are the typical classes of models that are to be supported by the Environment, what are their

typical interaction with the Environment, and is it efficient and possible at all to create generic interfaces to analytic models?

- What types of functionalities, libraries and tools are necessary in large-scale analytical environments to provide support to various analytical models in general and trading model in particular?
- Is it possible to identify typical modes of operation and information flows in such systems?
- What is the most suitable software architecture solution and what are the most appropriate technologies that can be used to support such experimental systems? Are virtualization, cloud computing, big data facilities and complex event processing (CEP) technologies applicable to this problem and how can we distribute the system to ensure sufficient performance and in-time responsiveness?
- Is it possible to identify and resolve various limitations that are imposed on these types of analytical systems, mainly with respect to scalability, data bandwidth & analytic capabilities and model execution in various modes?

Given the computational finance focus of the thesis, as well as the methodology assumed in the Introduction chapter it was decided that the most suitable way of addressing the above research questions is through construction of two environments that share a common software architecture. It was then assumed that such approach should enable evaluation of holistic operability of the designed software architecture. To support such work in financial economics and more generally in social science the following experimental computational simulation environments were designed:

- Algorithmic Trading & Risk Analytics Development Environment (ATRADE) for algorithmic trading systems, high-frequency trading and algorithmic risk. ATRADE can be used for experimentation with virtual and real trading, and has been designed specifically to study the behaviour and risk of trading algorithms.
- Social media streaming, storage and analytics platform (SocialSTORM) for social analytic algorithms. SocialSTORM supports the scraping and analysis of a wide range of social media data, RSS feeds and news etc. These two environments share a common systems architecture which the author believes is applicable to a wide range of experimental computational environments.

4.1.2 Common Systems Architecture

Computational Simulation Environments presented here employ a common systems architecture consisting of a set of distributed, multi-threaded, event-driven, real-time, Linux services communicating with each other via an asynchronous messaging system. For example, the ATRADE environment allows multi-user real and virtual trading. It provides a proprietary API to support development of algorithmic

trading models and strategies. It allows advanced trading-signal generation and analysis in real-time, with use of statistical and technical analysis as well as the data mining methods. It provides data aggregation functionalities to process and store market data feeds. Finally, the environment allows back and forward testing of trading strategies.

The common systems architecture comprises of the following modules:

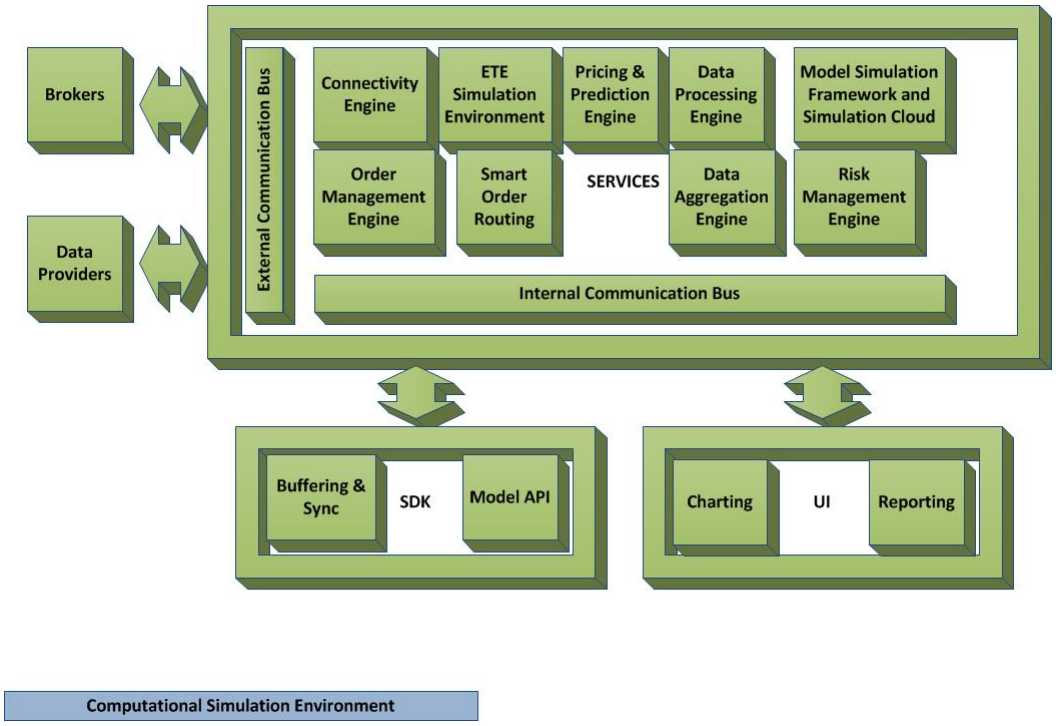


Figure 4.1: Key elements of a typical Computational Simulation Environment.

The key software components of the common system architecture are specified below:

Back-End Services: This provides the core Environment functionalities. It is a set of services that allows connection to data providers, propagation processing and aggregation of data feeds, execution and maintenance of models, as well as their management in a multi-user environment.

Front-End Clients: This provides a set of programmatic and graphical interfaces that can be used to interact with the Environment to implement and test analytical models. Programmatic access provides model-templates to simplify access to some of the functionalities and defines generic structure of every model in the Environment. The graphic user interface allows visual management of analytical models. It enables the user to visualize data in various forms, provides data watch-grid capabilities, provides a dynamic visualization of group behaviour of data, and allows users to observe information about events relevant to the users' environment.

When considering the Cisco's Trading Floor [Risca, 2008] architecture introduced in the Literature Review chapter of the thesis, one can relate the business logic of the trading floor to the domain specific

functionalities of ATRADE and SocialSTORM. In the two experimental environments the key business logic modules can be described as follows:

Connectivity Engine: This functionality provides ways of communication with the outside world; with financial brokers, new/social data providers and others. Each of the outside venues utilized by the Environment has a dedicated connector object responsible for control of communication. This is possible due to the fact that each of the outside institutions provides either a dedicated API or is using a communication protocol i.e. the FIX protocol, JSON/XML-based protocol. The Environment provides a generalized interface to allow standardization of a variety of connectors.

Internal Communication Bus: The idea behind the use of the internal messaging system in the Environment draws from the concept of event-driven programming. Analytical Environments utilize events as a main means of communication between their elements. The elements, in turn, are either producers or consumers of the events. The approach significantly simplifies the architecture of such system while making it scalable and flexible for further extensions.

Data Aggregation Engine: This provides a fast and robust storage functionality, for an entry-level aggregation of data, which is then filtered, enriched, restructured and stored in big data facilities. Aggregation facilities enable analytical Environments to store, extract and manipulate large amounts of data. The storage capabilities of the Aggregation element not only allows the replay of historical data for modelling purposes but also enables other, more sophisticated tasks related to functioning of the Environment, including model risk analysis, evaluation of performance of models and many more.

Client SDK: This is a complete set of APIs (Application Programming Interfaces) that enables development, implementation and testing of new analytical models with use of the developers favourite IDE (Integrated Development Environment). The SDK allows connection from the IDE to the server-side of the Environment to provide all the functionalities the user may need to develop and execute models.

Buffering & Synchronization: This provides a buffer-type functionality that speeds up the delivery of temporal/historical data to models and the analytics-related elements of the Environment (i.e. the statistical analysis library of methods), and, at the same time, reduces the memory usage requirement. The main idea is to have central points in the Environment that manage and provide a temporal/historical data from the current point of time up to a specified number of timestamps back in history). Since such central points are shared, the models have no need to keep and manage history. Moreover, since the information is kept in-memory, rather than in the files or the DBMS, access to it is instant and bounded only by the performance of hardware and the Environment on which the buffers work.

Model Templates: The Environment supports two generic types of models; push (Event-Driven) and pull (Service-Oriented). The 'push type' registers itself to listen to a specified set of data streams during initialization, and the execution of the model logic is triggered each time a new data feed arrives to the

Environment. This type is dedicated to very quick, low-latency, high-frequency models and the speed is achieved at the cost of small shared memory buffers. The 'pull model' template executes and requests data on its own, based on a schedule. Instead of using the memory buffers, it has a direct connection to the big data facilities and hence can request as much historical data as necessary, at the expense of speed.

4.1.3 ATRADE Functionality

To support research on trading algorithms and their risk, the following functionalities were identified as crucial for ATRADE Environment:

Simulation & Real Trading: The Environment allows users to trade virtually or with real money, with use of a dedicated API able to support algorithmic trading. To provide this capability a set of interfaces was designed that allows issuing and maintenance of orders, extraction and processing of market data (to generate trading signals), extraction and processing of order data, and management of information related to P&L and users' account information.

Rapid Prototyping: The Environment is a framework for development, trading, testing and evaluation of the algorithms' risk. Majority of ATRADE users come from academia and are relatively inexperienced in the field of algorithmic trading. Therefore, to boost their experience they need a framework for development, trading, testing and evaluation of their algorithms. Rapid prototyping requires usage of a simple programming language and also simple and powerful interfaces that provide as much embedded functionality as possible without a need of a heavy, low-level programming. Given the methodological evaluation based on observation of academic community described in the Evaluation chapter of the thesis, ATRADE is believed to meet the requirements.

Data Processing & Aggregation: The Environment is capable of aggregating and processing data on-the-fly. Information is a key to success in algorithmic trading, therefore, ATRADE aggregates and processes data in near real-time and delivers it to the users for analysis. Apart from delivering the data in near real-time, the Environment allows retrieval of historical data through a set of dedicated interfaces.

R & Matlab: The Environment provides capability of incorporating statistical/mathematical computing environments. Statistical and mathematical computing concepts are particularly applicable to modelling and implementation of algorithmic trading strategies. Therefore, incorporation of the two environments may simplify many, otherwise complex, objectives set by the users. The two environments may be incorporated with ATRADE by the users and utilized in the functionalities that analyse data, generate trading signals, evaluate risk and many more.

Black Box and Multiple Models: A particularly useful functionality of the Environment is a module for automated evaluation of risk of black box models that allows ranking of the models on such basis; without a need of handing the source of the models to anyone. This feature is useful in eliminating the IPR (Intellectual Property Rights) issues and ensures safety of the algorithmic trading models from a

potential third party threats. The Environment supports automated evaluation of multiple models concurrently, and also enables generation of statistical-performance reports for every evaluated model.

Secure Remote Access: ATRADE provides Client UIs that allow remote access to major functionalities of the Environment. In order to provide the users with a simple and easy way of access, the design permits the users to work remotely with provided API, and communicate the necessary data over the network. Consequently, an additional secure communication layer is in place to ensure the developed models and strategies are able to securely communicate with the Environment.

4.1.4 SocialSTORM Functionality

As introduced, the social media streaming, storage and analytics Environment (SocialSTORM) facilitates the acquisition of text-based data from on-line sources such as Twitter, Facebook, and RSS media. The system includes facilities to upload and run Java-coded simulation models to analyse the aggregated data; which may comprise scraped social data and/or users' own proprietary data.

Two key social media elements are: a) scraping collecting on-line data from social and news media sites; and b) sentiment analysis sentiment analysis or opinion mining refers to the application of natural language processing, computational linguistics, and text analytics to identify and extract subjective information in source materials.

Scraping: SocialSTORM supports scraping of social media, RSS feeds and news. APIs for the most popular social networks, such as Facebook and Twitter, currently make data easily accessible. Tweets from all the public accounts (including replies and mentions) are available in JSON format through Twitter's Streaming API (for near real-time data) and a Search API for batch requests of past data (http://search.twitter.com/search.json?q=APPLE). For example, Twitter's Streaming API allows data to be accessed via filtering (by keywords, user IDs or location) or by sampling of all updates from a selected number of users. The default access level (the Spritzer) allows sampling of roughly, 1% of all public statuses; with the option to retrieve c. 10% of all statuses via the Gardenhose access level (more suitable for data mining and research applications).

Sentiment Analysis: The major use of this social media data is sentiment analysis. Sentiment analysis refers to the application of natural language processing, computational linguistics, and text analytics to identify and extract subjective information in source materials. It generally aims to determine the attitude of a speaker/writer with respect to some topic or the overall contextual polarity of a document. Computers can perform automated sentiment analysis of digital texts, using elements from machine learning such as latent semantic analysis, support vector machines, "bag of words" and Semantic Orientation [Asur and Huberman, 2010].

SocialSTORM inherits its architectural design from the ATRADE system, which allows easy integration between the two systems; leading to real-time aggregation and analysis of social media data and financial

data. The following sections outline the key components of the overall system:

Connectivity Engine: These various connectivity modules communicate with the external data sources, including Twitter & Facebooks APIs, financial blogs and various RSS news feeds; and are continually expanding to incorporate other social media sources. Data is fed into SocialSTORM in near real-time and includes a random sample of all public updates from Twitter, as well as filtered data streams selected from a rich dictionary of stock symbols, currencies and other economic keywords; providing gigabytes of text-based data every day.

Messaging Bus: This serves as the internal communication layer which accepts the incoming data streams (messages) from the various connectivity engines, parses these (from either JSON or XML format) to an internal representation of data in the Environment, and distributes the information across all the interested modules.

Data Warehouse: This is home to terabytes of text-based entries which are accompanied by various types of meta-data to expand the potential avenues of research. Entries are organized by source and accurately time-stamped with the time of publication, as well as being tagged with topics for easy retrieval by simulation models.

Simulation Manager: This terminal provides the external API for clients to interact with the data for research purposes, including a web-based GUI whereby users can select various filters to apply to the datasets before uploading a Java-coded simulation model to perform the desired analysis on the data. This portal facilitates all client-access to the data warehouse, and also allows users to upload their own datasets for aggregation with previously stored social data for a particular simulation. There is also the option to switch between historical mode (which mines data existing at the time the simulation is started) or live mode (which consumes the incoming data streams and performs analysis in near real-time).

Considering the scale of the described Environments and the shared software architecture design, the remaining part of the experiment is focused on ATRADE only.

4.2 Algorithmic Trading Environment

The key ATRADE components were designed to work as standalone Linux services able to communicate with each other over a network. Such distribution ensures flexibility and scalability of the whole system. The software architecture divides the Environment into two distinctive parts. The first part consists of modules that are mainly utilized by the server-side, and the second part is made of clients that utilize the server-side services. The Environment supports work on different stages of strategy development (strategy lifespan management) with three distinctive UIs.

The core of the server-side are the modules that provide capabilities for virtual and real trading. These are the Order Management Engine (OME), the Connectivity Engine and the Electronic Market simu-

lation environment. Moreover, there is a number of sub-modules that support the functioning of the Environment, and these are the Internal Communication Bus, the Data Aggregation Engine and the Processing Engine amongst others. The rest of the sub-modules are utilized to support execution of other key modules, and these are the Smart Order Routing, the Risk Management Engine and the Pricing & Prediction Engine. A selection of the key modules is described below.

One can differentiate three core client User Interfaces (UIs), these are as follows: a software development kit (SDK) that allows lifespan management of algorithmic trading strategies, a graphic user interface for monitoring and control of algorithmic trading strategies, and a statistics client for ranking of users and statistical evaluation of their performance.

4.2.1 The Server-Side Services

The server-side functionalities are provided to users working with one or more of UIs from remote hosts. Two distinct provision approaches were assumed over the course of the experiment: a standard Linux Init Service, where services were managed directly from the OS layer, and an Enterprise Service Bus approach where services were managed within an OSGi environment (Apache ServiceMix). In both cases the services were instantiated during the start-up of the server and only one instance of each service was allowed to run at any possible time.

The wiring of the services is supported through utilisation of the Spring Framework for inversion of control and injection of functionalities. This ensures simplicity in configuration and reconfiguration of various components of the services. To simplify distributed, event-based communication between the various server-side services the Internal Communication Bus was constructed with use of the Apache ActiveMQ technology. This ensures high scalability of the services. Both Spring Framework and Apache ActiveMQ simplify the unit, integration and system testing of the services.

Apart from providing the trading functionalities of the Environment, the server-side services support health and performance monitoring with use of the Metrics Codehale library and the Ganglia Monitoring System.

4.2.2 The Internal Communication Bus

The Internal Communication Bus module of the Environment is based on the JMS (Java Message Service) technology. This set of functionalities lies in the heart of the Environment as it allows a scalable inner communication within the Environment. Every element that needs to communicate with other elements of the Environment is designed to extend one of the JMS handlers to be able to produce and consume event-messages. The JMS functionality utilizes the previously mentioned Apache ActiveMQ message broker to provide brokerage of event-messages.

The architecture of the Internal Communication Bus can be considered a three-layer architecture with the

first layer enabling communication of real-time feeds of market prices, the second layer allowing general communication with the key services and the third layer allowing session-based secure communication of users with the services. The first layer consists of a set of permanent JMS topics (history-less queues) where an individual topic maps one-to-one to tick prices of individual securities. The second layer consists of a set of JMS queues allowing request/response-based communication between the UIs and OME as well as the OME and CONN. The third layer consists of a set of session queues allowing communication between authenticated UIs and OME as well as registered connectors and OME.

4.2.3 The Connectivity Engine (CONN)

The engine is responsible for communication of the Environment with external and internal trading venues. Consequently, the architecture of the engine consists of four distinctive components: the Feed Stream, the Trade Stream, the Connector Manager and the Connectivity Engine. Furthermore, to support communication of the engine with all the other internal services of the Environment, it utilises the highly scalable Internal Communication Layer as well as a high-throughput, point-to-point Conveyor functionality, for Data Aggregation Engine. Finally, part of the communication process involves translation from venue-supported communication protocol to/from Environment-internal data representation.

The Connectivity Engine functionality allows management of a variety of Connector Managers by providing access to their methods via a standardized interface. The Engine can be considered as a container of all the connector-objects that, when required, can perform a quick lookup over a specified object and trigger a selected method. The Engine communicates with other elements of the Environment to support dynamic control of the state of connectors. The Engine is also responsible for error-detection and recovery when functionality of one of the connectors fails.

The Connector Managers provide means of communication with a particular trading venue/data provider through either an API supported by the broker or through a standardised protocol e.g., the FIX protocol. Every Connector creates a single instance of a Feed Stream to support real-time streams of market data from particular venue. Connectors also create multiple instances of Trade Streams to support issuing of order instructions to trading venues as well as to handle post-trade information received from the trading venues.

Over the course of the experiment the Connectivity Engine was designed to support connections with the following trading venues/data providers: Trading Technologies (https://www.tradingtechnologies.com/), Chicago Mercantile Exchange (www.cmegroup.com/), Eurex (www.eurexchange.com/), HotSpotFX (http://www.hotspotfx.com/), LMAX (http://www.lmax.com/), Chi-X (http://www.chi-x.com/), Currenex (http://www.currenex.com/). Furthermore, the Connectivity Engine provided access to the internal Market Exchange Simulator described in the previous chapter of this thesis.

4.2.4 The Order Management Engine (OME)

The functionality of the Engine is responsible for maintenance of an aggregated view of the current and recent state of the orders and prices, and for management of active orders. Consequently, it consists of four distinctive modules: the buffering, the accountancy, the risk policies, the request handler and the data acquisition.

The buffering functionality provides an aggregated view of the current as well as recent history of prices of all the securities supported by the environment. Furthermore, the buffer is designed to keep track of all the orders issued by the users and their state (as confirmed by the trading venues). The functionality also keeps track of authenticated sessions. The buffer is utilised by all other sub-modules of OME.

The request handler as well as the data acquisition functionalities are on the opposite side of the buffering functionality. The request handler is responsible for handling all the request-events generated by client UIs. These include the authentication requests, the buffer content requests, the risk policy-related requests as well as order issue requests. The data acquisition functionality, on the other hand, is responsible for periodical acquisition of information about the availability of connectors and the availability of security definitions. On such basis it then is able to subscribe or un-subscribe itself from the streams of price feeds. Moreover, it maintains communication with connectors supporting particular trade sessions of authenticated users and hence can be updated with the post-trade information that allows the OME to be up-to-date with the state of the orders.

The accountancy functionality and the risk policies functionality utilize all the mentioned sub-modules of OME in order to maintain an up-to-date state of user's account, including the P&L characteristics of every user in the system as well as activity of users with respect to their account; on the basis of privileges and risk-levels set by active risk policies associated with every account.

All the sub-modules of OME form a basis for the Smart Order Routing functionality.

4.2.5 The Smart Order Routing (SOR)

The Environment has an ability to communicate with multiple trading venues at the same time. Consequently, it is possible to trade either the same Securities individually on separate trading venues or a statistical aggregate of the Securities on multiple venues, at the same time. The former is targeted by the Smart Order Routing functionality.

The SOR module allows policy-based choice between connector-supported brokers and gateways, and thanks to the aggregated view of current prices and trading volumes (provided by OME) it enables finding the best trading route, suited for a given purpose, e.g., order execution based on Volume Weighted Average Price (VWAP). In other words, the SOR element splits/aggregates and directs orders through one or few connectors that at a given time are most suitable for the current policy.

Over the course of the experiment the SOR functionality was designed to utilize the Apache Camel technology to provide the event routing capabilities as well as a proprietary routing mechanism based on JMS. It also used a Complex Event Processing (Esper) functionality to allow a real-time processing of data feeds, which in turn enables execution of a particular policy.

4.2.6 The Authentication & Authorization

The Authentication and Authorization functionalities are not considered as an individual module of the Environment. Instead it is a set of functionalities responsible for securing access to elements of the Environment that require it. The functionalities can be summarized as responsible for the following: the security of internal communication layer, the security of communication between UIs and the server-side, the security of communication between connectors and trading venues, the security of accounts of individual users and, finally, the administration functionality to manage accounts and authorization privileges.

The security of internal communication layer as well as the security of communication between UIs and the server-side are both enforced by an internal mechanism of the JMS technology. The mechanism utilizes a Java Authentication and Authorization Service (JAAS) to handle authentication and authorization requests. In the instance of the Environment, JAAS integrates the JMS and the OME Accountancy modules. Furthermore Apache ActiveMQ enables SSL encryption of all protocols utilised by JMS.

The security of communication between connectors and trading venues, is enforced by the trading venues. One of the requirements of most of the trading venues is a secure communication either with use of the SSL (Secure Socket Layer) or the VPN (Virtual Private Network). This approach provides secure protocol layer that wraps the actual communication protocol used by the trading venues and their counterparties, in this instance, the Environment.

The security of accounts of individual users is enforced in the Environment through utilisation of sessions in communication between the server-side services and each instance of one of the clients. In order for a user to get assigned a session they need to provide a login and a password to the Environment as well as a login and a password to each individual connector (and consequently trading venue) the user intends to use for trading.

To complement utilisation of sessions in the Environment, an administrative functionality is used to enable management of accounts and assignment of authorization privileges to accounts. The service is exposed to administrators via an Apache Felix GoGo web-console that forms part of the Apache ServiceMix OSGi container.

4.2.7 The Data Aggregation & Processing Engines

The aggregation engine plays a passive role of a functionality capable of receiving and storing large amounts of information from the Connectivity Engine to a data storage functionality. Over the course of the experiment various storage technologies were utilized including JDBC-based aggregation to an Enterprise MySQL, Hibernate-base aggregation to the Enterprise MySQL and a Hadoop-based Big Data aggregation supported by a cluster. On the other hand, the processing engine is an active functionality that uses the Apache ActiveMQ, Apache Camel as well as Esper's CEP functionalities to stream-in, transform and stream-out one type of financial data into another type. Over the course of the experiment the processing engine was utilized to transform the bid/ask tick data into candlestick representation of financial information.

4.2.8 The Developer Client

The Developer Client can be considered a Software Development Kit (SDK) consisting of a set of APIs that developers can use in their favourite IDE (Integrated Development Environment) to design, implement, test and optimize new trading strategies. The following are the key sub-modules of the SDK: the strategy templates, the data access & trading functionality, the buffering & synchronization functionality and the back-tester & optimization functionalities.

The Strategy Templates can be considered a three-part functionality consisting of a Service-oriented Template, an Event-based Template and a Strategy Manager allowing management of all the registered strategies (including graphical management via User Client).

The Service-oriented Template is designed for execution of long-term, low-frequency trading models that require large amount of data to analyse, in order to take decisions and manage orders. Two key features of the template are a schedule-based execution and an access to large data-banks of financial information.

The Event-based Template allows execution of ultra-fast strategies by allowing registration to various event messages (delivered to every strategy of this type immediately after the event happens), including price messages, post-trade messages and risk messages. The framework uses a short-term buffering functionality to enable access to most recent history of events.

The Data Access & Trading functionality provides the Strategy Templates a low-level access to events and historic information as well as the capability to trade (it issues orders and receives post-trade information). The functionality enables communication with OME to trade all the asset classes supported by trading venues. The functionality supports generation of the new, modify and cancel orders as well as the close-position orders.

The Buffering & Synchronization functionality utilises the Data Access element to provide capability of buffering the market and candlestick data. The size of the individual buffers is predefined and the functionality exists mainly to support ultra-fast access to the data the buffers contain, in order to take a trading decision. The functionality contains a buffer manager element that plays a role of a container for all the asset buffers available to share. The container provides a quick lookup/retrieval access to the buffers, while the buffers provide access to the data. The element is shared between all the strategies contained within the Strategy Manager.

To obtain data from the historic dataset, the low-level element provides a variety of methods for querying historical data. The retrieval may be performed with regards to some specified historical boundaries (applicable during back-testing), on the basis of the size of the history window, or with regards to the current time (this functionality is particularly useful during initialization of the asset buffer).

To subscribe to the real-time feeds (once the asset buffer is initialized and synchronized), the element allows the buffer to declare the asset type which it wants to be subscribed to, and the data type which it is going to buffer (market or candlestick data). The declaration is then passed to the connector and the processing functionality, which then provides a requested stream of data.

The Backtester & Optimizer are also part of the Developer Client over the course of the experiment. The Backtester functionality allows developers to test performance of the strategies they design (with regards to the historical data of the assets that the users intended the strategy to trade with). The Optimization functionality may be used to optimize factors of the developed strategy with respect to a pre-defined utility function also designed by the developer.

4.2.9 The User Client

The User Client (Figure 4.2) is a group of functionalities that enables individual users to manipulate their trading strategies, visualise market data and when necessary trade manually, e.g., to correct orders issued by AT strategies. The client is a web-based GUI (Graphical User Interface) that generates a presentation layer with use of JSP (Java Server Pages), JFreeChart and JasperReports. The principal functionalities are presented below:

The Content Navigation functionality of the client lists all the strategies and chart-templates of the authenticated user, which are currently stored on the server. The functionality also allows management of the content, the upload of the new strategies and chart-templates and the removal of the existing ones. The underlying solution that provides content navigation is based on monitoring of specific folder locations on the server, and dynamic loading of a new content that appears in the locations.

The Market Watch Grid functionality visualizes the current state of a selected number of securities (supported by the Environment) in the form of a grid of real-time, market-data prices. The functionality also supports manual trading via a trading window. The idea behind the functionality is to provide

Figure 4.2: The Graphical User Interface screen shot of the User Client.

users a simple means of evaluation of direction of a particular security. Therefore, each grid emphasizes temporal direction of a security as well as its bid/ask price, spread and volume characteristics.

The Order Book Window provides a real-time visualisation of the state of positions in the user's order book. The window is a table of positions that emphasizes information and statistics, as well as the state of the positions. The window also allows expansion of selected positions to provide information on particular execution reports of selected orders.

The Chart Window provides charting capabilities to the Environment. Each user is able to initialize up to 10 individual charts with different assets (or a variety of assets, depending on the implementation of a chart panel). By default the Environment provides two types of charts (market data and candlestick charts), however, if necessary, users can implement (with use of Developer API) and upload their own chart panels that will be displayed by the Chart Window functionality. The Chart Window functionality is a main means of controlling the execution of trading strategies. Strategies can be attached or detached to/from selected instance of a chart window to visualise positions managed by the attached strategy.

The Connection Status is a simple functionality to monitor and visualize the state of connection of the client to the server-side services. This is particularly useful to detect issues related to Environment's networking infrastructure.

The Console Terminal functionality provides monitoring and visualisation of important events, related to the user of the client, that take place within the Environment. This is particularly useful to monitor and react to internal info, issues and errors. The Environment utilizes Log4J functionality to provide logging facility and the Console Terminal uses the functionality to log such info.

4.2.10 The Statistics Client

The Statistics Client is responsible for monitoring and visualisation of trading statistics of all the users of the Environment. It is a web-based client that utilizes JSP (Java Server Pages) technology to generate

the presentation layer. The web-site is managed by the Apache Tomcat container. The client provides two main functionalities: the ranking functionality and the dossier functionality.

The ranking functionality is a visual representation of a ranking of users of the Environment on the basis of their P&L performance. The OME's accountancy functionality is queried by the client to provide account information. This is then utilized to form a ranking where each place in the ranking is estimated on the basis of a set of performance parameters that can be selected from a list provided by the client.

The dossier is an additional functionality allowing provision of a more detailed information about each account and an associated user.

4.3 Summary Discussion

As mentioned in the Introduction Chapter of the thesis, this work focuses on computational finance rather than performance problems encountered in software engineering research. As a result the scientific effort is directed at holistic, operational results of the experimental systems and the applicability of proposed software architectures to a given financial economic problem. This is reflected in the structure of the thesis. An overall operational experiment that utilise all the designed environments is described in the Evaluation Chapter of the thesis. The discussion of results related to this particular experiment is nevertheless provided below. This is also followed by comparison of performance results against set expectations.

There were five key research questions set at the beginning of the experiment and these were as follows: 1) what may be the typical classes of models that are to be supported by the environment? 2) what types of functionalities, libraries and tools may be necessary in large-scale analytical environments? 3) is it possible to identify typical modes of operation and information flows in such systems? 4) what is the most suitable software architecture solution and what are the most appropriate technologies that can be used to support such experimental systems? and 5) is it possible to identify and resolve various limitations that are imposed on these types of analytical systems?

The typical classes of supported models are in case of the ATRADE Environment all the algorithmic trading models summarized in Chapter 2. Moreover, during the course of the trading competitions organized with use of this Environment the participants were not limited with respect to the utilized classes of models and it was always possible for the participants to utilize the provided interfaces with no indication of technical limitations. This suggests that the proposed Event-driven and Service-oriented Strategy Templates were designed to capture all intended interactions with the Environment and that the templates are flexible enough to support analytic models for algorithmic trading.

The types of functionalities suitable to support a large-scale analytic environment were also summarized in Chapter 2. Given that all the components of the Environment were able to support the trading

competitions, demonstrates that the minimal amount of necessary components was identified.

The identification of typical modes of operation has shown to be an open-end process, this also applies to the types of functionalities, libraries and tools that are applicable to this class of models. At this stage it is clear that both analytic and trading models are data-driven and consequently their modes of operation are related to the ways the models access data. The information-flows in such systems are typically sequential with feedback loops.

The most suitable software architecture has shown to be one that allows scalable provision of large amounts of information to and from strategy templates. Furthermore, a mixture of simulation and real trading capabilities has shown to be most flexible from the point of view of analytic models. The virtualisation technologies have shown not to be suitable to the problem due to the imposed network latencies, this is also applicable to cloud computing. Big data facilities, together with cluster/grid computing were suitable solution to deal with the scale of the problem and have shown to be suited to improve the overall performance of the system, especially in the simulation part of the problem. The Complex Event Processing technologies have shown to be practical when applicable to information transformations. However, they were not applicable (due to the lack of necessary flexibility in the way the models can be constructed) to support analytical model hosting. The strategy templates have shown to be a more practical approach in this scenario. The distribution of the Environment on the basis of specialized engines has shown to be practical from the point of view of performance and module responsiveness. However, it also introduced problems with potential lack of persistent availability of engines. This needs to be improved further as such types of Environments should be characterized by high-availability.

The key Environment limitations identified in the course of the experiment can be summarized as: the limitations with respect to the amount of data that can be delivered to the users at a particular time, the amount of users that can actively participate in the environment at a particular time, the amount of order instructions the users can issue at a given time, and the amount of calculations the system is able to perform with respect to the P&L account and risk tracking.

4.3.1 Performance Tests

To address the above questions and limitations quantitatively the following performance tests were conducted. The experimental simulation environment is believed to be unique in academia for financial economics modelling. To work around a lack of comparable environments, a set of common sense thresholds was selected to evaluate the achieved performance. The thresholds define the maximum amount of a combi-

CPU	Intel Xeon Quad Core 2.83GHz 6MB L2 cache
Memory	4GB DDR2 667MHz
HDD	500GB SATA (7.2K rpm)
Network	10/100/1000Mbps
OS	Linux Fedora 14 x64
Java	Oracle Java 1.7.0 x64

Figure 4.3: Server spec.

nation of instances of active trading strategies and software development kits (SDKs) that are acceptable, before the performance of the environment drastically drops.

For the purpose of performance evaluation, these sets of experiments were performed with an individual (non-scaled) instance of the experimental services, hosted on a single Dell Power Edge R300 Server running with a set-up described in Figure 4.3

All the instances of the Environment's SDK, that were handling the running strategies, were hosted on a single ACER ASPIRE 5935G Laptop, with a set-up described in Figure 4.4

CPU	Intel Core2 Duo 2.4GHz 3MB L2 cache
Memory	4GB DDR3 1066MHz
HDD	500GB SATA (5.4K rpm)
Network	10/100/1000Mbps
OS	Windows 7 x32
Java	Oracle Java 1.7.0

Figure 4.4: Client spec.

Figure 4.5 lists the Environment's performance statistics based on tick data provided by Trading Technologies (TT) and Knight Capital (HotSpotFX), and shows the current amount of inbound tick data that the experimental Environment is able to handle.

TT Connectors	HotSpotFX Connectors	
300	265	The average message size (Bytes)
20	60	Amount of supported streams
2	7	The average amount of messages per stream (second)
12	111.3	The average bandwidth usage (KBytes/second)

Figure 4.5: Inbound Tick Data.

Figure 4.6 lists the Environment's performance statistics related to the current level of users and strategies managed by the experimental Environment.

Expected Mean	Mean	Deviation	Amount of Trials	
200	230	52	100	Amount of concurrent instances of SDK (one per user), the ATRADE platform can manage without significant performance degradation
100	579	67	100	Amount of concurrent instances of Strategies (per single SDK), the ATRADE platform can handle without becoming unstable
200:100	160:75	24:48	100	An average amount of instances of SDK and Strategies the ATRADE platform can manage without significant performance degradation

Figure 4.6: User Statistics.

The above results emphasize the worst-case scenario performance, at an unscaled level, where the software is limited by the underlying performance of a single hardware unit. Thanks to the modularization of the Environment and utilization of highly scalable technologies, the results scale well with additional server and client machines.

To summarize, the key strength of this Computational Simulation Environment lies in the fact that it

allows practical experimentation, rather than limited theoretical research often encountered in academia. The performance of the ATRADE Environment has been thoroughly tested by the UCL Algorithmic Trading Competitions (discussed in details in Chapter 6) held annually for students, researchers and academics to test their trading algorithms.

Chapter 5

PORTFOLIO SELECTION SYSTEM (PSS)

This chapter describes three important concepts in the process of financial portfolio selection, namely the concept of time-series similarity measure, the concept of formation of clusters (portfolios) of similar/dissimilar time-series, and the concept of a combinatorial search algorithm for discovery of new clusters. This is followed by a description of the design and implementation of a portfolio selection system for algorithmic trading, that can be utilized in creation of better portfolio-based strategies, pairs-trading strategies, statistical arbitrage strategies, hedging and mean-reversion strategies.

5.1 Financial Portfolio Selection/Construction

Portfolio selection in finance can be considered as a process involving analysis of behaviour of a wide range of financial securities to select a group that is most fit for a given purpose. The process of portfolio construction involves definition of the fitness measure and the actual construction of a portfolio from the selected financial securities. Both processes are a crucial part of the financial portfolio optimisation.

Based on the appropriately constructed portfolio, portfolio managers can achieve anticipated level of risk of the portfolio and either hedge the portfolio or achieve excessive return. Similarly, quants and traders can create profitable strategies. Appropriately selected, constructed and optimised strategies enable diversification of assets, hedging of risk, and more stable speculation.

5.1.1 Research Questions

The security similarity/dissimilarity discovery problem addressed by the Portfolio Selection System is a four-dimensional, multidisciplinary problem with the key factors being identified as:

1. The mathematical problem of defining a decision space of possible solutions, that involves researching a suitable similarity/dissimilarity measure and ensuring stability of the measure through time by optimal selection of granularity of data.

2. The machine learning problem of conducting intelligent search over the previously defined decision space. One of the key problems to solve is the identification of the most influential search-guiding features, and another is the design of an appropriate utility function that will utilize such features.

3. The information technology problem of streaming and buffering large amounts of market information (including identification of optimal time-granularity of data).

4. The financial problem of appropriate definition of financial data, its key characteristics and the most suitable granularity extraction.

In the light of the above list of challenges, the following research questions were formulated:

1. What are the most suitable similarity/dissimilarity measures one can apply in the problem?

2. What are the most suitable combinatorial search algorithms that can be utilized?

3. How can we define the search space and the main dimensions of the search space?

4. How can we define a suitable utility function that will guide the search algorithm through the decision space towards optimality?

5. How can we define optimality in this type of search problems?

6. What is the most suitable software architecture for an algorithm that can tackle these types of problems?

7. What are the most suitable technologies that may be utilized in these sorts of algorithms?

Portfolio Selection/Construction and Algorithmic Trading: As mentioned, the discovery of exploitable relationships between financial time-series is important for many algorithmic trading (AT) strategies. For AT the challenge is identification of similarities/dissimilarities in behaviour of elements within portfolios of tradable and non-tradable securities. Recognition of sets of securities characterized by a very similar/dissimilar behaviour over time, can be beneficial from the perspective of risk management, recognition of statistical arbitrage and hedge opportunities, and can be also beneficial from the point of view of portfolio diversification.

The AT strategies are built out of multiple stages of data analysis that allow control over execution of order instructions issued by the strategy. The pre-trade analysis phase is typically the first amongst multiple data analysis stages in AT strategies. During this phase the strategy identifies exploitable relationships between data, on the basis of which it may generate profitable transactions. Identification of

co-movement patterns, in the securities that the strategy trades, can be considered one type of data relationship that can be exploited to generate profit in AT. To identify co-movement patterns one requires a similarity measure that can allow estimation on how similar/dissimilar securities behave. Cointegration is considered one of such measures, particularly applicable to estimate stable co-movements of financial time-series.

Portfolio Selection System in the context of portfolio selection/construction: The PSS, an experimental system described in this Chapter, is a large-scale search system for discovery of sets of securities with an AT domain-specific similarity characteristics that can be utilized in creation of better portfolio-based strategies, pairs-trading strategies, statistical arbitrage strategies, hedging and mean-reversion strategies. PSS is designed to support algorithmic trading and data analytics. The system is able to handle streams of live market information. Data analytics on the system requires historical data, consequently it provides a 24h buffer of all the streamed information. The available information contains time-series of high frequency, top-of-the-book, best bid/offer tradable security ticks. The buffer can be utilized as a test-bed for combinatorial search and discovery of cointegrated financial time-series. The discovered cointegrated sets of financial time-series can be then continuously monitored to improve their analysis results.

Definition of the problem domain: A more formal definition of the discussed problem may be described as follows:

1. Given a set S of financial time-series of different securities with the same time-sampling (averaging) granularity g, where all time-series are continuous and are available on the basis of a 24h rolling window.

2. And given a similarity/dissimilarity measure CMP normalized between real values CMP : $-1 \ldots 0 \ldots 1$, where -1 describes complete dissimilarity, 0 describes neutrality (a complete lack of either similarity or dissimilarity features/patterns), and 1 describes full similarity.

3. We want to continuously search, in deterministic time, through as many subsets $s : S$ as possible, for signs of similarities/dissimilarities such that all combinations of pairs of time-series in each subset are either (or both) similar or dissimilar to each other with a minimal/maximal threshold set to be bound between $CMP \leq sdmt1$ and $CMP \geq sdmt2$. We want to partition S into subsets. Since the algorithm has to run in deterministic time, the partition does not have to be complete (i.e. not all time series in S have to be taken into account). Nevertheless, the bigger proportion of S is taken into account, the better. The subsets in the partition should meet the following condition: all combinations of pairs of time-series in each subset are to be either (or both) similar or dissimilar to each other with a minimal/maximal threshold set to be bound between $CMP \leq sdmt1$ and $CMP \geq sdmt2$.

4. The search should have a preference towards maximization of a size of a subset and towards binarization of result (similarity/dissimilarity as close to borderlines $-1/1$ as possible).

5. The selection of time granularity g should be such that it minimizes the granularity of time-series sampling, yet at the same time stabilizes the maximum change in the average similarity regime of all the discovered subsets to no more than asr over the last $asrwn$ of 24h windows.

6. Finally, the selection of $g, sdmt1, sdmt2, asr, asrwn$ should be calibrated with respect to point 4), with the initial setup being: $g = 15min, sdmt1 = -0.7, sdmt2 = 0.7, asr = 10\%, asrwn = 30$.

Assumptions: The notion of similarity can be applied to the individual time-series where sections of a selected time-series can be compared to each other (including fractal-like self-similarity comparison of a time-series defined with different granularities), can be applied to two or more time-series where one would look for similarities between selected time-series, or can be applied to groups of time-series where one would look for a set-to-set similarity.

In the light of the above there should bet therefore a clear distinction in expressing the concepts of:

1. a time series $X = x1, x2, x3, \ldots, xn$ and comparison of the same time series with different granularity $CMP(Xa, Xb) = xa1 == xb1, xa2 == xb2, xa3 == xb3, \ldots, xan == xbn$.

2. a similar set $S = s1, s2, s3, \ldots, sn$ (where si are financial time-series of identical granularity g) is similar if and only if each pair (si, sj) for all i, j between 1 to n is similar (passes a similarity/dissimilarity threshold). And both si, sj belong to S.

3. two mutually exclusive similar sets $S1$ & $S2$ are said to be similar if and only if each pair (a, b) of financial time-series where a belongs to $S1$ and b belongs to $S2$ are similar.

   We are interested in the second (b) option where the algorithm is looking for pair-wise similarities within constructed set, and the search algorithm generates/selects the most promising sets.

4. We assume that combinatorial pair-to-pair similarity in accepted subsets is (as in the case of the equivalence relation):

   (a) a $\sim$ a (a is similar to itself, reflexive)

   (b) a $\sim$ b implies b $\sim$ a (symmetric)

   (c) a $\sim$ b, b $\sim$ c implies c $\sim$ a (transitive)

   where a, b and c are sets of similar ($\sim$) financial time-series

5. All values in all involved time-series are assumed to be non-negative.

6. Subsets containing elements passing the dissimilarity threshold are considered to be carrying more value, as these types or relationships are less obvious and therefore less exploited in construction of AT strategies.

7. We assume the weakest-link similarity, where the 'closest to the threshold' value of the combination of time-series in a subset (the least advantageous one), defines the global similarity/dissimilarity in this subset.

8. Although it is assumed that the algorithm will work on continuous data, with a 24h rolling window, our definition of the problem domain is not assuming forward continuity of the described similarity/dissimilarity beyond the last 24h (no predictions are made).

5.2 Time-Series Similarity Measures

In computational finance research the problems of securities co-movement and tradable instrument selection, as well as their application to the arbitrage, hedging, pairs-trading, mean-reverting and portfolio-based models is already recognized by the academic community, with C. Alexander [Alexander and Alexander, 1999], Caldeira & Moura [Lin and Shim, 1995] and Lucas [Cui et al., 2012] being important examples. C. Alexander [Alexander and Alexander, 1999] gives an account of a proprietary system (WIES) for equity selection; based on cointegration between various international indices where equity selection criterion used is *highly technical proprietary part of the model* and indicates the possibility of a huge number of possible combinations. Apart from giving a concise overview of application of cointegration similarity measure to commodity markets, assets markets, options pricing and market efficiency, the paper compares utilization of two most common similarity measures, namely correlation and cointegration.

For completeness, the following describes the most commonly used time-series similarity measures applicable to financial time series:

Classic approaches to financial time-series analysis involve estimation of mean, variance, trend, seasonality or forecast. Methods applied to analysis of such information involves exponential smoothing, auto-regressive moving average and spectral decomposition amongst others. [Gunopulos and Das, 2000] provides an overview of the classic similarity measures.

Euclidean distance: amongst the various similarity measures the most basic one can be considered the Euclidean distance-based measure. In this measure each time-series can be considered one point in multi-dimensional decision space and the similarity can be defined as a p-norm distance (which is a n-dimensional, finite dimension generalization of two dimensional Euclidean distance between points). An advantage of this approach is the simplicity of its calculation. A disadvantage lies in the lack of reflexivity, where although it is easy to evaluate the relative similarity of points in close proximity, it is

impossible to measure the negation of similarity e.g., if two time series are both trending in the same direction they will most likely be similar, however if they are both trending in opposite directions it will be impossible to evaluate how perfectly dissimilar they may be.

A slightly more advanced, approach was proposed by [Goldin and Kanellakis, 1995] and was based on the idea of comparison of normalized (mean, variance) sequences. Further to this line of approaches, [Jagadish et al., 1995] proposed a general similarity framework based on transformation rules, followed by [Mendelzon, 1998] combination of Euclidean distance with moving average transformations.

Dynamic Time Warping: proposed by [Berndt and Clifford, 1994] is an approach that allows to measure the similarity between two time-series that may vary in length and frequency, by finding an optimal match between the time-series. The time-series are assumed to be non-linearly warped in the time dimension, and the DTW aims to establish their similarity independently of the non-linear variations in the dimension. This is achieved by a pairwise comparison of non-equally length time-series. The total distance between the two being the sum or the mean between the individual distances of the feature elements of time series. The approach assumes a grid comparison of the time-series with the best match between the two, described by a path through the grid which minimises the total distance between the time-series. Once an optimal path has been established, the total distance between the two time-series can be calculated.

Landmarks and Probabilistic Approaches: [Perng et al., 2000] proposes to base the similarity definition on landmarks, where landmark distances are defined as tuples satisfying triangle inequality. [Keogh et al., 1997] takes this idea further by proposing one that involves matching sequential patterns to a pre-defined time-series database. In this approach a piecewise linear segmentation (for curve representation in a parametric form) is used as an underlying representation where features are defined with prior distribution of expected deviations from stored templates. Selection of prior distribution impacts sensitivity of the distance measure. Such probabilistic model builds out of local features composed into global shape sequences and allows a degree of deformation and elasticity in probabilistic distance.

Statistical Dependence, Correlation and Association: according to [Wikipedia, 2013a] the dependence phenomenon refers to a statistical relationship between sets of data or situation in which sets do not satisfy a mathematical condition of probabilistic independence. While the correlation can refer to any departure of two or more random variables from independence, it is a broad range of dependence relationships and can indicate a predictive relationship, however is not sufficient and does not imply the presence of causal relationship. Correlation only indicates the extent to which the relationship can be approximated by a linear relationship. While correlation refers to a linear relationship, an association is any relationship between two measured sets that makes them statistically dependent. The key measures from the family include: Pearson product-moment correlation coefficient, Spearman's rank correlation coeffi-

cient, Kendall tau rank correlation coefficient, Distance correlation, Polychoric correlation, Coefficient of determination, Odds ratio, Goodman and Kruskal's lambda, Tschuprow's T and Cramr's V.

Cointegration: according to [Zietz, 2000] given that the change in stochastic drift for a variable x is α and the change in stochastic drift for a variable y is a linear function of α, then x and y are cointegrated, in other words time-series are said to be cointegrated if they share a common stochastic drift. Cointegration is tied to common factors among the random errors that drive the stochastic trends. The technique doesn't imply economic significance of any of the common factors. However, if the course of one of the cointegrated time-series was to be predicted then the long-run drift values of the other time-series can be derived. The key tests for existence of cointegration include: the Engle-Granger two-step method, the Johansen test and the Phillips-Ouliaris cointegration test.

Given the reflexivity and stability of similarity in time, it was concluded that Correlation and Cointegration type measures are the most applicable. Furthermore, since financial time-series are characterized by non-linear relationships and the fact that cointegration is considered a more stable measure in time, it was selected as a distance measure for the state-space portfolio selection problem.

The next section will describe various search and clustering approaches that can be utilized on top of the distance measure to support discovery and construction of financial portfolios.

5.3 Time-Series Subset-Search and Clustering

Clustering aims to identify structure in an unlabelled data by minimizing distance (similarity) between the in-cluster objects and maximising distance (dissimilarity) between the out-cluster objects. [Warren Liao, 2005] divides clustering into five categories: partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. These are defined as follows:

The partitioning approaches build partitions of analysed data with each partition representing a cluster containing at least one object.

The hierarchical clustering approaches work by grouping information into a tree of clusters.

The density-based approaches continuously grow clusters as long as the density of the number of individual information-points in the neighbourhood exceeds some predefined threshold.

The grid-based approaches quantify the information state-space into a finite number of cells arranged in a grid. The approach uses levels of cells on such grid, corresponding to different levels of resolution and for each layer it computes a confidence interval, to reflect the cells relevance (with irrelevant cells being removed from further consideration).

The model-based approaches assume a model for each of the clusters and attempt optimal fit of information-points to the assumed model.

The PSS project was partially interested in the hierarchy of time-series, namely it was assumed that the similarity measure describing distances between individual time-series is transitive. Consequently the selected approach may be assumed to be partially hierarchical. The system was not aiming at estimation of densities of particular clusters and consequently the density-based approach was excluded. Due to the complexity of search space the grid-based approach was considered an inadequate representation. Similarly, the model-based approach was not applicable due to the lack of definition of fitness of particular clusters. In case of the PSS project the mixture of partitioning and hierarchical approaches has shown to be the most appropriate based on the graph representation of data with nodes being particular time-series and leafs being distances between the time-series.

The graph based approach allowed simple representation of relationships between time series and more importantly enables fast traversal over the decision-space. The construction of sets of securities for time series was based on a semi-random traversal through the constantly extended graph where the borders of particular cluster were defined by threshold-based selection on the basis of the similarity measure.

5.4 PSS Functionality

The key features of the designed Portfolio Selection System can be described as follows:

Subscription and on-the-fly handling of streams of prices: in order to provide information about potential portfolios the system needs access to streams of financial data. Therefore, one of the features of the system is an ability to communicate with external streams of data via a JMS interface. This enables the system to subscribe and un-subscribe to and from selected data feeds as well as to handle incoming financial feeds.

On-the-fly raw data buffering: the system works on the basis of time-series of received feeds, and consequently provides buffering capabilities of the received feeds. Consequently, another feature of the system is its ability to aggregate and buffer the data feeds for the period of 24 hours.

On-the-fly data granulation: the aggregated time-series needs to be processed prior to further usage, therefore, another feature of the system is a possibility to granulate a new, raw time-series on the basis of a pre-defined time-frame as well as to granulate and append, on-the-fly, the newly added feeds to the previously granulated time-series.

Cluster discovery: another feature of the system is its ability to search and cluster cointegrated time-series, to provide portfolio selection capabilities. This feature searches over the graph-based decision space for a new pairwise-cointegrated time-series and attempts to add to clusters the already discovered pairs or form new clusters.

Cluster evolution monitoring: it is beneficial to be able to observe changes in evolution of strength of similarity in discovered clusters over time. Consequently another feature of the PSS systems is its

ability to measure cointegration in each discovered cluster with a pre-defined frequency and record the achieved similarity results to achieve a time-series of changes in cointegration of the cluster. The system disregards traced clusters when the traced similarity/dissimilarity turns out to be unstable.

Cluster query API: is a feature of the system that enables programmatic queries from external systems. The API provides access to information containing all the discovered clusters and their history. It also allows filtering with respect to the size of clusters and the cointegration strengths amongst others.

Big data provisioning: all the aggregated data, including information on the discovered clusters and their change in time can be requested via a REST interface from within a set of Microsoft Azure-registered workers, able to store the information to a big data storage on the Azure cloud. This functionality was later redeveloped to enable aggregation of data on a customized Hadoop-based cluster.

5.5 PSS Software Architecture

Given the discussed methodology and the definition of the mathematical domain, the following Figure 5.1 is a software architecture proposed for a large-scale search algorithm for a discovery of similarities in the subsets of financial securities:

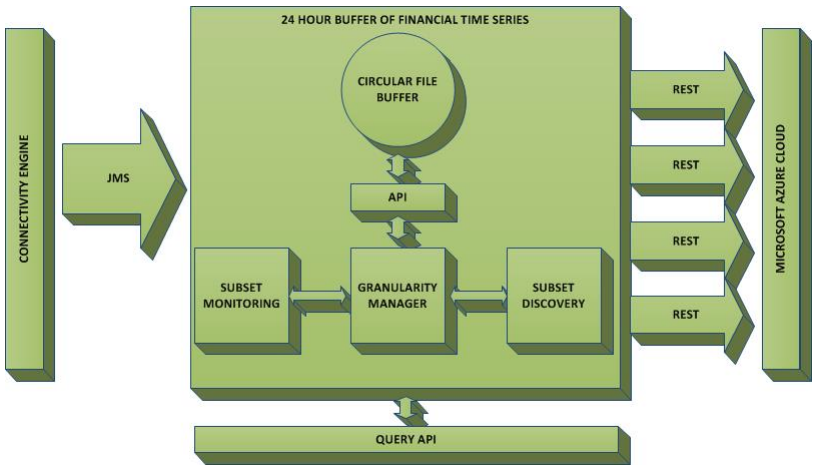


Figure 5.1: Key elements of the Portfolio Selection System.

Network Interface for JMS events: this functionality is supported by a set of classes and interfaces that enable PSS to utilize ATRADE's SDK interfaces to subscribe and handle tick events containing price information of tradable securities grouped by venues and gateways. The design of the ATRADE SDK interfaces followed the listener pattern and while implemented by PSS classes, allows subscription to streams provided by ATRADE. The key logic of this functionality requests from ATRADE a list of available trading venues, it then requests from each available venue a list of supported securities. The functionality then subscribes to all the available securities to receive feeds containing tick information. Each tick information contains information about tradable security as well as a temporal information about bid price, ask price, bid volume, ask volume and traded volume parameters of the security at a

time. The security information provides details on the symbol on the basis of which a particular security can be identified, as well as basic information on the origins and maturity of the security.

It is worth noting that due to the possibility of large amounts of data being streamed every second, a special version of the ATRADE API was designed and provided which attempted to compress the size of the sent information and to send optimally sized TCP messages. The compression approach involved the elimination of the security part from the content of every tick. Instead, a long identifier was used to uniquely identify a given security and during initialisation of the information feeding process, the maps of securities and their IDs were synchronized between the feed producers and consumers. Furthermore, if the bid and ask prices were represented as strings of numerical characters, in most cases they differ only by 2-3 characters. Consequently, a duplication removal technique was utilized that allowed to store the shared parts of numbers. Finally, in order to fully utilize the maximal binary packet-size of underlying TCP protocol, all ticks were turned into a binary representation and appended to TCP packets that exhibited free space. As a consequence, each packet carried on average 6-7 ticks and the achieved compression inside each tick was between 4-8 times better.

Additionally, the possibility of handling the full-book (the multiple levels of bid/offer prices) and the top-of-the-book (the best bid/offer) tick-information was provided with typical full-book number of levels averaging between 7-10.

Finally, the functionality has a possibility of dynamic management of subscriptions to the feed streams with periodic requests to ATRADE for an updated list of supported securities. On such basis, the functionality may un-subscribe the streams that no longer provide data (e.g., due to the end of the trading day or due to the maturity of a given security) and subscribe to the newly supported securities.

Network Interface for REST requests: while the previously described set of classes was responsible for a provision of large amounts of streamed data to the system, the REST network-interface can be considered a set of classes responsible for the provision of data out of the system. All the raw time-series buffered for the 24h in the PSS system need to be, at some point of that 24h, retrieved from the buffer and stored in their final destination on a big data storage cluster. Considering the fact that the cluster-workers pooling data from particular nodes of the cluster can at any point be disabled, it was necessary to ensure that this PSS interface is capable to serve multiple concurrent REST requests and that every request is able to state the point in time, from which on, the cluster-workers need to retrieve buffered information. The functionality is then able to provide history from the requested point in time as long as it is within the 24h buffer threshold. The same functionality is applied to the already processed data. Consequently, the Network Interface for REST requests is able to serve multiple cluster-workers at the same time with chunks of raw or analytic time-series.

An information compression algorithm similar to the one described in the previous paragraph was uti-

lized in reduction of the transported data. One major difference in the REST interface was that the ticks were grouped in the TCP packets on the basis of the security. This was due to the fact that the cluster storage organizes the tick time-series according to the security key and, consequently, the write time to the cluster-based hard-drives is shortened this way.

Network Interface for Queries: this set of classes is provided to support querying, filtration and retrieval of cluster-related analytic results achieved by the system. An SQL-based database management system is utilized by the functionality to enable efficient retrieval and filtration of stored and monitored clusters. A programmatic access was designed to provide access to e.g., ATRADE-based algorithmic trading systems interested in financial portfolio construction and management on the basis of clusters discovered and maintained by PSS. This interface provides a possibility of querying and filtering on the basis of the size of a cluster, the strength of the relationship in the cluster (both similarity and dissimilarity), the asset class of a security and on the basis of the particular security being part of the cluster.

24h Circular Buffer with an API: this functionality contains a set of classes that are responsible for an efficient, high-frequency read and write operations on a hard-disk-based storage, to support buffering of streams of data for 24 hours. This functionality is characterized by a smartly optimized design similar to the flat-file databases. The information is organized on the basis of securities and days. A directory structure that holds the storage file system is organized on the basis of uniquely identifying names of securities. In each one of such directories lies a set of 25 files holding one hour worth of information, the corresponding storage files are being named after the security and the file-creation time. Each file is organized on the basis of a map holding a binary information retrieved from the messages as well as their timestamps and a pointer to the end of the current binary representation. Such representation allows extremely fast traversal through the file system and through the particular files, to find a particular place in the file from which the functionality can start reading.

For the purpose of the speed, the processes of reading and writing are organized to utilize RAID mirroring with one serial writer on one hard-drive, and multiple readers on the mirroring hard-drive. Two 2TB SATA HDD were estimated to be optimal choice for storage. This came from the fact that an estimated 10000 different securities is the maximal amount to which the PSS system can have access via ATRADE. Considering the frequency and the average size of the data feeds it was estimated that an average of 1-1,5TB of data can be expected with the max. amount of available securities. Adding the space required by an OS, the 2TB was decided to be optimal, with a selection of HDD rather than SSD technology being price-wise optimal.

Granularity Manager: is a set of classes organized to provide functionality able to granulate the raw data stored in the 24h buffer into evenly sampled (with respect to the same timeframe, i.e. 1 minute timeframe) candle representations. This functionality is able to retrieve information from the buffer,

then granulate it and save the granulated data as a separate part of the buffer (for consideration of other functionalities in the next 24 hours). Based on the experiments with different granulation-timeframes, with an objective set to achieve the most stable clusters, it was estimated that one minute candles provided the highest stability. The utilised candle-representation of data is one of the standard financial charting representations that takes into account the open price at the beginning of the sample, the close price at the end of the sample as well as the highest and the lowest prices in the course of the sample. The manager functionality is supported by a thread-pool of workers, to which the manager delegates the process of granularity generation, by providing the location of data that is to be granulated as well as the granulation period.

Subset Discovery Manager: is a set of classes supporting the functionality able to represent the partial search-space known at a given time, able to expand the current representation with a newly estimated decision points as well as to form clusters out of the known decision space. The representation of a partially known search-space is maintained in a form of a graph of nodes representing granulated time-series and edges representing cointegration between the time-series. The process of discovery and expansion is based on the following rules: a) a time-series is chosen at random from amongst all the available raw financial time-series in the 24h buffer, b) if the selected time-series is part of any discovered graph the algorithm checks the availability of granulated data for the time-series and if it cannot find it then c) the time-series is granulated d) the algorithm picks at random another time-series that is already part of one of the discovered graphs, and if no graphs are available, it picks another raw time-series, granulates it and attempts to calculate cointegration e) if a cointegration strength passes a satisfactory threshold (a conservative two standard deviation confidence was selected for a threshold in PSS) it is assumed that the two elements form a cluster. Moreover, it was assumed that cointegration is a reflexive and transitive measure and therefore if any of the time-series in the considered pair forms a cluster with the other time-series it is assumed that a larger cluster is formed out of all the cointegrated pairs. If, on the other hand, the threshold is not matched or passed, the combination is disregarded and the algorithm moves to the next random selection. The algorithm is constructed to favour the extension of already existing graphs with the occasional random selection of ”non-associated” time-series. This may be considered similar to the ’genetic algorithms approach’ to evolution (the interaction between the cross-over operator and the mutation operator). The average process of granulation and evaluation of an individual pair is calculated to take on average between 20-21 minutes. The functionality is supported by a thread-pool providing a possibility of parallel execution of the estimation and discovery processes.

Subset Monitoring Manager: is a set of classes that extends the Subset Discovery Manager with a possibility to monitor the already created partial representation of search-space, as known at a given time, with scheduled recalculations of edges between the nodes. Given that clusters in PSS tend to contain or

be contained by other clusters, the goal of the functionality is to recalculate every edge in every cluster with a 24h sampling window. This creates a view of evolution of the strength of cointegration in time, for every available cluster. If, over a period of a week, a given cluster displays a significant volatility in its cointegration it is being dismembered, one weakest edge after another, until the measure becomes stable or the cluster disappears. The results of the recalculations are recorded to form a time-series of the cointegration changes for a cluster. Such time-series is available as part of the querying API. The monitoring manager, as in case of the discovery manager, is supported by a thread-pool to provide parallel execution of recalculations.

Azure-based Data Aggregator: is a set of classes providing extension to the core of the PSS system, to enable a wider experimental software infrastructure to aggregate and utilise all the data buffered and generated by the PSS. Microsoft Azure is one of the few available cloud-computing environments providing large-scale, distributed storage and computing environments. In order to enable storage of the PSS data in Azure a set of Azure-hosted worker-functionalities was implemented. These are capable to initiate and request chunks of data from the PSS via a REST communication protocol. Azure enforces some limitations on the external functionalities that are integrated with it: a) the transfer of information out of the cluster is expensive (while the transfer of information to the cloud isn't), b) only the REST communication protocol is supported in and out of the cluster, c) utilisation of global message queueing functionalities in Azure, as well as the MSSQL functionalities, is also relatively expensive while even a large binary storage is considered inexpensive, finally, d) all Azure-hosted services should be considered stateless because in any point in time the physical computer node that hosts the service can be taken down. With all the above in mind, an optimal solution for aggregation of large amounts of financial data to the cloud was to utilize a collection of Azure interfaces from AppFabric, Blob Storage and Azure SQL to implement a Worker Role with the following business logic: a) in order to specify the amount of separate active workers and their state, the AppFabric was configured with 5 workers and a configuration to keep them always active (in the instance a worker was taken down, together with its hosting node, the fabric was automatically detecting it and starting a new worker on an available node) b) all the workers have a shared state in a form of an identifier of a currently retrieved chunk of financial information from the PSS system. Such information, forming a state of each worker, is saved to the Azure's SQL storage and flagged as completed upon successful write of the currently held chunk to the Blob Storage c) in the instance that one of the workers was restarted by the AppFabric, it retrieves the state of all the workers from the SQL storage and picks up where the last worker finished, by either retrieving a new chunk of information or an old chunk, in the instance the previous write was unsuccessful. The organisation of the Azure blob storage, for the aggregation purpose, resembles the organisation of data on the PSS's 24h buffer functionality.

Design Assumptions , for completeness, the following set of points provide the key assumptions in the designed functionalities described above:

1. The part of the algorithm that provides the API access to the raw buffered data and the granularity manager that allows sampling (averaging) of the raw data to a more optimal granularity can be functionally described as follows:

   GIVEN a request for a time-series of a particular security with a defined granularity of sampling (averaging) THEN IF requested time-series with a specified granularity doesn't exist in the monitoring record THEN delegate to (C), set continuity of monitoring for 48h, wait for result RETRIEVE result and return it.

2. The part of the algorithm that provides the similarity search and discovery, over the financial time-series subsets, can be described as follows:

   WHILE the system is active THEN Pick the content of a new subset according to the guidance of the utility function, Calculate all time-series, as per the subset of securities with supported granularity, FOR all the combinations of pairs in the subset Calculate the pair-wise similarity IF the pair-wise similarity failed to pass the thresholds then reject subset and RETURN IF the similarity measure of all the pairs in the subset passes the thresholds, then assign the lowest achieved similarity value as the global value to the subset, set the continuity of monitoring for 48h and add to a list of monitored securities.

3. The on-the-fly tracking of similarity-changes in the previously discovered subsets (that have been qualified to be tracked), can be described as follows:

   WHILE the system is active THEN RETRIEVE the list of monitored securities FOR every element in the list IF the current element is outdated then mark it to be removed from the monitored list ELSE check the granularity and the last monitoring point AND IF the timestamp + granularity ¡= the current time THEN evaluate the similarity within the subset for the next point in time and update the historical track-record of the similarity for this subset

A typical user of the algorithm will try to execute the following queries to extract and filter available data:

1. Return all currently available (discovered) subsets of financial time-series
2. OR Return positive only (similar), currently available (discovered) subsets of financial time-series
3. OR Return negative only (dissimilar), currently available (discovered) subsets of financial time-series

4. AND/OR Return currently available (discovered) subsets of financial time-series with the amount of elements in the subset to be larger/smaller/equal to x

5. AND/OR Return currently available (discovered) subsets of financial time-series with the degree of similarity/dissimilarity to be larger/smaller/equal to x (where x is bounded between -1..-0.7 and 0.7..1)

6. Return the available history of similarities for the selected subset.

It is also assumed that the returned information will contain the following set of meta-properties:

1. Each element in the subset will have defined a similarity/dissimilarity value with respect to all other elements in that subset.

2. Each subset will have information about its general similarity/dissimilarity.

3. Each subset will have a beginning and end timestamp and a granularity of data over which the similarities were calculated.

5.6 Summary Discussion

As mentioned in the Introduction Chapter of the thesis, this work focuses on computational finance rather than performance problems encountered in software engineering research. As a result the scientific effort is directed at holistic, operational results of the experimental systems and the applicability of proposed software architectures to a given financial economic problem. This is reflected in the structure of the thesis. An overall operational experiment that utilise all the designed environments is described in the Evaluation Chapter of the thesis. The discussion of results related to this particular experiment is nevertheless provided below. This is also followed by comparison of performance results against set expectations.

Algorithmic trading strategies are I/O systems that, based on the market-related input information, generate outputs in the form of order instructions. One can differentiate the pre-trade analysis, the signal generation, the trade execution, and the post-trade analysis stages in AT that map input to output. The selection of input parameters in a strategy depends on a need for the information related to the securities that the strategy trades, the securities with relation to which the strategy may potentially trade, or the trade execution reports that provide feedback on the already issued order instructions.

The pre-trade analysis is intended to improve the indicativeness of input data in AT. The raw data in the analytic models seldom carry the desired level of 'importance' to support the decision-taking process. To improve the quality of data, the stage is further divided into the pre-processing, the feature selection/extraction and the cross-validation. The pre-processing ensures that the raw data is de-noised,

de-trended, normalized etc., before being considered further. The feature selection ensures the selection of a set of relevant statistical features of the pre-processed data. The feature extraction ensures he tcalculation and retrieval of the selected statistical features. Finally, the cross-validation ensures that only the most indicative features in a set are being used. Given such list of strong indicators, the trading algorithm may start to form patterns out of the features of the input data. The formed patterns may then be utilized in the signal generation and other decision-taking processes. The formation of the trade-exploitable patterns is often based on a recognition of co-relations between the different features of the input data.

The raw market information utilized in the feature extraction can be classified as either: an internal order book event from an exchange, a tick data type (full book or top of the book - best bid/offer tick data), and (if larger granularity is necessary) a candlestick data type (open/high/low/close prices). Typically, the features are formed from the raw data with use of statistics, technical analysis, data mining or econophysics. On the basis of such features, the strategies can form patterns and identify repetitive, exploitable relationships in data, either within the time-series of a particular security or between the time-series of one or more securities (where the latter is considered to be more valuable as it is applicable to a broader range of trading models). The co-movement patterns between multiple securities are one example of such relationships.

The various similarity/dissimilarity measures were developed over the years to estimate the strength of relations between two or more time-series. Some basic measures utilize Euclidean distance or statistical approaches to the problem. Two measures that were historically used to describe the co-movement in financial time-series are correlation and cointegration. Correlation is intrinsically a short-run measure as it reflects the co-movements in returns that are liable to significant instabilities over time. Cointegration is considered better-fitted for modelling of the dynamics for both the short-run and the long-run systems as it measures the longer-run co-movements in prices, which makes it intrinsically more stable.

As a result of the experiment, a large-scale search algorithm for discovery of co-relations in subsets of financial securities was built. This was supported by a functionality for the on-the-fly monitoring of fluctuations in already discovered similarities, to track the longer-time changes in behaviour of identified co-relations. The algorithm aims to be utilized to support the pre-trade analysis process in AT strategies on the ATRADE environment.

The Connectivity Engine, a part of ATRADE responsible for maintaining communication of the Environment with the electronic exchanges, is the main source of the streamed financial data, utilized in the proposed functionality. With use of an asynchronous messaging system it provides the market price information to the distributed elements of the Environment. One of such elements is a 24h buffer of financial time-series that buffers the security prices for 24 hours to make them available for other elements, per request. The 24h buffer functionality may be considered a test-bed for the developed PSS system.

The test-bed hosts a search engine that applies the cointegration similarity measure to combinations of securities and performs 'intelligently-guided' subset searches. The 24h buffer is also considered the main proxy of data from where the information is retrieved and stored to a big data facility on Microsoft Azure cloud (for further analysis and for simulation purposes).

Five categories of AT models were identified that may benefit from utilization of the proposed PSS system. These are the arbitrage models, the pairs trading models, the hedging models, the portfolio-based models and the mean-reversion models. The advantage the proposed solution provides to the AT models is the fact that, typically, the more similar/dissimilar behaviour of the underlying data, the better the arbitrage, the hedging, and the replication that can be achieved in the models.

The following research questions were formulated at the beginning of the experiment:

What are the most suitable similarity/dissimilarity measures one can apply in the problem? The conducted research was summarized with the selection of Cointegration as the most suitable similarity/dissimilarity measure for the problem. This was mainly due to the stability of the measure as well as its capability of capturing the non-linear relationships in time-series. Furthermore, the measure was able to capture the reflexivity, which is particularly important in finding dissimilar relationships that are found to be often exploitable in financial applications. Further research on the transitivity capabilities in cointegration is currently conducted to prove or disprove this assumption in the project.

What are the most suitable combinatorial search algorithms that can be utilized? How can we define a suitable utility function that will guide the search algorithm through the decision space towards optimality? How can we define optimality in this type of search problems? The application of a guided search algorithm to this problem has shown to be difficult to resolve, mainly due to the lack of suitable factors that could be used as a utility function to guide the search. Work is being done on a probabilistic representation of possible directions (edges in the graph), however, given the time-constraints, the author decided to exclude it from the thesis.

How can we define the search space and the main dimensions of the search space? A search-space is defined by the key features of elements that are part of it and, consequently, depends on the similarity measure. In case of the PSS problem no explicit features of granulated financial time-series were necessary and the graph-based search-space representation has shown to be particularly flexible, especially from the point of view of the extendibility (when the new nodes and edges were added or removed).

What is the most suitable software architecture for an algorithm that can tackle these types of problems? The author believes that the presented software architecture has shown to be suitable for the problem. The architecture evolved through the course of various experiments with different approaches to the portfolio selection problem and so far has shown to be working in practice. Some research still needs to be performed on proving or disproving the assumption of cointegration transitivity, this however

will not influence the architecture itself. This is due to the modularization of the architecture, namely the utilisation of interfaces between the JMS, the REST, the 24h buffer and the discovery & management elements. It should be relatively easy to replace one of the existing components with a more appropriate solution, if necessary.

What are the most suitable technologies that may be utilized in these sorts of algorithms? The problem required the fast streaming and the database management technologies with a significant but fully predictable (stable and typically high) information throughput. The considered database technologies available at the time were not fast enough (standard relational databases), not predictable enough (e.g., Hybernate), or too expensive (KDB+) with respect to the storage capabilities. Consequently, a proprietary solution was designed. Amongst the event-based technologies for data streaming, two were considered, ActiveMQ and ZeroMQ. However, due to the lack of time and the fact that ActiveMQ was already utilised by ATRADE, and supported more than the point-to-point communication (as in case of ZeroMQ) it was decided that ActiveMQ will be the technology of choice, complemented by the Kafka technology.

5.6.1 Performance Tests

To address the above questions and limitations quantitatively Figure 5.2 lists the key statistical experiments related to the performance evaluation of the PSS Environment. The experimental simulation environment is believed to be unique in academia for financial economics modelling. To work around a lack of comparable environments, a set of common sense thresholds was selected to evaluate the achieved performance. The thresholds define the maximum amount of processing time & information rate that is acceptable for various functionalities of the environment.

Portfolio Selection System				
The description of a PSS experiment	Expected Threshold Levels	Obtained Results		
	The maximal expected value	The minimal registered value	The average registered value	The maximal registered value
The rate with which PSS is able to consume market prices and aggregate it in the 24h Buffer.	1000/sec	14681/sec	15000/sec	15549/sec
The rate with which PSS is able to retrieve aggregated data from the 24h Buffer and perform data granulation.	500/sec	2200/sec	8000/sec	8150/sec
The amount of time necessary to select the most promising node from which the algorithm will expand selection.	30sec	4.8sec	5sec	15sec
The amount of time necessary to calculate transitivity probabilities from a selected node.	60sec	38sec	39sec	67sec
The amount of time necessary to calculate a new cointegration edge in the graph.	1200sec	80sec	81sec	153sec
The amount of time it takes to execute a selection query.	5sec	< 1sec	< 1sec	5sec

Figure 5.2: PSS Environment performance statistics.

The rate with which the PSS is able to consume the market prices and aggregate them in the 24h buffer defines how fast the Environment is capable of delivering the most up-to-date information to the calculation modules. Furthermore, this component of the Environment is particularly susceptible to the amount of data and consequently should be as fast as possible.

All the operations in the Environment are performed on the granulated, rather than raw data. Consequently, the rate with which the PSS is able to retrieve the aggregated data from the 24h buffer and perform the data granulation is important (as that is the intermediary step between the data retrieval and the cointegration discovery).

The following three statistics define the time necessary to perform an individual cointegration exploration, this involves: the selection of the most promising node in the graph, the evaluation of the cointegration transitivity probabilities from that node and, finally, the actual cointegration calculation from a selected edge. Consequently, all three statistics: the amount of time necessary to select the most promising node from which the algorithm will expand the selection, the amount of time necessary to calculate transitivity probabilities from a selected node, and the amount of time necessary to calculate a new cointegration edge in the graph are of interest to the software architecture researcher.

From the users' point of view the most crucial performance statistic is the amount of time it takes to execute a selection query. Given that the state of the cointegration graph is continuously recalculated and the actual query performs only a static selection procedure, this functionality scales well and has a relatively flat response time.

Finally, for completeness, all the core elements of this experiment were designed and implemented by the author with support of a group of BSc, MSc and PhD students. The author of the thesis was responsible for the research and design of the software architecture of the elements, for the implementation of the key functionalities, for the supervision of multiple groups of students and for the support of the hardware, software and support-services infrastructure.

Chapter 6

EVALUATION AND ASSESSMENT

This chapter presents the evaluation of the three performed experiments by verification of the quality, performance and stability of the created Computational Simulation Environments. The evaluation is performed in three stages: stage one presents the results of software testing through continuous integration, stage two presents the performance tests of the Environment, and stage three describes the holistic approach to testing the stability of the systems through utilisation in algorithmic and manual trading competitions.

Three key experiments have been conducted in this thesis: a) an experiment investigating the design of a Computational Simulation Environments, b) an experiment investigating the design of Electronic Market Simulation Models, and c) an experiment investigating algorithms and the design of a Portfolio Selection System. The evaluation of such large-scale computational simulation systems inevitably has shown to be troublesome due to their scale and the number of features, and functionalities they aimed to provide. Since the environments were predominantly designed for testing, optimisation and monitoring of algorithmic trading and risk models, running in virtual or real-trading mode, the author decided to approach evaluation of the described experiments holistically, in three distinctive ways. These three ways are representative of the working of all of the elements together: a) the first one being with use of an automated unit, integration and system tests, being part of the continuous-integration service supporting development of the experiments, b) the second one being with use of a set of system-performance tests oriented around the speed of information delivery and execution of models powered by the delivered information, c) the third and most extensive evaluation was undertaken through organization of an algorithmic and manual trading competitions, with use of the experimental software, carried out over the period of three years.

The following sections provide details of the three evaluation methods.

6.1 Evaluation of Environment quality

To increase the quality of the code, and consequently the reliability and performance of developed experimental systems up to the industry standard, it was decided to adopt a mixture of test-driven, continuous-integration, issue-tracking, build-management and source-code management methodologies. Consequently, the following technologies were used to support the methodologies, with their use described below: YouTrack (http://www.jetbrains.com/youtrack/), Git (http://git-scm.com/), Maven (http://maven.apache.org/), JUnit (http://junit.org/), Mockito (http://code.google.com/p/mockito/), TeamCity (http://www.jetbrains.com/teamcity/), Nexus (http://www.sonatype.org/nexus/), Sonar (http://www.sonarqube.org/), and ServiceMix (http://servicemix.apache.org/).

A typical action-flow within such methodology would be as follows:

1. A developer assigns himself/herself to one of available tasks on YouTrack (http://svarog.cs.ucl.ac.uk:8080/youtrack/); utilised as an issue tracking and agile project management system.

2. The developer then checks-out necessary source-code from Git (e.g., svarog.cs.ucl.ac.uk/srv/ATPCommonsRepo.git) to his/her favourite IDE. The checked-out code is managed by Maven. Git is utilised as a distributed revision control and source code management system, while Maven is utilised as a software tool for project management and automated builds.

3. The developer then implements the tests and pieces of code that satisfy requirements of the selected YouTrack task and the tests, this is then committed back to the Git repository. JUnit and Mockito frameworks are utilised for testing and mocking purposes.

4. TeamCity, a continuous-integration service (http://svarog.cs.ucl.ac.uk:8080/teamcity/), monitors all Git repositories and if the code is committed to one of the source-code repositories then the service compiles the code, executes all the tests, packages the compiled code and pushes such packaged binary repositories to a Nexus service, where Nexus is utilised as a manager of binary software artefacts (http://svarog.cs.ucl.ac.uk:8080/nexus/). In the instance the service encounters any problems with compilation or with any of the unit tests, it generates a report and e-mails it to the developer of the code, to prompt an issue that needs to be resolved.

5. Furthermore, when TeamCity executes all the tests, it pushes such information to a Sonar service (http://svarog.cs.ucl.ac.uk:8080/sonar/) which continuously analyses and measures the source-code quality, performs all additional code checks and visualises the current state of the source-code in Git repositories. Sonar is utilised as a quality management platform.

6. Once all the previous stages are successfully passed then the binary repositories committed to Nexus are deployed and initiated as software services in the ServiceMix container, which is utilised

as a an enterprise-class open-source distributed enterprise service bus (ESB) and a service-oriented architecture (SOA) toolkit. The experimental services reside in ServmiceMix's OSGI container.

The above mixture of support services allow automated and continuous unit-based evaluation of the developed experiments. The following four Figures present an outcome of such evaluation for one of the source-code repositories (a JMS source-code repository) which provides the event-based communication capabilities between the modules of experimental software. Figure 6.1 presents the key characteristics of the source-code in this repository. The repository at the time contained around 2500 lines of code. The total lines of code for all the repositories developed for the thesis exceeded 15000 lines of code. Rules compliance in the example package exceeded 90% while the average rule compliance in all the repositories oscillated around 80%. Unit test coverage in the example package exceeded 80% while the overall result for all repositories was slightly below 80%.

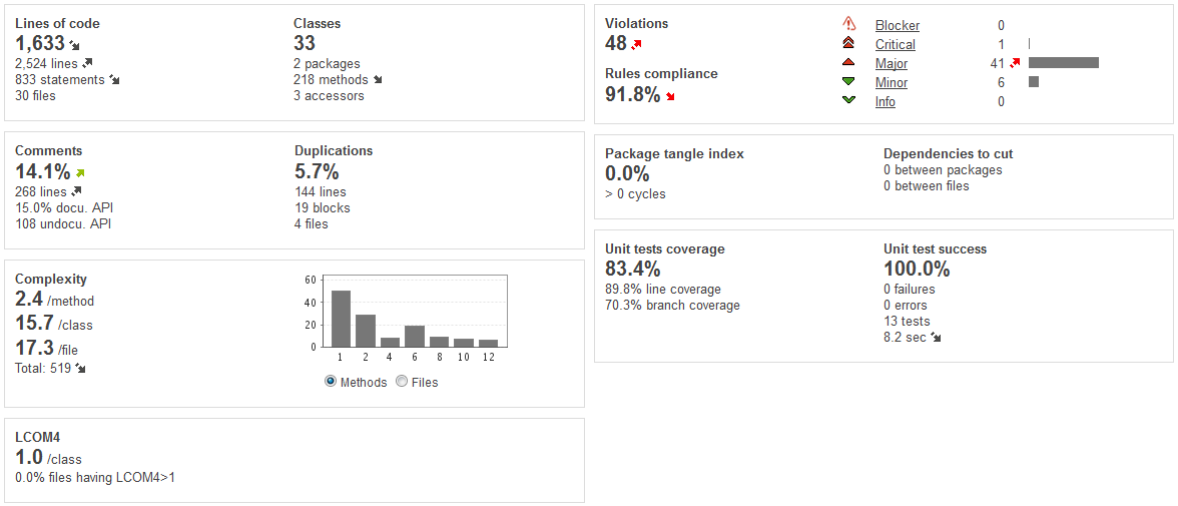


Figure 6.1: Example Sonar's Dashboard functionality for one of the evaluated source-code repositories (JMS).

Figure 6.2 presents the key Hotspots in the JMS repository, with a breakdown of key violations and levels of violations recognized by Sonar. Apart from violation breakdown, the Figure also presents duration of slowest tests, complexity of classes and most undocumented interfaces.

Given that this type of evaluation methodology is dynamic in nature, Figure 6.3 presents evolution of changes in the example JMS repository over a few months' period. This functionality is available for all the repositories utilised in experiments.

Figure 6.4 presents the success rate for all the steps performed in a build process of the example JMS repository. All the repositories are confirmed to compile the code, execute all the tests, package & deploy binaries to Nexus. The Figure also presents the overall time spent in the build process.

This concludes the evaluation through continuous-integration. The next evaluation approach focused on

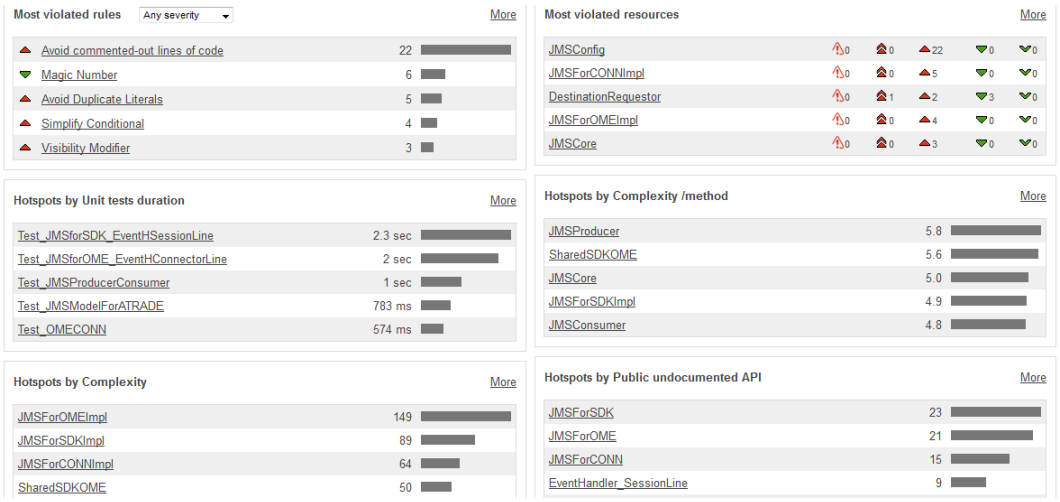


Figure 6.2: Example Sonar's Hotspots functionality for one of the evaluated source-code repositories (JMS).

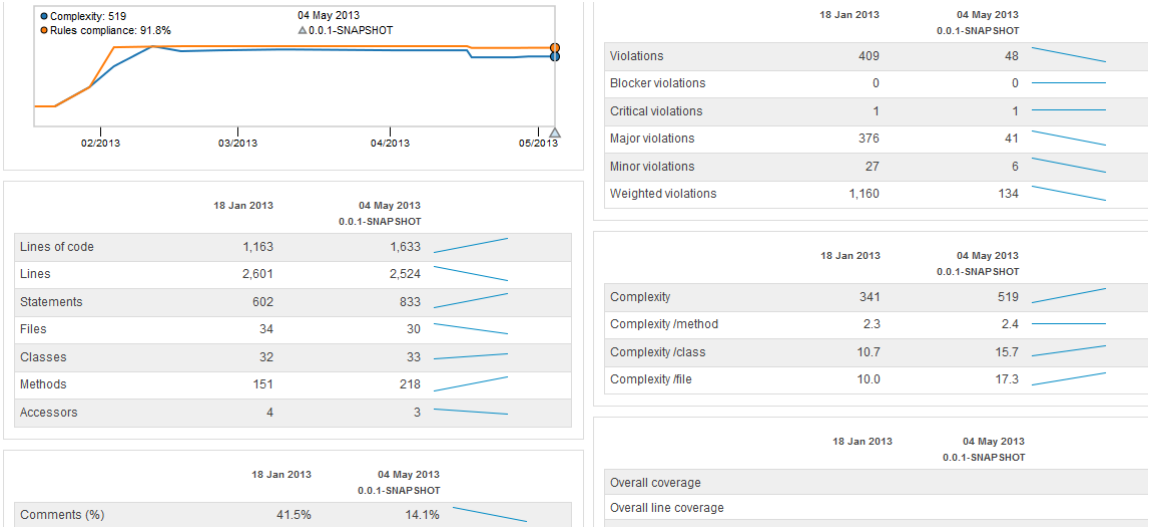


Figure 6.3: Example Sonar's Time Machine functionality for one of the evaluated source-code repositories (JMS).

the performance of experimental systems.

6.2 Evaluation of Environment performance

The key challenges in the evaluation of the enterprise-size financial systems are the performance, the scaling (with respect to the amount of data, the amount of users and the amount of trading algorithms being executed concurrently), the variety of types of latencies and the fault tolerance of the system. Different elements of the experimented systems are tested periodically during development, re-factoring and during the live events organized on the basis of the Environment.

In 2010 the stability of the Environment was tested with 100 high frequency trading models continuously running in parallel over a test period of one week. The models were designed to be able to subscribe to

Figure 6.4: Example TeamCity's statistics for one of the evaluated source-code repositories (JMS).

market events of 20 major currency pairs and to issue order instructions each time a new market event occurred. It is common that at least a few market events occur each second, for every currency pair, consequently triggering a few dozen orders a second in every strategy. During the 2011 competition over 300 traders used the Environment to trade with one or more models. Over the last two years the Environment has been tested with usage of between 10 to 70 streams of live market feeds of different liquid securities; it exhibited no significant issues with internal performance or latency.

For completeness, as already emphasized in Chapter 3, Figure 6.5 lists the Environment statistics on tick data from Trading Technologies (TT) and Knight Capital (HotSpotFX), and shows the current amount of inbound tick data that the experimental Environment is able to handle.

TT Connectors	HotSpotFX Connectors	
300	265	The average message size (Bytes)
20	60	Amount of supported streams
2	7	The average amount of messages per stream (second)
12	111.3	The average bandwidth usage (KBytes/second)

Figure 6.5: Inbound Tick Data.

Figure 6.6 lists the Environment statistics related to the current level of users and strategies managed by the experimental Environment.

Expected Mean	Mean	Deviation	Amount of Trials	
200	230	52	100	Amount of concurrent instances of SDK (one per user), the ATRADE platform can manage without significant performance degradation
100	579	67	100	Amount of concurrent instances of Strategies (per single SDK), the ATRADE platform can handle without becoming unstable
200:100	160:75	24:48	100	An average amount of instances of SDK and Strategies the ATRADE platform can manage without significant performance degradation

Figure 6.6: User Statistics.

For the purpose of performance evaluation, the above sets of experiments were performed with an individual (non-scaled) instance of the experimental services, hosted on a single Dell Power Edge R300 Server running with the following setup:

CPU	Intel Xeon Quad Core 2.83GHz 6MB L2 cache
Memory	4GB DDR2 667MHz
HDD	500GB SATA (7.2K rpm)
Network	10/100/1000Mbps
OS	Linux Fedora 14 x64
Java	Oracle Java 1.7.0 x64

Figure 6.7: Server spec.

All the instances of the Environment SDK, that were handling the running strategies, were hosted on a single ACER ASPIRE 5935G Laptop, with the following setup:

CPU	Intel Core2 Duo 2.4GHz 3MB L2 cache
Memory	4GB DDR3 1066MHz
HDD	500GB SATA (5.4K rpm)
Network	10/100/1000Mbps
OS	Windows 7 x32
Java	Oracle Java 1.7.0

Figure 6.8: Client spec.

The above results emphasize the worst-case scenario performance, at an unscaled level, where the software is limited by the underlying performance of a single hardware unit. Thanks to the modularization of the Environment and utilization of highly scalable technologies, the results scale well with the additional server and client machines.

This concludes the performance evaluation of the experimental Environment. The following section describes evaluation of the systems through organization of algorithmic and manual trading competitions on the basis of the Environment.

6.3 Evaluation with Trading Competitions

The stability and maturity of the experimented software has been thoroughly evaluated by the UCL Trading Competitions held annually for students, researchers and academics to test their trading algorithms.

Since 2009, each year the competition allowed participants to research and design their algorithmic solutions, get experience with trading by implementing their algorithms, and finally compete for prizes. It also allowed the author to test and evaluate the capabilities of the software.

6.3.1 Algorithmic Trading Competition 2010

The UCL Algorithmic Trading Competition 2010 was organised over a period of three months, between the 15th October 2010 and 15th December 2010. The competition was divided into three stages. During the first stage, the participating groups were registering themselves and researching different algorithmic trading strategies. During the second stage, the participating groups were given access to the experimental system and implemented the algorithmic trading strategies researched during the first stage. The third stage of the competition comprised the actual trading period, on the basis of which the winners were selected.

The organisation and a day-to-day running of the competition was supported in a variety of ways. A group of students was organised to monitor and maintain a smooth execution of algorithms on the experimental system. The participants were granted access to the system's website and a dedicated internet forum on which another group of students was providing support to the participants. All the participants received e-mail announcements about the progress of the competition.

6.3.1.1 The Participants

During the first stage of the competition the support team registered a total of 37 groups that submitted their interest in the competition. During the second stage of the competition the support team observed that slightly less than half of the registered groups were still actively developing and testing their algorithmic trading solutions. During the second stage of the competition the support team was also forced to warn three participating groups that their actions were against the Terms & Conditions of the competition. Over the course of the third and last stage of the competition, the support team have monitored performance of the remaining 12 groups that were actively trading.

The participants presented a variety of different trading styles, from High Frequency Trading, the Intraday, to Long Term (Strategic) approaches where only a few trades were placed during the course of the entire competition. The results of the competition are summarised and presented in the following sections. The analysis focused on the most interesting and significant cases amongst the participating groups as those are believed to reflect the working of the experimental system.

6.3.1.2 Trading Statistics

The trading statistics were derived from the data which the experimental system automatically gathered during the course of the competition. Some of the trading statistics presented were calculated on-the-fly by the Environment (e.g., the total balances that each one of the groups have traded). Other, more

refined, statistics were prepared by the support team.

The first Figure 6.9 presents the total profit that every participating group (groups are represented by their ID numbers) achieved after subtracting the initial (virtual) 10,000 provided for every group to invest. The majority of the groups lost a lot of their initial funds; some achieved significant losses. Some losses were expected due to the lack of experience of the participants. Slightly alarmingly, however, some of the high frequency traders generated major quantities of loss trades despite being able to derive statistics and correct the underlying issues.

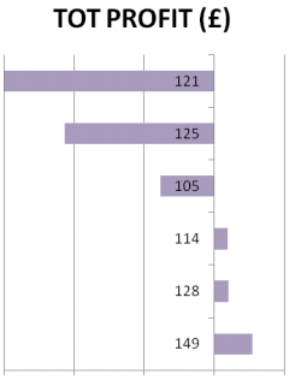


Figure 6.9: Total Profit.

Major losses were generated by groups with the following identification numbers: 121, 125, 105, 107 and 144 (records of groups 121, 125 and 105 are presented in the chart).

The group identified as 121 disastrously developed large positions without adequate consideration of the risk involved with the positions exposure. The analysis of the groups trades has shown that the group was using an intraday approach and on some occasions did not close their positions which consequently generated major losses. The closed positions on the other hand, were relatively successful.

The group identified by the number 125 started with an intraday strategy that was generating a small profit but seemed to have a relatively large risk-to-reward ratio as the losses were large in comparison to gains. The group appeared to spot the issue and seemed to have recalibrated the strategy to a more long-term (strategic) investing. This, however, has proven to be disastrous as the last few trades carried over a few days generated the majority of the losses.

The group identified by the number 105 was using a higher frequency intraday strategy generating from a few to a few hundred trades a day. The strategy however was generating a steady stream of loss trades with a few significant losing trades. The group did not manage to optimise performance of their algorithm.

The group identified by number 107 was also using a higher frequency intraday strategy generating around 50 trades every day. The strategy was focused on EUR/USD currency pair and was using flexible money management system. The final outcome, however, generally negative, indicated that the group, given enough time, would quite likely design a strategy that could potentially generate positive results. It seems that the lack of appropriate risk management contributed to the final outcome.

The group identified by number 144 designed a particularly uncoordinated strategy trading intraday with a more or less random currency pairs, and sizes. It is possible that the group did not manage to come up with any successful strategy during the development stage and decided to participate in the third

stage hoping that they would manage to finalise the development before the end of the competition. The attempt was unsuccessful.

Groups 114, 128 and 149 generated positive profits.

In case of group 114 the strategy generated a few dozen positive and negative trades, with one trade that was not closed. The trade was then carried over a few days and seems to have been closed manually.

A very similar behaviour was presented by group 128.

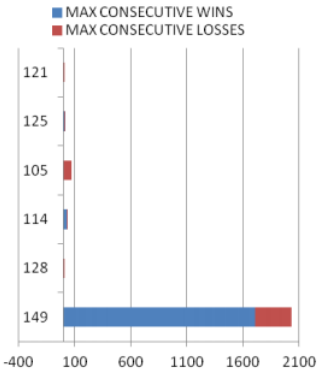


Figure 6.10: Maximal amount of consecutive win and loss trades.

A completely different strategy was implemented by group 149. The strategy created by the group has exceeded all the other groups in terms of the profit generated as well as the amount of issued trades. The group implemented a very high frequency trading solution that used all the available assets and had to be extremely quick in both the pre-trade analysis and signal generation processes. A trace of the trades indicated utilisation of a type of an arbitrage or statistical arbitrage strategy.

Another statistic used by the support team to evaluate the competition was a measure of the maximal amount of consecutive wins (profitable trades) and losses (loss trades). It represents the consistency with which the groups were trading (presented in the Figure 6.10). The measure shows clearly that the group identified by the number 149 was particularly consistent with their profitable trades. Groups 114, 111 and 107 have also exhibited some consistency in their profitable trades; however, the amount of consistent loss trades was also significant in those instances.

The total number of trades that every group performed over the entire period of the competition is presented in Figure 6.11. The Figure allowed the support team to differentiate between the various trading frequencies in algorithms.

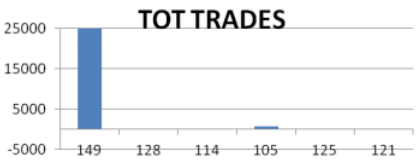


Figure 6.11: Total number of trades per group.

The Sharpe ratio was used to characterise how well the return of an asset compensates the investor for the risk taken. When comparing two assets, each with the expected return against the same benchmark, the asset with the higher Sharpe ratio gives more return for the same risk. The ratio was calculated with respect to the Bank of England daily interest rates (BOEBR) as the main comparison benchmark. The results of the risk analysis can be found in Figure 6.12. N.B. for those instances where it was appropriate and feasible.

RANKING	USER ID	TOT PROFIT (£)	TOT TRADES	MAX CONSECUTIVE WINS	MAX CONSECUTIVE LOSSES	TOT WINS	TOT LOSSES	MAX DRAWDOWN	SHARPE RATIO
1	149	5465.46	25323	1705	330	20839	4484	-0.01	0.57
2	128	2094.84	7	5	1	5	2	-1.36	N/A
3	114	1950.25	51	30	5	37	14	-0.04	-0.76
35	105	-7703.41	763	7	69	273	490	-1.58	N/A
36	125	-21306.56	15	10	5	10	5	-0.03	N/A
37	121	-31562.82	42	8	7	23	19	-0.01	N/A

Figure 6.12: Risk analysis results.

6.3.1.3 Competition Summary

After evaluating all the competitors, the group identified by the number 149 won the first prize in the Algorithmic Trading Competition 2010 with a total profit of 5465.46. In line with the competition rules, the teams received a prize of 7,500 (5,000 + 2,500). The second prize was won, again by the group identified by a number 149. In line with the competition rules, the team received 2,500.

The experimental system successfully handled both the large amount of competitors and their high frequency trading approaches.

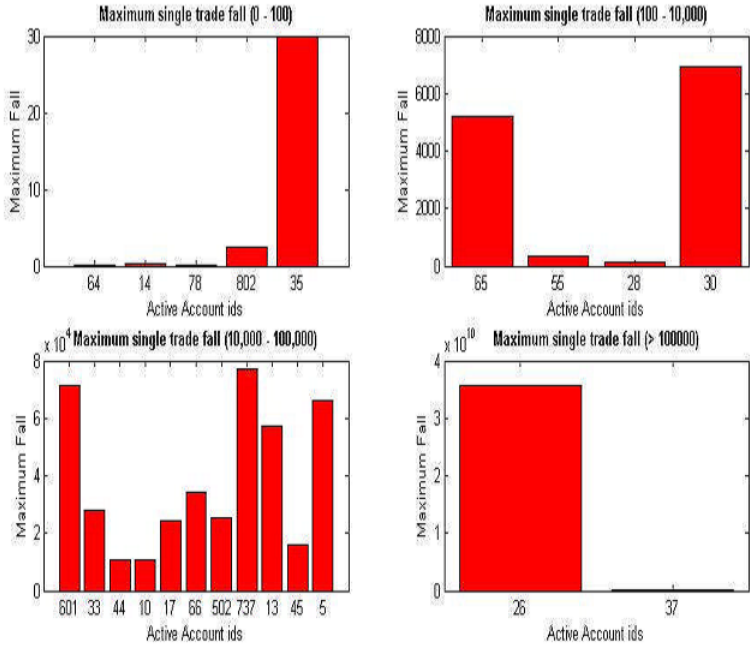
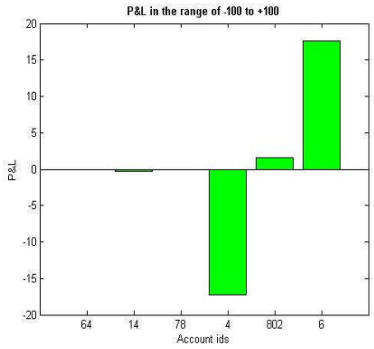


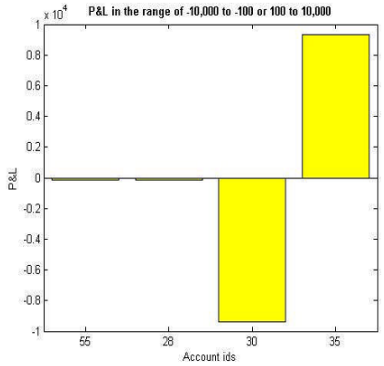
Figure 6.13: Maximum single trade fall.

6.3.2 Algorithmic Trading Competition 2011

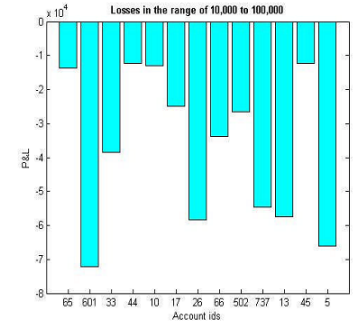
The UCL's Algorithmic Trading Competition 2011 took place over a period of three months. In September 2011 the participants registered for the competition on the dedicated website. Following registration they developed their trading algorithms in October 2011. Finally, the competition itself took place during the month of November 2011. For the development and testing of their trading strategies the participants were granted access to the experimental Environment. In the last stage, which lasted for a period of 30 days, participants conducted the actual trading using their strategies. Throughout the duration of the competition a group of UCL students provided support to the participants mainly with use of a dedicated website and forum where teams could ask questions about trading and utilisation of the Environment.



(a) P&L in the range of -100 to 100.



(b) P&L in the range of -10,000 to -100 or 100 to 10,000.



(c) Losses in the range of 10,000 and 100,000.

Figure 6.14: Profit & Loss

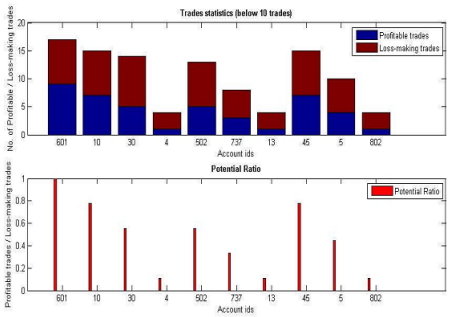
During the course of the competition a mixture of trading approaches and software systems was presented. Performance of the strategies in the trading period was quantified using various metrics and was taken as the basis of their evaluation for prizes. Additionally, the design and construction of the trading algorithms was taken into account.

During the course of the competition one strategy has proven to be an undisputed winner that generated substantial profits and maintained them in a consistent manner throughout the competition. The strategy consisted of an extremely effective risk control mechanism that prevented it from making an overall loss on all but one trading day. In addition some strategies demonstrated use of financial and social sources of data to generate trading signals, where one used Twitter feeds to gauge the overall traders mood. Strategies also demonstrated an advanced application of data analysis using machine learning methodologies.

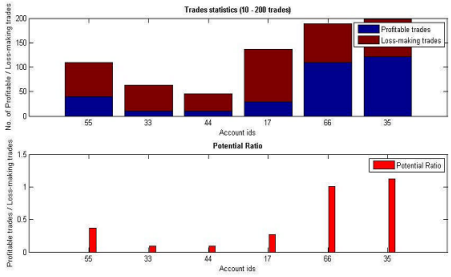
6.3.2.1 The Participants

The 97 registered teams demonstrated a number of different trading approaches ranging from high to low frequency, with strategies involving technical analysis, stat-arb techniques and pairs trading. A noticeable feature was that the participants' high-frequency strategies were particularly penalized in

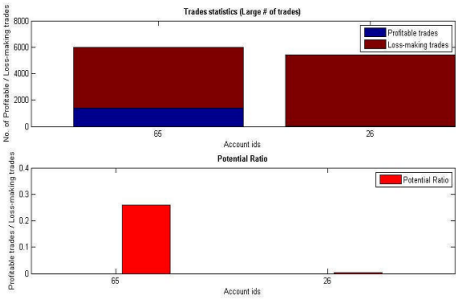
terms of the number of loss-making trades and consecutive losses. For example, participant ID 65, ended up piling massive losses early in the game after making a large number of trades, and then resulted in landing up a single trade loss of 5220. In terms of the largest single trade loss, some other participants did much worse as the Figure 6.13 shows.



(a) Small number of trades and their potential ratio.



(b) Medium number of trades and their potential ratio.



(c) Large number of trades and their potential ratio.

Figure 6.15: Statistics for trades and their potential ratio.

Interestingly, some medium frequency strategies were able to limit their downside to the minimum. Participant ID 35 who emerged with the highest profit at the end of the competition did not indulge in a high frequency strategy; the strength of the participant's performance was the algorithm deployed, resulting in a cumulative profit of 9343 on the final trading day.

6.3.2.2 Profit & Loss

Figure 6.14a indicates the P&L of the most active participants who are identified by their competition IDs. A common feature among most of the participants is that their trades had a much greater downside than upside; the amount of the largest cumulative loss incurred by one participant exceeded by far the largest cumulative profit made by others. This could mean either a failure of risk measures in place and/or an excessive appetite for risk-taking. The P&L reflects the profits or losses incurred over and above initial balance.

Profits for the most active accounts have been depicted below by dividing them into three categories depending on their total cumulative P&L as of 30/11/2011: a) P&L in the range of - 100 to 100, b) P&L in the range of -10,000 to -100 or 100 to 10,000, c) Losses in the range of 10,000 and 100,000.

Participant ID 6 had a positive profit on the last trading day however was not active through the competition and did not put on many trades. In terms of P&L, participant ID 35 is a winner by far; none of the other participants come close to this participant in terms of P&L. On the last trading day the cumulative profits of this participant were 9343.

A bulk of active participants, trading medium to low frequency strategies, ended up with large loss accumulations by the end of the trading period. One participant (ID 37) had a particularly large loss to the tune of 105,000,000.

6.3.2.3 Trends & Potential Ratio

Figure 6.15a depicts the total number of profitable trades and loss making trades the participants had accumulated by the last trading day. Based on this a 'potential ratio' was calculated which is the ratio of the total number of profitable trades versus the total number of loss-making trades a participant had managed to accumulate.

A greater than 1 potential ratio would indicate a 'potential' in the sense that the trader had accumulated more profitable trades than loss making ones. A potential ratio smaller than 1, on the other hand, would suggest that the trader had accumulated more loss-making trades than profitable ones. The level of disparity between profitable trades and loss-making trades witnessed would be indicated by how much this ratio deviates from 1.

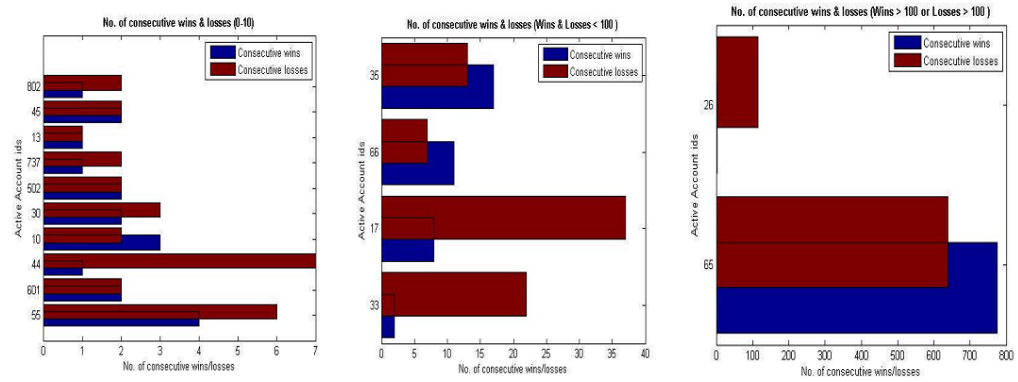
Figure 6.15c plots the total number of profitable trades and loss-making trades of the most active participants alongside their potential ratio. Most participants who put on very few trades had a mean ratio of 0.5. Participant ID 601, 10 and 45 had a potential ratio greater than 0.6. Participant ID 35 has a potential ratio marginally above 1, along with ID 66. The other participants who traded in this frequency had a ratio of less than 0.5. Participant ID 65 and 26 put on the maximum number of trades through the competition and participant ID 65 secured 1406 winning trades by the close of the competition. This was the largest number of winning trades obtained by a participant in the competition.

6.3.2.4 Max Consecutive Wins & Losses

The metric computes the length of consecutive winning trades and losing trades and picks out the instance of longest consecutive wins and consecutive losses witnessed by the user. This metric indicates the quality of algorithmic efficiency. Long losing streaks indicate poor monitoring on the side of the participant and lack of risk control measures in place.

Most participants have accumulated a large number of consecutive losing trades. One participant however, ID 65, created a strategy that early on in the game managed to secure a 776 consecutive-win streak. The participant nevertheless ended up having a long streak of losing trades as well (the longest in fact) and ended up in a cumulative loss. The participant ID 35, on the other hand, who secured the largest P&L by the end, only had a longest winning streak of 17 trades.

The Figures compare the length of winning and losing streaks for the most active accounts. Participant IDs 65, 66, 35 and 10 managed to secure longer consecutive win streaks than loss streaks however only participant ID 35 managed to wind up with a cumulative profit on the last trading day.



(a) Number of small consecutive win and loss transactions.

(b) Number of medium consecutive win and loss transactions.

(c) Number of large consecutive win and loss transactions.

Figure 6.16: Consecutive win and loss transactions.

6.3.2.5 Volatility

For each of the active participants in the competition, the standard deviation of their daily returns was used as a proxy for volatility. It is mainly computed to check for algorithmic stability and gauge the randomness component of returns. Participant ID 35 demonstrated a stable trajectory of positive returns as the Figure 6.17 shows. This indicates that the profits accumulated by the participant are an outcome of consistent profitable trades rather than a few lucky trades. If it was the latter, this would be captured by a high volatility metric and would necessitate penalizing the participant, even if he/she had high profits. It is common to see very high volatilities (indicating erratic returns) going hand in hand with very large losses.

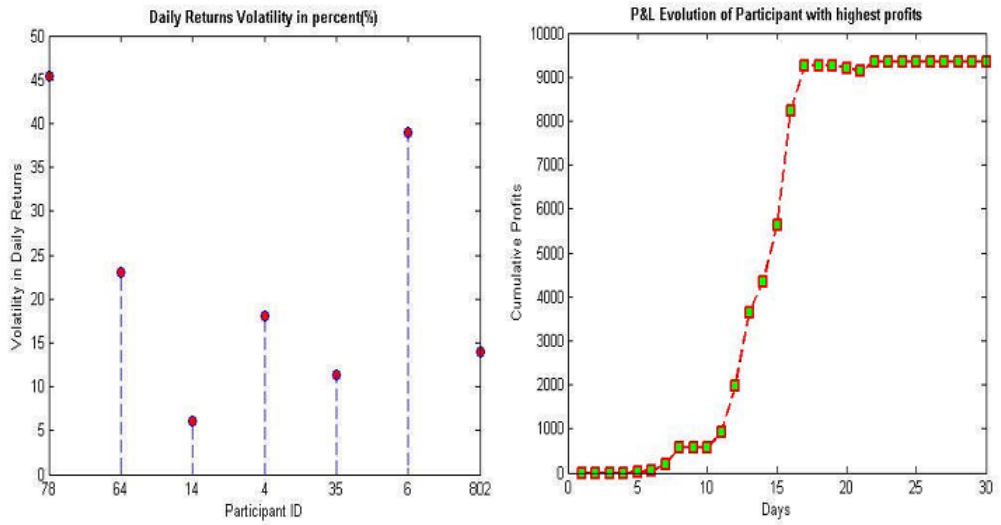


Figure 6.17: Daily Return Volatility and example P&L Evolution.

Participant ID 35 has managed to secure strong positive profits throughout the competition and his/her returns have exhibited very low volatility overall, likely due to the fact that his/her profits remained static

during the last week of the competition, as can be seen in the graph.

6.3.2.6 Sharpe Ratio

The Sharpe Ratio characterizes how well the return on the portfolio compensates the undertaken risk. The risk of the portfolio is measured using the standard deviation of the price returns of its various assets (the daily returns are used for the calculation). In short, the Sharpe Ratio is an attempt to summarize the return on the risk component of algorithms. Unsurprisingly, the participant with the highest profit and most stable returns realized the highest Sharpe Ratio of 1.16. The winner of last years competition had a Sharpe of 0.57. Participants ID 6 and 802 have small, yet positive Sharpe ratios.

6.3.2.7 Competition Summary

The 2011 Algorithmic Trading Competition showed improvement over the 2010 competition in terms of the total number of participants, the quality of competing teams and their activity over the trading period. The obtained results reflect the fact that the competition did indeed elicit some good talent from interested students and gave them the opportunity to engage with sponsors like Microsoft, Barclays Capital, Citi and Knight Capital, ABN-Amro Clearing bank and Eurex.

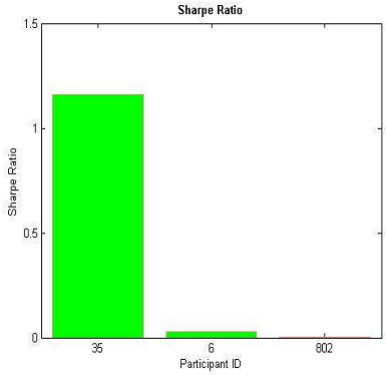


Figure 6.18: Sharpe ratio.

The experimental Environment which served as the backbone of the competition demonstrated its reliability in hosting competitions of such scale and magnitude. Each year, the competition reveals potential areas of improvement and fine-tuning of the existing architecture which couldn't be possible without such heavy testing and usage during the competition. Apart from some minor technical issues, the Environment was able to support trading at a wide range of frequencies.

Ranking by P&L

Rank	AccountID	Initial Balance £	Profit £	Maximum fall £	Sharpe Ratio	Volatility	No. of Profitable Trades	No. of Loss-making Trades
1	35	10000	9343.38794	29.81434641	1.16	11.3%	121	77
2	6	10000	17.64	0	0.03	39%	1	0
3	802	10000	1.52659354	2.385	0.0007	14%	1	3
			Minimum Loss participants					
4	78	10000	-0.0012005	7.21E-04	NA	45.4%	0	2
5	64	10000	-0.0451291	0.01815504	NA	23%	0	3
6	14	10000	-0.2764537	0.276453727	NA	6%	0	1
7	4	10000	-17.271789	0	NA	18%	1	3

Figure 6.19: Ranking of the 2011 competition ordered by P&L.

6.3.3 Manual Trading Competition 2012

The 2012 Eurex Manual Trading Competition was organized by the UK PhD Centre in Financial Computing and sponsored by Eurex, Europes leading derivatives exchange. The game involved manual trading across a set of pre-specified securities and benchmark indices traded on Eurex Exchange. The trading process was facilitated through ATRADEs graphical interface where participants were able to access Eurex data feeds on prices and make, amend and cancel orders. Additionally, the ATRADE interface supports a set of functionalities (e.g., the historical order viewing and the account-status information) to facilitate an enhanced trading experience for the participants of the game.

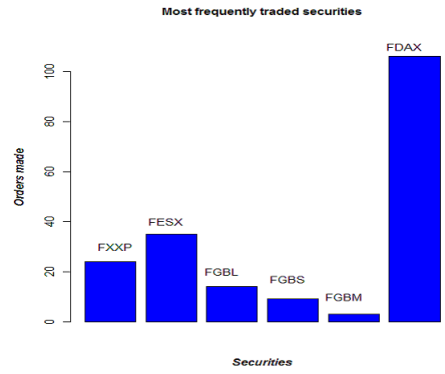
Participants registered their interest for the competition from 1st Feb 2012 to 7th March 2012. The first phase of the competition from 7th March to 4th April was called the beta-trading stage, participants were allowed access to the Environment and were able to issue orders and monitor their P&L from day to day. They were not assessed for their trading during this period.

On the 5th of April the actual trading stage commenced where participant accounts were reset to start with their original virtual cash balance of EUR 20M.

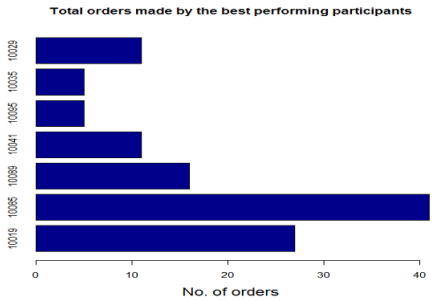
6.3.3.1 The Participants

Participants came from different Higher Education Institutions across the UK including UCL, Warwick, University of Reading, Cass Business School and Imperial College.

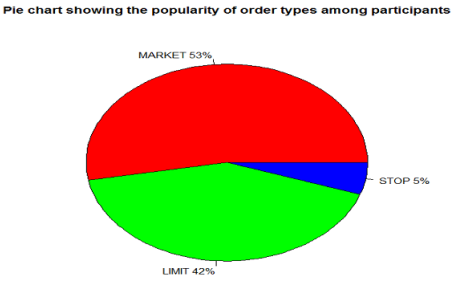
There were 49 registered manual traders at the beginning of the trading stage; 21 out of the 49 participants were the most active and over the 21 trading sessions between the 1st of April and 4th of May made over 200 orders through the Environment.



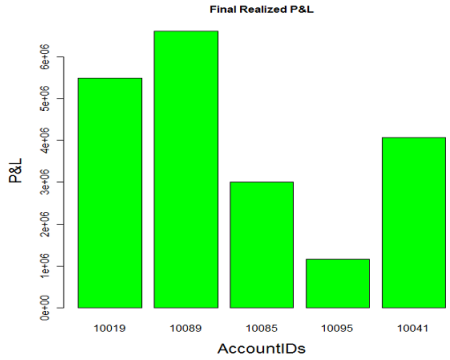
(a) Most frequently traded securities.



(b) Total amount of orders made by the best performing participants.



(c) Utilisation of order types.



(d) P&L of the top traders.

Figure 6.20: General Statistics of the competition.

At the end of the trading period, there were 5 participants

with significant net profits. Their details are summarized in the table below.

6.3.3.2 Profit & Loss

It was observed that participants who held diversified portfolios of bond futures and index futures were able to hedge themselves better. The fixed position worth EUR 100M in Stoxx Europe 600 Price Index was only worth EUR 95,456,921 on the last trading day contributing a EUR 5M loss to all participants portfolios. Participants who managed to realize profits of greater than EUR 5M ended up with a positive net profit.

Figure 6.20d summarizes the profits made by the top rankings traders on the trades made during the trading period. On the last trading day, all unrealized profits or losses were automatically converted to realized profits or losses by the experimental Environment.

6.3.3.3 Risk Management

A lot of participants assumed short FXXP (Stoxx Europe 600 Index Futures) to be a natural hedge against the long SXXP (Stoxx Europe 600 Index) position. However, participants who spread their funds across the whole range of available futures performed better over the trading period. Figure 6.21 indicates the list of futures held by the top ranking participants in their portfolios.

Apart from diversification, a continuous rebalancing of positions is a trademark of a good risk management. The top ranking participants are also the traders who balanced their positions more frequently than the rest.

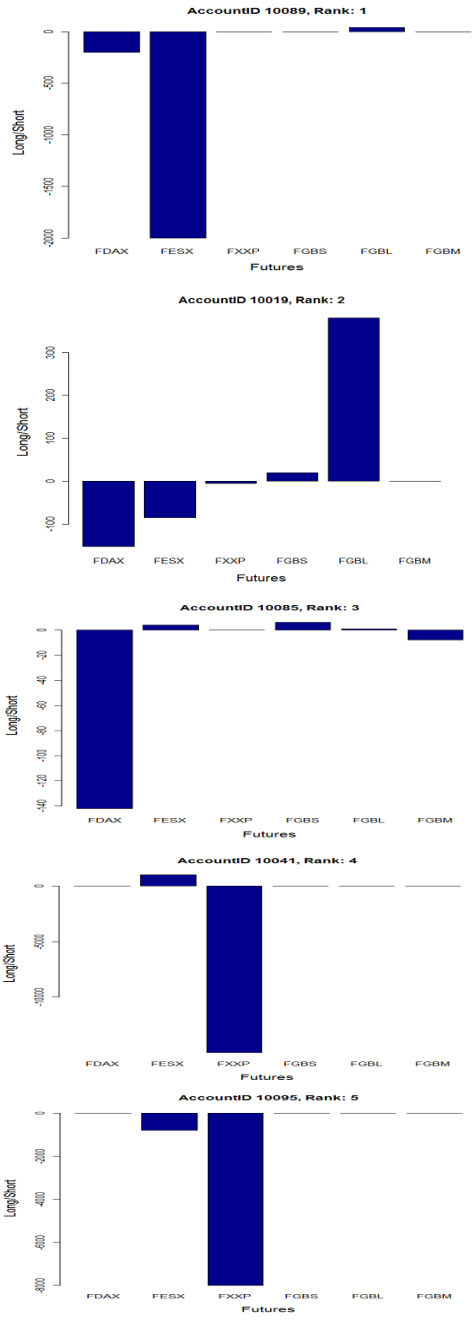


Figure 6.21: List of futures held by the top ranking participants.

6.3.3.4 Stability of Returns

Figure 6.22 depicts the trajectory of two stable performers who secured their profits frequently rather than exposing large open positions (and unrealized P&L) until the last day. They performed well with their overall profits towards the end of the competition

and achieved statistically-significant, stable returns from the beginning to the end.

This is particularly important, as ensuring stability of returns in a period of less stable prices is a challenge. Risk and returns are a trade-off, risk minimization through hedging eats away into profits that could be realized by maintaining some exposure, however, return maximization achieved without prudent hedging against market movements is short-lived and sooner than later may introduce losses. One of the best ways to ensure that returns are stable is to maintain a diversified portfolio starting with equal fund allocation to all the tradable futures and rebalance this allocation over time.

6.3.3.5 Utilization of Leverage

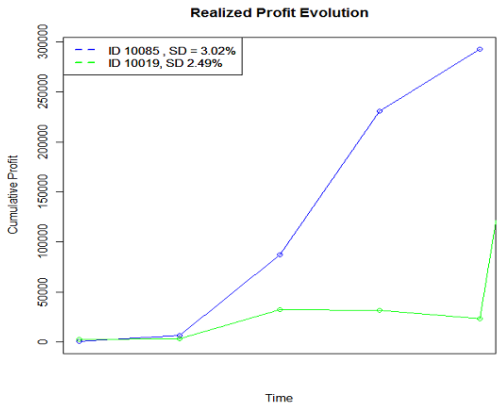


Figure 6.22: Evolution of P&L results of top traders.

It was observed that participants were not aggressive with utilizing their available margin, implying aversion to risk. The total value of positions held by most of the participants on the last day was less than 40% of the total margin allowance. Figure 6.23 summarizes the remaining margin for various participants on the last trading day of the competition.

6.3.3.6 Competition Summary

The experimental Environment utilised to support the competition demonstrated its maturity in hosting competitions of this scale. Virtual trading throughout the competition revealed areas of improvement and fine-tuning of the existing architecture which wouldn't be possible without such testing.

Rank	Account ID	Margin Remaining	% of Total Margin
1	10089	63,905,526	45.65 %
2	10019	65,330,659	46.66 %
3	10085	104,473,129	74.62 %
4	10041	3,284,971	2.35 %
5	10095	12,824,600	9.16 %
6	10029	41,818,266	29.87 %
7	10035	76,959,354	54.97 %
8	10039	115,468,076	82.48 %
9	10043	23,123,097	16.52 %
10	10053	114,792,235	81.99 %
11	10014	32,190,957	22.99 %

Figure 6.23: Remaining margin.

The results of the competition established that trading derivatives in an environment of falling prices is complex for inexperience traders. Future contracts are intricate instruments mainly used for neutralizing risk exposure and they are rarely traded as individual instruments for speculation or profit.

One of the important lessons derived from the results is that good hedging practice is imperative for

profit accumulation in the futures markets. Diversification and frequent rebalancing of portfolios are other key risk-reduction techniques that have proven to pay off. Introduction of risk policies as part of the experimental Environment's OME module may prove beneficial for management of risk.

6.4 Evaluation Summary

The goals of the three described experiments were set to experiment with the software architecture of the Computational Simulation Environments capable of supporting Algorithmic Trading. From the engineering point of view the Environments aimed to provide real and the virtual trading capabilities via a set of dedicated APIs. The Environments also aimed to provide users with rapid prototyping capabilities, that would allow deployment of their strategies in a minimum amount of time. Finally, the Environments aimed to provide aggregation and processing functionality to enable pre-trade analysis in a form of portfolio selection. The relationships between all three conducted experiments can be summarized by Figure 6.24.

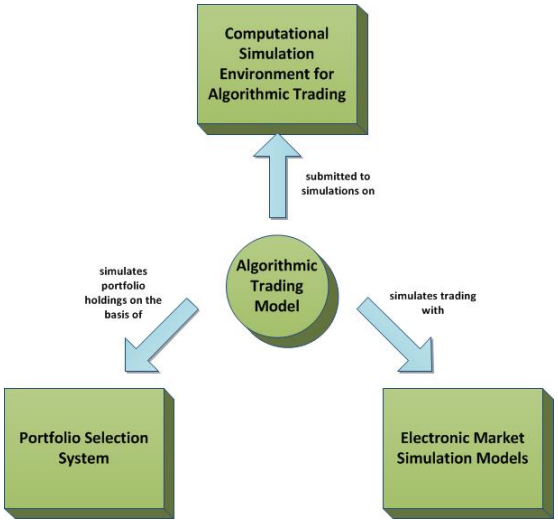


Figure 6.24: Experiments Relationship.

6.4.1 Comparison with other Environments

The Cisco's Trading Floor Architecture is particularly suitable for comparison with the proposed Computational Simulation Environments (CSE). Cisco distinguished three significant layers in the architecture of a trading floor that one can compare to the modules and elements of the CSE. The layers were labelled as: an external communication layer, internal communication layer and a business logic layer. The design of the CSE architecture resembles the Cisco's architecture in this way. The external communication layer resembles the Connectivity Engine that allows communication with external brokers and data providers. The internal communication layer resembles the JMS Internal Communication Bus used by the CSE to enable communication between its modules. Finally, the logic layer resembles all the modules of CSE.

Cisco's business logic layer distinguishes the Trading Cluster & Ticker Plant and the User Applications. The first two contain the Feed Handlers, Price Engine, FIX Engine, Algorithmic Trading, Risk Modelling and Order Management, while the User Applications consist of Execution Monitors, Alerting Systems, Trading Systems and Visualization Systems.

The Trading Cluster & Ticker Plant modules resemble the CSE's Connectors and Processing & Aggregation modules. The FIX Engine enables communication with the brokers, exchanges and data providers, precisely the way the Connectors are able to maintain connections to the three. The Feed Handlers allow aggregation of incoming feeds as well as processing the feeds and transforming them to more representative data structures, precisely the way the CSE's Processing & Aggregation does. The Order Management module provides management of execution of the issued orders, precisely the way that CSE's Order Management Engine does.

The User Applications logic resembles the CSE's Developer Client and all the UI clients in general. The Visualization Systems and Alerting Systems provide a means to graphically communicate to the user what the status of their strategies is, which is precisely what all the elements of the UI Clients do. The Trading Systems module that allows the algorithmic and manual trading resembles the CSE's Developer Clients functionalities.

6.4.2 Discussion on applicability of used technologies

Majority of the used technologies has shown to be suitable for the tasks they were chosen for. The Apache ActiveMQ, although highly scalable with respect to the amount of involved producers and consumers, has its limitations as to the volumes of events that the Environment needs to process every day. Consequently, when large volumes of data needed to be transferred over a network, the JMS broker was complemented with a high throughput, point-to-point Kafka broker.

Utilization of the Hibernate technology in a high frequency project was inefficient, due to the fact that an additional layer between the database and the logic layer, introduced extra delay to the processing of information. At the end the cluster-based Hadoop technologies were most suitable for the task.

The QuickFIX/J technology has shown to be flexible and mature, and fulfilled all expectations. The QuickFIX/J gives the CSE an advantage of speed both in terms of messaging and in terms of development. The technology is considered one of the fastest FIX engines on the market, moreover, the development of a single connector never took more than two weeks, from the beginning to the testing phase. The only problem experienced with the technology was its applicability when using more than one session. QuickFIX/J, when implemented in the initiator mode, is mainly used for single-session connections, while in cases of the Environments each user requires at least one session (and, if using more than one broker, even more than that).

6.4.2.1 Applicability of Esper rather than other Complex Event Processing engines

Complex Event Processing technologies are particularly useful in systems that need to process complex information in real-time. The selection of Esper, as a main CEP engine in the CSE project, was due to the fact that it is one of the most mature open-source project of its kind, and has a considerable group of supporters making development with Esper relatively simple. Similar Environments (e.g., Marketcetera) also utilize Esper. There are some Esper plug-ins that are already available, that provide parts of functionalities required by the CSE project for free. Esper has shown to work extremely well with other technologies utilized in the Environment (e.g., Apache ActiveMQ, Apache Esper, Hibernate and Spring Framework) which makes it even more applicable to the CSE.

6.4.2.2 Applicability of Linux Init service

Linux Init service is a well-established standard for service management in the Linux environment. The main goal of the Init manager in Linux is to spawn all other processes, including the background daemons. This feature is particularly applicable to the CSE problem because the main modules of the Environment need to be run in a persistent manner and provide its functionalities on request and through events. An alternative service management environment is the Apache ServiceMix environment, which is considered an Enterprise Service Bus (EBS) capable of combining functionalities of Service Oriented Architecture (SOA) and Event-Driven Architecture (EDA). The goal of the EBS is to provide an integrated solution for components and services, in a vendor-independent way. This significantly extends the functionality of the Linux Init service.

6.4.2.3 Applicability of Event-Driven Architecture

The CSE has utilized the Event-Driven Architecture for data streaming mainly for its speed. The event-oriented approach is faster than the standard service-oriented approach due to the fact that in this paradigm the data is accessible as soon as it arrives to the interested parties rather than as soon as it is requested. Another reason to use the Event-Driven Architecture is to simplify architecture of the elements that utilize it. It is easy and efficient to design a set of interfaces and handlers for the communication layer, and then to incorporate it in the logic layer. Also, building complex analytical tools can be made easier due to utilization of Complex Event Processing engines (these however require events to work on, hence it is beneficial to accommodate the EDA in the system).

6.4.2.4 GUI designs in Financial Computing

Majority of the GUIs that are parts of Financial Computing applications require significant amounts of processing and computation to deliver the visualization layer to the user. This implies that such GUIs should not only be attractive in the graphical terms, but also highly programmable. Moreover, due to the necessity of collocation, it is more efficient to provide web access to the GUIs than to make it a part of the underlying processes (there is always a possibility of virtualization but, from the authors experience,

it often makes the GUIs non-responsive in the crucial moments). Due to the above, the majority of technologies currently available on the market are either unsuitable, or have no required capabilities. Therefore, the author decided to use Java Swing, one of the older but well-established technologies that provide all the necessary capabilities.

6.4.2.5 Applicability of the strategy templates

The appropriate definition of interfaces simplifies further implementation. Because of this, the author decided to provide a set of predefined templates with partial implementations of strategies. Such templates present the user with a set of methods that simplify the development, maintenance and running of the strategies. The goal was also to further reduce the time of deployment of users' models as it is unnecessary to implement additional tools to handle strategies. The author also decided that (since there's a large selection of strategies, and most of them share common characteristics) the API needs to provide a higher level of abstraction to handle the different constraints that users may have. Based on the author's own experience with trading strategies and models, it was decided that the most general template types are the high frequency (Event-Driven) template and the low-frequency portfolio management (Service-Oriented) template.

6.4.2.6 Applicability of order and price buffers

The order and position buffering is an important feature of the CSE, as it allows local access to the information about the state of transactions committed by the users. Moreover, based on these states, the Environment is able to automatically evaluate the profit & loss of every user. Therefore, the location of the buffers needs to provide a low latency access to the information, whenever requested by users or their strategies. It should also allow a quick evaluation of the state of users' accounts by the system. Locating the buffering & synchronization functionality as part of every 'Clients' module provides the quickest possible response to users' requests. A disadvantage of this design is the fact that the further the physical location of the buffering from the server-side services, the higher the chance of discrepancies in the data stored in the buffers.

6.4.2.7 Data processing & aggregation techniques

A variety of different approaches is possible with regards to the market data, mainly with respect to the way it is processed and propagated through the system, and with respect to the way it is aggregated in the buffers and databases. The two most common approaches are to firstly aggregate and then process the market data, or to process and then aggregate. An advantage of the first one is the speed (as everything may be calculated on the run), the advantage of the second approach is the full availability of the market data (which makes the processing reliable). From the software engineering perspective, the processing can be implemented on the basis of the JMS and dedicated threads that will deliver the defined functionalities, or on the basis of Complex Event Processing engine. The former may be a slightly slower

solution but it is far more versatile thanks to its embedded querying language that allows creation of sophisticated analytical structures on the basis of the query.

Another issue that one may encounter during development of a similar system is the location of the processing & aggregation thread. If the implemented system is relatively small, there's no need to implement many services and it is advised to combine processing and aggregation with the connector service. In the instance that the implemented system is large (as in the case of the CSE) it is better to keep the modules separately and run them as separate services. This is due to the amount of information that may need to be processed and its effect on the stability and performance of processes handling it.

6.4.2.8 Design of Profit & Loss for every user

The calculation of Profit & Loss (P&L) is a complex process due to the fact that the values of financial instruments are obtained with relation to the other financial instruments. Therefore, there is always a need for an anchor which, in the instance of the CSE, is the base currency of the user's account. All the transactions are always evaluated on the basis of the base currency, which requires significant amount of on-the-fly queries to the prices of financial instruments that lead from the traded instrument to the base currency. This, on occasions, creates a chain of instrument-prices which, at provided timestamps, needs to be present in the system to succeed with the calculation of P&L for the trade. An additional degree of complexity is added due to the fact that an order can receive partial fills on different price levels, and that P&L of a position is build-out of at least two orders (the open order and the close order; in the simplest form of a position). To succeed with the above functionality, the author decided to use an in-memory-database, to hold all the accounts for every user. Each entry for a particular user, describes the current, most up-to-date state of the particular account, including positions that are closed and positions that are still exposed to the market. Since each execution report needs to be recalculated back to the base currency of the account, every time we receive an execution report, it needs to be accompanied by all the necessary queries to the relevant currencies. In order to obtain such data, an additional in-memory-database is designed to hold recent prices of all the instruments that the Environment is subscribed to. This allows a quick access to the recent prices. In the instance that a price is not found, an extra query is designed to look for the price in the aggregation database.

The goal of the above discussion was to provide justification and arguments that support the decisions behind the design and implementation of all the experimental modules and functionalities of the Environment. All the above provides a background to the conclusions of the described experiments.

Chapter 7

CONCLUSIONS AND FUTURE WORK

The chapter discusses the overall results obtained from the three experiments. The results are examined with respect to the initial objectives set in the Introduction chapter and are related to the literature and publications review in Chapter Two. A discussion on applicability of the software engineering methodology, algorithms and technologies in a variety of other problems are presented. Relevant conclusions are drawn, accompanied by a list of contributions to the field of Computational Finance and Algorithmic Trading. Finally, a discussion, on what was learned during the course of the experiments is presented with reflections on possibilities of further research.

7.1 Conclusions

This thesis investigated experimental computational simulation environments for the study of trading algorithms and their risk. As an outcome of the study a set of software architectures for computational simulation and modelling was developed. The thesis investigated the field of algorithmic trading and proposed software architectures that can support different analytical techniques from statistics, machine learning and economics to support algorithms capable of taking, executing and administering investment decisions with optimal levels of profit and risk. The proposed environments are crucial for effective lifespan management of trading algorithms, evaluation of their stability, estimation of their optimal parameters and their expected risk and profit profiles. This research Environments are predominantly designed for testing, optimisation and monitoring of algorithms running in virtual or real trading mode.

Three key studies have been conducted as part of this thesis: a) an experiment investigating design of an algorithmic trading environment, b) an experiment investigating design of simulators of electronic trading exchanges, and c) an experiment investigating algorithms and design of an environment for NP-complex co-relation search amongst financial securities.

The core elements of the described experiments were implemented in collaboration with a group of

roughly 60 BSc, MSc and PhD students over the period of four years. The author of the thesis was responsible for development of architecture of the software elements, for implementation of significant parts of the functionalities, for organization & management of multiple groups of students and for support of the hardware, software and support-services infrastructure. The ATRADE and LIVE EM systems were integrated and successfully deployed over a network of servers in UCL, and utilised for organization of algorithmic and manual trading competitions on a UK, European and global scale. A list of collaborators participating in organization of the competitions includes Euronex (Deutsche Borse), Barclays Capital, Knight, City, LMAX, Microsoft. Various components of the developed systems were utilised in commercial projects with LMAX, Tower Trading and Microsoft. The architecture of the Environment was utilised to build a twin social-analytics platform, SocialSTORM; described in an article by Financial Times (http://www.ft.com/cms/s/2/0664cd92-6277-11e1-872e-00144feabdc0.html#axzz1o11UrLZx).

The following set of paragraphs will provide a summary of results of all the conducted experiments.

7.1.1 Computational Simulation Environment for AT (ATRADE Platform)

The ATRADE (Algorithmic Trading & Risk Analytics Development Environment) was a study that investigated the design, implementation and testing of an experimental algorithmic trading Environment able to support a variety of AT strategies. The outcome of the study consists of a set of distributed, multi-threaded, event-driven, real-time, Linux services communicating with each other via an asynchronous messaging system. The Environment allows multi-user real and virtual trading. It provides a proprietary framework to support development of algorithmic trading models and strategies. It allows an advanced trading-signal generation and analysis in near real-time, with use of statistical and technical analysis as well as data mining methods. It provides data aggregation functionalities to process and store market data feeds.

The ATRADE Environment allows users to conduct experiments virtually or to trade with real funds, with use of a Software Development Kit (SDK) that provides access to a set of programmatic and graphical interfaces for both algorithmic and manual trading. To provide such capability the author has designed sets of frameworks, templates and functionalities for AT strategy implementation, including the price/order/account/position retrieval, the pre/post-trade analytics, the trade execution and the risk/money management.

The following server-side services were experimented with and thoroughly tested for signs of instability: the Connectivity Engine, the Internal Communication Bus, the Order Management Engine, the Processing Engine, and the Aggregation Engine. The elements were responsible for the connection to the external brokers, internal communication between the elements, data processing and aggregation of the market feeds and real & virtual trading. ATRADE has shown to be fully capable of supporting research

in algorithmic trading.

All the three clients: the Developer Client (SDK), the User Client (GUI) and the Statistics Client, together with their sub-modules: the Strategy Templates and the Buffering & Synchronization functionalities were implemented and provided to the users. Further work needs to be carried out on the graphical designs of the clients and the ways to simplify human interaction with the clients. Further work needs to be also invested into the charting and reporting capabilities of the clients, for research visualisation.

A set of technologies was introduced and considered for experimentation as part of the Environment. These were: QuickFIX/J (http://www.quickfixj.org/), Apache ActiveMQ (http://activemq.apache.org/), Apache Camel (http://camel.apache.org/), Apache Derby (http://db.apache.org/derby/), Apache HTTP Server (http://httpd.apache.org/), Apache Tomcat (http://tomcat.apache.org/), Apache ServiceMix (http://servicemix.apache.org/), Apache Hadoop (http://hadoop.apache.org/), Esper (http://esper.codehaus.org/), Hibernate (http://www.hibernate.org/), Enterprise MySQL (http://www.mysql.com/products/enterprise/), Spring Framework (http://www.springsource.org/) and many other, less significant technologies. Majority of the technologies have shown to be suitable to support the Environment.

To support external and internal communication of the Environment the author has chosen the QuickFIX/J and ActiveMQ technologies. The experiment has shown that these technologies are able to support event-communication of the Environment with no interruptions, for long periods of time, and without any major problems. The two technologies can be monitored either with JMX (http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html), web-console or logging services. This greatly simplified statistical experimentation with event throughput.

The Apache HTTP and Apache Tomcat Servers were a good selection of containers providing web-access for the User Client (GUI) and Statistics Clients of the Environment. Together with WordPress (http://wordpress.org/), a content management system, the technologies provided users with access to the Environment's website with documentation, libraries, examples and a support forum. Furthermore the technologies are highly configurable, and their support for a variety of external modules allowed integration with e.g., JAAS (http://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html) functionality for authentication and authorization between different elements of the Environment.

The results have also shown that utilization of Enterprise MySQL, Apache Derby and Hibernate technologies was justified, however Hadoop-based technologies has shown a more optimal solution for aggregation functionalities. The freely available Enterprise MySQL database was utilized in the first iteration of the experiment as the main engine behind the aggregation module and provided all the expected capabilities. The last iteration of the experiment used Hadoop-based cluster storage engine that enabled

not only aggregation but also collocation of computation with data. As for the buffering functionalities, the Apache Derby and the HSQLDB (http://hsqldb.org/) in-memory databases were the most mature solutions at the time. The Apache Derby database has shown to be more stable and therefore was more suitable for the Environment. The Hibernate technology, was a good choice for the less demanding services that did not require stable performance characteristics. The more demanding functionalities relied on the JDBC (http://www.oracle.com/technetwork/java/overview-141217.html) interfaces that were able to provide query speed and a more stable performance.

The mixture of Apache ActiveMQ, Apache Camel and Esper technologies has shown to be a suitable, generic solution for the on-the-fly event processing. However, when applied to the high-frequency, high-throughput processing of financial data, a simpler, lower-level Apache ActiveMQ, Apache Camel and a proprietary Java functionality was more adequate for the problem.

The initial utilization of Linux Init service (http://www.tldp.org/LDP/sag/html/init-process.html) was replaced with a more sophisticated Apache ServiceMix technology that enabled one-command deployment of the Environment's services, simplified service management and monitoring over the network. Furthermore, usage of the Spring Framework greatly simplified wiring of various modules and components of the Environment. Spring together with Maven (http://maven.apache.org/) allowed profile-based configuration of components for testing, development and production environments. This, in turn, simplified experimentation.

As indicated in Chapter 6, all three evaluation methodologies (evaluation of Environment quality, evaluation of Environment performance as well as the evaluation with Trading Competitions) have shown that the designed architecture, the approach taken to perform the experiment and the literature review of the subject enabled this Experimental Computational Simulation Environment to meet expectations set in the Introduction Chapter and address questions set in Chapter 3.

7.1.2 Electronic Market Simulation Models

The study of simulators of electronic financial exchanges was conducted to investigate models of such exchanges. This was to enable experiments with algorithmic trading strategies. The study has proven to be particularly valid for evaluation of trading paradigms. One consequence of the experiment was development of LIVE and EOD EM to trade virtually without the risk of losing real funds. This is nowadays a common practice amongst investment institutions to use EMs to fine tune their algorithms before allowing the algorithms to trade with real funds. The developed EMs simulators provide users with a chance to participate in a market, receive live data feeds and monitor their profit without bearing any of the risks associated with real-time market trading.

In order to understand the way electronic trading exchanges work the experiment presented the standard

steps utilised by a typical electronic trading exchange, identified the typical participants of the exchange, the steps utilised by the participants, the way the participants trade and their needs while developing their algorithms. This helped to unify a more generic communication protocol developed to communicate events between simulators and participants and consequently provide a unique insight on how such systems work.

Research performed during the course of the experiment enabled identification of a list of key electronic trading exchange simulation types. Two of the identified simulation types were developed and implemented, namely the LIVE and EOD EM simulation types for the on-the-fly and historic electronic trading exchange simulation of price-to-order matching, for individual users or for multi-agent modelling.

To identify a list of technologies suitable for electronic trading exchange simulators, various asynchronous messaging systems, in-memory databases, relational databases, column-oriented databases and concurrent processing techniques were examined. This experiment was summarized in the unique design of the exchange simulators.

To enable development and implementation of two exchange simulators, a list of typical usage and alternative usage scenarios was developed, then transformed into object-oriented class and interface representations. And finally the developed architecture was implemented in the Java programming language as a set of distributed services. Test-driven approach as well as a set of services (for continuous integration, project control, source control, source quality control, team management and deployment automation) were utilised during the development process. Consequently the simulators were submitted to continuous testing during development.

The two developed models of electronic trading exchanges were successfully utilized in UCLs Algorithmic Trading Competition 2010, 2011, and UCL Eurex Trading Game 2012. Their architectures have proven to be robust and scalable enough to handle between a few dozens to a few hundred users, with users issuing up to 300000 orders each, over the course of competition. Further work will be continued to enable more flexible configuration of the models for the purpose of large-scale experimentation and influence of market exchange features on trading agents.

7.1.3 Portfolio Selection System (PSS)

The Portfolio Selection System was a study that investigated an experimental engine for discovery of exploitable relationships between financial time-series applicable for algorithmic trading. The key solution allowed identification of similarities/dissimilarities in behaviour of elements within variable-size portfolios of tradable and non-tradable securities. Recognition of sets of securities characterized by a very similar/dissimilar behaviour over time, turned to be beneficial from the perspective of risk management, recognition of statistical arbitrage and hedge opportunities, and is also useful for portfolio diversifica-

tion. Consequently, this large-scale search algorithm can be utilized in creation of better portfolio-based strategies, pairs-trading strategies, statistical arbitrage strategies, hedging and mean-reversion strategies.

The conducted research was summarized with the selection of Cointegration [Zietz, 2000] as the most suitable similarity/dissimilarity measure suitable for the problem. This was mainly due to the fact of stability of the measure as well as its capability of capturing non-linear relationships in time-series. Furthermore the measure was able to deliver the reflexivity property; particularly important in finding dissimilar relationships that are found to be exploitable in financial applications. Further research on the transitivity property of Cointegration is currently conducted to prove or disprove this assumption in the project.

The application of a guided-search algorithm to the PSS problem proved to be hard to solve mainly due to a lack of suitable factors that could be utilized as a utility function which maximization or minimization of could guide the search. A work is being done on a probabilistic representation of probable directions (edges in the graph), however, given the time-constraints it was decided to exclude it from the thesis.

A search-space is defined by key features of elements that are part of it and consequently depends on the similarity measure. In case of the PSS problem no explicit features of granulated financial time-series were necessary and graph-based search-space representation proven to be particularly flexible, especially from the point of view of extendibility when new nodes and edges were to be added or removed.

The author believes that the presented software architecture has shown to be highly suitable for the problem. The architecture evolved through the course of various experiments with different approaches to the portfolio selection problem and so far have shown to be working in practice. Some research still needs to be performed on proving or disproving the assumption of Cointegration transitivity, this however will not influence the architecture itself. This is thanks to modularization of the architecture, namely utilisation of interfaces between the JMS, the REST, the 24h buffer components and the discovery & management elements. The design simplifies the process of replacing one of the existing components with a more appropriate solution, if necessary.

The PSS problem required a fast streaming and database management technologies with significant but fully predictable (stable and typically high) information throughput. The considered database technologies available at the time were not fast enough (standard relational databases), not predictable enough (e.g., Hybernate), or too expensive (KDB+, http://kx.com/kdb-plus-tick.php) with respect to storage capabilities, and consequently proprietary solution was designed. Amongst the event-based technologies for data streaming the following two were considered; ActiveMQ and ZeroMQ (http://zeromq.org/). Due to the fact that ActiveMQ was already utilised by ATRADE, and its functionalities supported more than a point-to-point communication (as in case of ZeroMQ at the time) it was decided that ActiveMQ will be

the technology of choice supported by Apache Kafka (http://kafka.apache.org/index.html).

7.2 Contributions to Science

The proposed architectures and the systems implementing the architectures are unique, domain-specific designs that aim to change the way scientists approach research in the social sciences (e.g., finance, economics). Experimental computational environments allow practical experimentation rather than only theoretical research often encountered in academia. This is believed to be proven in the example application to algorithmic trading and general social analytics.

This thesis introduced novelty and contributed to both science and industry in the following way:

It identifies key features, modes of operation and software architecture of an electronic financial exchange for simulated (virtual) trading. It also identifies key models of exchange emulation, defined a standard trading process, identified key market participants and their trading styles. The identified technical prerequisites for electronic trading exchange includes: a) a trade communication protocol - to define ways of communicating with EMs, b) a matching & exchange engine - to define transaction rules and enable trading, c) an accountancy and clearing functionality - to define who can trade and what is the state of their holdings, and d) a risk management functionality - to manage exposure to loss. The typical electronic trading exchange participants were divided into: a) quantitative developers, b) quantitative analysts, c) traders, d) portfolio managers, e) trade-floor managers and f) risk managers. The identified major simulation types were divided into: a) order vs. price, LIVE EM for multi-agent transaction simulation, b) order vs. price, EOD EM for end-of-day multi-agent transaction simulation, c) order vs. price, BACK-TEST EM for single-agent historic transaction simulation, d) order vs. order, LIVE EM for multi-agent financial system simulation, e) order vs. order, BACK-TEST EM for multi-agent financial system simulation.

It defines what is believed to be a unique software Environment for portfolio selection containing combinatorial framework for discovery of subsets of internally Cointegrated time-series of financial securities and a graph-guided search algorithm for combinatorial selection of such time-series subsets. The key features of the designed Portfolio Selection System were identified as: a) subscription and on-the-fly handling of streams of prices, b) on-the-fly raw data buffering, c) on-the-fly data granulation, d) cluster discovery, e) cluster evolution monitoring, f) cluster query API and g) big data provisioning.

It identifies the key elements and low-level features of algorithmic trading strategies and defines an API framework supporting event-driven and service-oriented paradigms for building such strategies. The major classes of strategies that one can differentiate were identified as: Arbitrage-based strategies, Hedging strategies, Directional strategies and Portfolio-based strategies.

It identifies key simulation processes that can be performed on AT models, defines a standard analytic

structures of AT models and introduces software architecture for large-scale simulation and model administration. It also describes two in-depth experiments with a free-run and back-test simulations. The most common types of simulations were identified as: a) a free-run, b) a back-test, c) a stress-test, d) a forward-test (a Monte-Carlo test), e) a sensitivity-test, f) an optimisation and g) a multi-agent version of the described simulations. The standard analytic structures of AT models were identified as: a) the pre-trade analysis (data analysis, state of the world analysis), b) the signal generation (decision taking process, policy formation), c) the trade execution (execution of actions, policy execution) and d) the post-trade analysis (evaluation of results, utility analysis).

7.3 Extensions & Further Work

The author of the thesis together with UCL and in collaboration with institutions including Bank of England, LMAX Exchange, Eurex(Deutsche Borse), Microsoft, Tower Trading and Banking Science has developed various computational environments on the basis of the proposed architectures. These can be considered extensions of the original ideas. Such extensions were developed for managing large volumes of real-time and archive data (financial, economic and social media) and supporting data mining, simulation modelling and stream processing analytics using high-performance computer clusters. The specific focus of environments was on algorithmic trading, risk and social sentiment analysis, and utilising real-time financial, social and news data.

The following may be considered extensions to the already described architectures as well as further work.

The social media streaming, storage and analytics platform (SocialSTORM): is a cloud-based central hub platform which facilitates the acquisition of text-based data from on-line sources such as Twitter, Facebook, RSS media and news. The system includes facilities to upload and run Java-coded simulation models to analyse the aggregated data; which may comprise scraped social data and/or users own proprietary data. There is also connectivity to the ATRADE Environment which supports access to further quantitative finance and economic data, and associated analytics. SocialSTORM consists of infrastructure tools to facilitate data acquisition, database connectivity, and various levels of access and administration along with data repositories for long and short term data storage. The Environment is able to operate in two simulation modes: 1) a historical mode which utilizes data already stored at the time of running the desired simulation (ideal for data-mining and back-testing), and 2) a live mode operating on a near real-time stream of data, which is continually monitored from the sources throughout the simulation (ideal for analysing financial markets and developing algorithmic trading strategies).

The distributed analytics, control & utilities system (DRACUS): is a large-scale experimental system for simulation of generic analytic models. DRACUS aims to provide a unified platform not only for

running the AT models in distributed environment but also the SocialSTORM models and the MASON (http://cs.gmu.edu/ eclab/projects/mason/) multi-agent models. This makes DRACUS particularly suitable for testing, simulation and optimization of complex social, financial and economic problems. DRACUS can be considered an implementation of all the simulation types described in the Introduction Chapter and involves creation of cluster-based environment to support such simulations.

BoE Financial Stability Models: the Bank of England was interested in designing new policies that would have stabilising effects on UK financial system. The institution was also looking at improved utilisation of existing policies to achieve the same goal. In order to contribute to BoEs vision the Oxford University and UCL collaborated with BoE on development and implementation of resilience models of UK financial systems and on modelling framework for multi-agent simulations. The DRACUS project together with the designs of multi-agent EM simulators were part of the supporting infrastructure.

Algorithmic Strategy Hub: UCL together with BankingScience and LMAX designed a system for students and inexperienced algorithmic traders that allows them to build directional algorithmic trading strategies in a graphical mode, and trade via ATRADE with the strategies when they are ready to use. The Strategy Hub started as an educational tool but thanks to its flexible design and efficient implementation can be utilised in real trading for profit. The design of the system enables multiple users to simultaneously design new strategies, to manage existing strategies and to trade with the strategies. Users can select amongst a variety of securities, pre-trade analysis elements, open & close signal generation rules and risk management elements to construct algorithmic strategies. This building process provides validation functionality that guides the user through the entire construction process. The heart of the system lies in its very powerful signal generation logic that is a novelty in the algorithmic trading world. Finally, the system provides an intelligently designed web-based GUI to enable users to authenticate, create and manage strategies.

New Generation of Trade Stations with Advanced Virtual Analytics & Trade-Aware Realms (AVATAR): UCL together with BankingScience and the support of Microsoft was exploring possibilities of a new generation of HCI tools for 3D management and visualization of information flows. The functionalities were to allow centralized visual management of data streams, data analytics, modelling, simulations and model runtime execution, all for users with limited exposure to information technologies. The functionalities were also supposed to allow streaming and visualization of data in a natural and meaningful way. For that purpose a demonstrator was to be developed with an aim to present a set of use-case scenarios for one of the Financial Wind Tunnel Observatories, dedicated to Algorithmic Trading. The Demonstrator was to present a possibility to interface a set of information chains to stream financial data from ATRADE-based source, through the event bus, to PCs, tablets and mobile phones. The demonstrator was also to present a possibility to interface analytical libraries, with CEP (Complex

Event Processing) engine and message bus and visualize the outcome information. Finally the demonstrator was to present a possibility to interface, control and monitor execution of algorithmic trading models/model simulations.

The initial architecture of the system was developed, together with a set of use-case scenarios, identification of applicable technologies and typical users of the system. However, due to financial constrains the project was not taken further.

Infrastructure for Traders' sentiment analysis and replication: UCL together with BankingScience and a significant London hedge-fund is involved in a research and development project involving creation of a system for informed trade replication on the basis of a continuous sentiment (group wisdom) analysis of a group of traders. The ATRADE Environment is utilised as a means of support to the sentiment and selection model. The extensions to the Environment were developed to provide trader tracing and evaluation functionalities.

Infrastructure for Trading Competitions: UCL together with BankingScience and a set of banks and hedge funds (Barclays, City and Knight amongst others) was involved in organization of algorithmic and proprietary trading competitions. The most recently organised, together with Eurex (Deutsche Borse), was a trading game on the basis of UCLs ATRADE infrastructure. The game involved manual trading of a selection of Eurex securities via the ATRADEs web interface. The trading process was supported by the EOD EM environment allowing an end-of-day execution of transactions against prices of selected securities. For the time of the competition, the ATRADE infrastructure was customised to support manual trading, with the ATRADE GUI being the main front-end means of executing transactions, and the EOD EM simulation engine being the back-end matching engine. The GUI provided support for visualisation of states of tradable securities, visualisation of active orders and transactions and, finally, for statistics of the user account. The EOD EM simulation environment allowed matching of the market, limit and stop orders against a selection of Eurex securities. The ATRADE infrastructure ensured a persistent connection to the Trading Technologies' (TT) gateway to the Eurex exchange, to feed Eurex market feeds to the simulation environment.

The organized competitions and the approach to integration of simulation environments to support such competitions can be considered another extension to the proposed software architectures and systems implementing it.

Bibliography

[Abouzeid et al., 2009] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., and Rasin, A. (2009). Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proceedings of the VLDB Endowment*, 2(1):922–933.

[Agrawal et al., 2011] Agrawal, D., Das, S., and El Abbadi, A. (2011). Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 530–533. ACM.

[Ainsworth, 2009] Ainsworth, M. (2009). Market data and business information two peas in a pod? *Business Information Review*, 26(2):81–89.

[Alexander and Alexander, 1999] Alexander, C. and Alexander, C. (1999). Optimal hedging using cointegration. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 357(1758):2039–2058.

[Andrade et al., 2009] Andrade, H., Gedik, B., Wu, K., and Yu, P. (2009). Scale-up strategies for processing high-rate data streams in system s. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1375–1378. IEEE.

[Ang and Timmermann, 2011] Ang, A. and Timmermann, A. (2011). Regime changes and financial markets. Technical report, National Bureau of Economic Research.

[Ashby, 1956] Ashby, W. (1956). *An introduction to cybernetics*. Taylor & Francis.

[Asur and Huberman, 2010] Asur, S. and Huberman, B. A. (2010). Predicting the future with social media. *arXiv preprint arXiv:1003.5699*.

[Barga et al., 2006] Barga, R., Goldstein, J., Ali, M., and Hong, M. (2006). Consistent streaming through time: A vision for event stream processing. *arXiv preprint cs/0612115*.

[Batten et al., 2012] Batten, L., Castleman, T., Chan, C., Coulthard, D., Savage, R., and Wilkins, L. (2012). Engaging suppliers in electronic trading across industry sectors. In *Multi-disciplinary solutions to industry & government's e-business challenges: proceedings of the IFIP WG8. 4 Working*

Conference on E-business, Salzburg, Austria, June 18-19 2004, pages 182–199. International Federation for Information Processing (IFIP).

[Berkowitz, 2001] Berkowitz, J. (2001). Testing density forecasts, with applications to risk management. *Journal of Business & Economic Statistics*, 19(4):465–474.

[Berndt and Clifford, 1994] Berndt, D. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370.

[Blog, 2012] Blog, J. L. (2012). Cloud, grid, or cluster? clarifying the terminology of distributed computing. [Online; accessed 19-January-2013].

[Bullock, 2011] Bullock, S. (2011). Prospects for large-scale financial systems simulation.

[Cameron, 2009] Cameron, J. (2009). Evolution of the fix engine.

[Cañete et al., 2008] Cañete, A., Constanzo, J., and Salinas, L. (2008). Kernel price pattern trading. *Applied Intelligence*, 29(2):152–156.

[Cesari et al., 2012] Cesari, R., Marzo, M., and Zagaglia, P. (2012). Effective trade execution.

[Chandramouli et al., 2010] Chandramouli, B., Ali, M., Goldstein, J., Sezgin, B., and Raman, B. (2010). Data stream management systems for computational finance. *Computer*, 43(12):45–52.

[Chootong and Sornil, 2012] Chootong, C. and Sornil, O. (2012). Trading signal generation using a combination of chart patterns and indicators.

[Chua et al., 2009] Chua, C., Milosavljevic, M., and Curran, J. (2009). A sentiment detection engine for internet stock message boards. In *Australasian Language Technology Association Workshop 2009*, page 89. Citeseer.

[Ciliendo, 2007] Ciliendo, E. (2007). Linux performance and tuning guidelines.

[Clark, 2010] Clark, C. (2010). Controlling risk in a lightning-speed trading environment. *Chicago Fed Letter*, 272.

[Cloudscaling, 2012] Cloudscaling (2012). Grid, cloud, hpc? what's the diff? [Online; accessed 19-January-2013].

[Consulting, 2008] Consulting, F. (2008). Fmr pricing engine.

[Cui et al., 2012] Cui, B., Zhao, Z., and Tok, W. H. (2012). A framework for similarity search of time series cliques with natural relations. *Knowledge and Data Engineering, IEEE Transactions on*, 24(3):385–398.

[Cukier and Mayer-Schoenberger, 2013] Cukier, K. and Mayer-Schoenberger, V. (2013). Rise of big data: How it's changing the way we think about the world, the.

[Domowitz, 1993] Domowitz, I. (1993). A taxonomy of automated trade execution systems. *Journal of International Money and Finance*, 12(6):607–631.

[Erl, 2006] Erl, T. (2006). *Service-oriented architecture: concepts, technology, and design*. Pearson Education India.

[Fiorenzani et al., 2012] Fiorenzani, S., Ravelli, S., and Edoli, E. (2012). Directional trading. *The Handbook of Energy Trading*, pages 33–80.

[Fixprotocol, 2013] Fixprotocol (2013). Fix protocol. [Online; accessed 19-January-2013].

[Foucault and Menkveld, 2008] Foucault, T. and Menkveld, A. (2008). Competition for order flow and smart order routing systems. *The Journal of Finance*, 63(1):119–158.

[Fusco et al., 2010] Fusco, F., Stoecklin, M., and Vlachos, M. (2010). Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic. *Proceedings of the VLDB Endowment*, 3(1-2):1382–1393.

[Gabrielsson et al., 2012] Gabrielsson, P., Konig, R., and Johansson, U. (2012). Hierarchical temporal memory-based algorithmic trading of financial markets. In *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on*, pages 1–8. IEEE.

[Gangadharan and Swami, 2004] Gangadharan, G. and Swami, S. (2004). Business intelligence systems: design and implementation strategies. In *Information Technology Interfaces, 2004. 26th International Conference on*, pages 139–144. IEEE.

[Goldin and Kanellakis, 1995] Goldin, D. and Kanellakis, P. (1995). On similarity queries for time-series data: constraint specification and implementation. In *Principles and Practice of Constraint Programming?CP'95*, pages 137–153. Springer.

[Gunopulos and Das, 2000] Gunopulos, D. and Das, G. (2000). Time series similarity measures (tutorial pm-2). In *Tutorial notes of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–307. ACM.

[Harris, 2004] Harris, N. (2004). Linux handbook: A guide to ibm linux solutions and resources.

[Hu et al., 2012] Hu, H., Dai, W., Dai, Y., Sun, H., Zhang, P., and Liu, X. (2012). Flexible business support system for stock index futures transaction. *Journal of Software*, 7(11):2630–2639.

[Huang et al., 2011] Huang, S., Lai, S., and Tai, S. (2011). A learning-based contrarian trading strategy via a dual-classifier model. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):20.

[Hughes Jr et al., 2012] Hughes Jr, J., Moore, D., Friedman, B., Vincent, T., et al. (2012). Order matching process and method. US Patent 8,244,622.

[IBM, 2008] IBM (2008). Current technologies in trading and execution venues.

[Jagadish et al., 1995] Jagadish, H., Mendelzon, A. O., and Milo, T. (1995). Similarity-based queries. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 36–45. ACM.

[Johansen and Sornette, 2010] Johansen, A. and Sornette, D. (2010). Shocks, crashes and bubbles in financial markets. *Brussels Economic Review (Cahiers economiques de Bruxelles)*, 53(2):201–253.

[Katz and McCORMICK, 2000] Katz, J. O. and McCORMICK, D. L. (2000). *The encyclopedia of trading strategies*. McGraw-Hill New York.

[Keogh et al., 1997] Keogh, E., Smyth, P., et al. (1997). A probabilistic approach to fast pattern matching in time series databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 24–30.

[Khan and Quadri, 2012] Khan, R. and Quadri, S. (2012). Business intelligence: An integrated approach. *Business Intelligence Journal*, 5:64–70.

[Kissell and Malamut, 2005] Kissell, R. and Malamut, R. (2005). Understanding the profit and loss distribution of trading algorithms. *Trading*, 2005(1):41–49.

[Kumar, 2009] Kumar, A. (2009). The risk management framework: Building a secure and regulatory compliant trading architecture.

[Lin and Shim, 1995] Lin, R. A. K.-l. and Shim, H. S. S. K. (1995). Fast similarity search in the presence of noise, scaling, and translation in time-series databases. *IBM Almaden Research Center*.

[Lo, 2004] Lo, A. (2004). The adaptive markets hypothesis: Market efficiency from an evolutionary perspective. *Journal of Portfolio Management, Forthcoming*.

[Luckham, 2008] Luckham, D. (2008). The power of events: an introduction to complex event processing in distributed enterprise systems. *Rule Representation, Interchange and Reasoning on the Web*, pages 3–3.

[Lybäck and Boman, 2004] Lybäck, D. and Boman, M. (2004). Agent trade servers in financial exchange systems. *ACM Transactions on Internet Technology (TOIT)*, 4(3):329–339.

[Madhavan, 2002] Madhavan, A. (2002). Implementation of hedge fund strategies. *Special Issues*, 2002(1):74–80.

[Madura, 2012] Madura, J. (2012). *Financial markets and institutions*. CengageBrain. com.

[Markowitz, 1952] Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1):77–91.

[Markowitz, 1959] Markowitz, H. (1959). Portfolio selection: efficient diversification of investments. cowles foundation monograph no. 16.

[Maruyama, 1963] Maruyama, M. (1963). The second cybernetics: Deviation-amplifying mutual causal processes. *American scientist*, pages 164–179.

[McNeil et al., 2005] McNeil, A., Frey, R., and Embrechts, P. (2005). *Quantitative risk management: concepts, techniques, and tools*. Princeton university press.

[Mendelzon, 1998] Mendelzon, A. (1998). Efficient retrieval of similar time sequences using dft.

[Michelson, 2006] Michelson, B. (2006). Event-driven architecture overview. *Patricia Seybold Group*, 2.

[Nuti et al., 2011] Nuti, G., Mirghaemi, M., Treleaven, P., and Yingsaeree, C. (2011). Algorithmic trading. *Computer*, 44(11):61–69.

[Perng et al., 2000] Perng, C.-S., Wang, H., Zhang, S. R., and Parker, D. S. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 33–42. IEEE.

[Risca, 2008] Risca, M. (2008). Trading floor architecture.

[Ross, 1973] Ross, S. (1973). *The arbitrage theory of capital asset pricing*.

[Shepherd, 1997] Shepherd, A. (1997). *Market information services: Theory and practice*, volume 125. Food & Agriculture Organization of the UN (FAO).

[StuIz, 1984] StuIz, R. (1984). Optimal hedging policies. *J. Financ. Quant. Anal*, 19:127–139.

[Su and Iyengar, 2012] Su, G. and Iyengar, A. (2012). A highly available transaction processing system with non-disruptive failure handling. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 409–416. IEEE.

[Su and Iyengar, 2013] Su, G. and Iyengar, A. (2013). Avoiding disruptive failovers in transaction processing systems with multiple active nodes. *Journal of Parallel and Distributed Computing*.

[Subrahmanyam and Titman, 2012] Subrahmanyam, A. and Titman, S. (2012). Financial market shocks and the macroeconomy. *Available at SSRN 2195184*.

[Treleaven et al., 2013] Treleaven, P., Galas, M., and Lalchand, V. (2013). Algorithmic trading review. *Communications of the ACM*, 56(11):76–85.

[Tsagaris et al., 2012] Tsagaris, T., Jasra, A., and Adams, N. (2012). Robust and adaptive algorithms for online portfolio selection.

[Turaga et al., 2010] Turaga, D., Andrade, H., Gedik, B., Venkatramani, C., Verscheure, O., Harris, J., Cox, J., Szewczyk, W., and Jones, P. (2010). Design principles for developing stream processing applications. *Software: Practice and Experience*, 40(12):1073–1104.

[Vincent and Armstrong, 2010] Vincent, A. and Armstrong, M. (2010). Predicting break-points in trading strategies with twitter. *SSRN eLibrary*.

[Vovsha and Passonneau, 2011] Vovsha, A. and Passonneau, O. (2011). Sentiment analysis of twitter data. *ACL HLT 2011*, page 30.

[Warren Liao, 2005] Warren Liao, T. (2005). Clustering of time series data, a survey. *Pattern Recognition*, 38(11):1857–1874.

[Wikipedia, 2013a] Wikipedia (2013a). Correlation and dependence. [Online; accessed 09-March-2013].

[Wikipedia, 2013b] Wikipedia (2013b). Scrum (development) — Wikipedia, the free encyclopedia. [Online; accessed 21-January-2013].

[Wu et al., 2006] Wu, E., Diao, Y., and Rizvi, S. (2006). High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM.

[Yoon et al., 1993] Yoon, E., Guffey, H., and Kijewski, V. (1993). The effects of information and company reputation on intentions to buy a business service. *Journal of Business Research*, 27(3):215–228.

[Zietz, 2000] Zietz, J. (2000). Cointegration versus traditional econometric techniques in applied economics. *Eastern Economic Journal*, 26(4):469–482.