MARKET SURVEILLANCE SYSTEM

SURVEILLANCE CLIENT

Author: Johan Örtenblad

Tutors: Vladimir Vlassov, KTH

Torkel Erhardsson, KTH

Håkan Carlbom, OM

Johan Norén, OM

Master's thesis 2001 for KTH (departments of Financial Mathematics and Microelectronics and Information Technology) and for OM

1. Table of contents

2. Abstract

Over the course of the last decades, the growth of the national and international capital markets has been tremendous. The activity, measured in number of traded contracts, on a typical market place has soared. At the same time, the number of electronically operated market places increases constantly. Facing this development, the need for market place surveillance is today more relevant than ever.

The present thesis regards the client side of a client-server based market surveillance system for use with an electronic exchange system. It was developed during the fall of 2001 at OM in Stockholm for use with the CLICK trading system on the JIWAY exchange, but with flexibility and scalability as core ambitions. During the same time, Peter Bergenwald developed the server side of the same market surveillance application. Naturally, there was a close collaboration regarding the design and implementation of the surveillance system, even though the client and the server could and should be considered separately.

We start off by introducing market surveillance, in terms of market manipulation and insider trading. Thereafter, a procedural and statistical framework for an automated surveillance process is presented, along with a specification for how to implement it on an exchange system. A fully functional pilot implementation has been implemented of the client, as well as the server, application. For obvious reasons, this report only covers the client part, which on the other hand is presented in detail. Consideration is also paid to the interaction with the server and the surveillance operator.

The report is concluded with a section on the performance of the pilot implementation, coupled with a discussion on how to move the pilot implementation forward towards being part of a release version of the surveillance system.

3. Acknowledgements

The author would like to thank the tutors at OM – Håkan Carlbom and Johan Norén – for their interest, support and exchange of ideas during the development process of this thesis. Their help was invaluable to the results presented herein.

Also at OM, Mats Danielsson contributed with much help in the development of the surveillance methodology.

A thank you is, of course, also directed towards the tutors at KTH – Torkel Erhardsson and Vladimir Vlassov – for their help in writing this report.

4. Introduction

During the fall of 2001, me and Peter Bergenwald finishied our thesis work at the OM group in Stockholm. The task we faced was that of investigating the possibilities to implement an automatic surveillance system for the CLICK bourse trading system, and also to implement a pilot application to illustrate our findings. I will in this report give an overview of the results of our work.

The interested reader should also consult the thesis report by Peter Bergenwald, since the present thesis solely aims at covering my part of our common work.

In this section, an introduction to the area of market surveillance is given, followed by a short description of OM. Finally, the task definition is presented.

4.1. Market surveillance

Over the last decades, the capital markets of the world have seen a tremendous development. The book by Grinblatt and Titman[1] gives a good overview over this development, in the US and abroad. Between 1970 and 1995, the total value of the outstanding equity in the US, for example, soared from just over $1 000 billions to almost $7 000 billions. Several trends are now working towards an even greater importance of these markets, as well as towards an increased access for different members of society. In the following, I give an overview to these patterns:

The increasing global presence of large, multinational businesses shopping around for cheap capital is forcing the governments of the world to streamline the legislation and tax policies for the capital markets. This is in turn likely to make the raising of capital in different geographic regions equally expensive, adding to the mobility of global capital. Conversely, it is becoming ever more difficult for countries to sustain regulations favouring large corporations at the cost of smaller ones, as the mobility of capital increases. Therefore, deregulation and capital mobility are walking hand in hand in a kind of spiral movement.

At the same time, the markets have never been this inventive before when it comes to coming up with new financial solutions to various problems. For example, today it is possible to hedge against an abundance of risks categories by the use of standardised instruments in the capital markets, from interest rate exposure to catastrophe-linked risk[2]. Also, companies can now securitise on many more types of assets than has been possible before. One illustrative example is the sell of a company's accounts receivable as a bundled security on the market.

Behind many of these overwhelming changes and the rapid expansion of the global capital markets is the constant development of the technology used for the actual trading. Today it is possible to simultaneously issue assets worth

[1] Grinblatt – Titman (1998)
[2] Örtenblad – Bogentoft [2001]

billions of dollars in several countries. It is virtually possible to trade 24 hours a day, following the sun around the globe[3]. It is, however, not only the big players that can benefit from this development. The spread of sophisticated software for retail trading, brokerage, clearing, etc. is beginning to allow many smaller players such as private persons to trade in the markets with almost the same ease as the large corporations.

All of these trends in combination are making the world's financial markets increasingly complex, sophisticated and extensive. In the long-term, the effects of this development are, of course, of much value. Most importantly, the spread of risk and the distribution of capital in the world can potentially be of tremendous benefit.

However, in the backwater of these great land-winnings, there are also problems. As larger and more complicated financial bonds tie the global market actors together, more opportunities for scams arise. The need for surveillance is therefore also on the increase. Nowadays the markets have more participants than ever, each wanting the turn of events to go the way of their portfolio. As the barriers of entry to the markets are lowered along with transaction costs, actors that would not have been thinking of trying to get a piece of the pie the dirty way start to look for ways to side-step the market systems.

In a perfect market, the prices of the instruments traded reflect the current information publicly available. If, however, the prices of some instruments do not obey to this rule, those who have the better information can use this fact to make money.

It is hence possible to make a profit with significantly lower risk[4] than the one faced by the rest of market by trading on an asset that you have some special, non-public knowledge of – e.g., information not yet released to the public. Ideally, the same information should be available to all market participants in order for the pricing to accurately reflect all available information. However, this is not precisely the case, since there are people working in all the listed companies. All of these people are potential insider traders. Furthermore, it would not be possible to monitor the trading activity of all of these persons. Instead, one chooses to focus on, in Yahoo Finance's[5] words, "*officers, directors, major stockholders, or others who hold private inside information allowing them to benefit from buying or selling stock.*" Still, the surveillance of these individuals' holdings is quite a task.

[3] The exceedingly capital-intensive and fragmented global money markets are, for example, active 24/7 around the year. Also, there are examples of centralised exchanges, such as the OM-run UK Power Exchange in London, that are operational 23,5 hours per day, seven days a week.

[4] One common measure of the relative risk of an investment (e.g. a company share) is the *Sharpe's ratio*, defined as the expected return divided by the historical standard deviation for the instrument. The higher the Sharpe's ratio, the higher the expected return compared to the risk of the investment. For more information, please see Bodie – Kane – Marcus (1999] p. 754.

[f] finance.yahoo.com

Another method of getting a cheap return in the market is to manipulate the market prices in favour of one's own holdings. By way of example, if I can affect the price of a stock so that it dips temporarily, I can buy at the lowest point (knowing that the price dip does not reflect any changes in the fundamental data associated with the stock). In this case, the potential wrongdoer is anyone trading in the market. Therefore, the surveillance task becomes even more difficult. Also, many of the techniques that can be used in order to affect prices on a market build upon ordinary sound market practice. As there are no clear-cut rules for what behaviour is price manipulative, the problem arises of defining limits for the activities that are to be considered legal.

On top of this, price manipulation is highly coupled with control of the information about the instrument being manipulated. Hence, a surveillance system potent of the detection of this kind of non-allowed behaviour needs to consider the news flow as well as the hard market data. For example, if the price of an instrument rises by 5%, it can be because of the fact that the company released new, positive information. On the other hand, it may equally well be a consequence of some kind of price manipulation on the part of one or several of the market actors involved in the trading of the instrument. When to anticipate manipulative behaviour is not obvious in this simple case, and real market situations are, of course, usually much more complicated.

Aitken and Berry give a good introduction to the emerging market surveillance concern[6]. They argue that different governments try to limit the information bias in the financial markets, through the use of legislation and for several reasons. Firstly, a publicly available and accurate flow of information increases the stability of the financial markets and lowers the systematic risk faced by the market participants. This is true both on a domestic and an international scale, the latter being propelled by the currently rapid expansion of the international financial markets. Secondly, a common economic and political goal for markets is that they should be fair and efficient. Fair means that no one actor should have any information advantage over another, effectively ruling out insider trading. Efficient means that the information that is equally available to every market participant accurately should reflect the actual status of the companies traded on the market, and that the asset prices observed are accurately set by the laws of supply and demand on the basis of this information. Therefore, price manipulation makes the market inefficient, giving rise to surplus economic costs.

At the same time as the governments of the world are getting increasingly interested in controlling this kind of behaviour in the financial markets, the exchanges themselves share this interest. The reason for this is of a similar nature – the existence of any given market place depends upon the public's trust in its ability to fairly and efficiently distribute risk and investment means between its participants. Therefore, it is of vital importance for the exchange itself to sustain the credibility of the market place it operates. But the threat also comes

[6] Aitken – Berry (1991)

from above – much of the regulations for the financial activity of today is initiated from within the financial markets themselves. In short, the financial institutions have a relatively large leeway for self-regulation. This arrangement cuts both ways – the business has control over its own structure and regulations, as long as the government trusts in their ambition to strive for the achievement of not only economic goals, but also for the social ones drafted by the government.

For these reasons, stock exchanges in the US and in Canada started to look more seriously into the possibilities of an active surveillance of their markets during the boom years of the 1980's. Since then, the market has developed and several software systems have been developed for the automatic detection of non-allowed market behaviour. Later in this chapter, we will shortly describe a couple of these other systems.

4.2. OM

OM is the world's leading supplier of transaction technology[7], providing exchange technology solutions to more than 25 stock exchanges and clearing houses around the globe. Among other things, the company markets two major exchange systems, and also operates the Stockholm Stock Exchange, as well as other exchanges in Calgary and in London. Thus, for OM the growing issue of market surveillance is highly relevant – both as a technology provider and as an exchange operator.

The company markets two major exchange systems under the names of CLICK and SAXESS, respectively. They constitute completely autonomous solutions, and are sold separately. Although their features are somewhat different, they can be used interchangeably by one single exchange using the proper configurations. One thing that unites them is that both keep a log of the transactions that have been processed by the system, making it possible to add surveillance functionality on top of the system without draining power from the exchange process itself.

One of the market places running SAXESS is the spot stock market in Stockholm. The derivatives trade in Stockholm operates using the CLICK platform. Another exchanges running the CLICK exchange system is JIWAY, the market place of primary concern to this thesis. JIWAY is aiming at providing European and US retail investors with a highly competitive international equity exchange for smaller transactions. By providing the brokers in the different countries that are connected (currently France, UK, Sweden, Germany, the Netherlands, Italy and the USA) the access to JIWAY, the retail public gains access to a cheap and easy way to trade in foreign stock.

One problem when developing a surveillance system to be run on top of either the CLICK- or SAXESS system is that the logs are differently represented on the two platforms. Another problem is that for different exchanges, such as JIWAY and

[7] One possible definition of the term "transaction technology" is *IT infrastructure for the processing of transactions at an exchange*.

the ISE[8], the actual transactions that drive the system have different informational formats. In order to use the same surveillance system in all of these cases, a common interface needs to be developed for the communication between the exchange system log and the surveillance application. In chapter 6, this issue of information mapping is addressed in detail.

4.3. Task definition

The aim of this thesis is to provide a suggestion for the architecture of an automatic market surveillance system, both theoretically and practically, to be used with the CLICK system on the JIWAY exchange. However, the solution should ideally be general and *flexible* enough to be adaptable for use on other exchanges and exchange systems, particularly with the SAXESS system.

Except for flexibility, *scalability* of the solution is an important evaluation criterion. This is natural, given the recent development on the international financial markets as well as the growth of the OM-run market places.

Figure 1 below gives an overview of the prerequisites.

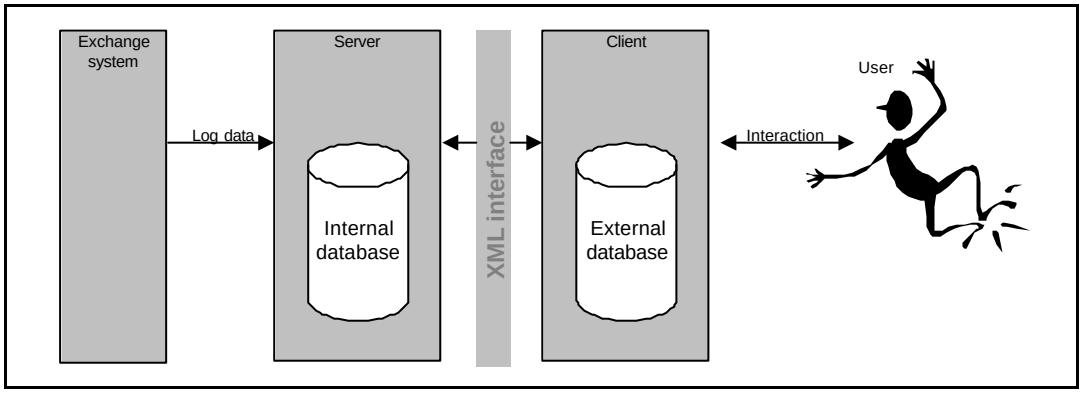


Figure 1 – Overall prerequisites for the thesis. The part focused on in this thesis is the Client application, whereas the thesis report by Peter Bergenwald is primarily interested in the server-side solution.

4.3.1. The server

The server-side part of the system (the part that Peter Bergenwald has developed) should be highly exchange system specific. It reads the log data produced by the exchange system in real-time and processes it. Through the use of the XML[9] interface, it provides the client application with a view of the log data that is more general than the specific log data format provided by the exchange. At the same time, a server-side database makes the log data (as

[8] ISE (International Securities Exchange) is the first electronic US option exchange and is also running CLICK.
[9] XML (eXtended Markup Language)

viewed through the XML interface) randomly accessible, making search operations of historic data possible.

4.3.2. The client

Via the XML interface, the client-side application (the one in focus for this thesis) communicates with the server. This part has been developed and tested for the CLICK-operated JIWAY exchange, but with the ambition to be flexible enough to be reconfigured to operate on other exchanges as well as under different systems. The general view of the system log data given by the server-side application should therefore also add system- and exchange independency. The definition of the XML interface itself was also part of our task.

The client application communicates with the end surveillance operator (the user) via a *Graphical User Interface* (GUI), displaying surveillance information and accepting configuration input.

4.4. Goals for the thesis report

In this thesis, I will try to give a thorough description of the theoretical design aspects of the automatic surveillance client mentioned above, given that a server-side solution exists. Also, a description of the actual pilot-version client, developed in the Java language, will be provided. This pilot application is built to interact with the pilot server application developed by Peter Bergenwald, communicating via a first-generation XML interface. The aim of exchange system- or exchange portability is not fulfilled to 100% in the current implementation. However, the design of the client application is intended to be easily configurable to handle new and more generic information. Also, a new and more general version of the suggested XML interface is presented, intended to be used with a generic exchange provided there is a server-side solution provided for each particular exchange and exchange system.

For a better understanding of the actual pilot implementation, a discussion of the theoretical framework will precede the description of the client application. This discussion spans over present surveillance targets, -methodologies, -informational prerequisites, and finally the models actually used.

Then, the actual pilot implementation is described, together with a thorough performance evaluation.

Technical details are, to as large an extent as possible, moved out of the text and placed in appendices found at the end of the report.

4.5. Delimitations

To investigate and build a full-scale surveillance agent is, of course, a very large task. In order to gain focus I have tried to limit the area to the most crucial or interesting parts of such a surveillance agent. In short, I will:

- limit the number of financial markets that are to be surveillable by the use of the thesis's surveillance application. For instance, derivatives markets

will not be possible to surveil, nor will there be a possibility to combine data from different markets in the surveillance process.

- limit the data input to the surveillance process. Additional data that could (and should) be incorporated in a release version include a news feed and clearing information.
- limit the scope of the surveillance process implementation to a small number of tests, in order to exemplify the theoretical surveillance framework. The level of mathematical complexity actually implemented is to be kept relatively low. By contrast, I will build this framework in rather general terms, in order for future enhancements and additions to be easily implemented. In particular, I will only consider market manipulation, leaving insider trading for future functionality expansions.
- not be able to present any results regarding the accuracy of the surveillance tests actually developed, since the access to real exchange data for different reasons is restricted.
- limit the scope of the GUI to a very rudimentary level, with the primary objective of making it possible to demonstrate and exemplify the workings of the surveillance process. Specifically, configuration of the system via the GUI will be strongly limited.

5. Theoretical framework

In this section, background information is given for market manipulation and insider trading and possible unlawful market conducts. Following this introduction, a discussion of the process of actually detecting these events is presented. Lastly, the general approach of the surveillance application is described, together with the statistical theory that it is based upon.

5.1. Types of unlawful conduct

As mentioned, there are two major categories of methods to gain a better estimated return while maintaining the risk in the market – market manipulation and insider trading. The article by Aitken and Berry[10] features a thorough discussion of these. Let us start with market manipulation.

5.1.1. Market manipulation

Motives

The reasons for manipulative behaviour affecting the price or the volume of a company's share vary. Individual investors, the company giving out the shares as well as other companies may have incentives to affect the price of shares one way or the other.

By raising the volume of an asset, for instance, liquidity increases, thereby helping an investor to an easier exit for an open position, or to magnify the price manipulation attempts currently pursued. A common factor for price manipulation is that because the fundamental information upon which the stock price is founded cannot be changed, the long-term price cannot be expected to change. Instead, market manipulation is all about strategies in the relatively short-term time frame.

The company itself may also want the price of its share to rise in the short run, because it is about to issue a new round of shares to the market. Naturally, it wants to sell the new stock for as high a price as possible, giving birth to the interest of raising the spot price in the short run. Also, old promises of good returns on shares bought by investors in the past might be hanging around, promises that the company wants to fulfil for one reason or another. By artificially raising the value of the company, genuine investors might also become more attracted, increasing the possibility of financing new projects. Under other circumstances (e.g., under financial distress), a price boost could possibly convince important shareholders not to leave the company. Companies will sometimes also want to affect the price of their own stock or that of other companies while facing potential take-over, either the take-over of another company or of the own company.

[10] Aitken – Berry (1991)

Also, individual investors (private or institutional) might want to temporarily affect the prices of the instruments, in which they trade or take ownership positions. The most fundamental case is where there exists a short-term, long position in either the stock itself or an option with the stock as underlying. There are, however, other ways to make money by price manipulation. For example, by decreasing the price of a security, the investor, knowing that the fundamentally implied price should be above the temporarily sunken market price, can take a larger long position at an advantageous cost. Other examples include investors in low-liquidity assets creating a false appearance of activity in the paper (thus inducing other parties to enter the market and artificially creating an increased liquidity), as well as different tax planning reasons. More complicated situations include the holding company, which might be interested in a value increase of the company. Finally, major private shareholders could give a shot at manipulating the price prior to selling part of their holdings or when so-called convertible loans have been issued.

Typical techniques

In this section, an overview of the ways in which a market actor can affect prices is given. For a more detailed listing of these techniques, please see section 8.1 of the appendices.

Market manipulation is either the manipulation of the price or the volume of a share. There are vast possibilities to affect these metrics both in the equity-, derivative- and fixed income markets, and by using combinations of positions in these markets. One common example is to affect the stock market in order to gain in the options market, or the other way around. These issues will however not be dealt with herein, as discussed above in the introduction.

In order to affect the price or the volume of an instrument, one usually has to be a major player in that particular asset, since one's actions need to be large-scale enough to affect the actual market for the asset. One common way to gain the control of an instrument is called *cornering*. It involves buying significant volumes, preferably at artificially set price levels, until one becomes one of the major shareholders in the market. Of course, the more illiquid the market, and the smaller the company, the more likely this is to succeed. Still, you will need substantial means in order to afford volumes high enough to gain significant controlling power.

Once a certain control over the instrument is gained (by the use of cornering or in some other way), the goal of the price manipulator can either be to raise or to lower the market price. Price increases are accomplished by *demand-side manipulation*, making buyers enter the market and thus driving the price. Similarly, *supply-side manipulation* affects the number of sellers (and short-sellers), lowering the price of the asset.

There are several techniques for both demand-side and supply-side price manipulation. They either include the manipulation-, or use, of various information

channels or special techniques for placing orders. All of these methods aim at giving the impression to the market that the price of the instrument should be something not equal to the currently prevailing.

In other cases, the manipulator will want to affect the turnover rate for a particular instrument. For example, when the turnover rises, the effects of the original manipulative attacks can be magnified. In the case of the so-called *chain-letter rally*, the volume increase is a natural consequence of the manipulation itself. The volume increase can also be a part of a manipulation attempt in the first time, as when the market is *short-squeezed*.

It is also possible to create a perceived turnover rate that is different from the actual market volume. One rationale for this would be to increase the attention given to the instrument that one tries to manipulate, hence making additional actors enter the market, possibly magnifying the manipulative effects. There may also be tax reasons behind an apparent attempt to raise the perceived turnover volume. So-called *wash sales*, e.g., where positions are regrouped in order to gain tax advantages, have the side effect of raising the noted turnover for the instrument involved.

Sometimes the objective of the manipulation is to hide the identity of the actor behind the volume created, rather than to affect the real or perceived volume itself. This can be the case when someone is trying to manipulate prices and when the secrecy of the identity of the manipulator is crucial for the manipulation to succeed (such as in a take-over situation). Anonymity can be obtained by letting someone else trade on one's own account or by more complicated chains of trade, that appear to affect other market actors than oneself. It is also possible to hide trading information from the public after a major deal.

Further, there are some price manipulative techniques that may be used by brokers, including churning and burning.

5.1.2. Insider trading

In the case of insider trading, the reasons for the conduct are more straightforward than in the case of price manipulation. Put simply, insider trading is about having access to information about the market that the rest of the public is not aware of. Hence, the market price of the security is somehow "wrong", and before the market gets around to correct this, the insider trader can prepare him- or herself with a position that will become lucrative when the price correction finally takes place.

One category of market participants that might partake in insider trading is of course the company insiders themselves. These include the upper managers or others with a good insight into the doings of a particular listed company. The insider trading in this case takes place when such an insider buys or sells shares prior to the release of some kind of price-sensitive information announcement about the company.

However, the company insiders are not the only ones that can have access to insider information. Brokers can also, in light of their special position giving insight into the deals of their individual clients, find themselves in an inside position. This is similar to the case of the broker-specific manipulation techniques described above.

It is also possible for insiders to manipulate the market prices by the mere power of their position. Since the market knows that they are insiders, attention will be given to their behaviour in the market place. Information about their trades becomes public information by the use of insider lists, etc. If such an insider wishes to affect the market one way or the other, all he needs to do is to let his actions in the market reflect the belief he wants to induce into the market. There has, however, been a discussion on whether or not such conduct can be a sustainable manner to affect prices (since an insider who repeatedly does this in the end is "seen through" and loses his credibility). For a more thorough discussion on these matters, please see the article by John and Narayanan[11] or the one by Benabou and Laroque[12].

5.2. Surveillance objects and goals

Above, we have discussed in what ways market inefficiency and unfairness in the financial markets can lead to economic damage. As a general rule, all actions aiming at misleadingly making the market appear differently than it should when in an effective and fair setting (by actually trading or by simply giving out offerings to buy or sell) are harmful to a given market. It is not, however, completely obvious where one should draw the line between legal and illegal conduct. For example, a person that enters an illiquid market can aggressively buy more shares in order to increase the apparent liquidity, thus attracting more liquidity to the market. In the end, the problem boils down to define where simply being a buyer in a market turns into trying to affect prices in the same.

On top of this, if the surveillance process is to be fruitful, the intent of the market actor has to be proven. Since such things as risk-taking, speculating and information trading are perfectly allowed in a market, it is often quite difficult to pinpoint an event as being the result of a specific illegal intent. If the market reasonably could anticipate the behaviour, then the situation becomes even more complicated. For example, when two companies are engaged in a cross holding, it is not difficult to foresee that that the one company has an interest in that the price of the other's shares are kept high. Whether this means that certain price manipulative behaviour is already discounted in the observed market prices or not is however not obvious.[13]

Thus, a complete surveillance system does not only have to identify possible breaching of the rules associated with the market place. This identification of

[11] John – Narayanan (2001)
[12] Benaboud – Laroque (1992)
[13] Aitken – Berry (1991)

insider trading and/or market manipulation is merely the first step to the ultimate goal of at the best interrupting the illegal activity, or at least to single out the wrongdoers and putting them to justice. To do this, it is also necessary to effectively secure evidence, and to alert the relevant authorities to carry through with the legal proceedings. Exactly what the appropriate proceedings are depends on the kind of conduct. It might be a trading member who has breached the trading rules, a securities regulator who has not followed the legislation or a company not obeying the listing rules.

In order to effectively treat the surveillance alerts triggered by the system, these alerts have to be of good quality in the first place. The difficult part here is to ensure that most of the illegal actions taken on the exchange are detected, at the same time as the number of false alarms are kept to a minimum, as every alarm has to be investigated one way or another. If the expected costs of surveillance are in excess of the average economic losses to unlawful conduct in the markets, there is no economic incentive to actually carry through with the surveillance program. Especially the number of false alarms can potentially be large, because of the stochastic nature of the trading activity itself. The key is to set the alarm levels so that they, on average, give a limited number of alerts per day, concentrating on the really significant ones by filtering them out of the general background noise.

5.2.1. Detection process

Generalities

In order to detect suspicious market behaviour, one has to carefully monitor the market and its actors, trying to single out suspicious actions based upon certain patterns that empirically have been observed in association with such unlawful conduct in the past. With respect to the massive amounts of market information available from a typical market place, the monitoring has to be very selective and efficient. In the case of insider trading, patterns are perhaps easier to pinpoint than in that of market manipulation – it is sufficient to monitor the trading of the legally-defined insiders, and possibly associated market participants that effectively affect the beneficial ownership of these insiders.

For the same reasons, the proofs are also easier to gather for insider trading than they are in the case of market manipulation. In the latter case, it is the information of the asset itself that is manipulated to be incorrect. Since prices, volumes, etc. are volatile by nature, and since complicated trading patterns may (and should) occur in a given market, the actual conduct, as well as the intent behind manipulative behaviour, is difficult to prove.

The surveillance system presented herein does not, however, detect insider trading. As it is a more challenging task, and because of the limited amount of time devoted to this project, the scope has been limited to market manipulation, leaving insider trading for a possible future functionality expansion.

So, how does one practically detect price manipulative behaviour, given sufficiently detailed market data? There are several basic strategies that may be used. First of all, one has to define certain *metrics*, deciding what to measure in the market. These metrics can be the observed market prices, individual orders, turnover volumes, etc. They thereafter need to somehow be compared to a number of *benchmarks*, in some sense defining normal or allowable market patterns.

The comparison of the metrics with the benchmarks can be carried out in one of two basic ways. A practical implementation is to have a number of predefined such comparison tests, which the user of the surveillance system has the power to fine-tune by the use of certain *configuration parameters*. This way, the user is up-and-running after a relatively short period of introductory time. On the other hand, real-life markets are often complex, intertwined and continuously changing. Therefore, it may be more effective for the surveillance system to offer the available metrics and benchmarks as configurable *entities*, that can be combined in any way or pattern to produce the user's own benchmark tests. This implementation offers more flexibility and diversity, but is more demanding on the account of the user.

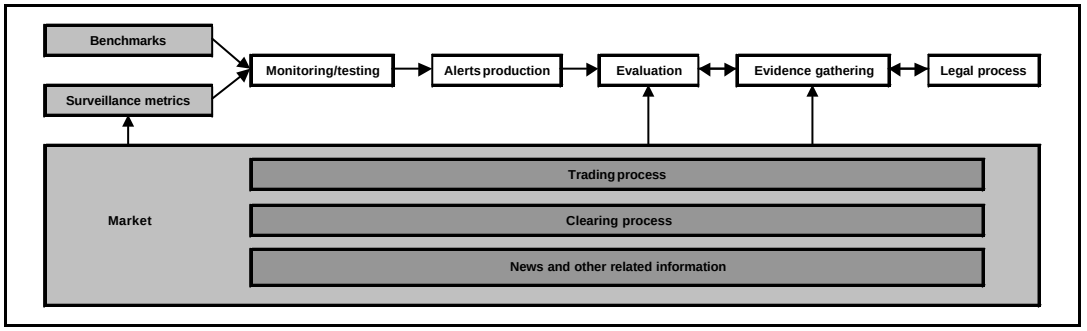


Figure 2 - Overview of the surveillance process

In the next step, the surveillance system developer has to decide whether the whole surveillance process should be contained in one single platform or if it should be split up into several modules. Analogously to the case with predefined or custom-made benchmark test, a single platform is easier to use. On the other hand, a module-based approach is a more flexible solution. As an example, the monitoring of market manipulation and insider trading, respectively, can be separated or integrated. Also, there can be a certain vertical integration of the parts of the process from monitoring to evidence securing. Alternatively, these parts can be divided into distinct modules, reflecting the organisation of the actual surveillance process more accurately. Naturally, there is a need for an information exchange between different such modules. The degree to which they are integrated can however vary.

As can be seen from Figure 2 above, the data inputs to the system should ideally not only include the raw trading activity data. There are other data sources that

may be of interest as well when deciding whether an observed market event should be considered suspicious or just business-as-usual trading activity. For example, such external data feeds might include the clearing process[14] and a news feed[15]. It is also easy to see additional funcionality in a surveillance system that would be of value. For example, an information bridge between different market places would make it easier to detect price manipulation relating to several markets at once, such as stock price manipulation in order to affect the prices of associated stock options, perhaps on a different exchange. Another example would be the possibility to play back the market activity during a certain time period, in order to observe what really happened in "real-time". Without this feature, the amounts of information to manually investigate when an alert has been triggered might be massive.

Typical symptoms of insider trading and market manipulation

Every action taken in the market place has its consequences, so also unlawful actions. More precisely, different kinds of unlawful conduct in the marketplace lead to different typical patterns in terms of the available metrics. Figure 3 below shows some of these patterns, among these notably the ones actually used in the pilot implementation of the surveillance client application. It thus depicts various ways to manipulate the market and insider trade, and the consequences (symptoms) these actions have on the market. The symptoms themselves are measurable by the use of carefully selected metrics.

One example is trading parties that trade non-anonymously with one another. This means that they strike a deal outside of the trading system, after which they place the orders simultaneously in the system (see under *matched orders* in the Figure 3. This way, the trade never appears on the trading screens before it is matched. One way to detect these trades is by detecting trades with orders that have been in the orderbook for a very short time (or not at all) before the trade was accomplished. Another is to use the fact that these deals often take place outside of the spread for the traded instrument, because this is a way to obtain a quick trade at a given price level. A better way of detecting matched orders is to use both of these symptoms in combination. Observe, however, that wash sales share both of these symptoms with matched orders, making these unlawful conducts difficult to contrast by only considering these two symptoms.

Another example is the symptom of a sudden spread[16] change, which can have several explanations. One possibility is that the market expects some important information to be announced shortly, such as an earnings announcement. When the uncertainty increases, the spread widens as a consequence. Opposedly, the reason can also be that an inside party has gotten information not publicly

[14] The clearing process can, e.g., be used for the consideration of cancelled orders, that add to the percieved volatility but that do not actually clear in the end. Also, on some trading systems, much of the historical information lies within the clearing process.
[15] Increased volatility or volume is not strange in the event of a major news event concerning a company.
[16] The spread is the difference between the best bid- and ask prices observable in the market.

known, and thus moved the outstanding orders further away from the best bid/offer spread.

When a symptom has been observed, it is up to the surveillance staff to try to figure out what has actually happened – has somebody done something unallowed – and in that case, what? This process can be very difficult, especially if the trading situation is complex. The only way to succeed is by experience. It should be possible to solve this by the use of an AI[17] implementation. In this thesis, this method is not investigated further. Instead, it is assumed that there are competent surveillance staff ready to interpret the various symptom alerts delivered by the system.

[17] AI, Artificial Intelligence

Unallowed action	Large volumes	Price movements	Orders far from BBO	Anormal orderbook	BBO crossing	Immediately traded orders	Spread changes	Beneficial ownership not changed	No relevant news	Comment
Orderbook-coupled techniques										
Price-affecting techniques										
Cornering	✓	✓							✓	
Highest bidder		✓	✓						✓	*One player consequently bidder outside spread*
Transactions at progressively higher prices	✓	✓		✓					✓	*One player fast follower in price raise sequence*
Pump and dump	✓	✓	✓	✓					✓	*The price moves up or down, then someone puts a large order to take advantage of the move.*
Ramping		✓	✓	✓						*Large orders are placed near closing, outside of the spread*
Window-dressing		✓	✓	✓						*Large orders are placed near closing, outside of the spread*
Volume-affecting techniques										
Churning	✓		✓					✓		*A player has both buy- and sell orders. Large trading with no beneficial ownership change.*
Passing the parcel	✓	✓	✓					✓		*A player has both buy- and sell orders. Large trading with no beneficial ownership change.*
Pools	✓							✓		*Large trading with no beneficial ownership change*
Short squeeze	✓	✓	✓	✓						*Large volumes*
Wash sale	✓		✓	✓	✓	✓		✓		*Large volume rders that never reach the order book between the same two parties. No beneficial ownership change.*
Anonymity-affecting techniques										
Matched orders			✓		✓	✓				*Large volume orders that never reach the order book between the same two parties. BBO crossing.*
News-coupled techniques										
Price-affecting techniques										
Bait-and-switch		✓							✓	*Stock price movements*
Hype and dump	✓	✓							✓	*Stock price movements in one direction, then a rapid movement*
Failure to disclose information										
Warehousing										
Nominee accounts										
Volume-affecting techniques										
Chain-letter rally	✓	✓							✓	*increased volatility and volume*
Insider trading										
Brokers										
Front-running							✓			
Inside market information							✓		✓	*Non-motivated volume or volatility*
Piggy-backing										*Correlation between the broker's and the customer's portfolios*
Company insiders										
Scalping	✓	✓					✓		✓	*Price movements before the release of the news. The spread widens prior to the announcement if anticipated.*
Classic insider trading	✓	✓					✓		✓	*Price movements before the release of the news. The spread widens prior to the announcement if anticipated.*

Figure 3 - Mapping unallowed actions to symptoms

Among the listed symtoms in Figure 3, some may need some additional explanation:

BBO crossing means that an instrument is bought at an unnecessarily expensive price, or sold unnecessarily cheap. This means that there was a seller in the market willing to sell to a lower price than that of the one used for the transaction, and analogously in the latter case. In the efficient market, this should never occur.

In order to observe the *anormal orderbook* symptom one first needs to define what is meant by "anormal". Later on in the thesis, this issue will be dealt with more in-depth. In the meantime, we can conclude that there is not a one-to-one mapping between this symptom and the BBO crossing symptom.

However, the picture is still slightly more complicated than. To be able to detect suspicious behaviour, it is not only necessary to monitor trading activities (measured by the available metrics) with respect to individual instruments and trading parties. One also has to consider the propagation of these metrics over time. In order to know what benchmark to compare the metric to, it is necessary to study the time process of the metric, in order to obtain some kind of estimation of the "normal" state of the process. There are several aspects to this.

Different metrics need to be treated differently. In some cases, such as when the time process can be approximated with a scaled white noise, a good method is to calculate the historical mean and variance, and then perform a simple hypothesis test to see if a value is to be considered to be normal or not. This white noise approximation can, e.g., be used for the return process in liquid stock exchange markets[18]. Other possibly usable metrics, such as the orderbook, cannot be treated in this simple manner. Instead, some more elaborate methods must be considered. One such method is a neural network approach. Another one is, as we shall see later on, to use a time series model such as ARMA and GARCH[19].

When one considers the propagation of the metrics over time, there might also be seasonal variations in the observations. In models such as the ARMA, this can be incorporated quite easily. For other models, the question of seasonality needs to be dealt with differently.

On top of this, the metrics themselves are often not clearly observable in the market data. For example, when considering the price process of an instrument, how does one define the price? One possibility is to use the last trade price. This definition is quite straight-forward to use, but does not take into consideration the fact that some trades take place on the bid side of the market, while others take place on the ask side. This oscillation gives rise to a certain descrepancy

[18] Gouriéroux (1997)
[19] See below for more on such time series models.

between consequtive trades without the real price of the asset being changed. This so-called *bid-ask bounce* adds to the perceived volatility of the price process.[20]

If, on the other hand, one chooses to define the price as a function of the best bid- and ask offers in the orderbook, the bid-ask bounce problem vanishes. However, other problems take its place:

Liquidity is defined as the possibility to convert cash to and from a security without affecting the prevailing price level.[21] For small, illiquid stocks, it might be difficult to find the price simply by looking at the orderbook. There might not even be both a bid- and an as price. Or, these might not change over time to reflect the actual change in market valuation of the asset.

On top of this, when an asset is very illiquid or not traded at all, it is also difficult to define a volatility measure. It is therefore also difficult to obtain a good overall historical understanding of what is to be considered "normal" return – in this case it might help to make use of so-called *asset groups*, grouped together by historical similarities. By observering metrics in light of some kind of average or aggregation over several instruments, a better understanding can be obtained.

At the same time, one needs to consider the fact that illiquid instruments are more prone to market manipulation, since per definition less cash is needed to affect the price of the instrument.

These issues will be dealt with on a more mathematical level below. For completeness, two of the existing automatic surveillance system solutions are presented next.

5.2.2. Other automatic surveillance systems

The typical automatic surveillance system is not a generic product. Instead, it is custom-made for a certain exchange system or a certain exchange. In this section, Peter Bergenwald and me briefly present one such custom-made-, and one more general surveillance system.

LM

LM (Local Modules[22]) is an integrated part of the CLICK system that, among other things, acts as a primitive market surveillance agent. It is CLICK-specific, and therefore only used on exchanges that run this exchange system (even though not all such exchanges have incorporated LM into their CLICK system). LM checks for the following events, and produces an alert if one of them is detected:

- **Internal trade**: a member that trades with himself.

[20] "Discovery" (2001)
[21] finance.yahoo.com
[22] The name "Local Modules" does not seem to have a rational explanation apart from historical reasons.

- **Large trade:** detects trades that are larger than what is considered "normal". The definition of this event uses the normal-state (historical) variance.
- **Trade price movements:** a member that sells and buys in steps, to move the trade price in a certain direction (without the net ownership being altered in the end).
- **Unintended BBO crossing:** deals that occur outside of the spread. This means that the buyer will pay more than the seller who will sell for the lowest price requires (see discussion above).

A surveillance system as simple as this one clearly has its limitations – one of the reasons why our study was conducted in the first place. However, it represents a possibility to at least carry out rudimentary market surveillance, although its tight dependancy on the CLICK system makes it non-portable to other exchange systems.

SMARTS

SMARTS[23] (Securities Markets Automated Research Trading Surveillance) is a generic surveillance system designed to run on various platforms, and in collaboration with various kinds of financial markets. It was originally developed by a group of researchers at the University of Sydney, but today it is developed and marketed by Computershare Limited. The system is made up of several different modules, each providing its own functionality. Among these are modules for benchmark visualisation, alerts triggering and post-trigger analysis, statistical reporting, real-time statistical monitoring and a market replay function.

The system was designed more like a general tool for financial market surveillance than with a specific exchange, or exchange system, in mind. Therefore, it is very versatile, at the same time as its complexity probably does demand some investments in learning time. When properly used, however, it is relatively powerful. Among other things, it features an own alerts definition language, allowing the user to define and use its own benchmarks and alerts through a collection of predefined metrics. This solution allows the user on an individual exchange to interactively gain an understanding of the market prerequisites prevailing at that particular market, and fine-tune alerts over time so that they are triggered not too often, nor too seldom, carrying relevant information about illegal market actions.

Today, an installation of the SMARTS system is used, for example, on the Oslo stock exchange.

5.3. Surveillance functions

After this more general overview of the field of market surveillance, and of the generic surveillance process, we now turn to study the theories developed during

[23] www.smarts.com.au

the work with this thesis. As much of the surveillance process can be described in terms of algorithms, an overview of the main surveillance algorithm in the client application is presented firstly. After this, the focus turns to how the actual detection of unlawful conduct is carried out in the system.

5.3.1. Overview

The main surveillance algorithm of the client application processes the incoming market data in two main loops, each operating on a different time scale. Firstly, the information is processed continuously, in real-time, over the so-called *aggregation period*. At the end of each such period, it is processed (aggregated) and dumped to the historical database. Each such aggregation thus takes place in discrete time, where each point in time is one aggregation period from the next in continuous time. We call the real-time loop the *continuous loop*, and the other one, wrapping the continuous loop, the *discrete loop*.

During both the continuous- and the discrete time loops, the market information is characterised into various *metrics*, each representing a certain characteristics of an individual instrument. Some metrics (*discreet* metrics) describe processes over time. They are collected over the aggregation period, and subsequently dumped into the historical database as some kind of sum or average value at each aggregation. Other metrics (*continuous* metrics) are associated with real-time individual events, such as a single order arriving into the system. They are not aggregated into the database. Rather, they take part in the associated discreet metric (where applicable), hence indirectly saved in the database as a part of the aggregated information.

There are a number of different ways to detect market manipulation in its different forms (please see 5.1 Types of unlawful conduct above for more on these forms). Each of the methods used carries out a comparison of one or several metrics to some statistical model, or *benchmark*. Ideally, such a comparison should be carried out for each traded instrument on the exchange. However, for different reasons[24], instruments are bundled into *instrument groups*, where all instruments belonging to the same group share the same benchmarks. Therefore, each comparison takes place for every instrument in the considered instrument group. The parameters of the statistical model are calculated using aggregated, historical data collected from the database. Then an alert is triggered for the considered instrument if the metric is "too unusual" by the definition of the benchmark used. As the model takes into consideration the historical data for all the instruments in the same instrument group, one single instrument can be compared to a group of other instruments. The historical data used from the database is constituted of historical values for different metrics. Thus, by carefully designing ways to fuse metric values associated with different instruments, one can end up with a single time series

[24] Please see the Configuration package section of 6.2.3 Client application for more information on instrument groups.

of metric values, which can be treated with the chosen statistical model in order to create a common benchmark for the instrument group.

The default statistical benchmark model is a simple confidence interval approach, in which the average value and the variance of the metric over time is observed under a time-constant normality assumption. Then it becomes possible to use the historical variance to measure how far from the historical mean the received metric value is, thereby deciding whether the current value of the metric is within the range of "normal" values or not.

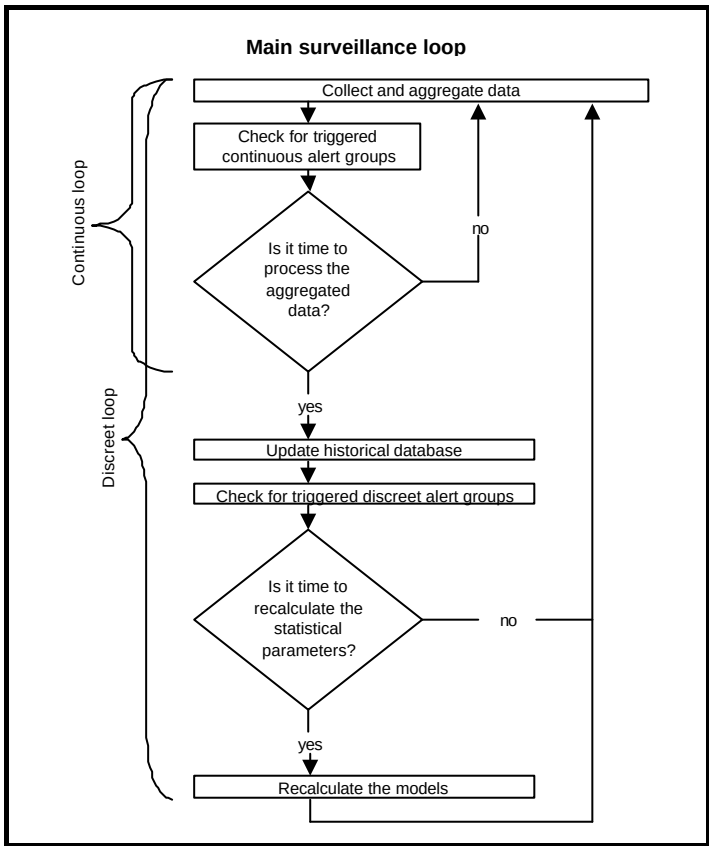


Figure 4 - Main surveillance loop

The system also allows for the definition and add-on of more elaborate models for the detection of different types of alerts. When such a model is used, the comparison of the metric to the historical data for the instrument group is carried out in a model-specific way, using model-specific parameters calculated from the historical data. For example, for one of the alert types, a GARCH(1,1) method is defined and used to describe the return metric process over time.

The flow chart in Figure 4 depicts the surveillance loop more in detail. The top half represents the continuous process, where the surveillance takes place in real-time. This process is punctuated at each aggregation time by the discreet surveillance process.

At each lap of the continuous process, one more item of market data is collected, followed by a check for triggered continuous alerts[25]. The type of test carried out in each loop depends on the type of market data collected. For the instrument to which this data relates, any configured alerts for this particular instrument are checked by comparing the discreet metric associated with this market data type with the statistical model for this metric derived from the

[25] Actually, alerts are bundled together in *alert groups*. Please see the section the Configuration package section of 6.2.3 Client application for more information.

historical aggregated data. For all alerts defined for an instrument, an alert check is performed for each piece of market data corresponding to the alert. The continuous surveillance process finally keeps track of all the data collected, producing aggregated measures of the different metrics as the data arrives.

The discreet-time surveillance process starts each turn by carrying out alert checks for all the defined discreet alerts defined in the surveillance system. Similarly to the continuous case, these alert checks are performed by the comparison of a metric to the statistical model of this metric. However, in the discreet case, the metric used for the comparison is not associated with a single piece of market data, but the latest value for the aggregated metric. Lastly, the discreet loop updates the historical database with the latest values for the aggregated metrics produced by the continuous process

5.3.2. section Metrics

As described above, there are two basic types of metrics used in the surveillance system; *discreet* and *continuous* metrics.

Discreet metrics are measured over a whole aggregation period in a simplified way, either as some kind of average or as an aggregated value over the period. Consequently, they are stored in the database to be used to standardise the current surveillance data. These metrics include the volume-, return-, spread- and orderbook metrics.

Continuous metrics, on the other hand, are measured continuously (as soon as the event to which they are tied take place). They are not simplified, but rather used as they are. They are not stored in the database either, since they are typically standardised by the use of some special rule or by the use of discreet metrics. They include the orders- and trades metrics (representing the characteristics of individual orders and trades that enter the exchange system).

Below follows a more detailed discussion of the different available metrics, starting with the discreet ones.

Discreet metrics

Volume

Historical volume data is interesting for standardising the current volume data in the system. The surveillance system measures both the volumes ordered and the ones traded for the active instruments.

Volume data is summed over the aggregation period, and is subsequently written to the database at the end of the aggregation period. Instruments are bundled into instrument classes[26], sharing the same surveillance settings. For different instrument classes, the aggregation period may differ. Hence, at the time of aggregation, data is written for all instruments belonging to the aggregated

[26] Again, see the section on the Configuration package section of 6.2.3 Client application for more information.

instrument class. However, the data for each instrument is saved distinctly from the data of other instruments belonging to the same instrument group or –class.

In addition, the volume data is split up along two further dimensions, the final trading client and exchange customer[27]. This division is done to be able to check such things as the correlation between the orders put by a certain customer and the orders put by its individual clients, and the real change in beneficial ownership during a period of trade in a certain paper.

The aggregate volume for each asset is thus calculated for every time period, and stored into two different database structures:

- A list of aggregated order- and trade volumes per instrument, client, customer and time period. Both the net and gross volumes are saved. At the time of alert checking, this list is used to check the "normal" pattern for the volume of trades and orders, both on a market level and on the level of individual market participants.
- A list of aggregated trade volumes per instrument, client and time period. Both the net and gross volume is saved. This list is only used for the calculation of the actual change in beneficial ownerships as compared to the gross trading activity in a certain paper.

As an alternative to using the "normal" volume patterns for ordering and trading, one can standardise ordering- and trading activity with the total number of outstanding stocks[28] for the considered instrument. This feature is currently not implemented in the pilot application, but it would mean that the number of outstanding stocks should be stored together with other instrument-specific data. For each instrument, the data should not be updated until the actual number of outstanding stocks is altered.

Return

The return process is interesting for the standardisation of return-related events. In this, price data could be of alternative interest. However, market share price data is typically not stable over time – i.e., the moving average of the price of a share tends to change over time. On the other hand, with a return transformation, the data typically becomes stationary and therefore has more usable statistical characteristics[29].

As is the case for the volume data metric, the discreet return data metric is measured over a certain aggregation period. However, it is only the final price of the instrument over the period that is of interest. Hence, no actual aggregation is produced during the aggregation period.

[27] The exchange's customer is called a *member* in the case of a CLICK system.
[28] I.e., the total number of stocks that a company has issued to the market, and hence are tradable.
[29] Séries Chronologiques (2000). A more general way to make a time series more stationary is to repeatedly differentiate the series until stationarity is reached.

In order to define a return measure for an instrument, one first has to define the price of the instrument at a given point in time. Two possible price definitions are offered by the configuration of the system:

- Average bid-ask price: $P_t = \frac{(ask_t + bid_t)}{2}$
- Latest Trade Price: $P_t = LTP$

Other definitions, including best bid and best ask, are possible but not implemented. The latter trade definition provides a price "close" to the actual market activity, in that it takes into consideration the trades actually taking place. This definition might be useful when surveilling instruments with very low liquidity, where the perceived market spread is sometimes not very significant for the actual price process of the instrument. For example, the downwards spread might be significantly larger when not many actors are interested in buying the security. Also, in the case of such a price definition being used, it is relatively easy to affect the instrument price by simply putting an order that changes the spread, or by withdrawing an order defining the price. On the other hand, for illiquid instruments it is relatively easy for one individual actor to affect the price by a single trade, thus affecting the price if a latest trade definition is used.

One extra advantage can be argued for the average bid-ask price definition. Namely, it does not lead to the excess volatility gained from the latest trade definition, a volatility that is due to the fact that some transactions take place on the bid side while others take place on the ask side of the orderbook. At the same time, the average bid-ask price is more abstract, relating primarily to the orderbook and not the actual trading activity.

Given the above definitions of price, it is now possible to define the return. In our case, this boils down to using one of the following available definitions:

1. Linear return: $r_t = \frac{P_{t+1} - P_t}{P_t}$.
2. Logarithmic return: $r_t = \ln(\frac{P_{t+1}}{P_t})$.

The linear return might be the most intuitive measure to use. However, the logarithmic return is most often used in practice, because it has the attractive feature of accumulatable rentability. If one adds subsequent returns the sum is equal to the return over the whole period considered:

$$\ln(\frac{P_{t+j}}{P_t}) = \sum_{k=0}^{j} \ln(\frac{P_{t+k+1}}{P_{t+k}}).$$

Also, the logarithmic return measure assumes values in the interval (-∞, ∞), just like the linear return does. When considering small returns, the two measures

yield approximately the same values. Depending on the price definition used, different historical data is of interest for calculating the return metric. These data are collected either from the best bid- and ask prices of the historical orderbook data, or from a list of historic trade prices that is stored for each instrument. In this latter list, each price is that of the trade that occurred last in the given aggregation period for that particular instrument. In order to be able to change the price definition over time, all data is saved regardless of the current price definition.

Spread

The spread process is measured in three ways, in order to capture market manipulation both on the bid- and the ask side:

1) relative bid-side spread: $$RS_{b,t} = \frac{(P_t - bid_t)}{P_t}$$

2) relative ask-side spread: $$RS_{a,t} = \frac{(ask_t - P_t)}{P_t}$$

3) relative total spread: $$RS_t = \frac{(ask_t - bid_t)}{P_t}$$

Obviously, all spread measures depend on the definition of the price of the instrument. When using the average bid-ask price, the relative bid- and ask-side spreads are per definition equal – in this case the only spread measure interesting for alert checking is the relative total spread. The values used are the final ones in each aggregation period.

Unlike the volume- and return metrics, the historic spread data is not stored explicitly as one or several lists. Instead, it is implicitly derived from the historical aggregated orderbook (as this includes the best ask- and best bid price – see below).

Orderbook

For each instrument, an aggregated (simplified) orderbook is saved for each aggregation period. When calculating and storing this metric, the data is simplified over several dimensions:

- The actual orderbook saved only includes the total aggregated volume at each price level, not the individual orders or quotes that make up this volume.
- Only a number of price depth levels are considered, discarding bid- and ask orderbook entries at prices that are more than the specified number of prices away from the BBO. This depth number is configurable. Each price level represents one available price at which a price is specified. For liquid assets, each price level should correspond to one tick mark, at least near the best bid/ask off levels. For less liquid ones, consecutive price levels may very well be several tick marks apart, as the orderbook is less dense.

- The orderbook that is saved in each aggregation is an aggregated snapshot of the actual orderbook at that point in time. Hence, no attention is given to any abnormal events taking place in the orderbook between two aggregations, not affecting the orderbook at the end of the aggregation period.

From the orderbook, the average ask/bid-definition price and spread metrics may be derived. Hence, the saved historical orderbook data is used for the calculation of these metrics as well.

Continuous metrics

Orders and trades

Orders and trades entering the system are measured by continuous metrics. The transactions are considered as they are, and are not stored in an aggregated form *per se*, even if they can be compared with aggregated historical data (discreet metrics) when checking for alerts.

The continuous metrics measure such things as the price of an order or the time an order resides in the orderbook before it is traded.

5.3.3. Alert types

The purpose of the surveillance system is to trigger alerts when possibly non-allowed market behaviour is detected for any one instrument. This is done in two ways; continuous metrics are evaluated as soon as they can be calculated, and discreet metrics are evaluated once every aggregation period.

Each alert type relates to one of the described metrics. Alerts are organised into alert groups. The purpose of this is to offer a way to define alerts whose definition build upon several comparisons between metrics and the corresponding benchmarks. By combining several alerts into alert groups, one can configure more complex alerts, built from the inspection of several of the metric processes in combination. Since a given illegal market conduct has its own "fingerprint" in terms of metric patterns (as is further explained in 5.2.1 Detection process), the surveillance user can define one alert group for every non-allowed action monitored. The fingerprint is built from individual metrics, and the whole alert group is triggered when all of the individual metrics display strange enough values at the same time.

Some alert types are quite straightforwardly tied to a metric, while others have a more complicated relationship to their underlying metrics. Every alert is associated with a *threshold level* variable, and every alert group has a *sensitivity* setting.

The evaluation itself depends on the types, and the number, of alerts in the alert group considered. The general rule is that if there are several alerts in the alert group, the defined threshold level value of the individual alerts is used for the evaluation of the alert group. The level is an absolute value for the metric

considered. In this case, if each individual alert threshold level is surpassed by the corresponding metric for the examined instrument, the alert group is triggered, and is thereafter made visible to the system user. This case is quite straightforward, and the actual result of the evaluation of the alert group depends on what kind of alerts that the group contains.

It gets more interesting when there is only one alert in the alert group. In this case, the defined sensitivity parameter of the alert group is used for the evaluation. This parameter measures the number of alerts wanted by the surveillance user during each time period. Hence, it is related to the statistical distribution of the metric considered. The way that the comparison between the metric and the historically founded statistical model is done depends on the statistical model used for the specific metric, and we will return to this issue shortly.

Ideally, it should be possible to tie the trigger level of several-alert alert groups to the statistical metric distributions as well. However, this requires a more thorough statistical treatment, and the present scope cannot cover this aspect.

In the following, each alert type is described, along with its relation to the underlying metric.

Alerts relating to discreet metrics

Volume alerts

There are in fact three different volume alerts. They are all based on different metrics and are standardised with historical volume data:

- **Total trade volume** – this alert is triggered if the total traded volume in the instrument is unnaturally high during an aggregation period. It is investigated by comparing the discreet trade volume metric to historic aggregated trade volume data.
- **Total order volume** – this alert is basically the same as the trade volume alert, except for the fact that it is the discreet order volume metric that is used in combination with historical aggregated order volume data.
- **Individual order volume** – here, the metric used is also the order volume metric, but this time measuring the individual order volume for each client and instrument over the aggregation period (thus, the continuous volume metric). It is standardised with the historic, aggregated such individual order volume data. Thus, the alert is triggered if the total orders of an individual client during one aggregation period have “too large” a volume as compared to the historic volumes for individual orders.

As an alternative standardisation for the volume metric (as opposed to historic volume data), the total amount of stocks outstanding can be used. This means that the alert is triggered whenever the order- or trade volume exceeds some predefined percentage (predefined by the user) of the outstanding shares for

that instrument. However, this feature is not implemented in the current version of the client application.

In the case of the volume alert, defining instrument groups should be useful when the liquidity of the instruments considered is low, enhancing the number of historical statistical observations for the volume and therefore improving the forecasting potential and the accuracy of the alert checking procedure.

Beneficial ownership alert

This alert could be considered a variant of the volume alerts. It should be triggered if trading takes place in significant volumes, affecting the holdings of a single ultimately beneficiary owner, but where the gross trade volumes are very elevated as compared to the net change in ownership for the beneficial owner in question. Clearly, this means that the beneficial owner somehow has traded with himself, for whatever reason. The trades themselves can take place by the effectuation of several clients, each representing the same beneficial owner[30]. Observe that this technique not only captures trading with oneself, but also circulatory trading, given that the trading takes place within one aggregation period and that the circle is fully completed – in other words, that the holdings come back to their original owner in the end.

The investigating of this alert is done by comparing the discreet net trade volume metric per beneficial owner and instrument, divided by its gross counterpart, to the historical, aggregated process of such metric quotes. If the current metric quote is “near enough” to zero, the alert is triggered.

A shortcoming in the current implementation of the client application is that no attention is given to the absolute trading volumes, only to the relative (net/gross). Ideally, smaller trading volumes should not trigger alerts.

Return alert

Similarly to the case of the volume alert, the return alert is produced if the return from an instrument suddenly leaps “too high” above- or “too low” below zero (indirectly indicating that the price also has leaped quickly). The metric used to check for this alert is the return alert, standardised by historical, aggregated return data.

The same type of low-liquidity instrument groups as in the volume alert case should be applicable also in the case of the return alert, given that the instruments in the same group are associated with companies similar enough.

Spread alerts

Just like with the volume alert, the spread alert is in fact three different alerts, each associated with a different variant of the spread metric. They include:

[30] A list of the parties representing each beneficial owner must be pre-configured by the user.

- **Relative bid-side spread alert** – using the discreet relative bid-side spread metric, standardised with historic, aggregated values of this metric, an alert is triggered if the bid-side spread is "too wide" during an aggregation period.
- **Relative ask-side spread alert** – essentially the same as the bid-side equivalent. However, this alert is rather triggered when the ask-side spread is "too wide", indicating suspicious activity on the ask side of the orderbook.
- **Relative total spread alert** – combining the two previous alerts, not taking into consideration the side of the orderbook where the spread is widest.

Possibly, defining instrument groups for less liquid instruments or in less liquid markets should also be usable in the case of abnormal spread detection. The reason is that when the trading in an instrument is scarce, one order can have a decisive influence on the orderbook, even affecting the best bid- or ask price available. Since few new orders arrive each given time period, the observed spread might not always well reflect the real liquidity for the asset. By averaging over several instruments, a better understanding can be gained for the spread process over time.

Correlation between actors' orders alert

This alert type should be triggered when the correlation between the orders placed by a client and the orders placed by the client's broker are "near enough" to unity. This indicates that a piggy-backing[31] activity is probable. The correlation estimator is calculated by the use of the well-known formula

$$r(X,Y) = \frac{C(X,Y)}{D(X)D(Y)}; \qquad C(X,Y) = E(XY) - E(X)E(Y)$$

$$r(X,Y) \in [-1,1].$$

Thus, the alert type is investigated by the correlation between the values of the discreet individual order volume metric for two different market participants. The standardisation takes place with respect to the historical values of this process, as calculated from the saved metric values from the database.

However, the checking for this alert is not currently implemented in the client application. One reason for this is that client/broker log data is not guaranteed to be accurate on the JIWAY platform.

Orderbook alert

This alert type is quite different from the ones described so far. As for, e.g., the volume- and return alerts, they relate directly to a measurable discreet metric. However, the orderbook alert, as well as the underlying metric, is more complicated. The underlying metric used is the simplified orderbook of an instrument, and should be triggered when the current value of this metric is

[31] See above under 5.1 Types of unlawful conduct

significantly “different” from the historical values of the metric, as estimated from aggregated data from the database.

How to define “different” is the core problem here. It is probably impossible to find a general definition of the term “different”, since different types of instruments have different typical orderbooks, depending heavily on the liquidity of the instrument and on the market type. Stocks with lower liquidity (that are more prone to price manipulation) usually have an orderbook that is more volatile. One large order can move up and down in the orderbook as the price of the instrument changes, hence causing large estimated values of the volatility. Also, a large class of orderbooks should be considered “normal”, or at least not abnormal enough to trigger an alert.

As the problem of defining “different” is so difficult, the generic evaluation model described below is not appropriate when checking for the orderbook alert type. Instead, specific models (that are also described below) are needed[32].

In the current implementation of the client application, the checking for this alert type is not possible.

Alerts relating to continuous metrics

Suspicious orders/trades

There are also a couple of alert types relating to continuous metrics. One of them is the suspicious orders/trades alert, which is triggered if an order enters at a price too far outside of the best bid/best ask spread, or if a trade is settled at such a price. To check for this alert, the discreet orderbook metric is used.

When it comes to the standardisation of continuous metrics for the evaluation of suspicious trade alerts, the current implementation of the client application is somewhat limited. Regarding orders, it only allows the user to indicate a minimum distance from the best ask- or best bid price, respectively, as a fraction of the price of the instrument, for an alert to be triggered. For trades, it suffices for the trade to take place outside the spread to trigger the alert. It would be better if these alerts were checked for by the standardisation of the continuous price- and volume metric (measuring the characteristics of the individual order or trade) with historical, discreet metrics in a more intelligent manner. For example, by relating the continuous order volume to the historical aggregated individual order volume, one could eliminate alert triggers for very small, malpriced orders. Also, by standardising with the discreet spread metric, a more realistic view on the magnitude of the price distance to the best bid- or ask price would be gained.

Immediately traded orders

This alert should be triggered when two orders are in the orderbook for only a short period of time before they are traded against each other. However, the

[32] See below for a more detailed discussion on these topics.

current implementation of the client application triggers the alert only when one order is matched without first entering the orderbook at all. It also has the major setback of not checking if both of the matched orders have been immediately traded.

The continuous metric of orderbook time should preferably be standardised with some minimum time value configured by the user. Finally, and in the same way as for the suspicious orders/trades alert, there should be a standardisation with respect to the order sizes, so that orders don't give rise to triggered alerts if they have not a significant volume.

5.4. Benchmarks

As we have seen, alert checking is done by comparing a relevant metric to some kind of benchmark for this metric. The metrics and benchmarks vary depending on the alert type checked for. However, each test is carried out in a similar fashion, aiming at controlling the triggering frequency of each alert. In this section, a general framework is presented for benchmark definitions, along with actual applications.

5.4.1. Time series

The propagation of a measured metric can be seen as a series over time $X(t_i), i \in \{0,1,2,\cdots\}$, where the t_i's represent discreet points in time. It is this time series that is to be compared to the relevant benchmark. In this, the benchmark must be represented with a statistical model of some sort, describing the propagation pattern of X over time.

The goal of the surveillance process is to capture non-normal behavior in the market. In order to accomplish this, it is necessary to know what is to be considered to be "normal". This is where the benchmark comes in. At t_i, the underlying statistical model is used to *forecast* the closest future value of X, i.e. $X(t_{i+1})$. As long as this forecast is of good quality, it can be used as the benchmark for what is to be considered a "normal" propagation of the metric considered. If the actual value of the metric at t_{i+1}, $X(t_{i+1})$ is "too far" from the forecasted value, a suitable alert should be triggered.

Definition: The *innovation* of the process is the variable $Y_{t_p} = X_{t_p} - X^*_{t_p}$, where $X^*_{t_p}$ is some regression on $\{X_{t_i}; i = 0,1,2,\cdots, p-1\}$, forecasting the value of X_{t_p} given all values of X up to (but not including) this point.

Using this definition, the value of X_{t_p} should be considered to be "non-normal" if $\left|Y_{t_p}\right|$ is too large.

In order to calculate the value of the forecasted value $X^*_{t_p}$, there are three major classes of forecasting procedures that can be used[33]:

- *Subjective forecasts* are based on judgment, intuition, know-how or any other type of intelligent use of information external to the actual time process.
- *Multivariate forecasts* are also dependant on external information. However, in this case the analysis is more formalized. Instead of taking into consideration such vague concepts as "market knowledge", one draws information from a broader field than just the time series *per se*. For example, the interest rate propagation can be used as an input variable in the forecasting of certain stock price curves.
- *Univariate forecasts*, finally, represent a more naïve, but simpler way to forecast a time series. Here, only the historical values of the time series itself are used in the actual forecast of its future values.

In the present surveillance process, a combination of univariate- and subjective forecast is used. The surveillance client needs to be fully automated at the same time as it is facing tough, time-critical performance demands. Therefore, a univariate forecast methodology has been judged to be accurate enough for the task at hand – to identify suspicious market behavior. However, a subjective forecast methodology has to be applied in order to judge the severity of the detected behavior, in light of the other information available to the surveillance personnel[34].

In a future version of the surveillance client, multivariate forecast methodologies should, however, be considered. Using a larger percentage of the total amount of available information, more precise alerting can be accomplished. The price to pay for this is more connections to external data sources and a larger calculations workload, leading to longer processing times. Also, the subjective aspect must always be incorporated before taking legal action.

5.4.2. Standard technique

By default, $\{X_i\}$ is assumed to be a *white noise process*[35] with a bias term. Thus, $X_i = m + e_i\big|_{i=0,1,2,\cdots}$, where μ is the (constant) bias term and ε_i is the white noise. This means that the different X_i's are assumed to be independently and normally distributed, with a mean equal to μ and with a constant variance[36]. This situation can also be viewed as if each new measured value for the metric is considered to be a random sample from a set of such values following a continuous normal distribution.

[33] Chatfield (1985)
[34] Here aspects such as the general market situation and the news- and clearing information becomes important.
[35] Anderson (1976)
[36] For more elaborate models, see below.

The normality assumption can often be motivated to some degree using the law of large numbers. No respect is paid to possible changes over time of this distribution. Rather, it is assumed to be completely described by its mean μ and variance σ^2.

These parameters are estimated using the following standard (efficient and expected value consistent) estimators:

$$m^{\bullet} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

$$s^{*} = \frac{1}{N-1}\sum_{i=1}^{N} (X_i - m)^2 .$$

N may be chosen to be any number large enough to give a fair estimation of the distribution. In the current implementation of the surveillance system, N is set via the configuration.

Under the assumption of a white-noise process, we can now forecast the next value of this time series. We know that the regression term $X^{*}_{t_p}$ is equal to μ^*, since the X_{t_i} are independent. Therefore, the best forecast for X_{t_p} is simply μ^*. Using the historical standard deviation σ^*, we can now form a forecast interval for X_{t_p}, on the form:

$$X^{*}_{t_p} = m^{*} \pm \sqrt{1+\frac{1}{N}}\, t^{N-1}_{a/2} \cdot s^{*},$$

where t is the student-t distribution quantile[37].

According to Pankratz[38], this forecast interval can be treated as an approximate confidence interval, under the condition that one uses a sufficient number of historical data points in the estimation of the distribution

Thus we have a way to construct an approximate confidence interval for X_{t_p} for any given value of λ_α. Given this, we are free to make tests of the type

$$H_0: m = X_{t_p},$$

on, e.g., a significance level α. If H_0 turns out not to be rejected, this means that the difference between the mean and the most recently measured value of the metric (i.e. $\left|X_{t_p} - m\right|$) is within the $1-a$ quartile of all values for X_{t_p}, as expected having the information contained in the series $\left\{X_{t_i}; i = 0,1,2,\cdots, p-1\right\}$. Thus, all

[37] We use the student-t distribution to form the interval since the standard deviation is approximated.
[38] Pankratz (1983)

values for which H_0 is rejected can be defined as "too far" away from the expected value to be "normal".

Expressed differently, on average α% of all values for X_{t_p} will be treated as "non-normal", under the assumption that the values are indeed normally distributed and that we use sufficiently many measurement points of historical data. By choosing α in relation to the average amount of measurement points for the considered metric, one can control the overall frequency of "non-normal" metric values, thus controlling the amount of alerts to tend to on, e.g., a daily basis.

In order to choose a suitable value for α, one can use the following relation:

$$\Phi(x) = a \Rightarrow \Phi^{-}(a) = x,$$

where Φ denotes the student t distribution function.

5.4.3. Alternatives

However, there are many situations where it is not very efficient to use the default alert production model. Often, the metric data cannot be properly represented by a simple one-dimensional time series. Also, even if we have one-dimensional data, there might be a more intelligent way to look at the metric data, less naïve than the simple normality assumption under which the standard model works.

In this section, we will look at a number of examples where this is the case. We start with an example of more complex metric data. Thereafter, some examples of more elaborate models for the treatment of a one-dimensional time series are presented.

Regression model

Regarding the orderbook alert, the metric is more complicated than a simple one-dimensional time series. Rather, the metric input is in the form of a simplified orderbook at every point in time. Every orderbook holds information about the number of instruments available to the market (sell or buy) on every distinct price level. An example of the structure of such an orderbook is shown in the following figure:

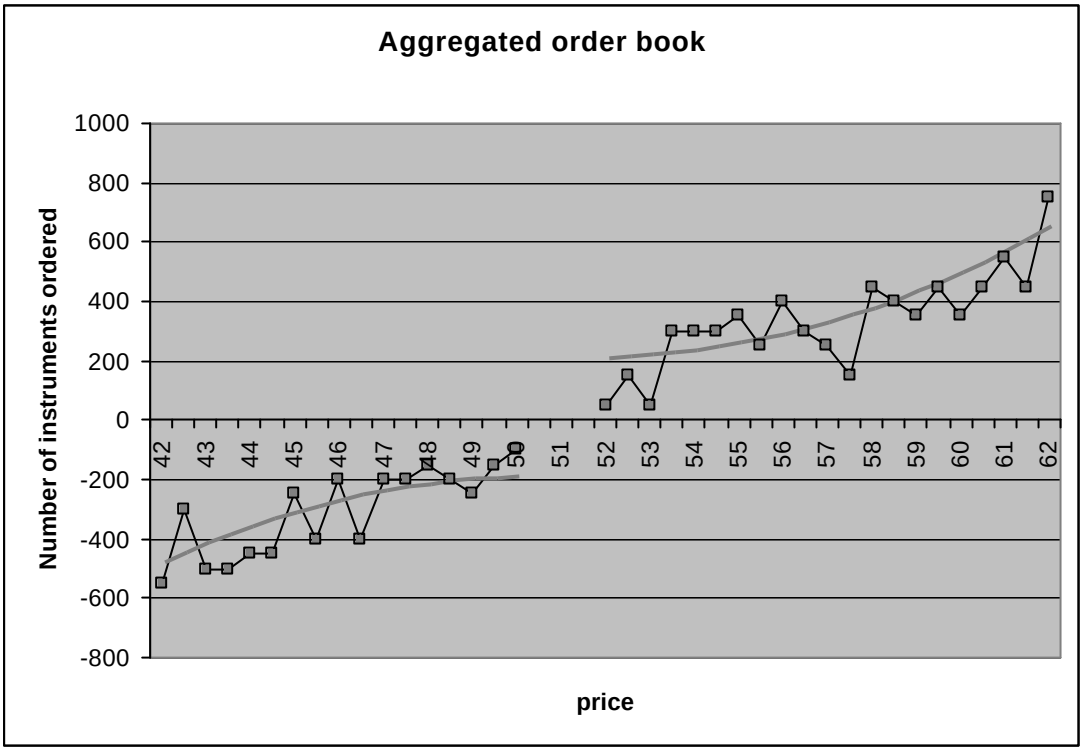


Figure 5 - A sample aggregated orderbook with regression line

The aggregated orderbook for some stock is shown at a particular moment in time. The aggregated order depth is shown for each tick level. The market spread is between 50 and 52. Below 50, market participants are willing to sell instruments. For example, at the 49,5 level, there are a total of 300 instruments offered for sale in the market. On the buy side, the converse apply. Of course sellers cannot be found on the buy side and vice versa, since these orders will be matched right away by the market place engine.

As can be seen, it is often possible to see a pattern in the aggregated orderbook. In this case, a simple parabola function has been fitted to the order depth function with a rough conformance. As a matter of fact, this method can very well be used in order to define the look of a "normal" orderbook. When the market undergoes drastic changes (because of attempted market manipulation or for some other reason), the parameters of the fitted function can be expected to change accordingly.

This way, as long as one has defined a regression with acceptable approximation to the "normal" state of the orderbook, it is possible to investigate the propagation over time of the regression parameters instead of the orderbook itself. What we have done is to construct a model on top of the input metric. Having defined such a model, the model parameters can be treated in a fashion similar to the standard technique described above. Naturally, the standard technique would have to be modified to account for possible non-linearities in the regression model.

Neural network

Another alternative method to treat metric data is to use an AI (Artificial Intelligence) solution, e.g. by the use of a neural network. The orderbook metric should be ideal for this, but one can imagine applying a neural network on any metric data. Specifically, the one-dimensional time series metric of the standard technique example above could very well be treated using such a method.

In order to gain a more concrete picture, consider the orderbook metric. In this case, the neural network is fed with such aggregated orderbook measurements until it has "learnt" how an orderbook normally looks. Then, it can be used as a filter, singling out the orderbooks that do not comply with the "normality" criteria as defined by the parameters of the network function.

5.4.4. Time series models

Box-Jenkins framework

Except for more imaginative methods (such as the regression model and AI approach described above), there are of course other, more elaborate, methods to treat simple univariate time series data such as the returns- or volume metric data. These methods often include manual judgment procedures in order to choose the model to use, how many parameters to incorporate, etc. However, it turns out that in the present surveillance example, the process can be automated without too much lack of relevancy.

In this section, a standard framework for the analysis of time series – the Box-Jenkins framework – will be discussed as an illustrative example of how the design of the automated surveillance process can be carried out. Thereafter, attention will be given to some specific time series models.

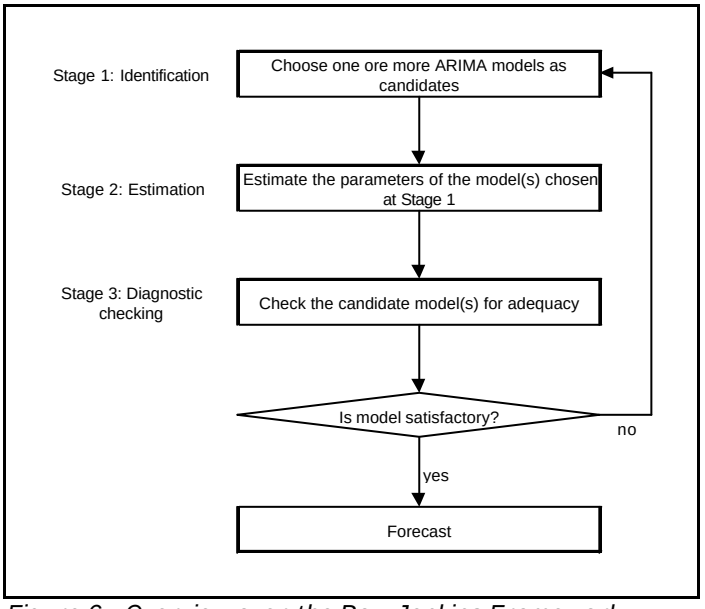


Figure 6 - Overview over the Box-Jenkins Framework

Consider the Box-Jenkins iterative approach to construct a model for a given time series data, adapting the model parameters to fit the historical data, and consequently using the model to make forecasts for the time series.

As can be seen in Figure 6[39], the Box-Jenkins approach uses so-called *ARIMA* (Auto-Regressive Integrated Moving Average) models to model the time series' propagation over time. These models constitute an extension of the classical *ARMA* (Auto-Regressive Moving Average) models. In this context, we will introduce the standard ARMA models on an intuitive level. Thereafter, we will quickly move on to *GARCH* (Generalised Auto-Regressive Conditionally Heteroscedastic) models. The reason for this is that these are more usable for the financial motives studied herein than what is the case for ARMA models.

However, the Box-Jenkins framework depicted above is highly illustrative for the methodology when using not only ARIMA models, but also, e.g., GARCH models. Its main components apply equally well to the modeling using GARCH models.

The ultimate goal for a time series analysis is to be able to forecast future values of a certain series. In our case the forecast deals only with the next value of the measured metric, but it is not uncommon to forecast more distant time series values. In order to do this, the first thing to do is to plot the actual given time series data. Unfortunately, there were (for different reasons) difficulties coupled with obtaining good data from the JIWAY exchange. Therefore, this paper cannot dwell further into this aspect of the process. From now on, it is assumed that the models used in the surveillance process are suggested and iteratively developed by a person with access to satisfactory historical time series data.

The next phase of the Box-Jenkins framework – to estimate the parameters of the model chosen – can be fully automated, given that a suitable model has been chosen in stage 1. Stage 3 can also be automated. Even though we do not deal with this aspect in this thesis, there exist tests to check for the accuracy of the estimated model parameters.

With the results from these tests, the surveillance process operator can modify the underlying time series model in correspondence with the specific needs of every metric. This brings us back to stage 1 in an iterative fashion. As long as the surveillance process is active, the underlying models need to be attended to, since the underlying financial metrics tend to change over time.

5.4.5. ARMA models

Generalities

So far, we have assumed that each time series is independent over time, i.e. $\{X_{t_i}; i \in Z\}$ are each independent variables. However, such an assumption is quite naïve. As a matter of fact, there are several ways that one can model time dependency for a given time series. One of the more classic methods is to model the time series as an ARMA process.

[39] Pankratz (1983)

We start with defining the process X on the discreet time $\{t = 0,1,2,\cdots\}$. Since the observations of our present applications all take place on constant time intervals this limitation does not pose a problem. Thus, each stationary process $\{X_t, t \in Z\}$ that obeys to the equation

$$X_t - j_1 X_{t-1} - \cdots - j_p X_{t-p} = q_0 + e_t - q_1 e_{t-1} - \cdots - q_q e_{t-q}, \quad (1)$$

under certain conditions described below admit to a minimal ARMA(p,q) representation. We can always eliminate the q_0 term by simply defining a new process $Y_t = X_t - E[X_t]$. Therefore, from now on we always assume that $q_0 = 0$. $\{e_t\}$ is a white noise process. (1) can be rewritten according to

$$\Phi(B) X_t = \Theta(B) e_t, \quad (2)$$

where B is the backward operator and the functions $\Phi(B)$ and $\Theta(B)$ are polynomials of order p and q, respectively.

If the time series indeed obeys to (1), and if

- $j_p \neq 0$ and $q_q \neq 0$;
- The roots of $\Phi(z)$ and $\Theta(z)$ are located outside of the unit circle;
- $\Phi(z)$ and $\Theta(z)$ do not share any common roots; and
- $\{e_t; t \in Z\}$ is the above said white noise process, with a variance $s^2 > 0$,

this means that the time series is auto-regressive of order p and has a moving average of order q. This implies that $E[X_t | \{X_{t-1}, \cdots, X_0\}] = E[X_t | \{X_{t-1}, \cdots, X_{t-m}\}]$; $m = \max(p,q)$, i.e. the expected value of the next element in the time series is only dependent on the m previous values of the series.

Forecasting

When applying an ARMA model to a given time series, the first thing to do is to choose the value of the parameters p and q, with guidance of the general look and pattern of the actual investigated time series sample. The next step is to figure out the proper values of the parameters of the ARMA(p,q) model $(j_1, \cdots, j_p, q_0, \cdots, q_q)$. There are several ways to do this estimation as well, including methods based on the auto-covariance function of the series as well as pseudo maximum likelihood methods. Once all the model parameters have been properly estimated, one can predict the expected value of the next value of the time series according to the following formula:

$$X_{t+1} = j_1 X_t + \cdots + j_p X_{t-p+1} + q_0 + q_1 e_t - \cdots - q_q e_{t-q+1} \quad (3)$$

Additionally, since we assume that the $\{e_t\}$ are normally distributed, we can say something about the distribution of X_{t+1}. Before we can specify this distribution, we need to say something about the innovation of the ARMA(p,q) process.

Theorem[40]: Let $\left\{z_j = \frac{1}{l_j}; j = 1, \cdots, p\right\}$ be all the roots of the polynomial $\Psi(z) = 1 - y_1 z - y_2 z^2 - \cdots - y_p z^p$, and let $\left\|z_j = \frac{1}{l_j}\right\| > 1 \forall j \in \{1, \cdots, p\}$. Then there exists a series $\Pi(z) = \sum_{i=0}^{\infty} p_i z^i$ such that $\Psi(z)\Pi(z) = 1$.

Lemma[41]: Under the same conditions, $\Psi(B)\Pi(B) = 1$, where B is the backward operator.

Thus, if $\Phi(z)$ does not have any roots on or inside the unit circle, (2) can be rewritten as $X_t = \Phi(B)^{-1}\Theta(B)e_t = \sum_{i=0}^{\infty} p_i e_{t-i} = \{p_0 = 1\} = e_t + \sum_{1=1}^{\infty} p_i e_{t-i}$. This, in turn, means that the innovation of the process $\{X_t, t \in Z\}$ is nothing but e_t.

To conclude, when we have estimated the ARMA(p,q) model and its parameter values, we can forecast the next value of the investigated time series using (3). This forecasted value has the same distribution as e_t. Since e_t is normally distributed, we know that the error of the forecast is also normally distributed. Finally, this result makes it possible to apply the hypothesis test of the standard technique described above on the forecasted value of X_t, obtained using the ARMA model.

For financial purposes, many processes (such as the price process) are seldom stationary over time. In order to solve this issue, one has to differentiate the process repeatedly until stationarity has been reached. In our case, we simply transform the price process into a returns process. If this process is not stationary, one can differentiate the process until so is the case. However, this last step is not taken in the current pilot implementation.

5.4.6. GARCH models

Generalities

The price process of a certain share, or any other process related to some financial instrument, does however not generally allow itself to be accurately

[40] "Séries Chronologiques" (2000)
[41] ibid

described by an ARMA model. The reason for this is that the volatility of the noise process $\{e_t; t \in Z\}$ is not generally constant. The individual noise terms are indeed normally distributed, but their respective variances $V(e_t)$ are not independent over time. For example, consider the price process of some listed stock. At certain time periods, the volatility tends to be high. Inbetween these periods, the volatility is lower.

At each time, the instantaneous volatility of the price process depends on the volatilities of the past, giving the price process non-linear characteristics. In order to include these properties into the analysis of financial time series, Engle[42] introduced the ARCH (Auto-Regressive Conditionally Heteroscedastic) models in 1982. These models are only concerned with the behavior of the noise process, and are able to describe many financial time series more accurately than other models, such as pure ARMA models (with homoscedastic errors). However, many problems are still not addressed, some of which include[43]:

- In reality, many series do not assume continuous values. For example, company share prices have a minimal resolution equal to the tick size.
- Sometimes, the fundamental underlying assumption of normal errors breaks down, due to, e.g, low liquidity.
- ARCH models are parametric specifications that operate best under relatively stable market conditions. Although ARCH is explicitly designed to model time-varying conditional variances, ARCH models often fail to capture highly irregular phenomena, including wild market fluctuations (e.g., crashes and subsequent rebounds), and other highly unanticipated events that can lead to significant structural change.

Gouriéroux (1997)[44] constitutes a good introduction to ARCH and GARCH models. In the following, these models are treated briefly for a general understanding.

In an ARCH model, the autocorrelation of the volatility of the terms in the error process $\{e_t; t \geq 0\}$ is modelled by simply adding such an autocorrelation expression to the ARMA(p,q) model studied before. Hence for an heteroscedastic model of order one:

$$\begin{cases} \Phi(B)X_t = \Theta(B)e_t \\ e_t^2 = c + a e_{t-1}^2 + m_t \end{cases} \qquad (4)$$

m_t is a white noise process. Hence, $E(e_t^2) = E(c) + aE(e_{t-1}^2) + E(m) \Leftrightarrow V(e_t) = c + aV(e_{t-1})$.

[42] Engle (1982)
[43] Gouriéroux (1997)
[44] ibid.

We start by investigating the $\{e^2\}$ process itself. A formal definition of the ARCH(p) process is then the following:

$$e_t^2 = c + \sum_{i=1}^{p} a_i e_{t-i}^2 + m_{i,} \quad \begin{cases} E(e_t | e_{t-1}) = 0 \\ V(e_t | e_{t-1}) = c + \sum_{i=1}^{p} a_i e_{t-i}^2 \end{cases}$$

One can find the following criteria for the existence of the process:

- If the process is to be unambiguously defined, we need to apply the initial condition $E(e_0^2) = \frac{c}{1-a}$.
- In order for the e_t^2 to be non-negative, sufficient conditions are $a > 0$ and $c + m_t \geq 0 \forall m_t$.

This autoregressive representation of the error process does come closer than a simple ARMA or ARIMA model to the behaviour of many actual financial time series. However, one can also add a moving average part. GARCH (Generalised Auto-Regressive Conditionally Heteroscedastic) models do this, by modelling the $\{e^2\}$ process with the use of an ARMA model. The formal definition is:

$$e_t^2 = c + \sum_{i=1}^{Max(p,q)} (a_i + b_i) e_{t-i}^2 + m_i + \sum_{j=1}^{q} b_j m_{t-j}, \quad \begin{cases} E(e_t | e_{t-1}) = 0 \\ V(e_t | e_{t-1}) = h_t = c + \sum_{i=1}^{p} a_i e_{t-i}^2 + \sum_{j=1}^{q} b_j h_{t-j} \\ m_t = e_t^2 - h_t \end{cases}$$

In fact, this is an ARMA(max(p,q),p) representation for the $\{e^2\}$ process. In this case, however, the error term μ_t does not necessarily have a constant variance (since it depends on ε^2 itself).

In the simplest case, one can describe a time series observation directly with an ARCH model. Otherwise, one can describe the innovation process rather than the initial process using the model. Also, there are many extensions of the ARCH model, except for the GARCH model family. For example, it is perfectly feasible to introduce an ARMA model with GARCH errors:

$\Phi(B) X_t = \Theta(B) e_t$, where $\{e^2\}$ satisfies a GARCH(p,q) model.

This latter model might be suitable for analysing volume data over time. On the other hand, the simple GARCH model (where the initial process is modelled, rather than the innovation process), does not apply to this case – the volume data is not a zero-mean, normally distributed time series. This model is more accurate for use with returns data. As always, an iterative development cycle must be applied,

where the models used are evaluated and adjusted over time with respect to the quality of the alerts produced.

5.4.7. Additional extensions

Seasonality

In general, we can express a time series X(t) as X(t) = T(t) + S(t) + r(t), where T(t) is a "slowly" changing trend part, S(t) is a seasonal part, and r(t) is the stochastic, centered residual part, respectively. Regarding T(t) and r(t), the above described ARMA and ARCH models are adequate tools for describing the time series. However, there are more effective ways to deal with seasonality.

The easiest solution is to deseasonalise the series before attempting to fit it to a model. Once the series has been cleared for the seasonal component and fitted, the estimated model can easily be used for forecasting by simply adding the seasonal component in retrospect.

One simple way to estimate the seasonal component is the following[45]. It assumes that the period p for the seasonality is known in advance. This assumption might seem unnecessarily restrictive at a first glance. However, this is not the case in our financial application. First of all, there are different seasonal patterns atop of each other – over the time of each trading day, there is a distinct pattern in the volume series data (sometimes even in the returns data), at the same time as there are several well-documented phenomena operating in the more long-term, such as the January boom, etc. Common to all of these known cyclic behaviors is that their respective periods are known in advance. What is left to estimate is the exact pattern of the seasonal component.

If there is no trend part T(t) to take into consideration, the problem is rather simple. In this case, we replace T(t) with m – the mean of the process. Since the seasonal component per definition has a zero mean, we can simply make the estimation $\hat{m} = \frac{1}{T}\sum_{i=0}^{T} X(i) = \bar{x}$, where T is a entire multiple of the period p. Then, we can write $S(t) = s_1 d_1(t) + \cdots + s_p d_p(t)$, where $d_i(t) = \begin{cases} 1; & t = i + kp; k \in Z \\ 0; & elsewhere \end{cases}$. Of course, we have that $\sum_{i=0}^{p} s_i = 0$.

If we form $X(t) = m + \sum_{i=0}^{p} s_i d_i(t) + r(t)$, we can estimate the s_i by the use of the ordinary least-square method. The more periods we use in the process, the more accurate the result. At the same time, one cannot use too much data if one wants a minimum of trend disturbance.

[45] "Séries Chronologiques" (2000)

However, in financial time series problems, there are often such tendencies present. For example, consider the investigation of the intra-day returns pattern. In order to get a detailed and accurate picture of this pattern, one typically needs to look at trading data for many consecutive days. But the market experiences times of hausse followed by baisse periods. Therefore, there is a general trend on top of the typical intra-day variations in the return to pay respect to.

In order to estimate the T(t) at the same time as the S(t), we first form:

$$Y(t)=\frac{1}{p}\sum_{i=-m}^{m}X(t+i)=\frac{1}{p}\sum_{i=-m}^{m}T(t+i)+0+\frac{1}{p}\sum_{i=-m}^{m}r(t+i)=j(t)+u(t);\quad m=\frac{p-1}{2}.$$

Here we assume that p is odd. The case when p is even is analogous. Since we sum over one whole period, Y(t) is approximately deseasonalised. Further, since T(t) is a slow component, $j(t)\approx T(t)$. Thus, we estimate T(t) and S(t) by the following iterative scheme:

1) Calculate Y(t).
2) Estimate $j(t)$ from Y(t).
3) Estimate $\hat{T}(t)\approx \hat{j}(t)$.
4) $\hat{X}^*(t)=X(t)-\hat{T}(t)\approx S(t)+r(t)$.
5) Estimate S(t) from $\hat{X}^*(t)$ and deseasonalise the series data.
6) Iterate using the new, deseasonalised data.

The estimation in the step 5) can be done as described above. The one in step 3) can be done by the use of a parametric model, which is estimated using the least squares method. For example, a polynomial of sufficient degree should be well-adapted enough to be able to capture most of the trend in many cases. In yet others, other function classes may be more appropriate.

Finally, one can also do the whole estimation by using a parametric model of both T(t) and S(t), estimated by the use of the least-squares method. An example of such a parametric model is:

$$\begin{cases} T(t)=a_0+a_1t+\cdots+a_kt^k \\ S(t)=a\cos(\frac{ip}{p}t)+b\sin(\frac{ip}{p}t) \end{cases},$$

where k should be chosen sufficiently large to capture the detail of the trend.

6. The pilot implementation

After having dealt with the general approach and the mathematical background of a generalised surveillance function, it is not time to move focus to the actual situation faced by the present thesis. The aim is to apply the methods developed above to the current situation.

We start by briefly describing the prerequisites on the JIWAY exchange, after which the application is treated in detail.

6.1. The JIWAY exchange

As stated above, for different reasons our access to good data from the JIWAY exchange has been somewhat limited. The total amounts of data is very large. However, only a small fraction relates to orders and trades at the JIWAY exchange itself – the rest of the data relates to off-exchange events. A consequence of this has been that for each instrument we have only had hard figures relating to a very limited number of orders per trading day. The situation has been even worse when it comes to matched trades. Sadly, this fact somewhat limits the scope of this thesis, because of two major reasons.

Firstly, considering the large sizes of the historical market place data available for the testing of the client/server system, it has not been possible to use more than a week's worth of data. During this time, the trading activity in each instrument is quite limited. This makes an extensive testing of the surveillance algorithms difficult. Instead, I have chosen to focus on the theoretical aspects of the task at hand, viewing the client application developed as merely a pilot implementation shedding some light over the possibilities and limitations of such an automatic surveillance system architecture.

Secondly, when the activity level is so low, it is difficult to estimate how usable the surveillance tools developed herein actually are. The price process for any given instrument based solely on its orderbook is with necessity a piece-wise linear function of time, in the case of JIWAY for the most part with quite long linear segments. It is difficult to see how this process could have normally distributed errors, hence making many of the considered metric/benchmark tests invalid. Specifically, this is the case for the default, normality-assuming test, as well as the different time series models discussed below. However, instead of letting this fact limit the methods developed, I have tried to assume a wider perspective, trying to present some usable methods in the general surveillance situation. Hence, the methods investigated herein should be usable in a generic market, preferably with a larger daily turnover per instrument than is the case for the available JIWAY data.

Having defined the scope and level of ambition, we now turn to study the pilot implementation itself.

6.2. The application

In this section, the system architecture of the client application is presented briefly. Where so is applicable, more detailed information and comments on specific features can be found in the Javadoc files created or the source files of the client directly.

6.2.1. System overview

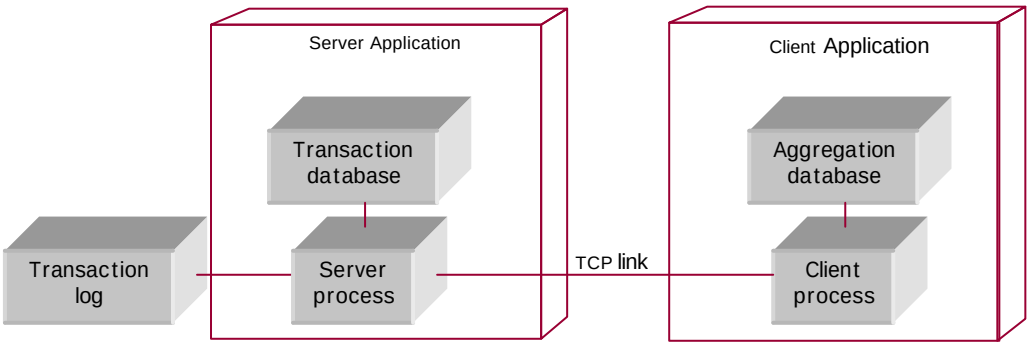


Figure 7 - Client-server system architecture overview

As can be seen in Figure 7, the market surveillance system is divided into two parts – a server and its client. There can be many clients connected to the same server. Usually, the server should run on one node, and be connected to one or several clients running on autonomous machines elsewhere on the network or via the Internet. However, as can seen in the figure below the server is connected to the client via a TCP link, so the server can in fact run on the same machine as the client. The Figure also shows that each application has its own separate database storage.

In the transaction log, market information from the trading system is queued, and subsequently sent to the server node. This information reflects all market events occurring in the market place (plus some additional information about the rest of the world) in the form of *transactions*. One transaction represents a distinct event and can be the entering of a quote, a trade that has occurred, a change in an orderbook, etc.

The server application reads the transaction records sent out from the transaction log, filters them, enriches the information by adding information about, e.g., the market participants and consequently furthers it to the different active client applications. Exactly what information is sent to each client depends on the demands of the client in question. Additionally, the server keeps an internal transaction database, in which the transaction data is saved intermediarily. The reason for this is to facilitate query functionality on historical data. Also, some initial data processing is done before the transaction data is put into the database.

All active clients have one TCP link to the server for each active information subscription. Also, each distinctive query from a given client node creates a new TCP link, over which the information is sent from the server. After the whole query has been delivered, the new TCP link is shut down.

The task of the client application is to communicate with the server, and carry out market surveillance using the data provided over the TCP link. Also, the client keeps a database holding aggregated, historical metric data to use in the surveillance process. Finally, the client also hosts a GUI to let the surveillance operator interact with the system.

The server application is entirely written in C++ on a Windows NT platform. It uses Microsoft MFC functionality for TCP/IP communication etc. For more information, please consult Peter Bergenwald's thesis report.

The client application, on the other hand, is written in Java. For the TCP/IP communication and the XML- and JDBC[46] functionality, the add-ons available from Sun Microsystem[47] are used. The GUI is based on Swing[48] components. As will be discussed more in depth below, the choice of Java as the implementation language has not led to ideal performance in time-critical situations. However, Java offers a way to quickly set up a working application. In the present situation, Java proved to be powerful enough to effectively demonstrate the thoughts behind the surveillance system. For the more processor intensive parts of the application, such as mathematical calculations, C++ native code is used for increased speed.

Both the server and the client uses a MySQL database. This is a simple, yet powerful, database that is available free of charge for non-commercial purposes online[49].

6.2.2. The XML Interface

As explained, it is the server that provides the client with all information to be used in the surveillance process, including the actions taken in the market place as well as details regarding instruments, brokers, market makers and end clients. The information is sent as responses to requests sent by the client to the server.

The communication takes place over a two-way XML protocol, specifically designed for the present surveillance system. It specifies the format in which the client poses requests for information to the server, as well as that in which the server sends information responses in reaction to these requests. In the Appendices of chapters 9 and 10, two different versions of such XML protocols are detailed. Here, only an overview of the most important characteristics of

[46] The JDBC (not an acronym, but often thought of as standing for "Java DataBase Connectivity") software is available for free at the java.sun.com web site.
[47] See java.sun.com for more information.
[48] Ibid.
[49] www.mysql.com is the official site of the MySQL database. Here the software is available free of charge.

these protocols is given – please refer to the Appendices for a more complete discussion.

The XML interface protocol version 1.0 (please see Appendix 2: XML Interface version 1.0) is quite strongly mapped to the JIWAY exchange, and to the CLICK exchange system. This works fine as long as the surveillance system is to be used solely with this system. Ideally, however, the thesis' solution should be capable of running on a broader selection of exchanges and exchange systems. Therefore, me and Peter Bergenwald have proposed a more general XML communication protocol (please see Appendix 3: XML Interface version 2.0) to be used over the interface. This new protocol incorporates as detailed information as possible at the same time as exchange- or exchange system-specific information is hidden. This way, the same client could be used with a variety of exchange systems, e.g., with the SAXESS system by OM. It is possible to incorporate exchange- or exchange system-specific functionality (such as the CLICK system's hedge orders) by only carrying out minor modifications to the client application. The same is the case for any special functionality on different exchanges. In order to format the transaction information from the exchange system into XML information, on the other hand, a new server has to be implemented on every new exchange to be surveilled.

One problem using XML as the base for the protocol is that XML is tagged and textual. Therefore, an XML communication tends to be very talkative. However, since the server eliminates redundancy in the data sent, the XML-formatted information sent to the client remains within the same magnitude as the original log data from the exchange (which in turn comes in a binary format). Please refer to the thesis report by Peter Bergenwald for a more thorough treatment of this topic.

Today, the bandwidth on a standard LAN is broad enough to handle the information load over the XML interface between a server and a client. Should this constitute a problem, however, it is always possible to add compression to the data actually sent. This way the bandwidth load could be decreased considerably.

Requests

Which records that are sent out by the server to each connected client depends on what the client has asked for. This is done in a *request*, using the same XML protocol. Such a query can either be a *query* or a *subscription*.

A subscription can only be placed for future exchange data – the start time for the subscription is automatically set to the current time. It can further be delimited in terms of transaction types, instruments, stop time, customers, users and clients. After the subscription is placed, the server produces XML records of all the relevant information from the exchange in real-time (or at least in semi-real time, i.e., as quickly as possible).

A query, on the other hand, is always concerned with historical data from the internal database. The user can limit the scope of the returned information in the same dimensions as for the subscription, with one exception. In the query case, an individual order number (the internal exchange order number) can be specified, to track an individual order's path through the system in retrospect.

There is always a third type of request, namely a "light" subscription. Here, the server sends out only the most vital information to the subscribing client. The exactly information to send out is configured in the server application.

The format for all requests is the same for both of the XML interface versions.

Responses

All data arriving to the client can be divided into *records*, each one representing an individual transaction[50] on the exchange. Each such record is informationally self-contained, i.e. it contains all data available for the respective trading parties and so forth. For instance, this means that individual instruments are referred to by name in a readable manner rather than by some internal code. As a matter of fact, there is no special query for instrument-, client- or customer data, since all this information is contained in the relevant records.

The mapping between the exchange transactions and the XML records is close, but not necessarily one to one. Specifically, many transactions are never sent – either because the client has not asked for them, or because of the fact that the server clears out some information-redundant transactions in the initial information processing.

By the introduction of the XML interface version 2.0 (presented in Appendix 2), the congruency is further decreased in favour of a more generic and flexible protocol. However, single transactions are still represented by single information records, since related transactions often are separated in time in an unpredictable manner.

As opposed to requests, the information sent from the server differs between the two XML interface versions for subscription. On the other hand, the response XML records are produced in the same format for both queries and subscriptions.

As an example, the information records that may arrive when using the CLICK-operated JIWAY exchange is derived from the transaction types presented below in Table 1:

[50] Recall that a transaction represents a single event on the exchange.

Information record type	Contains information about	Metric(s) affected
New order entry	Newly arrived order	Orders and trades
New quote entry	Newly arrived quote	None
New trade	New trade	Orders and trades Beneficial ownerships
External best bid / offer	The BBO of the far-away market place upon which the stock is parallel-noted	None
Internal best bid / offer	The BBO of the market system, as defined by the current orderbook	None
Hedge order entry	Newly arrived hedge order	None
Hedge order execution	Hedge order that has been traded on a far-away market place	Orders and trades
Improvement quote	The definition of a newly arrived definition of an improvement quote rule	None
Instrument status	The status of an instrument	None
Orderbook change	Any change in the current orderbook	Price-, volume and orderbook

Table 1 - Transactions on the JIWAY exchange

These transactions are mapped into response records by the server application.

Please see the Javadoc or the source files for the client application and the log documentation for the CLICK system for further information about the exact meaning of hedge orders, improvement quotes, etc., and about what information is relevant to-, and available from, these different types of information records

Information and surveillance

Using Table 1 as an illustrative example, the information available from a typical exchange is on a very detailed level. During the life span of one single order, there are several transactions, each reflecting an event in the market, such as the entering of the order into the orderbook or the trading of the whole order, or a part of the order. All of these transactions occur at different points in time, and information records describing each event will therefore be sent out to the client as they occur. In light of the currently large order volumes on the typical exchange, the total amount of information records sent to the client is massive.

The task of the client application is to extract as much relevant information as possible from the response records received, while still limiting the data load by cutting away the not-so-relevant information. This is done by the process of *aggregating* the data received over such dimensions as time, instruments, clients, etc. By aggregation, two purposes are fulfilled. Firstly, the information is cut considerably while still maintaining control over the relevant market patterns.

Secondly, the general trends in the market are isolated more concisely. Subsequently, this aggregated data is used, in conjunction with the incoming information on the most detailed level, for the surveillance of the market. In the following section, we will study how this is done in practice.

6.2.3. Client application

As stated above, the client application is coded in Java. The code consists of 13 packages, with a total of 73 classes and approximately 10,000 rows of code. This is quite a substantial amount of code, owing greatly to the diverse tasks of the application. The majority of the code, however, does not relate to the core surveillance loop. Rather, since this is a pilot implementation for demonstration purposes, there are many functions (such as the GUI), that have been implemented without flexibility and robustness as the primary concern. The core functionality, however, has been more thoroughly tested – both under different conditions and under large workloads. Javadoc with comments has been produced for all classes of the application.

In Figure 8 below follows a (slightly modified) UML-style[51] overview of the packages of the client application. Not shown in the Figure are data wrapper classes. The code is divided into packages, shown in the Figure along with their interaction, though in a somewhat simplified way. It does not take into consideration that some parts of the application are run in separate threads and may be instantiated in several copies at the same time. For a more detailed explanation, please see the description of the individual packages below. For a full understanding of the functionality, please see the Javadoc[52] for the client application classes or the source code itself. In the UML diagram, the naming convention used is the following:

[package task description] : [Java package name]

The system communicates with two external actors – the surveillance system user and the XML TCP link. The user receives information from-, and sends configuration input to, the system via a client GUI. The XML link receives and sends XML information via the Link manager. The XML translator takes care of the actual translation of data into the XML format.

In the heart of the client application, the central data manager is found. It communicates with the client GUI and the XML link manager, and processes the surveillance data. In the surveillance process it uses a MySQL database (via the Database IO package, using the Java Sun JDBC functionality), where aggregated historical data is saved. Special surveillance- and math packages are also used. Variables global to the system are kept in separate classes. This is the case for the configuration of the surveillance process as well as for different constant variables used for various purposes.

[51] UML, Unified Modelling Language.
[52] This, as well as the source code, can be obtained from the author upon request.

Normally, the flow of financial information runs via the central data manager to the client GUI. However, it is also possible for the system user to request specific information. When this is the case, the information flow is handled by the use of a special client GUI data manager, set up in essentially the same way as the central data manager to communicate with the XML interface. For every new query, an additional XML interface is started in order to handle the incoming response XML records.

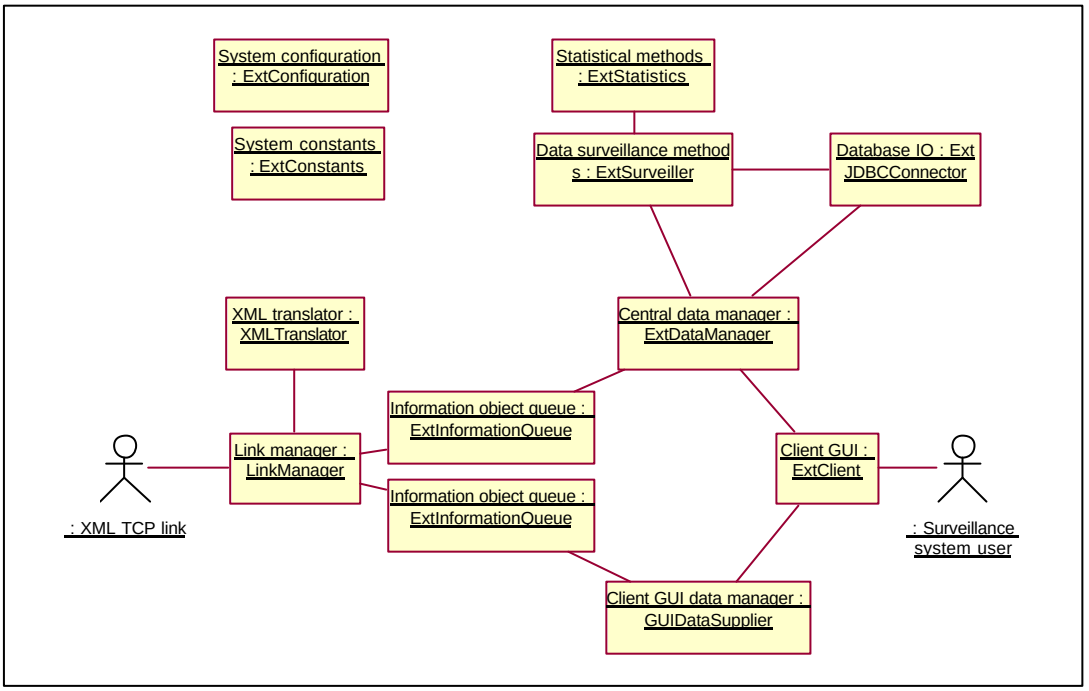


Figure 8 - Client Application overview

Application dynamics

In this section sequence diagrams will be used to describe the dynamic behaviour of the major parts of the application.

Start-up

Figure 9 depicts the start-up dynamics of the client application.

The client GUI package contains the main method of the application. Its client class starts by instantiating the main data manager class, and sets it off running the main surveillance loop. The data manager, in turn, instantiates the surveillance- and statistics classes. Thereafter, the database connectivity is set up. It is assumed that the MySQL database engine is up and running before the client application is started. The information queue is also created.

Thereafter, the data manager creates the XML link manager, which in turn creates an XML translator. Submitting the information queue as an argument, a new default surveillance data subscription is set up with the server application. This default subscription asks for the relevant surveillance data for all traded instruments on the exchange. After this point, surveillance data is sent to a queue in the link manager as soon as it arrives to the server from the exchange.

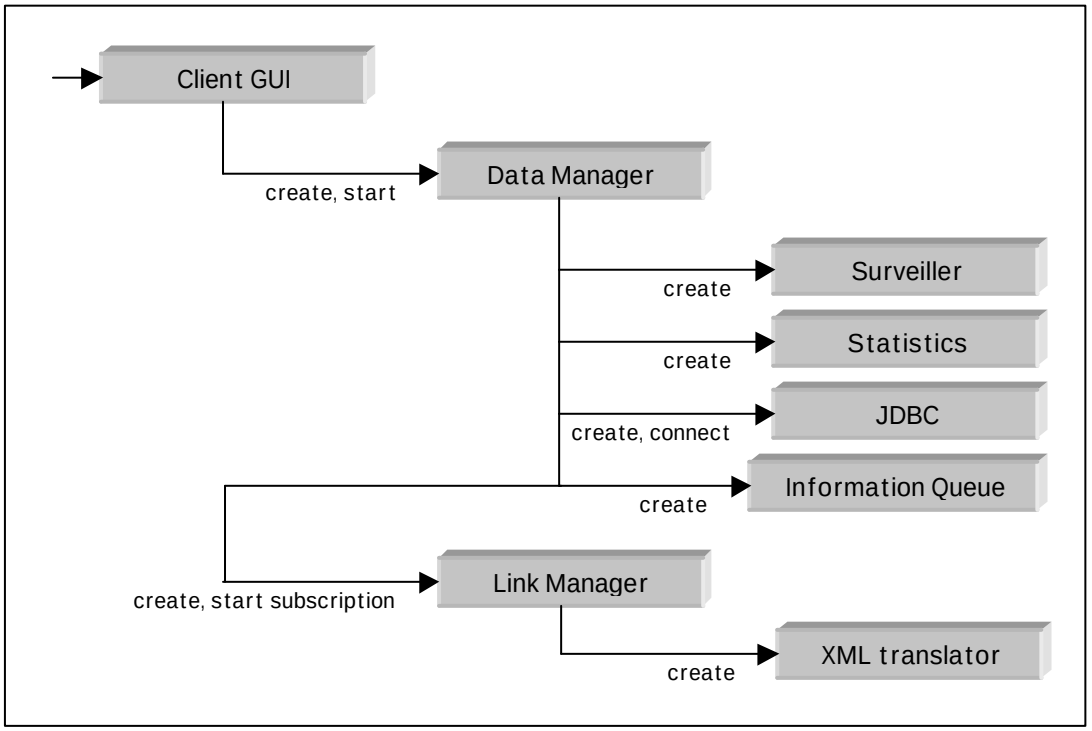


Figure 9 - Start-up dynamics

Receiving information from the server

Please refer to Figure 10 for the information receival dynamics. For every active subscription, there is one dedicated link manager. Since there is exactly one subscription open when the client application has been started (the default surveillance subscription), there is only one link manager running just after start-up. If the user makes queries for specific information at a later stage, a new link manager is set up to deal with the query response. Every link manager works in the same way receiving information records from the server.

The TCP/IP connection to the server is polled continuously for newly arrived information records. When there is a new piece of information to gather, the link manager reads it, and passes it on to the XML translator for parsing. The return from the XML translator is an internal Java object containing the same information as the XML record received from the server.

This Java object is placed in an information queue recognized by the data manager. For the default subscription, the data manager is the central data

manager. If the information records relate to a query, the data manager is a purpose-defined query data manager.

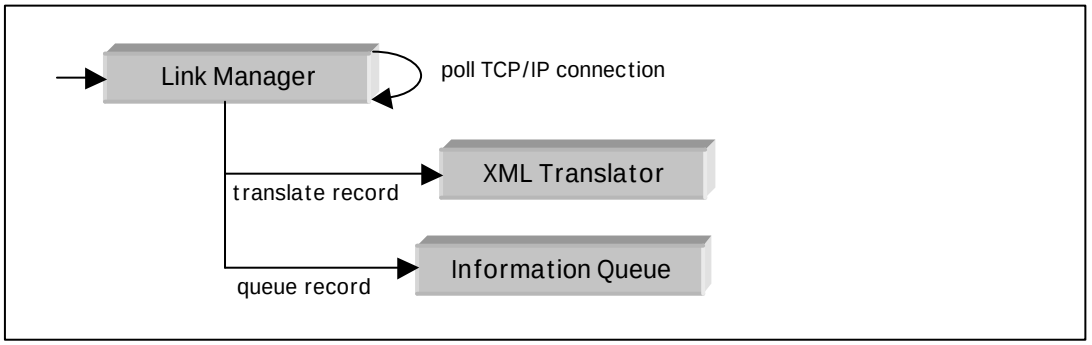


Figure 10 - Information receiving dynamics

Information processing

We now turn to study the internal information processing more in detail, with reference to Figure 11.

Recall that the surveillance loop features two loops. The continuous loop continuously polls the link manager-associated information queue for new information records. When such new information is found, it is stored in the internal memory structure in an aggregated fashion together with the rest of the data received since the last aggregation for that instrument class.

For each information record received, two additional tasks are performed. Firstly, the surveiller class is asked to check for any continuous alert groups that should be triggered for the newly income piece of information. The surveiller uses the statistics class to do the statistical calculations. This class, in turn, uses native C++ code for heavy mathematical calculations. Exactly what calculations are done depends on what benchmarks models are defined.

Secondly, the client GUI is updated with such information as price- and volume data for display. If there has been one or many triggered alert groups, information about these are also transferred to the GUI package. The communication between the data manager and the client GUI takes place via a collection of common data queues, that map directly to list boxes in the GUI. This solution is perhaps not the optimal with regards to flexibility demands for a full-fledged GUI. Remembering that the GUI is only a rudimentary prototype, though, this method works fine for these purposes.

Every time that the data for a certain instrument class needs to be aggregated, a new turn is conducted in the discreet surveillance loop. Firstly, the data that has been aggregated during the previous aggregation period is saved into the database, using the JDBC package. Thereafter, the surveiller class is used to check for triggered alert groups in the considered instrument class, in the same manner as in the continuous loop. However, in the discreet loop, the benchmark models may be recalculated prior to the actual checking. The frequency of this

recalculation is configurable. In order to recalculate the benchmark models for the instrument class, historical data is needed. This data is collected from the database before the recalculation takes place.

If any alert groups have been found to trigger, information about these alert states is communicated to the GUI by the use of the above said common data.

During the whole surveillance process, the data manager also updates the database with information about new exchange data (instruments, clients, customers, etc.) and triggered alerts. This is not, however, showed in the figure.

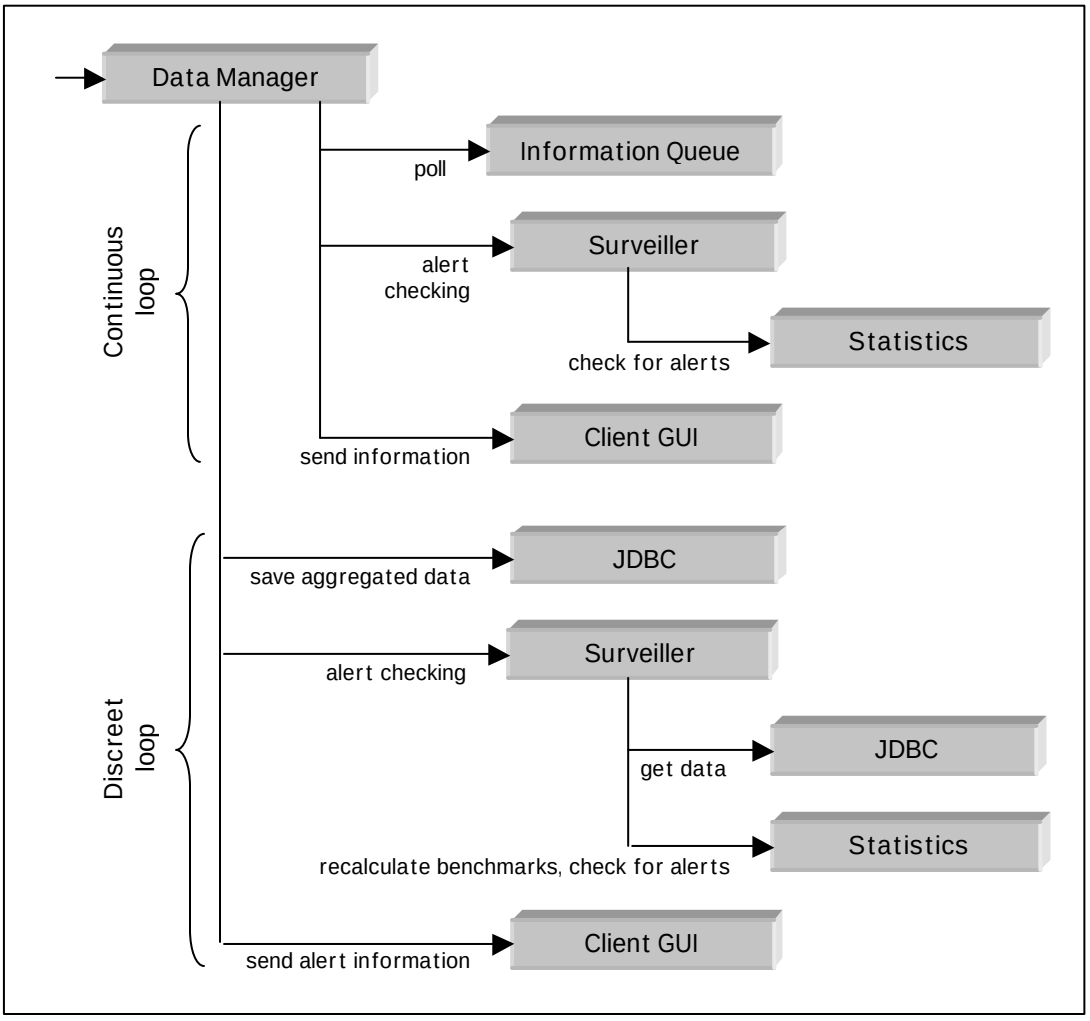


Figure 11 - Information processing dynamics

Requesting additional information

From the GUI, it is possible for the user to request additional historical market information. Presently, this is only possible to do for information produced in the market around the time for a triggered alert state and stored in the server-side

database. However, it would be rather straight-forward to implement more elaborate ways for the user to exploit the information in this database, as well as that stored in the client-side aggregated database.

As can be seen in Figure 12 below, when the user requests for additional information, the query produced by the client GUI results in very much the same execution pattern as the start-up procedure described above. As a matter of fact, the GUI data supplier is nothing but a simplified data manager. The main difference is that no surveillance is performed in the GUI data provider – rather, the collected information is simply fed to the client GUI for display in a pop-up window.

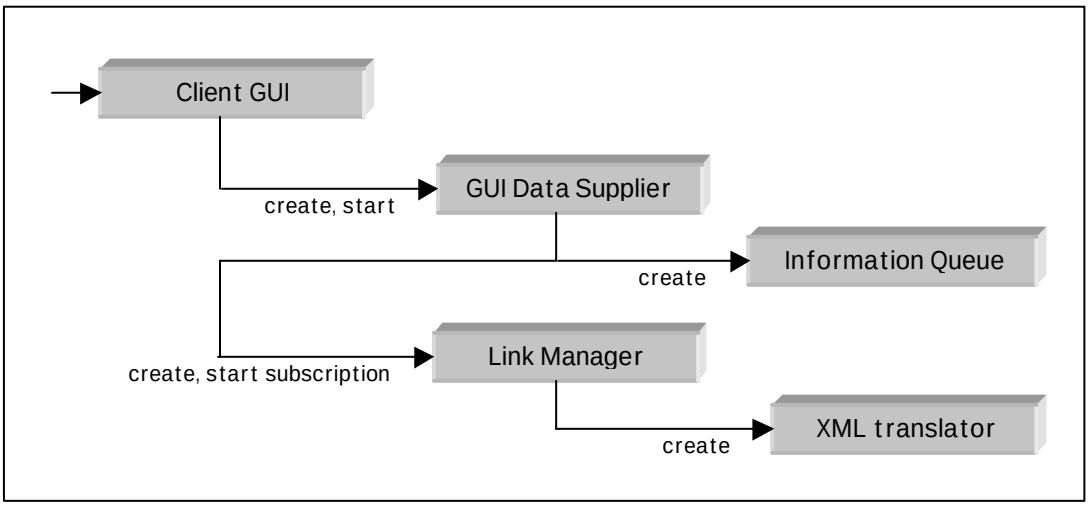


Figure 12 - Additional information request dynamics

Thread design

In the present implementation, the client application runs three main threads:

1) The link manager runs in an autonomous thread, polling the TCP/IP connection for new information and putting the information found into an information queue.
2) The central data manager runs in another thread, polling the information queue and processing the information found in the main surveillance loop. It has a data area common with the client GUI, where it puts the results from the surveillance process.
3) The client GUI runs in the third thread, showing information to the user and accepting configuration input.

When a request for additional information is made by the user, an additional thread is created for each such request. Each request thread is structured in a similar way as the central data manager.

With the above discussion on the dynamical behaviour of client application, let's now turn to study the individual packages more in detail.

Constants package

This package holds the global constants of the surveillance system. Both the constants- and the configuration package variables are declared static – that way they are available to any class that knows about the package.

The variables themselves are of a diverse nature. They include definitions of the source of XML information, different default values for the system as well as certain global surveillance settings.

Configuration package

The configuration package holds the entire configuration of the surveillance system. It acts as a container holding global variables – each class that knows of the package can read from- and change the configuration settings.

In a release version of the surveillance system, the configuration should be fully supported by the GUI, in order to allow run-time configuration. However, in the present implementation of the client application, the configuration of the surveillance process has to be done in the code. On the other hand, for the purpose of demonstrating the theoretical surveillance framework this has not constituted a problem.

The configuration settings can be divided into three major areas: instrument categorisation, alerts and other parameters. Since the instrument- and alerts configuration more or less defines the surveillance scope and –process, these areas will be carefully described in the following.

Instrument categorisation

Instrument categorisation provides a way for the surveillance system operator to structure the surveilled assets in manageable way.

Instruments

The instrument is the basic unit in the surveillance process, representing a listed company stock type. New instruments are automatically added to the configuration when they are discovered in the data received from the server[53].

Instrument groups

For certain instruments, the market liquidity is low. It can therefore be practical to have the possibility to cluster some instruments together when analysing the market data. For instance, by looking at the return development over time for a small portfolio of low-liquidity instruments, one might obtain a fairer picture of

[53] Recall that all XML data sent from the server application is informationally self-contained.

the current market picture for each of these instruments by observing them as a group rather than by looking at the individual return histories. This is because since the liquidity is low, the price curve (which is piecewisely linear) is often stable over a substantial time before jumping to a new, possible quite different, level. Hence, an estimation of the volatility for the individual instrument becomes very imprecise.

This clustering is done by the use of *instrument groups*. Each instrument in an instrument group is analysed individually, but the individual instrument data is compared to some sort of aggregated statistics over every instrument in the instrument group. Thus, the statistical groundwork is more reliable for the individual instrument. The drawback of this strategy is, of course, that there is not a perfect correlation between the market data of any two instruments. Som of what is won in statistical precision is lost in relevancy. In order to obtain a good trade-off, one has to carefully choose the instruments that should belong to the same instrument group.

Every instrument must belong to an instrument group. If the liquidity of the instrument is sufficiently high to treat it separately, it should be put in an instrument group of its own. One instrument can also take part in several instrument groups – it is, e.g., possible to have several instrument groups, each with a wider selection of instruments than the previous. This way, a varying market liquidity can be handled by simply choosing which instrument group to use in the alert checking.

Instrument classes

Subsequently, instrument groups are put into *instrument classes*, clustering the surveillance settings for different instrument groups. The instruments of every instrument group have the same settings for the resolution of the market data that is saved to the historical database. The settings for how often the parameters of the statistical models used for forecasting and alerts production should be updated are also shared within the same instrument class. Therefore, the instrument class abstraction is practical when looking at many instruments that approximately share the same market characteristics.

As is the case for instruments in instrument groups, each instrument group must be put into an instrument class, at the same time as a given instrument group very well can be part of several instrument classes.

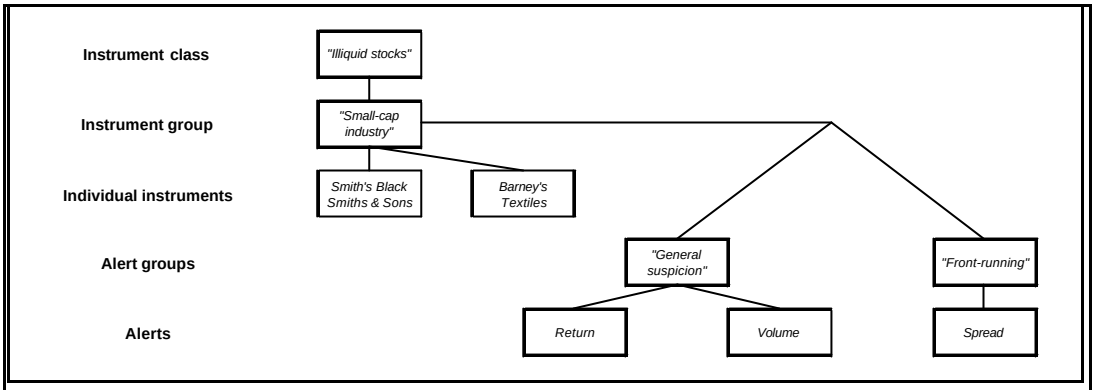


Figure 13 - Example of an instrument structure

Consider Figure 13 above, illustrating a part of the configuration that a particular surveillance system user (Bill) has set up. One of the configured instrument classes is called "Illiquid stocks". In this class, all instrument groups that contain low-cap instruments are put. For all of these instruments, Bill reckons that the need for historical data is about the same, and he also thinks that the statistical models used can be updated at roughly the same intervals and still be valid.

The "Illiquid stocks" class contains, among others, an instrument group called "Small-cap industry". In this instrument group, Bill has put some of the low-liquid, small industry stocks that he is currently surveilling. Since he does not think that any one of these stocks has liquidity enough to effectively provide a historical statistical record to compare the collected metrics to (and since he knows that they usually behave similarly in the market), he clusters them into one instrument group. So far, he has added Smith's Black Smith & Sons and Barney's Textiles, two companies that have been unusually quiet on the stock market so far this year.

As can be seen in the Figure, Bill has also added some alerting configuration. Let's see how this works:

Alerts and alert groups

As the surveillance client receives new data from the server, it compares this data with historical data. If the data is somehow suspicious – showing proof of possible non-allowed market behaviour – it triggers alerts in order to focus the attention of the surveillance system operator on the suspicious situation. The way to tell the system for what instruments to do alert checking, and what alert types should be considered for these instruments, is to define *alert groups*, each containing a variable number of *alerts*.

Very much the same way as for individual instruments, an alert group always belongs to an instrument group. As a consequence, every instrument that belongs to a certain instrument group will be checked for the alerts contained in the alert groups that belong to the same group.

Analogously, every alert must belong to an alert group, at the same time as one alert can belong to several alert groups. As each alert represents one basic alert type related to the investigation of a single surveillance metric, this should in fact frequently be the case. The type of the alert simply represents the metric to check for suspicious market behaviour – return, order volume, etc., reflecting different assumptions about how the market behaves when being manipulated.

Alert groups come in two types – the ones containing only one alert and the ones containing several alerts. For the first type, it is (in the current implementation) possible to set a *sensitivity* parameter for the alert group. This is a measure on how sensitive the analysis should be, expressed as the desired average number of alerts per day. For example, if the sensitivity parameter is set to 10, the system calculates the level of the considered statistics that needs to be surpassed in order for the expected value of the number of produced such alerts per day becomes 10. This is done on basis of the historical data for the metric in question. [54]

For the other type of alert group – the one with several alerts in it – the alert has an *absolute threshold level* parameter. In this case, an alert is produced if the level of each individual alert of an alert group is surpassed, at the same time, by the respective metric value for the considered instrument.

Some types of alerts (*discreet alerts*, such as the unnatural returns alert) are investigated every time the data is aggregated into the database. Others (*continuous alerts*, such as the alert for suspicious individual orders) are tested for in real-time, as the transaction is sent from the server to the client application. Because of the fact that they are investigated at different times, discreet and continuous alerts are not allowed in the same alert group.

Returning to the case of Bill and his surveillance configuration process, take another glance at Figure 13 above. For his Small-cap industry instrument group, Bill has defined two different alert groups. The purpose of the first one ("General suspicion") is to catch a broad palette of possible market interference, such as cornering, pump-and-dump, etc. Consult Figure 3 (also above) for information on what to look for when checking for different alerts. The alert group contains two individual alerts – Volume and Return – each checking for unnatural metric values.

Since the General Suspicion group holds more than one alert, the absolute level parameters of the alerts are used when deciding whether an alert is triggered or not. As Bill argues, all of the instruments in the instrument group considered have approximately the same market characteristics. Therefore, it is relatively

[54] See the section on benchmarks above for more information on how this calculation is actually carried out.

straightforward to find maximum (and also minimum, in the case of the Return alert) allowed levels for the values of the volume- and return- metric.

However, Bill is also interested in checking for front-running activity in these instruments. Therefore, he has defined the alert group "Front-running", containing the single individual Spread alert (since Bill knows that if the spread suddenly increases, this might be a sign of front-running activity). Since the alert group has only one alert, the sensitivity parameter of the alert group is used when deciding if it should be triggered or not. Hence, Bill simply assigns a value indicating how many alerts that he wishes to receive from this instrument group every day. After this, the system itself will calculate the appropriate threshold level.

Found below is Table 2, summoning the different alert types available. Some of these are implemented in the pilot application, some should be implemented in a release version.

Alert type	***Alert category***	***Implemented***
Excessive volume (trades/orders)	Discreet (checked at every aggregation)	Yes
Excessive return	"	yes
Excessive spread	"	yes
Non-normal orderbook	"	no
Beneficial ownerships	"	yes
Correlation between orders	"	no
Suspicious orders	Continuous (checked at the arrival of the information)	yes
Immediately traded orders	"	yes
Suspicious trades	"	yes

Table 2 - Available alert types

Other parameters

Apart from the definitions of instruments, instrument groups, instrument classes, alerts and alert groups, there are also some other settings available in the configuration of the surveillance client:

- A list of instruments that will send out *historic transaction information* to the client GUI when an alert has been triggered on them. The time span of this information is also configurable, and is defined as the time before- and after the actual trigger of the alert from which to query information. The query is made with the server, hence producing data on a transactions level. Presently, this feature is not in use. Instead, the GUI package itself will query

for this historical information when the user clicks on the presentation of a triggered alert group in the GUI window.

- Lists of the instruments that send *out price-, order volume-, trade volume- and orderbook information*, respectively, to the client GUI in real-time. As mentioned above, the information sent to the GUI is not in the form of transaction data, rather in a GUI-specific data format. The actual transferral of data takes place via a common data structure, making it possible for the GUI to graph the activity of instruments continuously.
- The *price* definition that the system uses. The alternatives available at the moment are the average of the best bid- and ask prices for the instrument or the price of the last trade made[55].
- The definition of *return* that is used.
- The definition of a *suspicious order*. This definition is made in terms of a constant specifying, as a fraction of the instrument price, how far from the best ask- or bid price that an order can be without being "suspicious".[56]
- A number of constants specifying whether or not the surveillance system should check for a number of different alert types at all. These include the correlation between the orders of third-party clients as compared to their brokerage firms, suspicious orders and –trades, and whether the system should check for orders that are traded without first entering the orderbook. These parameters should be used for testing purposes only.
- The choice of *statistical method* for the different alert types. The default setting is a mean/variance analysis, but for alerts having some other statistical model associated with them, this setting can indicate that this alternative model should be used. An example of such an alternative model is the GARCH model used with the returns alert. See the section 5.4 Benchmarks above for more information about this.

Link manager- and XML translator packages

These packages handle communication to- and from the XML interface. Normally, this communication will take place over a TCP link with another computer on the same network or elsewhere. However, the link manager also has the capability to read from a file containing XML in the appropriate format (for testing purposes). There are two different versions of the XML protocol, of which the version 1.0 is the one currently in use.

However, as described above, this version lacks some generality because it is tightly coupled with the JIWAY exchange and the CLICK trading system. As a

[55] See section 5.3.2 for a more detailed discussion on different price- and return measures.
[56] Other ways of defining a suspicious order are possible. Please see Appendix 4: To do list for the client application.

matter of fact, much of the information treatable over the XML interface version 1.0 (Appendix 2: XML Interface version 1.0) is directly related to specific features of the CLICK system running on JIWAY.

Version 2.0 (Appendix 3: XML Interface version 2.0), on the other hand, has not been practically implemented. But, since it is a much more general approach, it should constitute a more extendable solution. When version 1.0 in many ways is a simple translation of the information arriving from the JIWAY exchange, version 2.0 is a way to send generic market surveillance information in real-time.

Both of the XML interface versions incorporate functionality for sending information requests (subscriptions and queries) as well as receiving responses. The requests are the same in both cases.

There is no specific login- or logout procedure. Rather, a surveillance client is considered logged in when it places a subscription or a query to the server. Further, there is no encryption, nor compression in the data flow. In a release version of the XML interface, the security aspect needs of course be taken much more seriously than has been the case for the present thesis. The information sent over the XML link is classified, and encryption would be an absolute necessity. However, as discussed above compression might not be necessary as long as the information transfer itself is not a bottleneck for the surveillance system.

In Appendix 2: XML Interface version 1.0 and Appendix 3: XML Interface version 2.0, respectively, the two interface versions are presented in detail. Nowadays, the standard procedure for specifying XML protocols is with an XML schema. For the present purposes, however, the authors have judged it fully sufficient with the simpler DTD (Document Type Definition) of the XML interface protocols.

As can be seen in Figure 9 - Start-up dynamics and Figure 12 - Additional information request dynamics, the client GUI always runs an XML link manager in order to get information from the server. Every time this happens, the link manager is run as a separate thread. The information request is either in the form of a subscription for real-time information or a query for some specified set of historic information. The link manager translates the query or subscription into the appropriate XML text string and sends it to the server over the TCP/IP connection. Simultaneously, it sets up an XML parser waiting for information records to arrive in response to the request. This parser is a standard SAX[57] solution, using libraries provided by Sun at the java.sun.com web site. The information is received, parsed and put into the appropriate internal data structures for further transferral to the data manager that originally initiated the link manager. When the data collection is done (i.e., when the XML document ends), the XML parser dies. Since there is no special session management in the XML protocol used, this can be seen as a user logout.

[57] SAX, Simple API for XML.

The information thus received is sent to the data manager via an information queue, to which the XML parser as well as the data manager has access. This queue serves the double purpose of evening out the data flow and letting the link manager thread and the data manager thread communicate.

Figure 14 below shows the activity over the XML TCP/IP link more in detail. A couple of things are worth pointing out specifically. Firstly, the link manager thread continuously polls the TCP/IP connection for new information. When it has something to receive, the information is summoned in the textual XML format. Thereafter, it is passed to the XML translator for translation into one of a collection of internal information objects. The XML translator is used in the other direction when formulating a query or a subscription to be sent to the server, going from internal information objects into XML text.

Secondly, the session management is tightly coupled with the XML structure itself. The “login” of a client is the request itself. Thereafter, the server accepts the “login” by sending the XML header of the response document. The server later “logs the client out” by sending the end tag of the XML document.

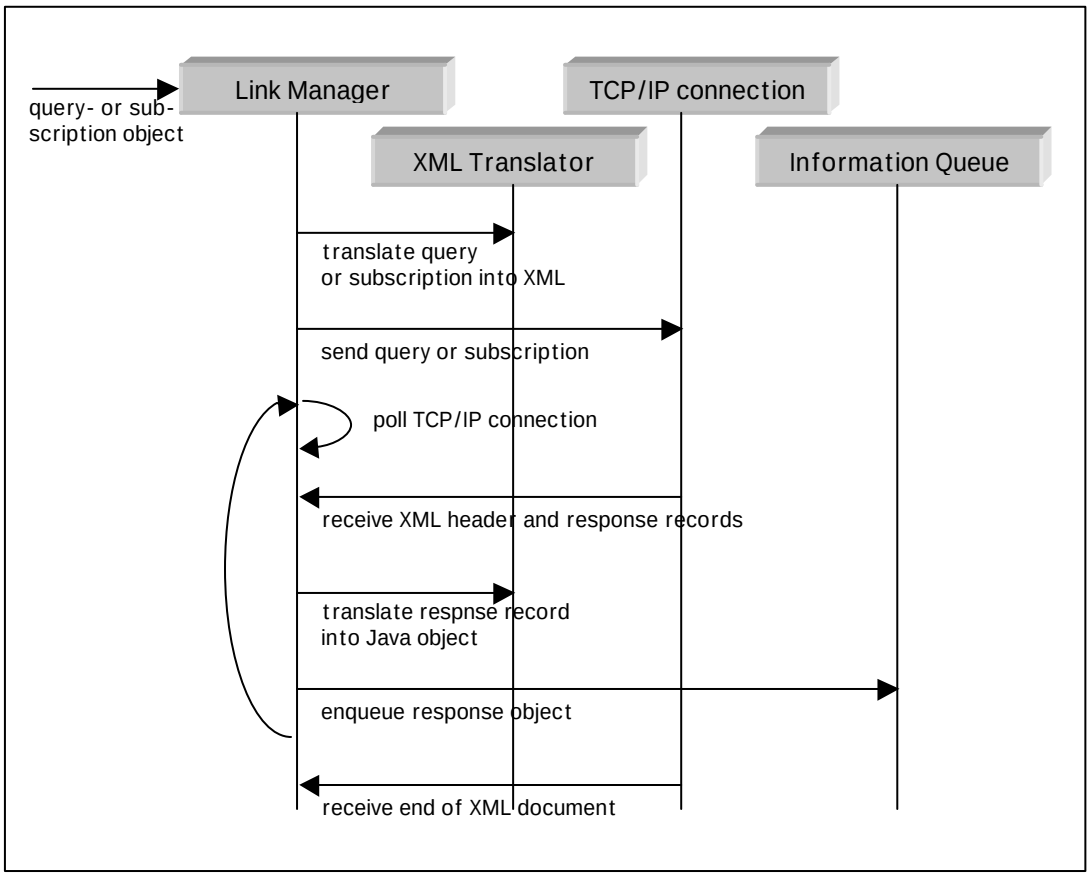


Figure 14 - XML link manager details

Data manager package

This package is the central handler of surveillance information in the client application. Initiated by the client GUI package with a subscription for surveillance information, it forwards the subscription to a link manager that in turn is initiated by the data manager itself. After these start-up actions, the data manager starts reading from the information queue that continuously is filled with information records, arriving from the server and processed by the link manager and the XML translator as described above. Each information record is read and is, depending on its type, processed and saved for future reference. Refer to Figure 11 above for a schematic view of the data manager algorithm.

Data segmentation

Surveillance data is categorised in several dimensions. One of these is by instrument, directly affecting the behaviour of the data manager when handling a piece of incoming information. As described above, all instruments that are active in the system belong to at least one instrument group, and all instrument groups belong to at least one instrument class. Therefore, the surveillance actions performed as a response to each information record depends on which instrument the piece of information relates to. In short, all data analysis and aggregation is completely separated with respect to different instruments. The instruments belonging to the same class are saved at the same place in the internal memory structure, aggregated at the same point in time as well as treated statistically at the same time. The instrument grouping is considered only when calculating the various parameters for the historically based statistical models used to check for market alerts.

Every instrument, for which information arrives in the information queue, is added to the internal memory structures. Also, recall that the incoming XML information is self-contained. Specifically, this means that full information on each instrument, client, customer, etc. is incorporated into the information, and that it is possible to maintain the database solely with the contained information. Therefore, the instrument data is also added to the database if not already added. All such new instruments are assigned to a default instrument class and a default instrument group, defined in the constants package. In the current implementation, new information about existing instruments, clients, customers, etc. is not updated in the configuration and the database. However, this function should be implemented in a release version of the client application.

The data is further categorised depending on what statistical interest it has for the instrument in question. Three internal memory structures exist, each concerning one distinct statistical interest:

- Price-, volume- or orderbook data
- Orders- and trades data
- Beneficial ownership data

Some information affects several of these areas. Depending on the statistical interest, the information contained in the record read from the information queue is added to the appropriate internal memory structure.

Database interaction

All of these three data structures are of an aggregation type. That is, they contain the currently aggregated value since the last aggregation time. Depending on the settings for the configured instrument class, all aggregated data for the instrument class at hand is written to the database every so often, and the internal memory structures relating to this instrument class are zeroed. Therefore, the adding of the aggregated data to the database itself is not very time consuming (apart from the actual writing to the database). Instead, the aggregation takes place immediately and continuously.

However, the writing itself to the database does not take place in a thread of its own. Rather, program control is taken over by the database IO package. As the writing operation may take quite some time, a sound add-on to the client application should be to put this into a separate thread[58]. See below for more information about how the data is stored in the database.

GUI interaction

The configuration also gives the opportunity to define certain instruments for which information (price, volume and/or orderbook data) is supplied to the client GUI in real-time. If the record read regards one of these configured instruments and information, it is branched to the client GUI thread as well. As mentioned above, the connection is through the use of a client GUI data structure known to the data manager, so that the client GUI is updated immediately.

Surveillance actions

The supplied information is checked for suspicious market behaviour in two ways, depending on whether the alert group considered contains discreet or continuous alerts. As can be seen from Table 2 above, certain alerts relate to data collected over some time period and somehow aggregated or averaged over this time period. These alerts are only checked for in conjunction with the aggregation of the data for the affected instrument class. Other alerts can be related to a particular order, trade or other information entering the system. They are checked for immediately on arrival of the order or trade in question.

When an alert is triggered on an instrument, information about this alert is sent to the client GUI for display. This communication is dealt with in the same way as for the real-time information distribution to the client GUI described above – i.e., the data manager has knowledge of some of the client GUI's data structures and write directly to them.

[58] Please see section 6.3 for further details on database performance.

At certain intervals, the surveiller package is called upon to refit the statistical models that are used to compare the different statistics with when checking for alerts. See below for more information on this matter.

Please see the sections 5.3 and 5.4 for a more detailed explanation on how the data collection and alerts checking processes actually work.

When the XML document ends, and the link manager thus stops sending data to the information queue, the central data manager dies. In order to continue the data collection, a new central data manager needs to be started. However, this new central data manager starts with its internal memory structures blank, so it needs to recalculate the statistical models in order to be able to perform alert checking.

The only time the data flow is stopped as a consequence of an ended XML document is of course time-limited subscriptions and queries (that contain a limited number of records per definition). However, the issue of a stopped central data manager brings light upon a problem in the current implementation of the client application.

Namely, there is currently a weak support for day-to-day operation of the client. The internal memory structures are adapted to a new day at a configurable time during the night, but this only works if the data manager is up-and-running 24 hours per day without interruption. In a realistic setting, this is not practical. Instead, an operations unit should be implemented on top of the client application. Such a unit should take care of restarts, calendar management, etc.

GUI data provider package

Once the client GUI has received information about an alert that has been triggered, it has the power to generate a query for more precise information about the market situation around the time for the alert. This query is executed by the initiation of another data manager, called a GUI data provider and running in a different thread from the central data manager.

In many ways, this new data manager proceeds in the same way as the central data manager. It initiates a link manager with the request for information, and then awaits the information records put into an information queue by the link manager. However, it does not carry out any surveillance, nor does it send any information to the database. Instead, it simply forwards all the information received to an internal client GUI data structure, which (in a similar manner to the situation above, when sending information to the client GUI) is known by the data manager.

When the data collection is done, the new data manager dies. Several such data managers may be initiated by the client GUI, several of which may also run simultaneously. Every new such GUI provider runs in an individual thread. Therefore, the processor load might be considerable when initiating several queries at the same time to the server.

It is also possible to start a new data manager to answer a query not directly related to the triggering of an alert. In the current version of the client application, it is not possible to pose queries to the external database. All queries must be addressed to the server (and thus pass by the XML interface), yielding answers in terms of detailed transaction information. It would be desirable to have the opportunity to pose requests for aggregated historical information to the external database directly. Please see the Appendix 4: To do list for the client application for more information.

Client GUI package

This package is of a temporary nature, merely providing an illustrative surveillance GUI draft, and is therefore not dealt with in depth here. However, it might be of interest to give an overview image of the relationship between the client GUI and the rest of the client application.

In short, the client GUI receives three different types of data, all from the central data manager or the GUI data providers when present:

- real-time information on price, volume and/or orderbook changes for the instruments so configured;
- real-time information about triggered alerts; and
- historic trading information as responses to queries.

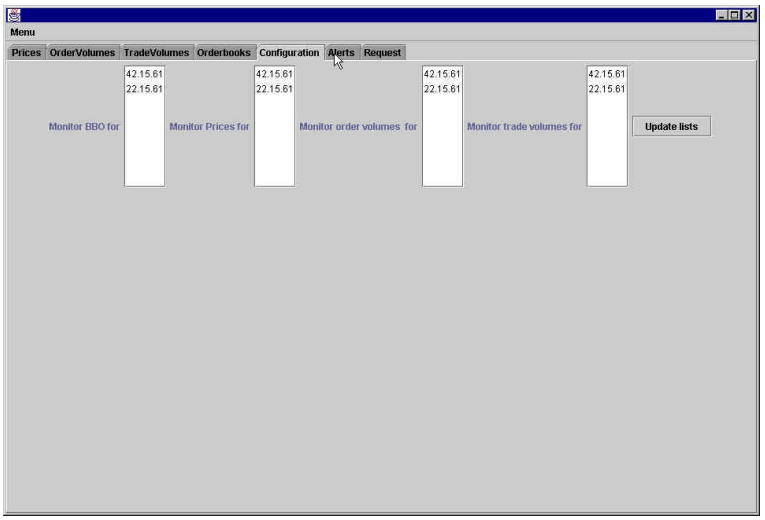
Further, the current GUI implementation can:

- create and start data managers and GUI data providers;
- show information received from these data managers; and
- let the user pose queries to the surveillance system for clarifying market information.

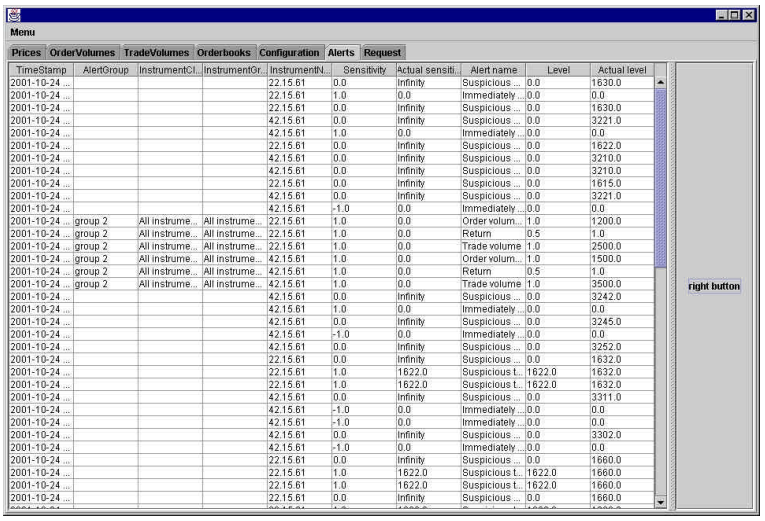
Below follows a number of screenshots of the current implementation of the client GUI, together with some brief explanations. These pictures could add some understanding of how such a GUI could look in a release version. Such a version of the GUI should of course be more comprehensive. It should specifically constitute a complete UI in terms of system configuration, operation and data investigation, and be completely integrated with the underlying surveillance functionality.

The current Java implementation of the GUI is structured around a central tab control. Each tab represents a basic function of the GUI – either information display or user input.

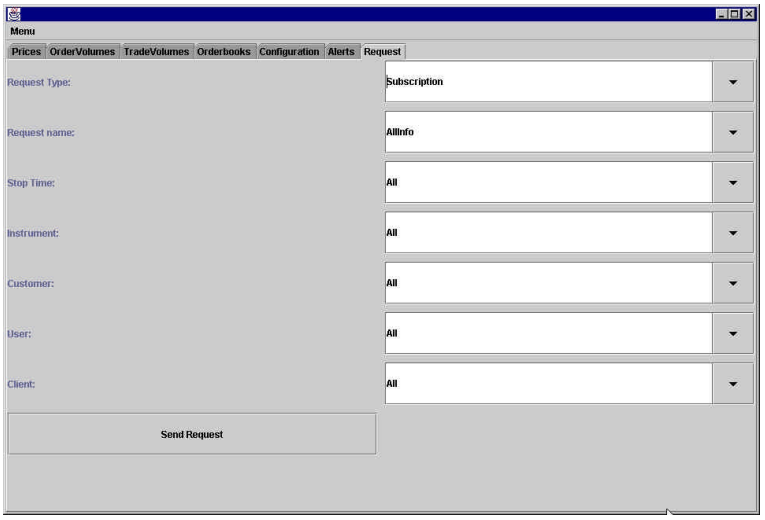
The configuration tab of the current GUI implementation merely allows the surveillance user to choose for what instruments to view real-time information. Such information is available for best bid/offer, prices and order- and trade volumes.



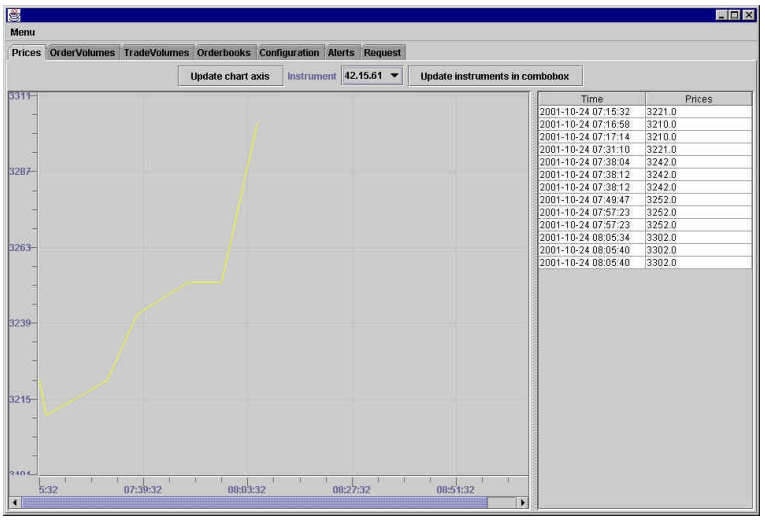
Under the alerts tab, the user can view information on triggered alert groups. Every row in the table represents one alert – several such alerts make up an alert group. By double clicking on an alert, additional information about the market is displayed.



In the request tab, the surveillance user has the ability to pose custom-made requests to the system. As of now, only the possibility of queries to the server-side application is implemented.



Under the prices-, order volumes-, trade volumes- and orderbooks tabs, real-time information is displayed. The available instruments to view information for are specified under the configuration tab.



Database IO package

This package encapsulates the database I/O operations of the client application. It uses a standard JDBC connection to access a relational MySQL database. Tables are created and deleted, and records are added, deleted and read by the use of standard SQL[59] statements via the JDBC connector classes. Notably, the central data manager creates the entire database structure on start-up.

Below follows a brief summary of the relational database structure of the client application. For a more detailed view, see Appendix 5: Database structure. The database In the database, the following data is stored:

- Aggregated instrument-specific data for each instrument.
- Aggregated data about orders and trades from the market system.
- Aggregated beneficial ownership data.
- Configuration of the system, including instrument classification and alert structure.
- Information about instruments, clients, customers and special information about what clients represent the same ultimately beneficial owner.
- Triggered alerts history.

Database table	Information contained	Primary Key
Surveillancedata_X (one table for each instrument on the exchange)	Orderbook, trades and volumes for a single instrument. X is the internal instrument number.	Time + date

[59] SQL, Structured Query Language.

Database table	Information contained	Primary Key
Orderstrades	Net and gross trades and orders for a specific instrument, a client and a customer.	Customer + customer's country + client + client's country + instrument + date + time
Instruments	Instruments and related information.	Instrument
Groupedinstruments	Instruments along with the instrument's respective instrument group.	Instrument + instrument group
Instrumentgroups	Instrument groups along with the instrument group's respective instrument class as well as other related information.	Instrument group
Instrumentclasses	Instrument classes along with relevant information.	Instrument class
Clientswithsamebeneficialowner	Five clients with the same beneficial owner. Some clients may be blank.	Client 1 + … + client 5
Beneficialownerships	Net and gross traded volume per instrument and client.	Instrument + client + client's category + date + time
Alerts	Alerts and the alert's respective alert type and level setting, as well as the respective alert group.	Alert
Alertgroups	Alert groups along with related information and instrument group belonging.	Alert group
Alerttypes	Alert type.	Alert type
Triggeredalerts	Alert group, alert along with threshold- and actual sensitivity level, as well as information on client and customer.	Alert group + alert name
Triggeredalertgroups	Alert group along with configured- and actual sensitivity value, as well as information on instrument classification.	Alert group
Clients	Client name and category.	Client
Clientcategories	Client category.	Client category
Customers	Customer along with related information.	Customer
Countries	Country.	Country

Database table	Information contained	Primary Key
Clientcustomerrelations	Related client and customer.	Client + customer
Automaticquerystatuses	Instruments.	Instrument
Automaticquerybefore	Days and time.	Days + time
Automaticqueryafter	Days and time.	Days + time
Automaticreturngraphinstruments	Instruments.	Instrument
Automatictradevolumegraphinstruments	Instruments.	Instrument
Automaticordervolumegraphinstruments	Instruments.	Instrument
Automaticbbographinstruments	Instruments.	Instrument

Table 3 - Relational database layout

As the database I/O is not threaded in the current implementation, there are some performance issues that need to be considered. Please see the section 6.3 below for further information.

Surveiller package

This package handles the actual surveillance of the collected, aggregated data. It has two main functions – statistical model recalculation and alert checking. In order to follow the development of the statistical pattern of the time series of the instruments in an instrument class, it is necessary to recalculate the parameters of the models used to forecast future time series values. However, this recalculation takes some time, and should therefore not be carried out more often than necessary.

Alert checking is performed each time the instrument class data is aggregated, while recalculating is only performed once in a (configurable) while – even though every recalculation is performed in conjunction with the alert checking process. The configurable dimension of freedom for the time elapsed between every calculation is namely the number of alert checks between every statistical recalculation.

The recalculation procedure itself is contingent upon the specific model chosen for the statistical interpretation of the individual data for the instrument class at hand. Here, only an overview of the available options is given. For a more thorough discussion on these matters, see the section 5.4 above.

The base case is a simple mean/variance analysis. This interprets to calculating the estimated mean and variance for the historical data, and evaluating new data by the use of these estimated parameters. If the new data is "too far" from the statistical mean (i.e., too many standard deviations from the mean), the alert is triggered. By varying, and making additions to, this method, it can be used for all of the alerts that are investigated using aggregated data.

For certain types of alerts, it is possible to define more elaborate statistical models, taking into consideration such phenomena as seasonal- or daily variation, time-variable volatility and trends. In the current implementation, such a model (a GARCH(1,1) model) is defined for returns data. However, it is straightforward to add statistical functionality to the surveillance system in the future. Models of interest should be ARMA-GARCH models for volume and returns, some type of regression model for the orderbook or an AI (Artificial Intelligence) implementation. However, there is an abundance of statistical models to pick from. As financial time series are studied intensely, more and more elaborate models are constantly on the uprise.

The historical (aggregated) data for the recalculation process is retrieved from the database. This retrieval is quite slow in the current implementation (please see the section 6.3 below), slowing down the whole recalculation process. When the data has been fetched, the calculation is performed by the statistics package, either internally or by calling upon Java native code in C++. The latter alternative is used for the GARCH analysis, but could be extended to every calculation that is time-consuming (in order to speed things up). After the recalculation is performed, the model parameters for every metric and every instrument group are stored in the internal memory structure of the surveiller package, and can be retrieved when needed for alert checking.

Each time the information is aggregated into the database, the surveiller is thus called upon by the data manager, to check if any of the alerts that are defined for the active instrument groups have been triggered. This alert checking is done by comparing the most recent data aggregated for each instrument to the historical statistical model parameters calculated for the relevant instrument group during the last recalculation. This way, the statistical foundation of the analysis is provided by all the instruments in the instrument group, whereas the actual alert is triggered on the individual instrument. The criteria for the instrument to have an alert being triggered is therefore to be too far from the expected value for the rest of the instruments in the instrument group. Less liquid instruments can be compared to a more solid history of data than would be possible only with the data from that particular instrument.

For the alert types that are checked for on a continuous basis, the checking itself is carried out by the central data manager package as the relevant information arrives.

6.3. Performance evaluation

One of the great challenges with this thesis work has been to implement a system not only flexible yet powerful in the market surveillance task, but that also is capable of handling the sometimes very large amounts of data produced in an exchange system. In this section, we will investigate what the possible obstacles for achieving this goal are. In conjunction therewith, we will describe some areas of success, along with issues yet to be solved in a release version of the surveillance client.

6.3.1. Capacity

Continuous information processing

The most basic criterion for the client application to function well is that it should be able to handle all the data sent to it by the server without an excess queue build-up. Since the treatment of data by necessity is serial, all links must have a relatively high lowest level of performance. Several possibly weak links can be identified:

If, on the one hand, the link manager / XML translator thread is to slow, the server-side queue for the client will start to grow. Eventually, when the client's performance is too poor as compared to the server speed, the server will disconnect the client from the flow of information.

If, on the other hand, the central data manager thread is too slow, the client-side information queue will start to grow. In the current implementation, the queue will not grow over 1000 records before the link manager starts to wait for the central data manager to catch up. Such a standstill on the link manager side soon also makes the situation critical for the server/client relationship, eventually leading to a disconnection of the client.

As the trading day goes by, the typical mix of trading information received changes, as well as the typical instantaneous data load. For example, in the beginning of the day, a great number of records regarding the current status of the traded instruments, as well as the best bid/offer rates at far-away market places[60], are sent in large chunks almost at the same time. This kind of behaviour produces heavy data load peaks at certain times of the day. Depending on the expected size and shape of these peaks, and the dips between them, the client can be temporarily unable to keep up with the instantaneous data load.

[60] A far-away market is a reference market where a contract is traded apart from on the considered exchange.

In Figure 15 below, minute-wise data load over one trading day at JIWAY is presented. As can be seen, the data flow is quite irregular, displaying relatively large peaks every now and then. Also, there are large longer-term variations over the day, with higher loads around the opening and closing of the market place. It should however be noted that the client needs never have a maximum capacity higher than that of the server, since all information arriving at the client has been processed sequentially in the server beforehand. The exact effect of this relationship depends on the maximum capacity of the server, but since a release-product server application probably should have a larger maximum capacity as

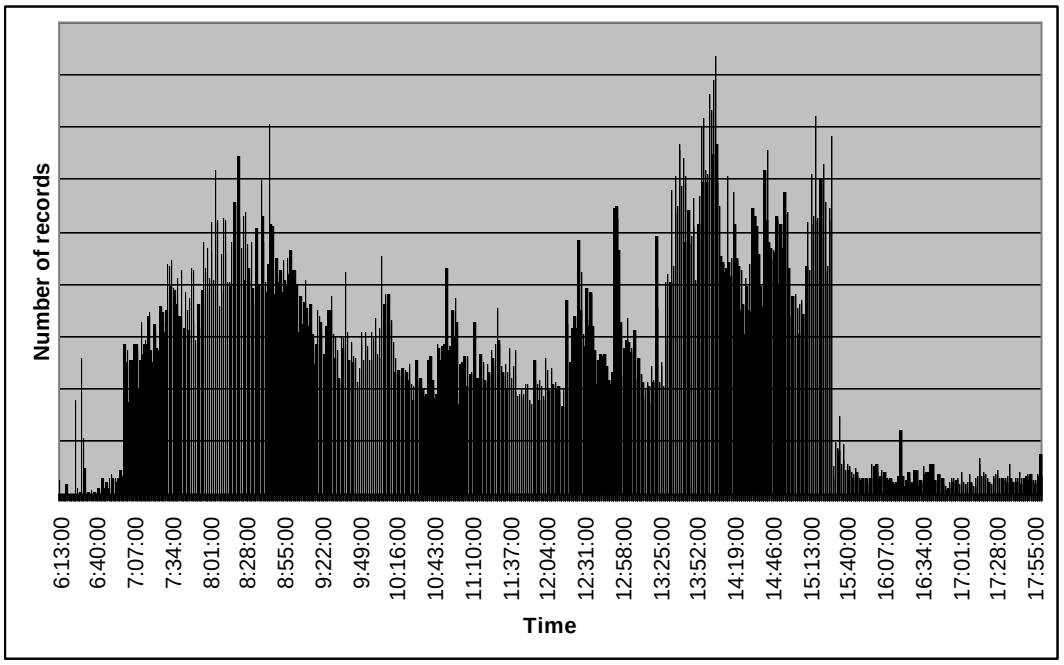


Figure 15 – Relative data load per minute during one trading day on JIWAY

compared to those of the individual clients, this aspect might be negligible. Please see the thesis report of Peter Bergenwald for more information on this aspect.

In the current implementation, and depending on the informational structure, the client application has been tested to handle in about 2000 to 5000 records per second, not counting aggregation or recalculation of data. It makes sense to compare these numbers to the number of records in the CLICK system today, and to the number of records that might be expected to be produced in the future, on CLICK-run exchanges and others.

Let us assume that the log file data every day in a CLICK system amounts to between 3 and 15 Gb[61], and that the average size of a log record is 50 bytes[62].

[61] These are the approximate data loads today on the JIWAY and the ISE exchanges, respectively.

Therefore, the number of records produced over a trading day amounts to between 60 and 300 million records, giving the average frequency of records μ as:

$$m_1 = \frac{60000000}{(8 \cdot 60 \cdot 60)} \approx 2100 \text{ records/s}$$

$$m_2 = \frac{300000000}{(8 \cdot 60 \cdot 60)} \approx 10400 \text{ records/s}$$

First of all, every information record sent to the client has a counterpart in a record from the log file. However, not all log records read in the server application are sent to the client, since there is some redundancy in the log information, and since not all the log data is of interest to the surveillance process at all. In the end, only about roughly 50% of the records are sent to the client application[63]. So, we finally land in the vicinity of 1000-5000 records/s on average, with the typical variations roughly depicted in the graph above.

As described in the introductory part of this report, the financial markets of the world are growing at a tremendous pace. This is true also in the case of individual market places[64]. It is not difficult to argue for the probability of a strong future increase in the data load on a surveillance system like the present. Therefore, the figures discussed above will need to be modified upwards as time goes by.

Other surveillance tasks

The processing of information records is, however, not the only thing that the application needs to do. There are also some actions that are carried out not continuously, but every now and then.

- The most frequently run, and therefore most time-consuming, of these is the data aggregation. Here, no heavy calculation is done, but a potentially large amount of data records are written to the database. As this process is not implemented as a thread, the execution of the incoming information records comes to a halt when aggregating the data, making the client's performance drop to almost zero during these periods (provided the XML interface package fills up the information queue). Data is typically aggregated on an hourly- or daily basis.
- Another action that is run every now and then is the recalculation of the statistical models. This process should be run less frequently than the data aggregation (typically daily, weekly or even monthly), but consumes about as much processor time reading from the database as the

[62] The size of the log file is expected to grow over time, so the surveillance system should be able to scale as the demands grow. The average log record size is taken from our own investigation of log files.
[63] Please see the thesis report by Peter Bergenwald for more precise information on how the information is cut.
[64] Some rapidly growing exchanges include BrokerTech in New York and Stockholmsbörsen.

aggregation counterpart does writing to it. More resources are needed for calculations than when aggregating, but the time needed for the maths diminishes when compared to the database I/O. At least, this is true for the current implementation. If even more elaborate mathematical benchmark models are to be added, the processing time for these will naturally increase as compared to presently. Also, because of the lack of data experienced, tests have not been performed for the behaviour when the data series to regress are longer than a couple of hundred time points, nor when the model calculated is near a singularity[65]. These issues need to be tested and matched with an appropriate mathematical library for the task at hand. Like the database aggregation, the recalculation process is not run in a separate thread, producing a heavy load on the server-side queue at the beginning of the data chain.

- Thirdly, there is the checking for triggered alerts, taking place every time the data is aggregated. In the base case, this task is not very time-consuming relative the database I/O. However, there is a dependency on the average number of database records with information about beneficial ownerships and orders/trades that need to be read database during the process of alert checking. See below for more information on this problem.
- Fourthly, the user can instruct the client GUI to send queries to the surveillance system. Currently, the only type of query allowed is one that is posed to the internal server, via the XML interface. As described above, the client GUI basically sets up a new data manager thread, which together with an equally additional link manager handles the reading of information records from the TCP link. During the collection of the information records that constitute the answer to the posed query, this leads to a situation where several data manager threads run in parallel, giving the (time-critical) central data manager less processor attention. Queries posed to the external database (concerning historical aggregated data) are not yet implemented, but the addition of such queries would produce the same kind of problems for the client performance.

Database I/O

During the read phase, the database access of the client application is not considerable. The operations performed include loading newly triggered real-time alerts and new instrument information into the database. It is the aggregation of surveillance data and the recalculation of the statistical foundations that demand heavy loads on the database. When aggregating, one surveillance data record is written for each instrument, followed by an additional number of

[65] Even though singular points should be handled by the mathematical algorithm in an efficient manner.

orders- and trades data records. When recalculating, one surveillance data record is read for each instrument.

It turns out that, ignoring the problem of the beneficial ownership- and orders/trades data amounts (see below), it is the reading and writing of surveillance data for each instrument that puts the far greatest load on the database. Therefore, the time taken for these operations for some different numbers of active instruments has been measured. The results from these measurements can be seen in Figure 16 below.

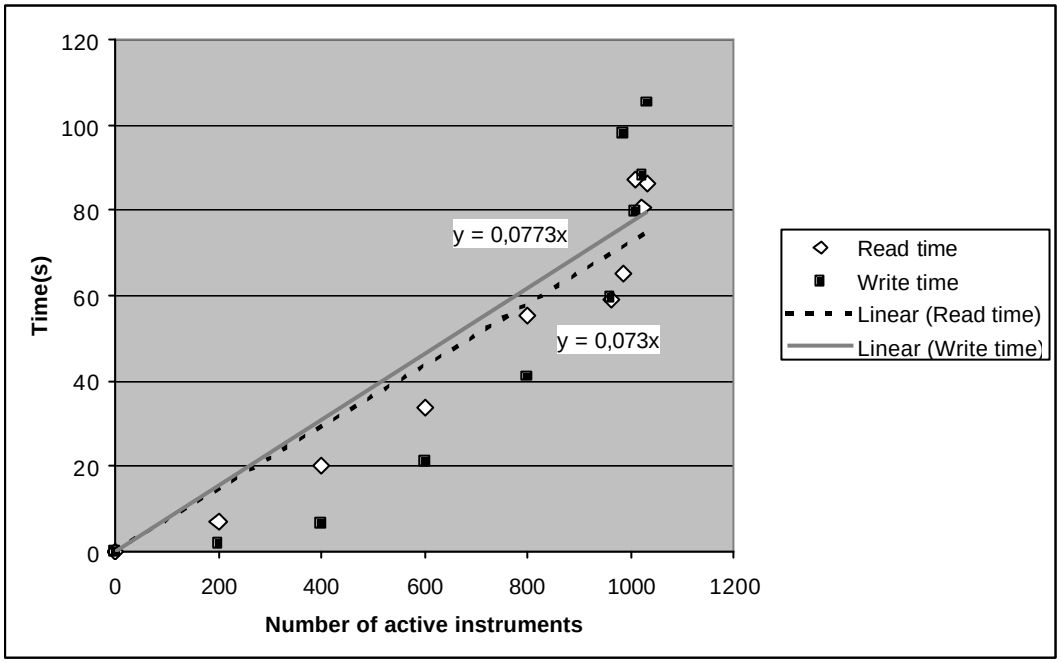


Figure 16 - Database access times for reading and writing

In the graph, the total access times are shown for reading or writing, respectively, a specified number of surveillance data records under normal surveillance conditions. As can be seen from the figure, these access times are quite long. From the regression lines we see that on average, it takes approximately one tenth of a second to read or write one record. As one record contains 45 standard data type variables, this performance could probably be improved considerably.

One problem is the way data is stored in the database. In an attempt to avoid one table to grow too large, each instrument was given its own table, where surveillance data records are written for this single instrument. As there may be several thousands of instruments active in the surveillance process, the number of tables can thus be considerable. When facing the task of accessing a database, it is however much less efficient to access each record from different

tables than to have them all in one large table, containing the surveillance data for all the instruments.

By accessing several tables, a number of factors slow down the process of writing to- and reading from the database. First of all, it becomes impossible to write several records before committing the transaction to the database. Secondly, on a lower level, the internal memory handling of the database becomes more fragmented and demanding, since it has to deal with more than one table at a time. With a reasonable aggregation period[66], and some thousands of instruments, the data load should not be overwhelming if one instead chooses to use only one, central, surveillance data table.

Database data storage capacity

The growth of the database as a whole must, however, be considered for a market system with a larger data load than on JIWAY. Information is added to the database each time the surveillance information is aggregated. As this is done periodically in a configurable manner, the user has a certain degree of freedom customising how large the database should become. Referring again to the database appendix, the following can be said about the size of the different tables in the database:

Tables that can potentially grow large include surveillancedata_X, beneficialownership and ordertrades. The number of surveillancedata_X tables can also potentially be large. The size of each of these records depends linearly on the number of aggregations saved into the database, in turn depending inversely on the aggregation time used for each instrument class. On top of this, the number of beneficialownership- and orderstrades records depends on the number of parties trading.

Let's take an example. Assume that there are 3000 instruments and 10000 different clients. Further, suppose that each client, during each aggregation period, on average trades through 2 different customers and in 2 different instruments. Then we get the following growth, in terms of the number of records on average per aggregation:

Table name	**Number of records written**
Surveillancedata_X	3000 new records (one in each table)
Beneficialownerships	$10000 \cdot 2 = 20000$ records.
Orderstrades	$10000 \cdot 2 \cdot 2 = 40000$ records.

As we can see by this rough approximation, the number of surveillancedata_X records does not grow considerably over time – only with 3000 per aggregation.

[66] Say, once every hour.

In this tempo, one can aggregate quite often and still save the data for a long time afterwards, for use in queries and/or for recalculation of the statistical models used.

Nor does the growth of the beneficial ownerships- and orders/trades tables threat the long-term stability of the system. On the other hand, the sensitivity of these tables to the number of trading clients and the average number of instruments in which these clients trade, as well as the number of customers these clients use for the placement of their orders, is high. Our test runs on JIWAY have not presented any problems, but when used with a larger CLICK market place, these problems can become serious. Therefore, two main tasks have been identified that need to be implemented in a release version of the client application, described below.

One nice side-effect of running the client application on an electronic market place is that the aggregated information is well formatted to use in other situations than market surveillance. For instance, many statistical applications could use the data from the client database. In order to achieve this extra advantage, one would have to save the data over longer periods of time than what is necessary for the surveillance function itself.

However, the older the data, the higher level of aggregation could be used. One is seldom interested in a fine-grained view of very old market data[67]. Therefore, additional aggregations should be performed on the older data saved over longer periods of time.

6.3.2. Summing up

It is, as we have seen, necessary for the client application to have the capacity to handle the arriving information records fast enough for the server not to be obliged to wait for the client to read the records sent. Ideally, this also incorporates the peaks of the information arrived. Now the crucial question is if the application stands up to meet these demands, and whether it is scalable enough to grow as the demands increase. This is the issue for the present section.

When simply processing information records arriving from the server, the client application does quite all right. With the current levels of transaction frequency (and with the current maximum speed of the server[68]), it even has some surplus capacity. Temporary peaks can be handled by the use of a larger maximum queue size in the link manager, even if this strategy increases the maximum time from the actual market event to the actual processing of the event in the client application. However, it is not a top priority to be as “real-time” as possible. A lag of a couple of minutes, even, should not disrupt the surveillance process or the follow-up activity unacceptably.

[67] Also, there already exist such data sources in various databases. One such example is the commercially available EcoWin database.
[68] See the thesis report by Peter Bergenwald.

It is when considering the aggregation and recalculation of data and the statistical models that the problems start to arise. Presently on JIWAY, it is feasible to stop the processing for several minutes at the time every hour or so to fill the database with new data. However, as the information load increases, this will become more and more difficult to handle. For example, the internal memory will suffer greatly from reading several hundreds of thousands of bytes per second into the memory, without any processing of the data, during any longer periods of time. The same goes for the recalculation of the statistical models.

Another problem is that there is no upper bound for the number of simultaneous queries that the user can pose from the GUI. Neither is there a limit to the size of the query responses. Therefore, the load from processing queries can potentially get very heavy for the system.

At the same time, it is important to remember that the tests carried out with the current version of the client application are done with a 600 MHz computer equipped with 128 Mb of internal memory and nothing done in particular to increase the speed of disk I/O. A simple increase of the machine performance should be enough to at least double the performance of the client. Also, it should be possible to quite easily put the database itself on another node, lifting these operations off the host system. Finally, the effect of restructuring the surveillancedata_X database tables as discussed above should have a significant effect.

7. Conclusions and future work

7.1. Brief summary

In this thesis report, a basic theoretical framework for the construction of an automatic surveillance system has been developed. The underlying problem of the increasing need for market surveillance on the world's financial markets was presented, alongside with the different ways market manipulation and insider trading is conducted. Thereafter, a general detection method for these unlawful conducts was presented, based upon characteristic footprints associated with the different conducts.

The pilot surveillance system is a practical implementation of the described detection method. Herein, the client part of this implementation was presented in detail. For the sake of completeness, a brief documentation of the XML interface between the client and the server was also provided. Having presented all the important aspects of the surveillance client, the attention was shifted towards a performance evaluation. Included in this was the identification of some areas of special concern, constituting potential threats to the functioning, flexibility or scalability of the client application.

7.2. Conclusions and future recommendations

During the work with this thesis, me and Peter Bergenwald were faced with the challenge of developing and describing a surveillance methodology flexible and scalable enough for the high demands of today's exchanges.

The next task was to make a demonstrable pilot implementation of the surveillance methodology. There were certain prerequisites for this pilot system, further discussed in section 4.3. Most importantly, the system should be split into a server- and a client part. Between the two, an interface based upon an XML protocol should be implemented. The server should read the raw log data from the exchange. Apart from these demands, the solution should of course be robust.

In short, the overall goals in terms of prerequisites, flexibility, scalability and robustness have all been fulfilled. However, there remains some major and a (larger) number of minor shortcomings in the pilot implementation. In the following, some of these are presented, with reference to Appendix 4: To do list for the client application, where a more complete list of flaws is presented.

Management of system resources

Two capacity problems could quite easily be solved by a more controlled behaviour of the application code. Firstly, the size of the queue of inbound information records is in the current version set to a maximum of 1000 records. When this limit is reached, the client forces the server to wait for the client to move ahead.

This number could very well be increased substantially, especially with a computer having a little extra RAM. More preferably, the maximum queue size should be set relative to the available system memory, maximally transferring the data load from the server onto the client.

Secondly, a central control unit should be implemented that monitors the usage of threads. There should be some maximum amount of processor power given to the query threads, so that they do not interfere too much with the time-critical workings of the central data manager. This central control unit should also be equipped with scheduling functionality and act as the system operation control.

Database I/O

Database I/O is the largest threat to the capacity of the client application as the information load and the number of instruments per server grows in the future. Delegating the database I/O work to special threads would take off the processing breaks now experienced by the data managers when the data is aggregated or the models are recalculated. Instead, the workload for the database operations could be spread out over time, allowing the central data manager to obtain a smoother processing curve. Also, the processor attention given to the database operations could be capped, not jeopardising the capacity of the system.

At the same time, as indicated above, there is probably quite some processing time to be saved by storing the surveillance data for different instruments in the same table – not as the implementation is done now, with one table for each instrument.

Another area of concern is the amounts of data being saved in the database. In the current implementation it might not be possible in all situations to save all the wanted information. Therefore, in a release version, it would be desirable to have the opportunity to select certain combinations of clients/customers/instruments to surveil, ignoring combinations not likely to produce constructive alerts. This way, the amounts of data can probably be cut considerably.

Also, it would be preferable to carry out an extra aggregation on top of the aggregated data periodically, when the data becomes older than a certain limit age. This would also add to the cutting of data load in the database. This extra aggregation could possibly take place in several steps, making older data gradually more summarised.

Separation into several instances

One advantage of the surveillance process is that there are few possible interactions between the market data of different instruments. If we divide the active instruments into broader categories of instruments, it becomes, by the simple rule of not allowing instruments from different to be in the same

instrument group, possible to run one client application for every such instrument category. This way, the processing burden of the surveillance system can be shared between several computers, each running an instance of the client application. One possible way to define these instrument categories would be to use the division already incorporated in the CLICK system[69]. By assigning one server application to the log output of each CLICK node, it is possible to run a client directly on each such server, hence obtaining a complete division of the amount of data processed.

Language of implementation

The language chosen in which to implement the client application is presently Java. Considering the sometimes time-critical setting, this might seem like an odd choice. However, the current implementation should be seen as a test pilot, the object of which is investigating the possibility of creating a flexible yet robust platform for market surveillance. In this perspective, Java has proved very good in its ability to quickly be up-and-running, and offering flexible and extensive resources for quickly producing a GUI, the database connection and an XML parser. Especially when considering the limited amount of time for the implementation of the surveillance client, Java has greatly helped holding the implementation-specific problems on a minimum, giving time to concentrate on the theoretical framework for the task.

However, in a release version of the surveillance system, it would be desirable to use a high-level language offering better run-time performance than Java, such as C++.

[69] Today, it is possible to divide the market between several CLICK nodes, each taking care of a specified number of instruments.

8. Appendix 1: Unallowed market actions

This appendix lists some of the common ways that different parties of an exchange use price manipulation and insider trading in order to gain an unlawful advantage in the market place.

8.1. Market Manipulation

8.1.1. Affecting the price

Bait and switch

Via the issuance of buy recommendations in media, a group of customers are offered to buy at one price and then resell at a slightly higher level (after the stock recommendation has been incorporated into the price). This in turn captures the interest of other investors, who step in at the higher price level. This process can be repeated over and over, until the price level is the desired or until the market gets its eyes open for the scam.

Hype and dump / Slur and slurp

The hype-and-dump technique, like bait-and-switch, uses the information flow to "hype" the instrument, thereby raising its market price. The hyping can take place through rumours, exaggerated reports, recommendations or otherwise. When the price has risen as intended, the holdings of the manipulator are sold rapidly, thus dumping the share price. The inverse (talking down the stock and then buying) is called slur-and slurp.

Highest bidder

By constantly and aggressively appearing as the highest bidder in an instrument, the supply of bids in the orderbook at lower prices is exhausted. This technique can also be used on the supply-side, in order to lower prices.

Pump and dump

By progressively staging transactions at higher and higher prices, other investors get a wrongful image of real market interest for the security, thus raising prices. It is then possible to sell with a nice return.

Ramping / Window dressing

This technique is used either on the demand- or the supply side to affect prices. Near the closing of the market at the end of the day, the manipulator places significant bid- or ask orders for the instrument. Played properly, this technique alters the closing price of the instrument. As the closing price is often used for

price comparisons over time, the perceived market price is affected. The order itself can be cancelled during the night or first thing when the market re-opens in the morning. When an institutional investor is affecting prices this way it is called window-dressing.

8.1.2. Affecting the real turnover volume

Chain letter rally

This is the volume-increasing effect of speculators that enter the market in reaction to some event caused by market manipulation. The increased number of participants increases the turnover and thus the effect of the original manipulation.

Short squeeze

In assets where there are many short-sellers, some of these will be forced to cover their short position by buying the security if the price increases enough. These transactions help raising the price even more, forcing others to follow suit. By cornering the market and squeezing the short-sellers, the price can thus be driven in several steps.

8.1.3. Affecting the perceived turnover volume

Churning / Passing the parcel

After cornering the market, the manipulator can place simultaneous bid- and ask orders, creating excess trading activity (with him- or herself), this way increasing the apparent volume. When the orders are placed at prices at consecutive levels – either upwards or downwards – this technique of affecting the volume and price at the same time by trading with oneself is called passing the parcel.

Pools

There are several ways to hide the fact that you are in fact trading with yourself when churning an instrument. One way is to use several different brokers, or to set up a pool of investors, all sharing the same volume- and price goal. By trading with each other, the churning can effectively be hidden.

Wash sales

Trading with oneself can also be useful for tax reasons, to “wash” some open positions. Thus, by staging trades between several actors with the same beneficial owner in the end, or by first selling and then buying back a short time afterwards (or the other way around), the trade can take place without really changing the ownership of the shares.

8.1.4. Affecting the trading participant anonymity

Matched orders

This is a way to control the turnover of an instrument in more detail. Two parties agree on that one of them places a bid order at the same time as the other places an ask order, exactly matching the bid order. Then, the orders are probably matched right away, and consequently never displayed on the computer screens of other brokers.

Warehousing / Parking / Nominee accounts

By using a middleman in a transaction, the identity of the real seller and/or buyer can be kept secret, side-stepping the rules of transparency on the exchange. By the use of so-called nominee accounts, the same effect can be obtained. It is possible to set up many such accounts, where the same beneficial owner controls each one, but the trading activity carried out by the nominee is not associated back to this ultimate owner.

Failure to disclose

Another way to accomplish the same thing is to simply fail to disclose one's newly taken position, thus disobeying exchange rules. This way, the market might for example get the impression of a genuine interest in a share, in a situation where there in reality is only one (but major) buyer.

8.1.5. Price manipulation by brokers

Churning and burning

Since the broker obtains commission income on the transactions closed, there might be an interest in increasing the volume of these transactions. In the case that the broker him- or herself takes the initiative to an increased number of transactions, this is called *churning*. The client, who gets to pay excess commission, is said to be *burnt*.

Guarantees or payments

Some brokers might want to offer extra service to valued customers, either by the guarantee to specified prices in the market, or by affecting the transactions taken place by subsidies to the actors loosing on the induced transactions as compared to the prevailing market price. In both cases, the effectivity of the market is jeopardised.

8.2. Insider trading

Classic insider trading

Prior to the release of some kind of price-sensitive information announcement about the company, a company insider can trade on this information, knowing that the market price of the stock is going to change in the near future.

Scalping

If the information traded upon in classic insider trading is a research report, the insider trading is called scalping.

Front-running

This is when a broker gains before-hand knowledge by observing the trading activity of large client, and uses this information for placing orders on his or her own account, or for another client. For example, this can be the case when a player places a large buy order on a company and the broker quickly jumps on the train before the rest of the market can observe the order in the orderbook.

Piggy-backing

The broker can also take things one step further, observing the successful trading of one of his clients. Simply replicating the trading decisions of this client, the broker can "piggy-back" on the research or information of this client. Unlike front-running, this technique is applicable after the placement of the order itself, since the broker needs to observe the outcome of the trading before he or she can decide whether or not the client really is successful in its trading.

Inside information dissemination

Instead of taking advantage of insider information for the own trading, a broker can disseminate the information to certain key players or to the public.

9. Appendix 2: XML Interface version 1.0

This is the first and currently implemented XML interface for the communication over the TCP link between the client and the server. In essence, it is a simple request – response type protocol. A client sends in a *request* and the server responds with appropriate transaction data. The request can be either in the form of a *subscription* for future trading activity data or a *query* for historical data. Both requests and responses in the interface are sent over the interface in an XML format (a Document Type Definition (DTD) specifying this format is found below). The request is the only command defined by this interface, and the client is further restricted to sending only one request per connection. After the server has sent all of the data in response to the request, the connection is closed by the server.

9.1. Protocol

Here protocol is used in the sense that it defines the way the client and the server interacts, for example how communication is set up and ended. The exact specification of the messages used in the communication is described later. A client sends requests to the server of either subscription or query type. The protocol also requires the client to precede every request with the size of the message in bytes in a textual format. This is a temporary solution to let the server read the whole request from the socket before starting to parse it.

The server responds to the request by sending an XML document. Both query and subscription responses follow the same XML format. In the case of a query, however, all of the information is sent once it has been retrieved from the database. The signal to end the communication is the end tag of the document. If the database does not contain any of the data demanded, an empty XML document is sent.

If the request is a subscription, on the other hand, the beginning of the XML document is sent to the client when the subscription is added to the subscription list. This works as a message to the client that the subscription is accepted. Then, as long as the subscription remains valid, i.e. as long as specified in the request, subscribed information is sent as XML to the client immediately when it is received to the server from the CLICK system. Thus, the response forms a complete and valid XML document being sent successively at the same tempo as the information arriving from the exchange system. When the subscription expires, the server ends the communication by sending the XML document's end tag (the document is not a valid XML document until the end tag has been sent). The figure below illustrates the basic flow of messages between a client and the server.

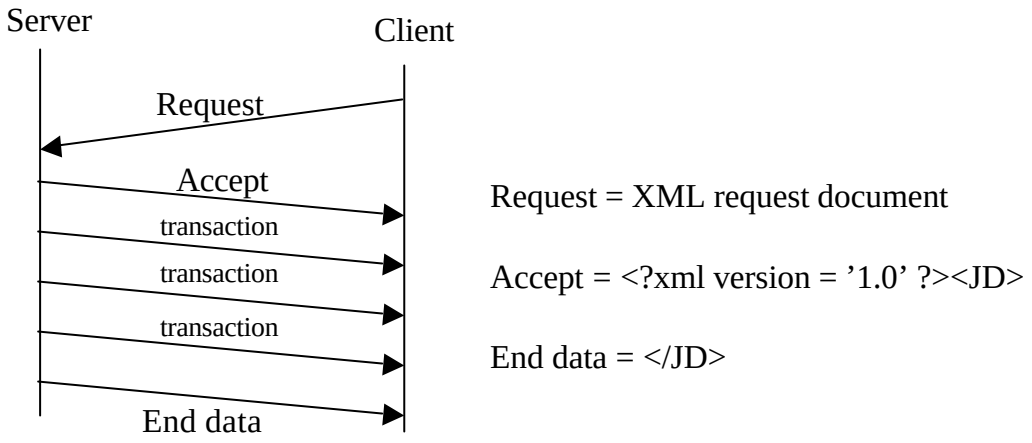


Figure 17, Communication flow client - server

Below follows a detailed description of the information in a request and the type of transaction messages the XML interface defines.

9.2. Request

The requests that the interface can handle are predefined. The format of the request has ten different parameters that define the query, where some of the parameters also have a predefined set of values that are valid.

9.2.1. Data in requests

The parameters in a request are:

Request type – one of the following predefined values:

1. Subscription
2. SubscriptionLight
3. Query
4. QueryLight

The light types return a response where a configurable amount of the information in the transaction messages is suppressed. The same transaction messages are used but with some fields suppressed. For example, it is possible to configure the messages so that they only contain information about what instrument and which participant it concerns, price and volume information. The configuration takes place in the server application.

Parameters	Predefined values	Explanation
	Subscription	Future data.
	SubscriptionLight	Future data where some fields in the transaction messages are suppressed.
	Query	Stored data.

	QueryLight	Stored data where some fields in the transaction messages are suppressed.
Request name	All	This value is only valid for subscriptions in order to avoid queries for a too large amount of information.
	Surveillancedata	This is value returns the minimal subset of the available information necessary for the client to perform its surveillance functionality. For our implementation of the client the necessary information is quotes, orders, trades and orderbook changes.
	Transaction name	The name of a specific transaction message, one of the following: • Trade • Order • ExtBBO • IntBBO • Hedge • HedgeExecution • Quote • PriceImprovementQuote • OBChange
Instrument	All	Returns information for all instruments traded at the exchange. This only a valid value for subscriptions also to avoid large queries
	Instrument name	The name of an instrument should be in a readable form and not some numerical code. A name could for example be Volvo A, SSE. This is not implemented in the prototype due to the time it would take to develop. Instead the internal numerical representations are used.
StartTime	YYYY-MM-DD hh:mm:ss	The field is only considered for queries. Subscriptions are assumed to start when they are received.
StopTime	YYYY-MM-DD hh:mm:ss	The value is used in both queries and subscriptions
Customer	All	
	Empty string	
	Customer name	The name of a customer to the exchange, i.e. a broker firm.
User	All	
	Empty string	
	User name	The name of a user account at a brokerage firm, i.e. a specific broker.

Client	All	
	Empty string	
	Client name	The end client, this information is not available in the trading system today.
Order number	Empty string	
	Number	If this field is used the request is assumed to be a query, and the return is all the information about that order number found in the database.

Table 4, Request variables

9.2.2. Request DTD

```
<?xml version = '1.0' ?>
<!DOCTYPE REQUEST [
<!ELEMENT REQUEST (REQUESTTYPE , REQUESTNAME , INSTRUMENT,
STARTTIME, STOPTIME, CUSTOMER, USER, CLIENT, ORDERNUMBER)>
    <!ELEMENT REQUESTTYPE (#PCDATA)>
    <!ELEMENT REQUESTNAME (#PCDATA)>
    <!ELEMENT INSTRUMENT (#PCDATA)>
    <!ELEMENT STARTTIME (#PCDATA)>
    <!ELEMENT STOPTIME (#PCDATA)>
    <!ELEMENT CUSTOMER (#PCDATA)>
    <!ELEMENT USER (#PCDATA)>
    <!ELEMENT CLIENT (#PCDATA)>
    <!ELEMENT ORDERNUMBER (#PCDATA)>
]>
```

9.3. Trading information responses

The format of the trading information sent as XML to the client corresponds closely to the format of the information from the trading system itself. The messages are found in the table below. For an explanation of the meaning of the tags used for specific data fields, see the CLICK log documentation.

Message	Explanation
Order	This is a transaction message with information about a new order in the system.
Quote	This message contains information from a market maker about the quotation of an instrument.
Trade	Information about a trade with references to the two matched orders.
Orderbook change	Information about any change in the orderbook. This record is generated after each trade, order entry and quote, among other events.
Internal Best Bid Offer	Information about the current best bid and best offer for a certain asset at the market place.
External Best Bid Offer	BBO at the away-market of the asset.
Hedge Order	This record indicates that an order is destined to be traded at the away market (hedged), because the BBO at the away market is better than at JIWAY.
Hedge Execution Report	Report of a successful hedge execution, where the original hedge order has been traded at the away market in part or fully.
Price improvement quote	Record with information that a market maker will trade a certain volume of an asset at the market place to the price specified by the external BBO.

9.3.1. Tags used

Since an XML representation of data is string-based, and can be quite "talkative", the tags used in the interface are kept at a minimum length. The following tables present the tag names used, and their respective meaning.

XML transactions tags	Meaning
EB	External BBO
HO	Hedge Order
IB	Internal BBO
IQ	Improvement Quote
O	Order

OB	Orderbook Change
Q	Quote
S	Instrument Status Record
HE	Hedge Execution Report

XML data tags	**Meaning**
AEB	Ask External BBO
AIB	Ask Internal BBO
BA	BidAsk
BEB	Bid External BBO
BIB	Bid Internal BBO
C0	Customer Country
CC	Client Category
CL	Client
CN	Country Number
CON	Check OrderNumber
CU	Customer
DP	Deal Price
EN	Exchange/Country Number
ER	End Of Report
GSN	Global Sequence Number
HI	Hedge Order ID
HR	Hedge Execution Reference
HS	Hedge Status
IC	Instrument/Commodity Code
IG	Instrument Group
IN	Instrument/Series
IS	Instrument/Series Status
LR	Log Reason
LT	Lock Type
MC	Market Code

MOD	Modifier
MOV	Market Order Volume
OC	Order category
ON	OrderNumber
OT	Order Type
PR	Premium
QY	Quantity
SN	Sequence Number
TA	Trade Ask side
TB	Trade Bid side
TON	Target Order Number
US	User
VM	Volume Multiplier
VO	Volume
VT	Validity Time
TO	Trade Order

9.3.2. XML response DTD

```
<?xml version = '1.0' ?>
<!DOCTYPE JD [
<!ELEMENT JD (O | Q | IQ | OB | T | EB | IB | H | HE | S)*>
    <!ELEMENT O (ON, TS, SN, IN, CO, CU, US, BA, QY, PR, VT, OT, CC, CL)>
    <!ELEMENT Q (ON, TS, SN, IN, CO, CU, US, BA, QY, PR)>
    <!ELEMENT IQ (ON, TS, SN, IN, CO, CU, US, BA, VO, VM)>
    <!ELEMENT EB (TS, SN, IN,  BEB, AEB)>
        <!ELEMENT BEB (PR, VO)>
        <!ELEMENT AEB (PR, VO)>
    <!ELEMENT IB (TS, SN, IN,  BIB, AIB)>
        <!ELEMENT BIB (PR, VO, MOV)>
        <!ELEMENT AIB (PR, VO, MOV)>
            <!ELEMENT MOV (VO)>
```

```
<!ELEMENT OB (ON, TS, SN, IN, CO, CU, US, BA, QY, PR, LR, CON, OC, VT, OT,
CC, CL, LT )>
        <!ELEMENT CON (ON)>
<!ELEMENT H (ON, TS, SN, IN, CO, CU, US, BA, QY, PR, HI, OT, EN)>
<!ELEMENT HE  (ON, TS, SN, IN, CO, CU, US, BA, QY, PR, TON, HI, ER, HS, HR)>
        <!ELEMENT TON (ON)>
<!ELEMENT T (TS, SN, IN, CO, CU, US, QY, PR, TB, TA)>
        <!ELEMENT TB (TO)>
        <!ELEMENT TA (TO)>
        <!ELEMENT TO (ON,CO,CU,US,QY,PR,VT,OC,OT,CC,CL)>
<!ELEMENT S ( TS, SN, IN, CO, CU, US,  IS)>


        <!ELEMENT ON (#PCDATA)>
        <!ELEMENT TS (#PCDATA)>
        <!ELEMENT SN (#PCDATA)>
        <!ELEMENT IN (#PCDATA)>
        <!ELEMENT CO (#PCDATA)>
        <!ELEMENT CU (#PCDATA)>
        <!ELEMENT US (#PCDATA)>
        <!ELEMENT BA (#PCDATA)>
        <!ELEMENT QY (#PCDATA)>
        <!ELEMENT PR (#PCDATA)>
        <!ELEMENT VT (#PCDATA)>
        <!ELEMENT OT (#PCDATA)>
        <!ELEMENT CC (#PCDATA)>
        <!ELEMENT CL (#PCDATA)>
        <!ELEMENT VO (#PCDATA)>
        <!ELEMENT VM (#PCDATA)>
        <!ELEMENT LR (#PCDATA)>
        <!ELEMENT OC (#PCDATA)>
        <!ELEMENT LT (#PCDATA)>
        <!ELEMENT HI (#PCDATA)>
```

```
        <!ELEMENT EN (#PCDATA)>
        <!ELEMENT ER (#PCDATA)>
        <!ELEMENT HS (#PCDATA)>
        <!ELEMENT HR (#PCDATA)>
        <!ELEMENT IS (#PCDATA)>
]>
```

10. Appendix 3: XML Interface version 2.0

10.1. Introduction

With the experience gained from the work done with the surveillance client and surveillance functionality, it was clear that the XML interface version 1.0 was quite CLICK- and JIWAY specific. One desirable feature of the XML interface is to be usable on different exchanges, as well as with different exchange systems, notably also the SAXESS system. With this in mind, the XML interface has undergone a thorough redesign. The aim of this has been to make it simpler, and at the same time more general and flexible. Also, the robustness of a strictly defined structure has been preserved in as much as possible of the new design.

10.2. Protocol

The protocol (i.e. the way the client and the server interact) remains the same in the new version. It is only the format and content of the messages containing trading information that change. The format for the requests also remains unchanged.

10.3. Trading information responses

The number of messages has been greatly reduced as compared to version 1.0, and their content is designed to reflect the information needed for market surveillance only. In this, the idea is to only use messages general enough to be valid for most electronic marketplaces. All flow of surveillance-relevant information on an exchange relates to *orders*. This is the most general entity – a market actor requesting to sell or buy a certain quantity of an asset to a certain price. An order may also change or be changed during its way through the exchange system. This leads to the following two main message types:

- Order entry
- Order change

Each of these messages contain data about that specific event. The table below illustrates the minimal set of such data identified as necessary for the surveillance process. The choice of what data is considered necessary is based on the experience gained from the pilot implementation of the surveillance client. Basically, the idea is that the surveillance process on any exchange is primarily concerned with the actions taken by its participants, and not internal exchange information. This follows from the fact that all market manipulation and insider trading deals with the information that is actually visible to the trading public. How this information (about orders, trades, prices, volumes, etc.) has been produced (on a technical level) is not interesting from this viewpoint. Thus, the surveillance client needs to know what was done at the exchange when, and who did it.

Order entry	**Order change**
Order identity	Event time
Instrument	Bid order identity
Event time	Ask order identity
Bid or Ask	Reason
Brokerage firm	
Broker	
Type	

As can be seen from the table, both of the basic messages include certain predefined fields, containing information that are generic to an “order”, or to an “order change”, respectively. Hence, an order will always be identified with an instrument and an event time, for example. In the case of the order change, either one or both of the bid- and ask order identity will be used, depending on the reason for the order change.

However, there are many types of orders, as well as reasons for an order change, so there should be a way to make the messages flexible and the interface extendable. Therefore, the messages were designed with a set of data fields common to all types, together with a part that is specific for each type of order or order change. This way, the interface is extendable; it is possible to add new order types and new reasons for order changes while the header part of the message remains the same.

Practically, one defines a new DTD for each exchange system that is to be used with the surveillance client application. This DTD must be compliant with the XML interface version 2.0. Therefore, the DTD needs to have the order entry and the order change entities at the central structural unit. It is then possible to define several different values for the Type entities of the Order entry, and the Reason entities of the Order change, respectively, in order to customise the DTD for use with the particular system considered. The Type- and Reason entity definitions are simply XML definitions of the allowable sub-entities of compound elements, reflecting the specific order entry or change that needs to be described. Examples include a simple order entry (e.g., with a price, volume and ordering market participant) and a trade (an order change possibly including a price and a volume).

Below a more detailed explanation is given to the XML interface version 2.0, described in the light of two examples. The first such example is an example mapping of the XML interface version 2.0 to the CLICK system run at JIWAY. After this, another example is presented in the mapping of the XML interface to a SAXESS-run exchange.

10.4. Mapping the interface to JIWAY

For JIWAY (running the CLICK exchange system), there are a couple of special events that can occur, and which are interesting for the surveillance process. The different possible order types are:

1. Simple order
2. Quote
3. Price Improvement Quote

The reason for an order change field can be of any one of the above types or, in addition, any one of the following:

4. Trade
5. Hedge
6. Delete

The respective Type- and Reason entity definitions include the following fields:

Type/ Reason	**Simple order**	**Quote**	**Price I. Q.**	**Trade**	**Hedge**	**Delete**
Data	Price	Price	Volume multiplier	Price	Price	
	Volume	Volume	Volume limit	Volume	Volume	
	End party					

Due to the fact that this particular DTD definition deals with the JIWAY exchange only, certain CLICK-supported transactions that are not used on JIWAY, such as combination orders, are not defined here. Combination orders are different in the way that they contain several orders that all need to be traded together. An extension including combination orders would be quite straightforward. An example implementation could include an additional Type definition, specifying an order message as a part of a combination order with a combination order identity (each combination order would be sent over a number of individual Order entities). There are other things to take into consideration as well, for example different price and volume types used by different exchange systems. The general guideline is that the interface needs to transfer the information that is public on the market to the surveillance process.

10.4.1. Tags used

As in the XML interface version 1.0, the tags used in the JIWAY version of the interface are abbreviated in order to reduce the XML overhead. The two tables below present the tags used.

Compound XML tags	Meaning
OE	Order entry
OC	Order change
IQ	Price Improvement Quote
SO	Simple Order
Q	Quote
TR	Trade
H	Hedge
D	Delete
T	Type
R	Reason

Table 5 – Tags defining compound entities

XML data tags	Meaning
ID	Order identity / Order change identity
P	Price
V	Volume
BA	Bid or Ask
IN	Instrument
TS	Time stamp
BR	Brokerage firm
DE	Dealer
EP	End party
VM	Volume Multiplier
VL	Volume Limit

Table 6 – Tags defining simple data types

10.4.2. XML response DTD

```
<?xml version = '1.0' ?>
<!DOCTYPE JD [
<!ELEMENT JD (OE | OC )*>
    <!ELEMENT OE (ID,IN,TS,BA,BR,DE,T)>
        <!ELEMENT T( SO | Q | IQ )>
            <!ELEMENT SO (P, V, EP)>
            <!ELEMENT Q (P, V)>
            <!ELEMENT IQ (VM, VL)>

    <!ELEMENT OC (TS,BO,AO,R)>
        <!ELEMENT R( SO | Q | IQ | TR| H | D)>
            <!ELEMENT TR (P, V)>
            <!ELEMENT H (P, V)>
            <!ELEMENT D (P, V)>

<!ELEMENT ID (#PCDATA)>
<!ELEMENT IN (#PCDATA)>
<!ELEMENT TS (#PCDATA)>
<!ELEMENT BA (#PCDATA)>
<!ELEMENT BR (#PCDATA)>
<!ELEMENT DE (#PCDATA)>
<!ELEMENT EP (#PCDATA)>
<!ELEMENT V (#PCDATA)>
<!ELEMENT P (#PCDATA)>
<!ELEMENT D (#PCDATA)>
```

10.4.3. Transaction message mapping

For each exchange implementing the XML interface version 2.0, a server has to be set up in order to map the informational content to the XML interface. Mapping the interface to the JIWAY exchange is quite straightforward, since the order types defined above match the transaction messages from the exchange

closely. In the XML interface version 2.0, external and internal BBO records should be disregarded (since they are not used in the surveillance process), which reduces the message load from the CLICK exchange system. The mapping would be as follows:

Jiway message	**Interface message**
Order entry	Order – simple order
Quote	Order – quote
Price improvement quote	Order – price improvement quote

For the messages mapping to order changes, some extra consideration needs to be taken. The table below shows a draft for how this could be done.

Jiway message	**Interface message**
Trade	Order change – trade
Order/Quote in orderbook, trade	Disregarded, redundant
Hedge Execution order	Order change – hedge
Order delete	Disregarded, redundant
External BBO	Disregarded, redundant
Internal BBO	Disregarded, redundant
Order/Quote in orderbook, deleted	Order change – deleted

As can bee seen from the table, all order/quote in orderbook messages are ignored except the ones concerning a deletion of an order.

10.5. Mapping the interface to SAXESS

An exchange running SAXESS would have to adapt the above-presented XML format for translation from the messages used in the SAXESS Interprocess Protocol. This section constitutes an example outline of how this could be done. A certain care should be put into the interpretations of this outline, however – the mapping may contain errors due to a misunderstanding of the SAXESS system. The reason for this is a shortage of time to study the SAXESS system.

The order types identified are:

1. Simple order
2. Quote
3. Combination order
4. Linked Order
5. Stop loss order

The reason for an order change field can be of any one of the above types or, in addition, any one of the following:

6. Trade
7. Delete

The respective Type- and Reason entity definitions include the following fields:

Type/ Reason	**Simple order**	**Linked Order**	**Combination order**	**Stop loss order**
Data	Price	Price	Price	Price
	Volume	Volume	Volume	Volume
	Open volume			Stop loss trigger price
	Price condition			Stop loss new price
				Price type

Type/ Reason	**Quote**	**Trade**	**Delete**
Data	Price	Price	
	Volume	Volume	

In the SAXESS system it is possible to specify market participants acting for other participants. A decision of how to handle this needs to be taken. One simple approach would be to disregard the participant acting for another participant and only use the participant from where the transaction originates. Otherwise it would be necessary to keep lists with the participants that act in the name of other participants, so that the XML information correctly reflects the real intentions of the active exchange members.

Another thing to consider is how the price and volume information is to be treated, since it is possible to specify price and volume conditions for incoming orders. In the current implementation, the price condition of the order is specified in the XML message. For volume conditions, the volume field contains

the total volume left to be traded and the open volume field contains the remaining volume shown on the exchange.

The SAXESS system also supports so-called combination- and linked orders. Combination orders are merely a combination of 2-8 individual legs, associated with an AND condition. Linked orders are like combination orders, except that the condition is a XOR logic. Combination- and linked order messages in the SAXESS system are broken up into one XML order message for each leg in the SAXESS combination order. The XML messages are linked together simply by having the same order identity in the header.

Finally, there is an order type called stop-loss order, indicating an order with a price- or volume condition attached to it. At the time when this condition is fulfilled, the order changes price and/or volume following a predefined pattern. In the present adaptation of the XML interface, the stop-loss order has its own message specification with values for the trigger price and the new price when the trigger price is reached. A definition of the trigger price type is also included.

10.5.1. Tags used

The tags used in the interface are the same except that the Jiway specific message types price improvement quote and hedge are not used. Instead, a tag for combination orders is added along with some other tags, such as one for the open volume field. Please see the table below for the complete listing of the new tags used.

New XML tags	Meaning
CO	Combination order
LO	Linked order
STO	Stop-loss order
OV	Open volume
TP	Stop loss trigger price
NP	Stop loss new price
PT	Price type
PC	Price condition

Table 7 – New tags for SAXESS

10.5.2. XML response DTD

```
<?xml version = '1.0' ?>
<!DOCTYPE SX [
<!ELEMENT SX (OE | OC )*>
    <!ELEMENT OE (ID,IN,TS,BA,BR,DE,T)>
        <!ELEMENT T( SO | Q | CO | LO | STO)>
            <!ELEMENT SO (P, V, OV, PC)>
            <!ELEMENT Q (P, V)>
            <!ELEMENT CO (P, V)>
            <!ELEMENT LO (P, V)>
            <!ELEMENT STO (P, V, TP, NP, PT)>

    <!ELEMENT OC (TS,BO,AO,R)>
        <!ELEMENT R( SO | STO | Q | TR| D )>
            <!ELEMENT TR (P, V)>


<!ELEMENT ID (#PCDATA)>
<!ELEMENT IN (#PCDATA)>
<!ELEMENT TS (#PCDATA)>
<!ELEMENT BA (#PCDATA)>
<!ELEMENT BR (#PCDATA)>
<!ELEMENT DE (#PCDATA)>
<!ELEMENT EP (#PCDATA)>
<!ELEMENT V (#PCDATA)>
<!ELEMENT P (#PCDATA)>
<!ELEMENT D (#PCDATA)>
<!ELEMENT OV (#PCDATA)>
<!ELEMENT NP (#PCDATA)>
<!ELEMENT TP (#PCDATA)>
<!ELEMENT PT (#PCDATA)>
```

<!ELEMENT PC (#PCDATA)>

10.5.3. SX Session message mapping

As an illustrative example, the SX messages from SAXESS can be mapped into this XML format in the following way:

XML order messages

- The XML message of type simple order (SO) maps from the SX order insert message when the order is not a stop-loss order. If the SX order message is indeed a stop-loss order, it maps to the XML stop-loss order.
- Several XML messages of combination order type map from each SX combination order insert message, and several XML messages of linked order type map from each SX linked order insert.
- XML quote messages map from either a leg in a bundled quote message or an order insert message where the isQuote flag is set to yes.

XML order change messages

- XML trade messages are mapped from SX trade insert messages.
- XML order, quote and stop loss order messages for order changes map from the SX update and adjust messages for single, linked and combination orders.
- Delete messages in XML are mapped from the cancel messages for the different types of orders.

11. Appendix 4: To do list for the client application

11.1. Performance issues

- The size of the queue of inbound information records is in the current version set to a maximum of 1000 records. Instead, the maximum queue size should be set relatively the available system memory, maximally transferring the load from the server onto the client.
- A central control unit should be implemented that monitors the usage of threads. There should be some maximum amount of processor power given to the query threads, so that they do not interfere too much with the workings of the central data manager.
- It should be possible to use one client application for every predefined instrument category. This way, the processing burden of the surveillance system can be shared between several computers, each running an instance of the client application. An example of a division into categories is to follow the already-made division in the log files.
- By delegating the database I/O work to special-purpose threads, this not so time critical work load could be evened out over time, allowing the central data manager to obtain a smoother processing curve. Also, the processor attention given to the database operations could be capped, not jeopardising the capacity of the system.
- If it would be possible to select certain combinations of clients/customers/instruments to surveil, the amounts of data saved in the database could be cut considerably.
- It would also be of use to carry out an extra aggregation on top of the aggregated data periodically, when the data becomes older than a certain limit. This would also add to the cutting of data load in the database.
- By porting the client code to C++ or some other high-performance compiled language, overall performance could be elevated for the application.

11.2. General implementation shortcomings

- The GUI is somewhat rudimentary. Its main purpose is to serve as an illustration of the functionings of a full-fledged GUI. However, some things are missing from the basic required features, such as the possibility to choose from all available instruments for real-time graphing. As it is now, it is necessary to predefine the instruments that should be available in the Client class in order to view the graph data associated with these instruments.
- Also, the queries from the GUI are lacking some fundamental functionality. It is only possible to query for detailed historical information from the server

database, and the result is presented in a simple table form. Instead, it should be possible to query for either detailed or aggregated historical information (in the latter case it suffices to ask the client database for the information, releiving the server from the extra work load). Also, the result should be presented in a more customised way, sorting and graphing relevant data, etc.

- In a release version, the XML information arriving from the server should be completely self-contained. I.e., no additional information should be needed to obtain the actual names of instruments, market actors, etc. Therefore, it would be necessary to implement the client application in such a way so that it automatically updates the information stored in the database according to changes of the characteristics of instruments, clients, customers, etc. that arrive from the XML interface. In the current implementation, all that is done is to simply add new instruments, etc., as they arrive.
- Additionaly, the information content sent over the XML link would have to be encrypted in order to guarantee the integrity of information for the exchange. A login/session functionality should also prove necessary.
- At the moment, all configuration parameters are not saved into the database (e.g. the model/default statistics parameter).
- A day-to-day operations unit should be implemented on top of the client application, to take care of the start-up, shut-down and restart of the system. This operation should be seen to by the use of a calendar function, and should make it possible to set up more long-term schedules for the surveillance function.

11.3. Surveillance model shortcomings

- Presently, there are some unlawful conducts that are not checked for by the surveillance process, such as ramping/window dressing.
- The possibility to use outstanding stocks as the benchmark definition for volume alerts is not implemented either.
- For all volume-related alerts, it would be of use to have a lower volume limit, in order not to trigger an alert if the magnitude of the event is not large enough. This limit could either be calculated from the total amount of outstanding stocks or from the historical volume data.
- It would also be desirable to implement a way to standardise the suspicious orders/trades alert with historic orderbook data (in order only to detect orders or trades that are “far enough” from the BBO, for example).
- For the immediately traded orders alert, there are two concerns. Firstly, “immediately traded” should be understood as “quickly traded” (with a configurable maximum time in the orderbook prio to trade) instead of “never entered the orderbook”. Secondly, the alert is presently triggered not if both orders are traded immediately, but if this is true for at least one of them.

Naturally, both orders need to be traded immediately for a suspicioun to be in place. Additionally, trades involving a market maker on one side might not be relevant.

- When calculating the values of metrics for instrument groups, the historical values of the individual instrument metrics are simply added to one another. For many metrics (such as for the volume) this is the natural thing to do. For others, however (such as the price), other functions should be used. This shortcoming currently strongly restricts the actual use for instrument groups in the client application.
- As they are evaluated now, the hypothesis tests are always double-sided. This is correct in many situations (such as for the price process, where the price of an instruments is supposed to be checked both for sudden increases and decreases). However, for metrics such as the volume, a single-sided hypothesis test would be the appropriate one to use (as unnaturally low volumes are not to be associated with unlawful conduct in a market).
- According to my objectives, it should be possible to specify the sensitivity of an alert group in terms of "wanted triggers per day" as a percentage. In the current implementation, however, it is only possible to specify the threshold as the number of standard deviations from the estimated mean. With the use of an inverse normal distribution, this should however not present a big problem to implement.
- Also regarding sensitivity measures, it should be possible to define alert levels not in absolute values, but rather relative the other alerts in the alert group. This way, it would be possible to receive a given number of alerts per time period even for alert groups with several individual alerts.
- Presently, there is a table in the database where it is possible to store a list with the clients that in reality represent the same beneficial owner. The legal issues of keeping such a list are however not investigated. Also, the client application does not take the contents of this list into consideration when checking for alerts.
- The structure with alerts organised into alert groups is a flexible tool for the definition of what states should be checked for when investigating the metrics as compared the available benchmarks. However, one could easily imagine some ways to extend the usability. One way to do this is to let each alert have both a level and a sensitivity, and let it be configurable which one to use. This way, one does not have to create several single-alert alert groups to check for several alerts at once using the sensitivity method. Another example of such an extension would be to introduce some kind of alert group "logic", such as conditional alert checking or the possibility of an AND/OR/XOR/NOT-type configuration among the different individual alerts in the alert group.

12. Appendix 5: Database structure

The client-side database is a MySQL relational database, connected to the Java client by the use of a JDBC connection. Below follows a detailed description of the database structure. All datatypes are MySQL types. Please see the online documentation for MySQL[70] for details.

12.1. surveillancedata_X

There is one surveillancedata_ table for each active instrument in the surveillance system, used to store aggregated surveillance data for the instrument in question. This is basically a simplified orderbook, but the table also contains the last trade price for the instrument. The internal instrument id number is a SMALLINT value, starting at 1 for the instrument first added to the configuration. ORDERBOOKSIZE is a system constant (defined in the ExtConstants package), defining up until what depth the orderbook should be saved in the database.

Each table has the following columns, the field values of which reflect data aggregated over the aggregation period of the instrument class to which the instrument belongs. If the instrument belongs to several instrument classes, the shortest aggregation period is used.

Column name	**MySQL data type**	**Primary key**
AskPremium1	FLOAT	
AskVolume1	BIGINT	
BidPremium1	FLOAT	
BidVolume1	BIGINT	
AskPremium2	FLOAT	
AskVolume2	BIGINT	
BidPremium2	FLOAT	
BidVolume2	BIGINT	
…	…	
…	…	
…	…	
…	…	
AskPremium[ORDERBOOKSIZE]	FLOAT	
AskVolume[ORDERBOOKSIZE]	BIGINT	
BidPremium[ORDERBOOKSIZE]	FLOAT	

[70] www.mysql.com/documentation/index.html

BidVolume[ORDERBOOKSIZE]	BIGINT	
LastTradePrice	FLOAT	
TimePeriod	TIME	
DaysPeriod	INTEGER	
TimeStart	TIME	*
DateStart	DATE	*

12.2. orderstrades

This table is used to store time-aggregated information about who has ordered and traded what, both on the net and gross side. The data is subdivided by the instrument and the trading client and -customer.

Each table has the following columns, the field values of which reflect data aggregated over the aggregation period of the instrument class to which the instrument belongs. If the instrument belongs to several instrument classes, the shortes aggregation period is used.

Column name	**MySQL data type**	**Primary key**
TradedVolumeNet	BIGINT	
OrderedVolumeNet	BIGINT	
TradedVolumeGross	BIGINT	
OrderedVolumeGross	BIGINT	
CustomerID	SMALLINT	*
CustomerCountryID	SMALLINT	*
ClientID	SMALLINT	*
ClientCategoryID	SMALLINT	*
InstrumentID	SMALLINT	*
Time	TIME	*
Date	DATE	*

1.1.instruments

This table is part of the configuration. Each record lists an active instrument.

Column name	**MySQL data type**	**Primary key**
OutstandingStocks	BIGINT	
InstrumentName	CHAR(20)	
InstrumentID	SMALLINT	*

1.2. groupedinstruments

This table is part of the configuration. Each record lists a grouping relationship between an instrument and an instrument group.

Column name	MySQL data type	Primary key
InstrumentGroupID	SMALLINT	*
InstrumentID	SMALLINT	*

1.3. instrumentgroups

This table is part of the configuration. Each record lists a configured instrument group, together with its assigned instrument class. If an instrument group needs to be assigned to several instrument classes, one has to create several, identical instrument groups and assign each one to one of the instrument classes.

Column name	MySQL data type	Primary key
InstrumentGroupID	SMALLINT	*
InstrumentClassID	SMALLINT	
InstrumentGroupName	CHAR(80)	

1.4. instrumentclasses

This table is part of the configuration. Each record lists a configured instrument class, together with its configuration parameters in terms of aggregation periods and the amount of historical data used in the recalculation of the statistical model for this instrument class.

Column name	MySQL data type	Primary key
InstrumentClassID	SMALLINT	*
InstrumentClassName	CHAR(80)	
AggregationPeriodDay	SMALLINT	
AggregationPeriodTime	TIME	
OrderbookHistoryLengthDay	SMALLINT	
OrderbookHistoryLengthTime	TIME	
ReturnHistoryLengthDay	SMALLINT	
ReturnHistoryLengthTime	TIME	
VolumeHistoryLengthDay	SMALLINT	
VolumeHistoryLengthTime	TIME	
SpreadHistoryLengthDay	SMALLINT	
SpreadHistoryLengthTime	TIME	

1.5. Clientswithsamebeneficialowner

This table is part of the configuration. Each record lists between 2 and 5 clients that represent the same beneficial owner, i.e. aliases for one trading party. They are not currently used, but should be taken into consideration when the beneficial ownership alert is tested for.

Each record contains the following columns:

Column name	MySQL data type	Primary key
ClientID1	SMALLINT	*
ClientID2	SMALLINT	*
ClientID3	SMALLINT	*
ClientID4	SMALLINT	*
ClientID5	SMALLINT	*

1.6. beneficialownerships

This table is used to store time-aggregated information about who has traded what, and how many shares have really changed owner when adding up. The data is subdivided by the instrument and the trading client and –customer, respectively.

Each table has the following columns, the field values of which reflect data aggregated over the aggregation period of the instrument class to which the instrument belongs. If the instrument belongs to several instrument classes, the shortes aggregation period is used.

Column name	MySQL data type	Primary key
NetTraded	BIGINT	
TotalVolume	BIGINT	
InstrumentID	SMALLINT	*
ClientID	SMALLINT	*
ClientCategoryID	SMALLINT	*
Time	TIME	*
Date	DATE	*

1.7.alerts

This table is part of the configuration. Each record lists configured alert, together with its alert group identity and the level of this alert.

Column name	MySQL data type	Primary key
AlertID	SMALLINT	*
AlertTypeID	SMALLINT	
AlertGroupID	SMALLINT	
Level	FLOAT	

1.8. alertgroups

This table is part of the configuration. Each record lists a configured alert group, together with its instrument group identity and the sensitivity variable of this alert group.

Column name	MySQL data type	Primary key
AlertGroupName	CHAR(80)	
InstrumentGroupID	SMALLINT	
AlertGroupID	SMALLINT	*
Sensitivity	FLOAT	

1.9. alerttypes

This table is part of the configuration. Each record lists a configured alert type.

Column name	MySQL data type	Primary key
AlertTypeName	CHAR(80)	
AlertTypeID	SMALLINT	*

1.10. triggeredalerts

Each record lists a triggered alert, part of a referenced configured alert group, together with data about the actual triggered alert.

Column name	MySQL data type	Primary key
TriggeredAlertGroupID	BIGINT	*
BidClient	CHAR(80)	
BidClientCategory	CHAR(80)	
AskClient	CHAR(80)	
AskClientCategory	CHAR(80)	
Customer	CHAR(80)	

CustomerCountry	CHAR(80)	
AlertName	CHAR(80)	*
Level	FLOAT	
ActualLevel	FLOAT	

1.11. triggeredalertgroups

Each record lists a triggered alert group, part of a referenced configured instrument group, in turn part of a referenced instrument class. There is also data about the actual triggered alert group, such as the instrument and the actual sensitivity value at the time of triggering.

Column name	**MySQL data type**	**Primary key**
TriggeredAlertGroupID	BIGINT	*
InstrumentClassName	CHAR(80)	
InstrumentGroupName	CHAR(80)	
InstrumentName	CHAR(80)	
AlertGroupName	CHAR(80)	
Sensitivity	FLOAT	
ActualSensitivity	FLOAT	
Time	TIME	
Date	DATE	

1.12. clients

This table lists the clients that have been referenced in the received data. The client category is necessary to fully identify the client. Please see the CLICK system documentation for further information.

Column name	**MySQL data type**	**Primary key**
ClientName	CHAR(80)	
ClientCategoryID	SMALLINT	
ClientID	SMALLINT	*

1.13. clientcategories

Ths table lists the client categories that have been referenced in the received data. Please see the CLICK system documentation for further information.

Column name	**MySQL data type**	**Primary key**

ClientCategoryName	CHAR(80)	
ClientCategoryID	SMALLINT	*

1.14. customers

Ths table lists the customers that have been referenced in the received data. The customer country is necessary to fully identify the customer. Please see the CLICK system documentation for further information.

Column name	MySQL data type	Primary key
CustomerName	CHAR(80)	
CountryID	SMALLINT	
CustomerID	SMALLINT	*

1.15. countries

Ths table lists the countries that have been referenced in the received data (in the current implementation only as a reference to a "customer country"). Please see the CLICK system documentation for further information.

Column name	MySQL data type	Primary key
CountryName	CHAR(80)	
CountryID	SMALLINT	*

1.16. clientcustomerrelations

This table lists the clients and customer that have been associated. Please see the CLICK system documentation for further information.

Column name	MySQL data type	Primary key
ClientID	SMALLINT	*
CustomerID	SMALLINT	*

1.17. automaticquerystatuses

This table is part of the configuration. It lists all instruments that have been configured to send out an automatically generated query for market information around the time for the trigger of an alert group. In the current implementation, this query is initated by the user (via the GUI), and is hence not used.

Column name	MySQL data type	Primary key
InstrumentID	SMALLINT	*

1.18. automaticquerybefore

This table is part of the configuration. It lists only one value, namely the amount of time that an automatically generated query should cover before the trigger of the alert group. This value is the same for all instrument classes.

Column name	MySQL data type	Primary key
Days	SMALLINT	*
Time	TIME	*

1.19. automaticqueryafter

This table is part of the configuration. It lists only one value, namely the amount of time that an automatically generated query should cover after the trigger of the alert group. This value is the same for all instrument classes.

Column name	MySQL data type	Primary key
Days	SMALLINT	*
Time	TIME	*

1.20. automaticreturngraphinstruments

This table is part of the configuration. It lists all the instruments that are configured to send out real-time price data from the central data manager to the client GUI.

Column name	MySQL data type	Primary key
InstrumentID	SMALLINT	*

1.21. automatictradevolumegraphinstruments

This table is part of the configuration. It lists all the instruments that are configured to send out real-time trade volume data from the central data manager to the client GUI.

Column name	MySQL data type	Primary key
InstrumentID	SMALLINT	*

1.22. automaticordervolumegraphinstruments

This table is part of the configuration. It lists all the instruments that are configured to send out real-time order volume data from the central data manager to the client GUI.

Column name	MySQL data type	Primary key
InstrumentID	SMALLINT	*

1.23. automaticbbographinstruments

This table is part of the configuration. It lists all the instruments that are configured to send out real-time orderbook data from the central data manager to the client GUI.

Column name	MySQL data type	Primary key
InstrumentID	SMALLINT	*

13. References

13.1. Text books

Anderson, O.D. - Time Series Analysis and Forecasting (Butterworth & Co (Publishers) Ltd, 1976)

Bodie, Z. – Kane, A. – Marcus, A.J.: Investments (McGraw-Hill 1999)

Chatfield, C. – The Analysis of Time Series – An Introduction (Chapman and Hall, 1985)

Gouriéroux, C. – ARCH Models and Financial Applications (Springer-Verlag 1997)

Grinblatt, M – Titman, S: Financial Markets and Corporate Strategy (McGraw-Hill 1998)

Pankratz, A. - Forecasting with Univariate Box-Jenkins Models (John Wiley & Sons, Inc., 1983)

13.2. Periodicals and magazines

Benabou, R. – Laroque, G. – "Using Priviliged Information to Manipulate Markets Insiders, Gurus, and Credibility" (The Quarterly Journal of Economics, August 1992)

Engle, R.F. – "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of U.K. Inflation" (Econometrica, nr. 50, p. 987-1008, 1982)

"Evolving Cooperation and Coordination in Financial Market Surveillance" – Finance & Development (periodical from the International Monetary Fund), September 1999, vol. 36, nr. 3.

John, K – Narayanan, R. – "Market Manipulation and the Role of Insider Trading Regulations" (Journal of Business, 1997, vol. 70, nr. 2)

13.3. Personal meetings

Meeting with Mats Danielsson and Per-Åke Sundström 2001-09-17

Discussions with various people at OM, including the tutors and other informed persons

13.4. World Wide Web

Aitken, M – Berry, J. – "Market Surveillance at the Australian Stock Exchange: An Overview" (1991, available [online]: www.smarts.com.au, accessed 2001, 2002)

"Discovery" (Issue 3, 2001, newsletter from SMARTS, available [online]: www.smarts.com.au/09_DiscoveryIntro.html, accessed 2001, 2002)

finance.yahoo.com (accessed 2001, 2002)

java.sun.com (accessed 2001, 2002)

smarts.com.au (accessed 2001, 2002)

www.jiway.com (accessed 2001, 2002)

www.mysql.com (accessed 2001, 2002)

www.om.com (accessed 2001, 2002)

13.5. Other material

Lecture notes from the course “Séries Chronologiques” (2000, ENSIMAG, Grenoble, France)

Örtenblad,J. – Bogentoft, E – “Catastrophe-linked Securities” (Master’s thesis, Stockholm School of Economics, 2001)