Architecture on the execution of algorithmic trading and high frequency algorithmic trading

PhD student Giulio Carlone
Department of Management and Business Administration
University "G.D'Annunzio" of Chieti and Pescara
gcarlone@unich.it

Algorithmic trading and, in particular, the high frequency algorithmic trading of recent years, has had a significant effect thanks to technological advances and financial innovations surrounding the trading activities. This trading technique involves use of algorithms to acquire, process and react to market information at a high speed. Exchanges performed with this technique have grown a great deal and in Europe account for between 10% and 40% of all exchanges, depending on the European country involved. This paper will show a transaction of this type from an architectural point of view. The scenario will show all phases of the connection between the client and the platform of interest, the information structure and the dynamics of the data as processed by the algorithm and the response from the trading platform. The paper is a very useful tool in order to understand the structure of the components involved in this issue and at the same time the simplicity of the discussion of these objects can imagine further combinations, more sophisticated financial instruments in the performance of high frequency trading.

Introduction

The goal of this paper is the presentation of the components involved during the execution of high frequency trading and in their combination through the realization of a simple algorithm, obtaining as a result a very useful base case of application to be used as an point of departure to hypothesize more complex scenarios.

In this discussion, the case of algorithmic trading (AT) will be considered, and the specific case of high frequency algorithmic trading (HFT) on the Italian Stock Exchange, where the platform used to perform this type of transactions is called MillenniumIT (MIT), which runs the Electronic Share Market (MTA) and the markets bond of Exchange Traded Funds and Covered Warrants.

The recent literature on the economics was used to commence the study [12], [13] and to provide insights about the risks and criticality [1], [2], [6] of HFT, from a point of view of liquidity [15], volatility [3] and quality of the market [14]. We observe that the HFT is a subset of AT. At the same time there has been investigative work undertaken by the European Commission [4], [11], by the supervisory authorities of the financial markets [5], and the Italian Stock Exchange [8], [9]. Their main task in the face of this modern financial technique was to ensure the transparency of negotiations, the organized execution of the trades, and surpervision as a precaution to those who undertake the investment. All the initiatives taken by the

authorities are aimed at taking actions with the purpose of carrying out monitoring activities by checking for any possible adverse consequences.

In the academic literature there are many papers about the HFT but compared to these works, the innovative contribution of the current proposal is to examine in detail the life cycle of this method through the application of a base case. In the following sections we will describe the steps required to perform this technique initially through a high-level architecture, and subsequently from a low-level architecture, describing its components.

The usefulness of this proposal consists in producing a base case as a starting point for studies to deal with more complex trades, since the results can be used to predict more sophisticated scenarios. This proposal consists of a contribution of great originality because it will examine a specific architecture, the trading on the Italian Stock Exchange, through a HFT base case using a particular algorithm, ad-hoc, it is need to explain some of the steps crossed during its execution.

In this study, we hypothesized a maximum algorithm optimization and a maximum processing speed, conditions necessary to ensure that the execution has the high frequency characteristics of a real transaction.

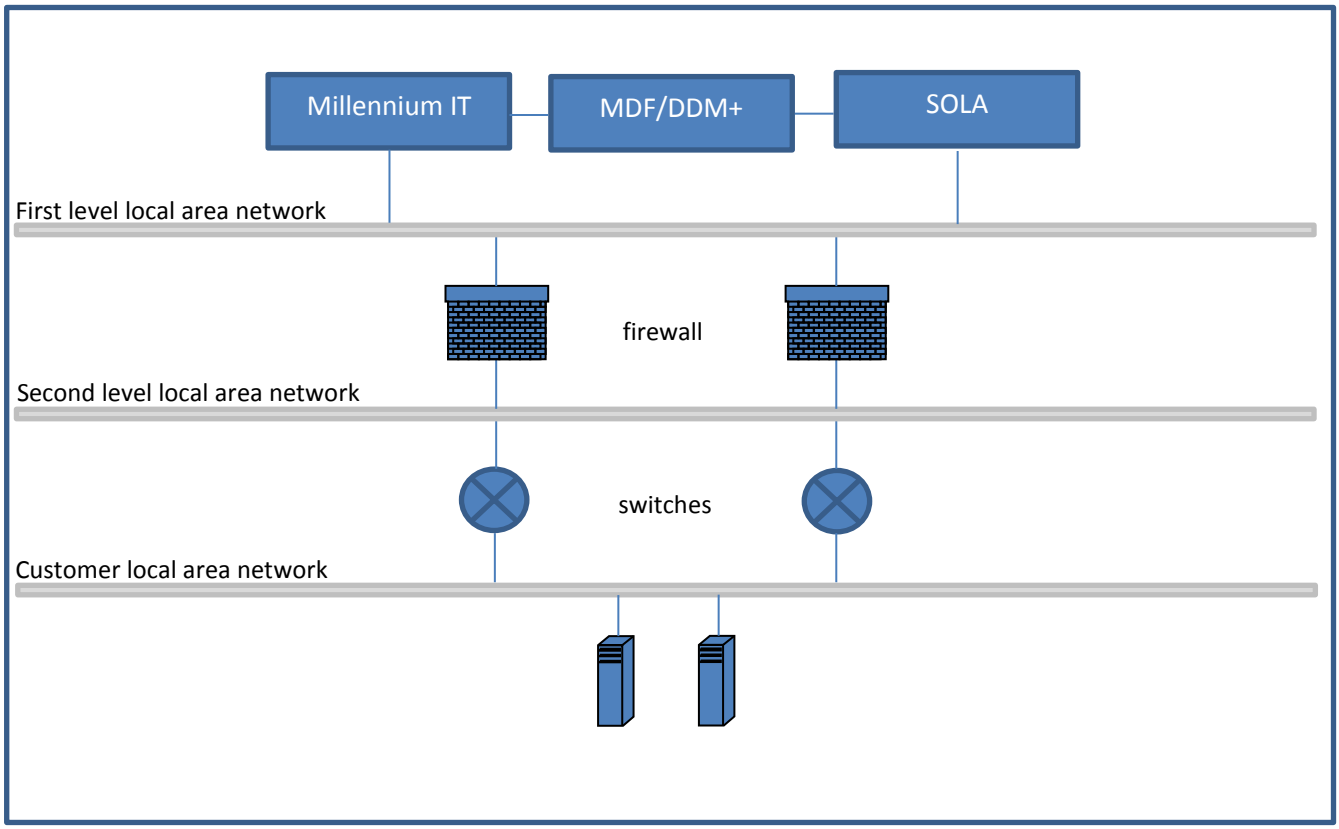
We will examine the characteristics of the data flow in real time, the characteristics of the data that runs a transaction, the characteristics of connectivity between the server and the client platform MIT, both from a point of view of reception of the information and from the transmission of the information. It is important to understand the type and the format of the data that must run in this architecture and what are all the steps that allow access to the system and the output from the system; the steps that allow the transmission of orders inclusive of all the controls. In this architecture will show all the steps required for the execution of an order, and we will then introduce in detail the operation of the algorithm that performs these functions.

The contents of the proposal in the first part will deal with the architectural structure showing the various levels that the customers have to achieve in order to reach the market. The second part deals with the connection mode required to start sending messages to the market and also introduces the characteristics of the administrative message sent with standard FIX. The third part is an example of a message written in the FIX protocol and will be examined in detail through of its components. The fourth part will introducee the structure of the application type message, sent with standard FIX, which is necessary for the management of the orders. The fifth part will show the application type messages answered by the system after sending a new request. In the sixth part is shown a flow chart, from both the customer side and the system side, which traces the path of a new message. The seventh part presents the approaches of risk measurement, the risks facing an organization that runs HFT, the technique of measuring the Value at Risk through the variance-covariance formula. These are introduced with a programming language method that calculates all the variables in play, and is finally provided with a subsequent algorithm indicated schematically by a flow chart that shows the rounds of recursive calls of the various methods. In the eighth and final part there is a simple implementation of the algorithm mentioned.

Architectural landscape

Consider the MIT platform and the location of the customer of the Italian Stock Exchange. In this graphical representation they also play a part of the Italian market of derivatives (IDEM) performed through the SOLA platform and the central system MDF/DDM +, composed of Member Data Feed (MDF), which shows the information service market data for those who belong to the markets and the service dissemination of market data (DDM+) that distributes real time prices.

Network service gives the possibility to displace the services of the market and the management of the connections on a server. At the lowest level there is the local area network of the customers connected to the switches, in turn they are connected to the second level of the local area network, which is then connected to the firewall of first level, which is itself connected to the first level of the local area network that accesses MIT, SOLA and to the central MDF/DDM+.



Mode connection and message components from the administrative side.

To connect with the MIT you must first establish a connection with the ip of its corresponding server (IPSERVERNAME) and the respective port (PORTNAME), using the protocol for the exchange of financial information, the Financial Information eXchange (FIX) using Transmission Control Protocol/Internet Protocol (TCP/IP) for communication. Once the connection has been made, operations are administratively possible as follows: logon, logout, heartbeat, test request, resend request, reject and reset sequence.

Logon: id=A, establish a connection between the client and the server;
Logout: id = 5, allows you to terminate a connection between the client and the server;
Heartbeat: id=0, allows the client and the server to test the communication line during periods of inactivity and to verify that the interfaces are available at the ends;
Test request: id=1, allows the client or the server to request a response from the other party if it is determined an inactivity;
Resend request: id=2, allows the recovery of lost messages during a malfunction of the layers of communication;
Reject: id=3, a message used to reject non-conforming;
Sequence reset: id=4, enables the client or server to increase the sequence number expected arrival the other side.

Example of a typical data transmitted with the FIX protocol

We see below the structure of a message sent by the client through the FIX protocol (version 4.2) it is composed of a header, a body and a final part. Let us examine in detail the components of the following message that allows you to send a quote for the individual instrument USD/CAD at 15:25:20 GMT January 21, 2013:

8=FIX.4.2 | 9=309 | 35=S | 49=ML-FIX-FX |
56=ECHO2-QTS-TEST | 34=5015 | 52=20130121-15:25:20 |
131=1185895365 | 117=ECHO2-QTS-TEST.00043690C8A8D6B9.00043690D14044C6 | 301=0 |
55=USD/CAD | 167=FOR | 15=USD | 132=1.065450 |
133=1.065850 | 134=5000000.0 | 135=5000000.0 |
647=2000001.0 | 648=200000.0 | 188=1.06545 |
190=1.06585 | 60= 20130121-15:25:20 | 40=H | 64=20130122 |
10=178

Dissecting the message we note the following parts:
8=FIX.4.2: version of the FIX protocol used;
9=309: the body of the message is 309 characters long;
35=S: this message is carrying a quote;
49=ML-FIX-FX: internal identification of the message; in this case, the sender is Merrill Lynch FX desk;
56=ECHO2-QTS-TEST: internal identification of the message recipient;
34=5015: sequential message number;this number is used to track all the messages sent;
52=20130121-15:25:20: timestamp corresponding to the time transmission originated;
131=1185895365: unique identifier corresponding to a message containing an original quote request for a given security;
117=ECHO2-QTS-TEST.00043690C8A8D6B9.00043690D14044C6: unique identifier for the quote;
301=0: level of response requested from recipient of the quote message;
55=USD/CAD: the ticker symbol of the quoted instrument;
167=FOR: the type of the security quoted (es: FUT=futures, OPT=option,…);
15=USD: based currency used for price;
132=1.065450: bid price;
133=1.065850: ask price;
134=5000000.0: bid quantity;
135=5000000.0: ask quantity;
647=2000001.0: minimum quantity for a bid;

648=200000.0: minimum quantity for an ask;
188=1.06545: bid Foreign exchange spot rate;
190=1.06585: ask Foreign exchange spot rate
60= 20130121-15:25:20: timestamp of the quote creation;
40=H: available order types;
64=20130122: trade settlement date;
10=178: checksum, a sum of computer codes of all characters in the message.

Message components from the application launched by the customer for the management of orders.

As regards the management of orders from the customer perspective, the phases of sending a request have the following possible choices:
new order-single; cancel order request; order mass cancel request; order cancel/replace request; cross new order message; cross-order cancel request.

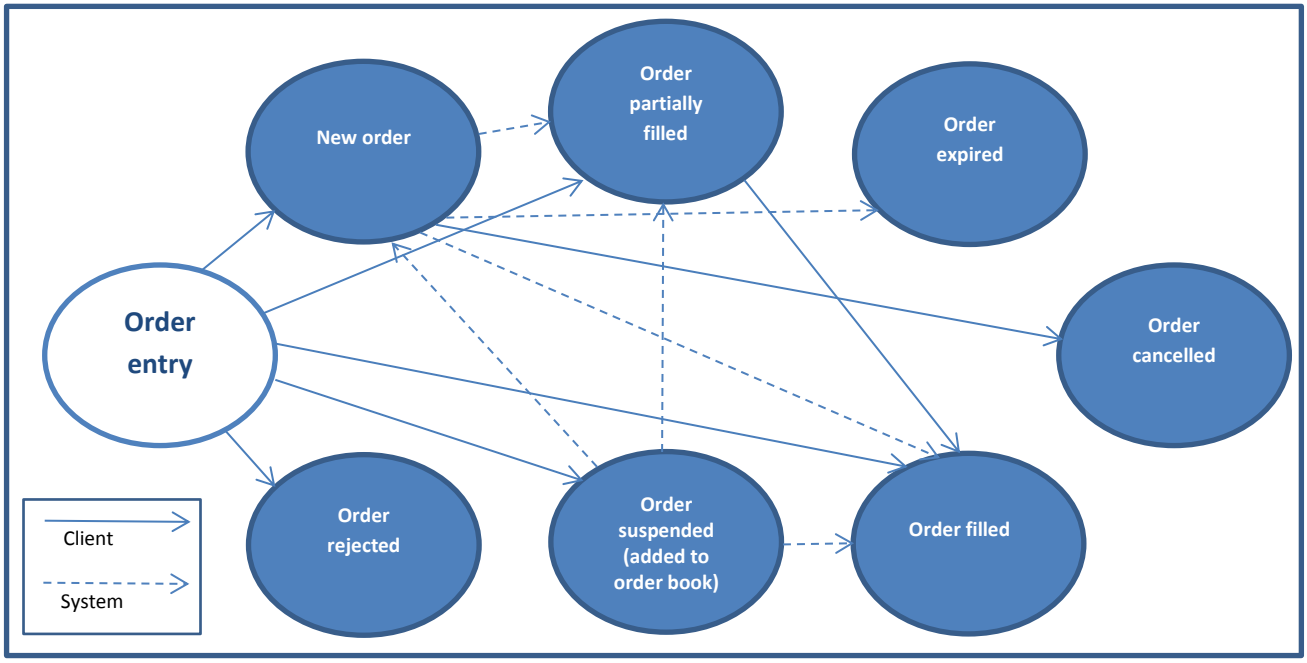
New order - single: id = D, allows the customer to submit a new order;
Order cancel request: id = F, allows the customer to cancel an order in real time;
Order mass cancel request: id = q; allows the customer to cancel the mass of (i) all orders in real time, (ii) all orders in real time to a particular instrument, (iii) all the orders in time real for a particular segment. The mass cancellation can be applied to orders for a particular group of performs trading or to all orders of the Company.
Order cancel / replace request: id = G, allows the customer to cancel/replace an order in real time;
New order cross message: id = s, allows the customer to place an order Cross/BlockTradeFacility;
Cross order cancel request: id = u, allows the customer to cancel an order Committed Cross/BlockTradeFacility.

Message components from when the application answered by the server to the management of orders.

From the server side some of the main messages which may occur in response to receiving a message from the client side for the management of orders, are the following: execution reports, order cancel reject, order mass cancel report.

Let us see what these messages communicate to the customer.
Execution report: id = 8, indicates one of the following actions: (i) order accepted, (ii) order rejected, (iii) order executed, (iv) order expired, (v) order canceled, (vi) order canceled/replaced (vii) exchange deleted, (viii) order suspended, (ix) order active;
Order cancel reject: id = 9, indicates that the order of cancellation request or the order of request for cancellation/replacement were rejected;
Order mass cancel report: id = r, indicates one of the following actions: (i) the order of cancellation of mass has been accepted, (ii) the cancellation of mass was rejected.

Flow chart relating to the management of orders



Market risk management strategy with the application of an algorithm that performs high frequency trading

Effective management of risk in trading operations is just as important as are the signals that motivate purchases. A proper function of well-designed and well executed risk management strategy is the key to sustained profitability in all organizations. Risk management in relation to high transaction frequency trading is particularly important since potential large losses can be quickly reduced to minimal losses. You will be shown what are the strategies for risk measurement and what are the risks that an organization faces in specifically in relation to high frequency trading.

1. Approaches to risk measurement can be divided into four broad categories:
Statistical models: generating estimates of the worst future outcome based on past information. These models are the preferred methodology to estimate if there are conditions to achieve them;
The scalar models: establishing maximum levels of expected loss rates with business parameters such as revenue and operating costs;
The scenario analysis: determining a base for the best case and a base for the worst case, depending on the risk indicators (Key Risk Indicators, Kris);
Modeling random: it involves the identification of the causes and effects of potential losses.

2. Looks at the risks that an organization faces, especially in HFT:
Market risk: corresponding to the risk of losses in those positions arising from movements in market prices;
Credit risk: corresponding to the risk that a borrower will default on any type of debt, and not making the payment that is due;
Counterparty risk: corresponding to the risk that the counterparty fails to fulfill its obligations under an agreement;
Liquidity risk: which is the risk of being unable to sell an asset in a timely manner;
Operational risk: defined as the risk about the internal workings of an organization, and which are within its control;
Legal risk: refers to the risk associated with the impact of a defect in the legal documentation which would negatively impact the cash flows or debt service capability.

Market risk models have been developed that allow us to quantify, compare and aggregate the risk associated with individual items or entire portfolios. These models are the models of value at risk, Value at Risk (VaR), corresponding to the measure of the maximum loss that a position or a portfolio of positions can undergo. We consider a parametric measurement approach of market risk. The formula for calculating the VaR of a portfolio P of N positions with a parametric approach is the following:

$$VaR_P = f^{-1}(1-c)\delta t^{\frac{1}{2}} \sqrt{\sum_{j=1}^{N}\sum_{i=1}^{N} w_i \; w_j \sigma_i \sigma_j \rho_{ij}}$$

Where:
N= number of positions;
σ_i = Volatility of the i-th position;
ρ_{ij} = Coefficient of correlation between the positions of the i-th and j-th;
w_i = Fraction of wealth invested in the i-th position;
f^{-1} = Inverse of the standard normal cumulative distribution function;
c = confidence interval;
$\delta t^{\frac{1}{2}}$ = Time interval.

The corresponding method for the calculation of VaR is given by the following code:

```
public double vaR(double c, double δ, double t, List σ, List ω, List ρ){...};
```

Where $\sigma = (\sigma_1 \; , \ldots , \sigma_P)$ and $\omega = (\omega_1 \; , \ldots , \omega_P \;)$ with P the number of position of our portfolio and ρ is the vector $(\ldots, \rho_{i,j} \; , \ldots)$.
Since we have to constantly monitor the value of VaR, we must therefore constantly monitor the value of the parameters that form part of its calculation. In this regard, we must take into consideration the calculation of volatility, which is expressed by the following formula:

$$\sigma_t = \sqrt{\frac{\sum_{i=t-n}^{t-1} r_i^2}{n-1}}$$

where the variables are as follows:
r_i = i-th value, observed at time t;
n = number of observations.

The corresponding method for calculating the volatility will be given by the following code:

```
public double volatility(double t, int n, double r_i){...};
```

With regard to the interaction with the system,we define two types of methods that represent communication with the platform MIT through the FIX protocol. From a point of view of methods of an administrative nature we define the following:

```
public String login(String IPSERVERNAME, int PORTNAME){...};
public String logout(String FIXSTRING){...};
```

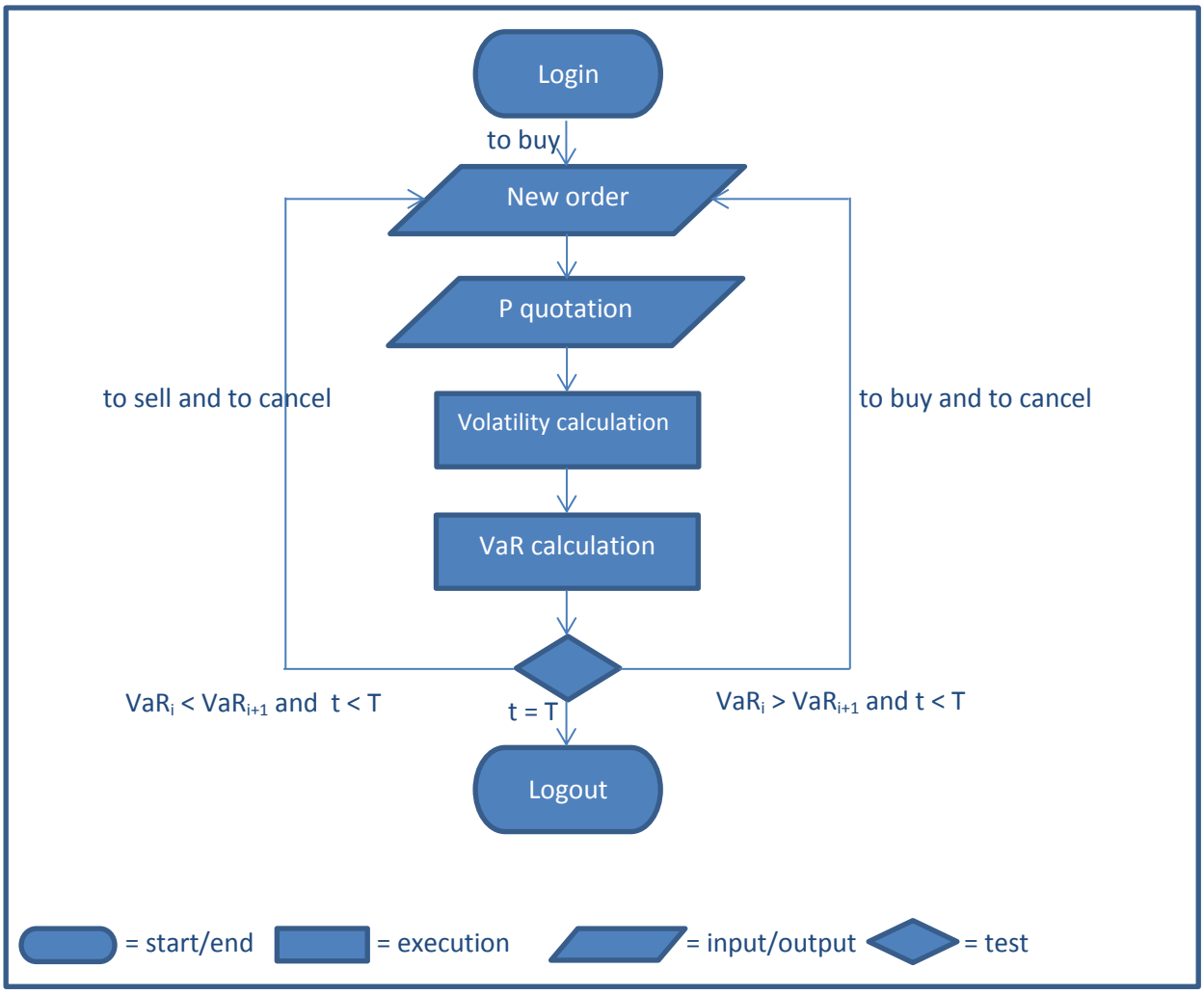
With regard to communication with the system from a practical point of view we define as follows:

```
public String newOrder(String FIXSTRING){...};
public String newQuote(String FIXSTRING){...};
```

At this point, specifying the application context, we describe the scenario of our interest showing characteristics of our algorithm.

Context and algorithm: *Establish a portfolio of k securities at time t and assume that the potential loss of these investment positions with a confidence level of 99% is appropriate to a particular VaR and assume that this has a value less than a predetermined threshold, then proceed as follows. We provide access to the system. Send a new purchase order system corresponding to the choices of our portfolio. We constantly monitor variables volatility after receiving quotes of the positions of our interest after a specific request to the system and recalculate the VaR. At this point we can obtain a value greater or a smaller value. If the new VaR is greater, then we identify the positions corresponding to the growth of the VaR and send a new sell order to the system and if they have outstanding purchase orders then send the system a new order of cancellation of purchase orders for such positions. If the new VaR is less then identify the positions corresponding to the decrease of VaR and send the system a new buy order with respect to such positions and if they have outstanding sell orders then send the system a new order of cancellation of sell orders for such positions. We proceed recursively to perform these operations from time t to a time t. We update our database with all the data that changes at each iteration, which will be the subject of further studies. At the end of the execution of a series of cycles, at time T we provide the output from the system.*

The flow chart represents this algorithm:



Light implementation of the algorithm that performs high frequency trading

Below is a prototype for the implementation of the algorithm represented graphically in the diagram flow using jruleengine.org library. A best implementation could be realized with an account, very expensive, on the Millennium IT test environment platform, in Italian Stock Exchange.

File MyRule.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 Title: HFT Project
 Description: Demo about high frequency trading technique
       The author may be contacted at:
           gcarlone@unich.it
           giulio.carlone@gmail.com
@author Giulio Carlone
@version 1.0
-->
<rule-execution-set>
 <name>RuleExecutionSet</name>
 <description>Rule Execution Set</description>
 <synonymn name="StockManagement" class="hft.carlone.unich.it.StockManagement" />
 <!--    -->
 <rule name="Rule1" description="VARTi+1 greter VARTi" >
 <if leftTerm="StockManagement.getVarTk" op=">" rightTerm="hft.carlone.unich.it.Status.getVarT" />
 <then method="StockManagement.sell" arg1="hft.carlone.unich.it.Status.getVarT"
        arg2="hft.carlone.unich.it.Status.getVarI" arg3="hft.carlone.unich.it.Status.getPortfolio"/>
 <then method="StockManagement.updateVarIk" arg1="hft.carlone.unich.it.Status.getVarI"
        arg2="hft.carlone.unich.it.Status.getPortfolio" />
 <then method="StockManagement.updateVarTk" arg1="hft.carlone.unich.it.Status.updateVarTk" />
 <then method="hft.carlone.unich.it.Status.setPortfolio" arg1="StockManagement.getPortfolio" />
 <then method="hft.carlone.unich.it.Status.setVarT" arg1="StockManagement.getVarTk" />
 <then method="hft.carlone.unich.it.Status.setVarI" arg1="StockManagement.getVarIk" />
 </rule>
 <rule name="Rule2" description="VARTi+1 less VARTi" >
 <if leftTerm="hft.carlone.unich.it.Status.getVarT" op=">" rightTerm="StockManagement.getVarTk" />
 <then method="StockManagement.updateVarIk" arg1="hft.carlone.unich.it.Status.getVarI"
        arg2="hft.carlone.unich.it.Status.getPortfolio" />
 <then method="StockManagement.updateVarTk" arg1="hft.carlone.unich.it.Status.updateVarTk" />
 <then method="StockManagement.buy" arg1="hft.carlone.unich.it.Status.getVarT"
        arg2="hft.carlone.unich.it.Status.getVarI" arg3="hft.carlone.unich.it.Status.getPortfolio"/>
 <then method="hft.carlone.unich.it.Status.setPortfolio" arg1="StockManagement.getPortfolio" />
 <then method="hft.carlone.unich.it.Status.setVarT" arg1="StockManagement.getVarTk" />
 <then method="hft.carlone.unich.it.Status.setVarI" arg1="StockManagement.getVarIk" />
 </rule>
 </rule-execution-set>
```

File Engine.java

```
package hft.carlone.unich.it;

import hft.carlone.unich.it.handling.Order;
import hft.carlone.unich.it.riskmanagement.RiskManagement;
import hft.carlone.unich.it.admin.Admin;
import javax.rules.InvalidRuleSessionException;
import javax.rules.RuleRuntime;
import javax.rules.RuleServiceProvider;
import javax.rules.RuleServiceProviderManager;
import javax.rules.StatefulRuleSession;
import javax.rules.StatelessRuleSession;
import javax.rules.admin.RuleAdministrator;
import javax.rules.admin.RuleExecutionSet;
import java.io.*;
import java.util.*;
/*
* <p>Title: HFT Project</p>
* <p>Description: Demo about high frequency trading technique</p>
*
* <p>The author may be contacted at:
*           gcarlone@unich.it
*           giulio.carlone@gmail.com</p>
*
```

```
* @author Giulio Carlone
* @version 1.0
*/
public class Engine {

        private String FIXSTRING = "";

        public Engine() {
        try
        {

                Admin admin = new Admin();
                String IPSERVERNAME = "127.0.0.1";
                int PORTNAME = 8080;
                admin.logon(IPSERVERNAME, PORTNAME);
                // Load the rule service provider of the reference
            // implementation.
            // Loading this class will automatically register this
            // provider with the provider manager.
            Class.forName( "rule.engine.RuleServiceProviderImpl" );

            // Get the rule service provider from the provider manager.
            RuleServiceProvider serviceProvider = RuleServiceProviderManager.getRuleServiceProvider(
"rule.engine" );
            // get the RuleAdministrator
            RuleAdministrator ruleAdministrator = serviceProvider.getRuleAdministrator();
            System.out.println("\nAdministration API\n");
            System.out.println( "Acquired RuleAdministrator: " + ruleAdministrator );
            // get an input stream to a test XML ruleset
            // This rule execution set is part of the TCK.
            InputStream inStream = new FileInputStream( "C:\\pathname\\MyRule.xml" );
            System.out.println("Acquired InputStream to rule.xml: " + inStream );
            // parse the ruleset from the XML document
            RuleExecutionSet res1 = ruleAdministrator.getLocalRuleExecutionSetProvider( null
).createRuleExecutionSet( inStream, null );
            inStream.close();
            System.out.println( "Loaded RuleExecutionSet: " + res1);
            // register the RuleExecutionSet
            String uri = res1.getName();
            ruleAdministrator.registerRuleExecutionSet(uri, res1, null );
            System.out.println( "Bound RuleExecutionSet to URI: " + uri);
            // Get a RuleRuntime and invoke the rule engine.
            System.out.println( "\nRuntime API\n" );
            RuleRuntime ruleRuntime = serviceProvider.getRuleRuntime();
            System.out.println( "Acquired RuleRuntime: " + ruleRuntime );
            // create a StatelessRuleSession
            StatelessRuleSession statelessRuleSession = (StatelessRuleSession)
ruleRuntime.createRuleSession(uri,new HashMap(), RuleRuntime.STATELESS_SESSION_TYPE);
            System.out.println( "Got Stateless Rule Session: " + statelessRuleSession );
            // call executeRules with some input objects

            // set up varI value, varT value, portfolio value
           List varI = new ArrayList<Double>();
           double varT = 0;
           List portfolio = new ArrayList();

                Status inputStatus = new Status();
                FIXSTRING = "fixStringIdForTestRequest";
        String answer = admin.testRequest(FIXSTRING);
        //management of the answer
                inputStatus.setVarT(varT);
           inputStatus.setVarI(varI);
           inputStatus.setPortfolio(portfolio);
           //Stock management inputSM
           StockManagement inputSM = new StockManagement();
        inputSM.updateVarIk(varI, portfolio);
        inputSM.updateVarTk(varI);
        inputSM.updatePortfoliok(varT,varI,portfolio);
                        int t = 100;
        int k = 1;
        // for 100 iterations
            while (t != k){
                k = k+1;
                // Create a input list.
                List input = new ArrayList();
                input.add(inputStatus);
```

```
                input.add(inputSM);
                // Print the input.
                System.out.println("Calling rule session with the following data");
                // Execute the rules without a filter.
                System.out.println(input.toString());
                List results = statelessRuleSession.executeRules(input);
                System.out.println( "Called executeRules on Stateless Rule Session: " + statelessRuleSession
);
                System.out.println( "Result of calling executeRules: " + results.size() + " results." );
                // Loop over the results.
                Iterator itr = results.iterator();
                while(itr.hasNext()) {
                    Object obj = itr.next();
                    if (obj instanceof Status)
                            System.out.println("Status instance: varI, varT,portfolio ");
                    if (obj instanceof StockManagement)
                            System.out.println("Stock Management instance: varI, varT,portfolio");
                }
                FIXSTRING = "fixStringIdForTestRequest";
                answer = admin.testRequest(FIXSTRING);
                // set up new status values
                inputStatus.getVarI();
                inputStatus.getVarT();
                inputStatus.getPortfolio();
                FIXSTRING = "fixStringIdForTestRequest";
                answer = admin.testRequest(FIXSTRING);
                //management of the answer
                inputSM.getVarIk();
                inputSM.getVarTk();
                inputSM.getPortfoliok();
            }
                // Release the session.
                statelessRuleSession.release();
                System.out.println( "Released Stateless Rule Session." );
                System.out.println();
                // create a StatefulRuleSession
                StatefulRuleSession statefulRuleSession = (StatefulRuleSession)
ruleRuntime.createRuleSession( uri,new HashMap(),RuleRuntime.STATEFUL_SESSION_TYPE );
                System.out.println( "Got Stateful Rule Session: " + statefulRuleSession );
                System.out.println( "Released Stateful Rule Session." );
                System.out.println();
                FIXSTRING = "fixStringIdForExit";
                admin.logout(FIXSTRING);
    }
    catch (NoClassDefFoundError e) {
      if (e.getMessage().indexOf("Exception") != -1) {
        System.err.println("Error: The Rule Engine Implementation could not be found.");
      }
      else {
        System.err.println("Error: " + e.getMessage());
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  public static void main(String[] args) {
  }
}
```

File Status.java

```
package hft.carlone.unich.it;

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
/*
* <p>Title: HFT Project</p>
* <p>Description: Demo about high frequency trading technique</p>
*
* <p>The author may be contacted at:
*           gcarlone@unich.it
*           giulio.carlone@gmail.com</p>
*
```

```
* @author Giulio Carlone
* @version 1.0
*/
public class Status {

	private double varT;
	private List varI;
	private List portfolio;


	public Status() {
		// TODO Auto-generated constructor stub
	}

	public void setVarT(double varT){
		this.varT = varT;
	}

	public double getVarT() {
		return varT;
	}

	public void setVarI(List varI){
		this.varI = varI;
	}
	public List getVarI() {
		return varI;
	}

	public void setPortfolio(List portfolio){
		this.portfolio = portfolio;
	}
	public List getPortfolio() {
		return portfolio;
	}
}
```

File StockManagement.java

```
package hft.carlone.unich.it;

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import hft.carlone.unich.it.handling.Order;
import hft.carlone.unich.it.handling.Quote;
import hft.carlone.unich.it.riskmanagement.RiskManagement;
/*
* <p>Title: HFT Project</p>
* <p>Description: Demo about high frequency trading technique</p>
*
* <p>The author may be contacted at:
*           gcarlone@unich.it
*           giulio.carlone@gmail.com</p>
*
* @author Giulio Carlone
* @version 1.0
*/
public class StockManagement {

	private double varTk;
	private List varIk;
	private List portfoliok;
	private String tempB;
	private String tempS;
	private String tempC;
	private String tempQ;
	private List positionPortfolio;
	private double quotePosition;
	private List quotePortfolio;

	public StockManagement(){}

	public void setVarTk(double varTk){
		this.varTk = varTk;
```

```
	}

	public double getVarTk() {
		return varTk;
	}
	RiskManagement RM = new RiskManagement();
	Quote q = new Quote();
	private double tk;
	private int nk;
	private double rik;
	private double var1,var2,var3;

	public void updateVarTk(List varI){
		double tempVar = formula(varI);
		this.varTk = tempVar;
	}

	public double formula(List varI){
	double tempVar = 0;
		//formula for var calculation
		return tempVar;
	}

	public void updateVarIk(List varIk,List portfolio){

		String quoteFIXSTRING = "";
		double volatilityk = 0;
		List updateVar = varIk;
		Iterator p = portfolio.iterator();
		while (p.hasNext()){
			ArrayList position = new ArrayList ((ArrayList)p.next());
			quoteFIXSTRING = genFIXSTRINGquote(position);
			this.tempQ = q.quoteUpdate(quoteFIXSTRING);
			this.quotePosition = quoteUpdate(tempQ, position);

			volatilityk =
RM.volatilityK(extractTK(this.quotePosition),extractNK(this.quotePosition),extractRIK(this.quotePosition));

// some different input value volatility var1 var2 var3 for vari calculation
			double vari = RM.vari(volatilityk,var1,var2,var3);
			updateVar = updateVar(vari,updateVar);
		}
		this.varIk = updateVar;
	}

	public double extractTK(double quotePositionk){
		return tk;
	}

	public int extractNK(double quotePositionk){
		return nk;
	}

	public double extractRIK(double quotePositionk){
		return rik;
	}

	public List updateVar(double vari,List varI){
		// update the i-th var value of varI vector
		return varIk;
	}

	public List updatePortfoliok(double varT, List varI, List portfolio){
		// update of portfolio
		// sell-cancel or buy-cancel metod
		return portfoliok;
	}

	public void setVarIk(List varI){
		this.varIk = varI;
	}

	public List getVarIk() {
		return varIk;
	}
```

```
	public void setPortfoliok(List portfoliok){
		this.portfoliok = portfoliok;
	}
	public List getPortfoliok() {
		return portfoliok;
	}

	public void sell(double varT, List varI, List portfolio){
		Order o = new Order();
		String sellFIXSTRING = "";
		String cancelFIXSTRING = "";
		List positionUpdate = portfolio;
		if (this.varTk < varT){
			Iterator it1 = this.varIk.iterator();
			Iterator it2 = varI.iterator();
			Iterator p = portfolio.iterator();
			while (it1.hasNext() && it2.hasNext() && p.hasNext()){
				ArrayList position = new ArrayList ((ArrayList)p.next());
				double val1 = new Double ((Double) it1.next());
				double val2 = new Double ((Double) it2.next());
				int k = Double.compare(val1, val2);
				if (k == -1){
					sellFIXSTRING = genFIXSTRINGsell(position);
					this.tempS = o.newOrderSingle(sellFIXSTRING);
					cancelFIXSTRING = genFIXSTRINGcancel(position);
					this.tempC = o.cancelOrderRequest(cancelFIXSTRING);
					positionUpdate = positionUpdate(tempB, tempC, position,
positionUpdate);
				}
				portfolio = positionUpdate;
			}
			this.portfoliok = portfolio;
		}
	}

	public void buy(double varT, List varI, List portfolio){
		Order o = new Order();
		String buyFIXSTRING = "";
		String cancelFIXSTRING = "";
		List positionUpdate = portfolio;
		if (this.varTk > varT){
			Iterator it1 = this.varIk.iterator();
			Iterator it2 = varI.iterator();
			Iterator p = portfolio.iterator();
			while (it1.hasNext() && it2.hasNext() && p.hasNext()){
				ArrayList position = new ArrayList ((ArrayList)p.next());
				double val1 = new Double ((Double) it1.next());
				double val2 = new Double ((Double) it2.next());
				int k = Double.compare(val1, val2);
				if (k == 1){
					buyFIXSTRING = genFIXSTRINGbuy(position);
					this.tempB = o.newOrderSingle(buyFIXSTRING);
					cancelFIXSTRING = genFIXSTRINGcancel(position);
					this.tempC = o.cancelOrderRequest(cancelFIXSTRING);
					positionUpdate = positionUpdate(tempB, tempC, position,
positionUpdate);
				}
				portfolio = positionUpdate;
			}
			this.portfoliok = portfolio;
		}
	}

	public List positionUpdate(String temp,String tempC, ArrayList position,List portfolio){

	  /* calculation of state position order sell temp="tempS"
	* state position order buy temp="tempB"
	   * state position order cancell "tempC"
	   * of position i-th
	   * update of i-th position value of portfolio
	  */

		return positionPortfolio;
	}

	public double quoteUpdate(String tempQ, ArrayList position){
```

```
                /*
                * calculation of quote position
                 * of position i-th
                 * update of i-th value value of portfolio
                 */
                 return quotePosition;
        }

        public String genFIXSTRINGsell(ArrayList position){
                String sellFIXSTRING = "";
                //sell position
                return sellFIXSTRING;
        }

        public String genFIXSTRINGbuy(ArrayList position){
                String buyFIXSTRING = "";
                //buy position
                return buyFIXSTRING;
        }

        public String genFIXSTRINGcancel(ArrayList position){
                String cancelFIXSTRING = "";
                //cancel action sell/buy
                return cancelFIXSTRING;
        }

        public String genFIXSTRINGquote(ArrayList position){
                String cancelFIXSTRING = "";
                //cancel action sell/buy
                return cancelFIXSTRING;
        }

}
```

File Admin.java

```
package hft.carlone.unich.it.admin;
/*
* <p>Title: HFT Project</p>
* <p>Description: Demo about high frequency trading technique</p>
*
* <p>The author may be contacted at:
*           gcarlone@unich.it
*           giulio.carlone@gmail.com</p>
*
* @author Giulio Carlone
* @version 1.0
*/
public class Admin {

        private String loginval;
        private String logoutval;
        private String fixstringval;

        public Admin(){}

        public String logon(String IPSERVERNAME,int PORTNAME){
                return loginval;
        }

        public String logout(String FIXSTRING){
                /*
                 *
                 *
                 */
                return logoutval;
        }

        public String heartbeat(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
```

```
        }

        public String testRequest(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
        }

        public String resendRequest(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
        }

        public String reject(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
        }

        public String resetSequence(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
        }
}
```

File Order.java

```
package hft.carlone.unich.it.handling;
/*
* <p>Title: HFT Project</p>
* <p>Description: Demo about high frequency trading technique</p>
*
* <p>The author may be contacted at:
*           gcarlone@unich.it
*           giulio.carlone@gmail.com</p>
*
* @author Giulio Carlone
* @version 1.0
*/
public class Order {

        private String fixstringanswer;

        public Order() {}

        public String newOrderSingle(String FIXSTRING){

                return fixstringanswer;
        }

        public String cancelOrderRequest(String FIXSTRING){
                return fixstringanswer;
        }

        public String orderMassCancelRequest(String FIXSTRING){
                return fixstringanswer;
        }

        public String orderCancel(String FIXSTRING){
                return fixstringanswer;
        }

        public String orderReplaceRequest(String FIXSTRING){
                return fixstringanswer;
        }

        public String crossNewOrderMessage(String FIXSTRING){
                return fixstringanswer;
```

```
        }

        public String crossOrderCancelRequest(String FIXSTRING){
                return fixstringanswer;

        }
}
```

File Quote.java

```
package hft.carlone.unich.it.handling;
/*
* <p>Title: HFT Project</p>
* <p>Description: Demo about high frequency trading technique</p>
*
* <p>The author may be contacted at:
*           gcarlone@unich.it
*           giulio.carlone@gmail.com</p>
*
* @author Giulio Carlone
* @version 1.0
*/
public class Quote {

        private String fixstringval;
        public Quote(){}

        public String quoteUpdate(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
        }

        public String quoteCancel(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
        }

        public String quotesMassCancel(String FIXSTRING){
                /*
                 *
                 */
                return fixstringval;
        }
}
```

File RiskManagement.java

```
package hft.carlone.unich.it.riskmanagement;

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
/*
* <p>Title: HFT Project</p>
* <p>Description: Demo about high frequency trading technique</p>
*
* <p>The author may be contacted at:
*           gcarlone@unich.it
*           giulio.carlone@gmail.com</p>
*
* @author Giulio Carlone
* @version 1.0
*/
public class RiskManagement {

        private double varTot;
        private double vari;
        private double volatility;

        public RiskManagement(){ }
```

```
        public double varT(double a, double b, double t, List c, List e, List f){
        /* total var of portfolio P (1...I...N) positions
         *
         * N = number of positions;
         * c = Volatility vector of the (i-N) position;
         * f = Coefficient vector of correlation between the positions
         * of the i-th and j-th in the c vector
         * d = Fraction vector (1-N) of wealth invested in the i-th position;
         * a = confidence interval;
         * b*(t(exp(1/2))= Time interval.
         */
                return varTot;
        }

        public double vari(double volatility, double var1, double var2,double var3){
                // var of the i-th position
                return vari;
        }

        public double volatilityK(double t, int n, double ri){
        /* the volatility of the I-th position
         * ri= i-th value, observed at time t;
         * n = number of observations.
         */
                return volatility;
        }

}
```

Conclusion

In this proposal we considered the type and syntax of messages that must be sent to the system to be accepted by MIT. We also introduced the literature about the strategies and the risks that a organization should manage. Finally, we examined a case of explicit algorithm with the corresponding flowchart and relative mild implementation.

The examination of our case has as a result a single original combination of elements structured in a manner that provides a well-defined idea of the risks what you are dealing with, representing a step forward in research since they explicitly disclose all theobjects involved in the technical HFT Italian Stock Exchange, and which must therefore provide the basis for further theoretical suppositions combining financial instruments. A further development would be to operate on the MTA and simultaneously using the same algorithm on the derivatives market, the SOLA platform, with a trading technique at high frequency.

We can say that even after gaining cyclically of such prices, the information does not interfere with the measurement of the correlation between changes in the performance of the stock relative to the market. The effect cannot be attributed to more efficient pricing that adapt quickly to new information and that, therefore, will not affect the calculation of the' index of variation percentage of prices over time.

Other studies that lead to greater volatility of the index have achieved an increase in the performance of AT/HFT; greater volatility of specific indices to achieve a reduction in the performance of AT/HFT and that the higher the activity of AT/HFT then lowers the volatility. The AT/HFT improves efficiency in general. AT/HFT may routinely have an adverse effect on the ease with which the action may be treated on the market and that in general the effects of AT/HFT on market quality may vary over time considering high degree of transparency of static indicators' s, the multiplicity of maturities and issuers, the low spreads and high liquidity.

References

[1] E. Boehmer et al. (2012) "*International Evidence on Algorithmic Trading"*, working paper, SSRN.
[2] V. Caivano, S. Ciccarelli, G. Di Stefano, M. Fratini,G. Gasparri, M. Giliberti, N. Linciano, I. Tarola (2012) *"High frequency trading Definition, effects, policy issues"* , discussion papers CONSOB
[3] Broogard (2012) "*High Frequency Trading and Volatility*", working paper SSRN.
[4] European Commission (2004) Directive 2004/39/CE on markets in financial instruments.
[5] ESMA (2012) "*Guidelines: Systems and controls in an automated trading environment for trading platforms, investment firms and competent authorities*".
[6] Foresight: (2012) "*The Future of Computer Trading in Financial Markets",* Final Project Report, The Government Office for Science.
[7] van Kervel V. (2012) "*Liquidity: What you see is what you get?*", working paper.
[8] Borsa Italiana (2012) Istruzioni al Regolamento dei Mercati Organizzati e Gestiti da Borsa Italiana S.p.A.
[9] Borsa Italiana (2012) Regolamento dei Mercati Organizzati e Gestiti da Borsa Italiana S.p.A.
[10] London Stock Exchange Group (2012) FIX Trading Gateway (FIX 5.0).
[11] European Commission (2011) proposals for a Directive on markets in financial instruments repealing Directive 2004/39/EC.
[12] Irene Aldridge (2010) "*High-Frequency Trading, A Practical Guide to Algorithmic Strategies and Trading Systems"*, John Wiley & Sons, Inc.
[13] Biais, B. and P. Woolley "*High frequency trading*", Working paper, IDEI Toulouse.
[14] Brogaard, J., (2010) "*High frequency trading and its impact on market quality*", Working paper, University of Washington.
[15] Hendershott, T., C. Jones, and A. Menkveld (2010) "*Does algorithmic trading improve liquidity?*", Journal of Finance 66.
[16] Giulio Carlone (2013) "*Algorithmic Trading*", working paper 2013 Rmetrics Meielisalp Workshop