# Algorithmic Trading at Bucharest Stock Exchange

Adrian Victor SĂNDIŢĂ[1]
[1]University of Craiova, Faculty of Mathematics and Natural Sciences
asandita@inf.ucv.ro

Abstract: *Very conservative estimates indicate that over 40% of transactions on the stock exchanges in the United States are based on automatically generated orders. Such systems are designed to do algorithmic trading on the basis of a predefined set of rules that determine the composition of the portfolio and the moment in which the purchases and sales of securities are done. Applying highly diverse trading strategies, algorithmic trading systems ultimately aim to maximize profit and minimize risk taking. Algorithmic trading on the Bucharest Stock Exchange is still in its incipient phase. Now, automated trading systems are used only for participants who act as Market Makers for the actions of a few issuers. Estimates indicate a volume of algorithmic trading on the Bucharest Stock Exchange of under 1% of* its total transactions*. This paper aims to describe a general way that algorithmic trading systems can be connected to the Bucharest Stock Exchange and to present some of our results in the implementation of such a system.*

**Keywords: Algorithmic trading; Multi Agent Systems; Information management; Bucharest Stock Exchange**

**JEL classification: C41, C81**

## 1. Introduction

Applications that are oriented to agents represent a new paradigm of software engineering that appeared through the merger of two technologies: artificial intelligence and distributed computing (Odell & Burkhart, 1998).

The term "agent" or "software agent" is increasingly common in recent years in computer networks, artificial intelligence and databases. Although there is no universally accepted definition of the term, all definitions indicate the agent as being a specialized software component, having a relative autonomy and behaving like a human agent, interacting with the environment in which they operate and possibly with other agents (F. Bellifemine, G. Caire, D. Greenwood, 2007).

Although one can imagine situations in which certain issues can be solved by a single agent that interacts with the environment and communicate with users, complex systems require the presence of several agents. These systems, called multi-agent systems (MAS) can model complex situations and may involve defining agents with common or contradictory objectives. Agents can interact with each other directly, through communication and if needed through negotiation or indirectly through joint environmental action. Agencies may decide to cooperate for mutual benefit or to compete in order to fulfill their tasks.

Thus, the relevant characteristics of agents are: *autonomy*, as it can operate without the direct intervention from the user and possess the capacity to control their actions and internal conditions; *social skills*, which allow them to collaborate with each other (or the user); *reactivity*, which represents the ability to perceive the environment in which they act and react to changes.

Oriented Modeling allows agents to have an unconventional approach to system design, including defining the components and system integration. A multi-agent system is used to solve a complex problem that can not be solved by a single entity. Coordination conduct independent agents is a central part of the multi-agent system design.

Agents have the ability to efficiently process local data and communicate with other agents when necessary, for example when the tasks are beyond their scope of knowledge or exceed their processing ability.

Multi-agent systems have been used in a wide range of applications such as e-commerce, e-learning, data mining, simulation, robotics, transportation systems and grid computing.

## 2.Objectives

This paper aims to present the interface of a multi-agent system designed to generate, insert and automatically cancel purchase orders and sales in the BSE system.

Trading decisions are based on technical analysis algorithms and techniques to minimize and control risk.

Introducing and modifying or canceling orders is performed using the Trading Gateway Arena system provided by BSE.

## 3. System Analysis

- Arena Gateway

Produced by BSE and available to agents acting in the romanian stock market in, Arena Gateway (AG) is a complex application that facilitates message transfer between the central system and applications dedicated to the participants (ADP). It offers services request / response, event-based services, and connectivity. Using a system of XML messages transmitted over the network, AG receives orders and requests from ADP, sends them to the stockmarket's central system and provides answers and market data to the applicant. The Arena Gateway is installed on a computer from the user's internal network and connects via VPN to the stockmarket's system. The Client Arena Gateway, ADP is the dedicated application, acquired or produced by the participant connected to the AG, that can send commands can receive and process responses.

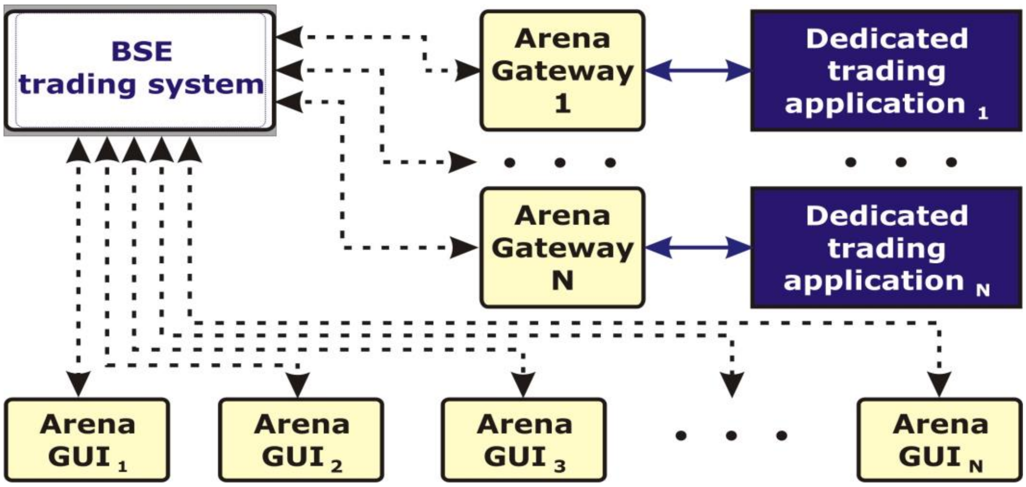

**Fig. 1.** BSE transaction system connections

The communication protocol between the AG and ADP is a concatenation of standardized sequences, the first sequence being start, followed by the message body, a MD5 checksum on the message body and a final length. Start and end sequences are 9 bytes each, 2-byte checksum, while the message body has a variable length. For the encoding of the message body the UTF-8 character set (***U**niversal Coded Character Set **T**ransformation **F**ormat* + ***8**-bit*) is used.

To avoid confusion, the body of the message may not contain the start or end sequences. ADP works with two types of messages:

• outgoing messages: messages that are sent to the central stock market system by the ADP using AG;

• incoming messages: messages from the central system via AG.

For the application, the message body is interpreted as an XML formatted text. XML message structure is fully described using XML schema file, supplied with the application. Each outgoing message will be reviewed and validated using the schema file. If it detects a message that does not comply with the constraints scheme AG sends an error report to the ADP and ignores the message.
It should be noted that some of the messages that describe the same market condition may vary from one participant to another. For example, when a participant introduces a market order, the Broker Code field of the message is filled with the Id if the message is sent to the one who introduced order and is void if message sent to any of the other participants.

*Outgoing messages description*

Any outgoing message consists of a command designed for the central control system, followed (if needed) by specific parameters (if the certain command has parameters) and a numerical sequence, called client sequence – *csq*, generated and managed by the client application that enables it to identify the answer received after the order processing of the stock market system. The csq field is not modified by the stock market system.
An outgoing message has the following general structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<m:outgoingEngineMessage xmlns:c="http://www.bvb.ro/xml/ns/arena/gw/constraints"
xmlns:m="http://www.bvb.ro/xml/ns/arena/gw/msg"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <content xsi:type=…>
   …
   </content>
   <csq>nnnnn</csq>
</m:outgoingEngineMessage>
```

*Input messages description*

Each received message has a header and an internal structure that incorporates an order confirming a report, information on the change of market data etc. All these structures are called **D**ata **T**ransfer **O**bjects (**DTA**).

An incoming message has the following general structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<m:incomingEngineMessage xmlns:c="http://www.bvb.ro/xml/ns/arena/gw/constraints"
xmlns:m="http://www.bvb.ro/xml/ns/arena/gw/msg"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
   <content xsi:type=…>
      …
   </content>
   <csq>nnnnn</csq>
   <err>false</err>
   <id>213</id>
   <kmsg>to ADMIN: Hello</kmsg>
   <ktime>2015070827174849441</ktime>
</m:incomingEngineMessage>
```

A recieved message can either be a response to a previous command, in which case it has a sequence equal to the client sequence order or can be a market event desription, in which case the *csq* value is zero. The common fields of a message entry are described below:

| **Field name** | **Description** |
|---|---|
| id | Message ID. In relation to the value of this field, the ADP can interpret the message content. |
| ktime | The moment in which the message was generated by the central system, in the form of yyyyMMddHHmmssSSS: an+lună+zi+ore+minute+secunde+milisec. |
| kmsg | Comment, message or error description |
| err | Error indicator. Can be true or false |
| csq | Client sequence, used to identify the receiving message after transmitting a command with the same csq. If the value is 0, the message is unsolicited. |
| content | Message content, built from one or more data structures packed together. This can also have a null value. |

*Monitoring the connection to the Gateway Arena*

Once a connection is established, the ODA can test the connection status by sending Heartbeat (*HeartBeatCmd*) messages to the gateway. When AG receives such a message, it replies with a void content message.

*Sending orders and the confirmation process*

ADP can send commands to the AG at any time after logging in. Any command that reaches the central system is followed by a response confirming the execution of the order or a specific error message if the command could not be executed. In the event of an error message, the message body is null, but the *kmsg* field contains explanations about the error.

*Report requests and response processing*

ADP may request complex AG structural information in the form of complex structured reports. A report request sent by the Central System AG will determine the issue by requesting a response containing one or more pages of records or an error message. If an error message appears, the entry message content is null, but the *kmsg* field contains the description of the error that occurred. As the number of entries on a page is limited, in order to get all the information in a report, ADP has to request further data until it receives the last page of records.

*Subscription mechanism*

In order to receive information about a symbol, ADP must inform the central system that it is interested in the certain symbol by using the *LoadSymbolCmd* command. The central system memorizes the symbol and provides date to the customer until it is no longer requested and explicitly solicits to not receive more data (the *UnloadSymbolCMD* Command). By a similar mechanism, to receive market data for a symbol-market entity, ADP subscribes to the pair symbol-market through the *AddSubscribeCmd* command. The data will be received until the ADP opts out of receiving them via the command *DelsubscribeCmd*.

*Change processing*

After a successful authentication, ADP will receive a *MarketPictureDto* structure containing a list of market entities: symbols, symbol-market pairs, indices etc. Typically, each entity contains two types of information: its properties and its current state. Whenever an entity or the state of ownership of the entity changes, ADP is notified so that they can build an exact local image of the entity.

A symbol-market entity is a composed entity, consisting of the symbol and the market in which it is traded. For a symbol there may be several symbol-market pairs, as a symbol may be traded on multiple markets using different trading mechanisms.

The properties of an entity may be delivered by the Gateway as an integrated *ExchangeExplorerDto* structure in various messages. For example, an instance in which such information is available is within a *UserEnvDto* after reaching the connection or after login every time those properties are changed during a session within the *TickersPack* message.

The summary of an entity (except the symbol-market) will be present at the time of login as a structure incorporated in a package *UserEnvDto* or during the session as a list of *PublicTickerMsg* packed in a message *TickersPack*.

Structural changes (for example when a symbol is removed from a given market) will arrive during the session *ExchangeExplorerDto* like structures packed in a *TickersPack* message.

*The processing of Level 1 market*

Level 1 refers to information on the current trading symbol-market entities (best bid/ask, the last transaction, the total volume, the total value variation, the opening price, the minimum price, maximum, etc.). Level 1 data are supplied in two different ways: first, when logging in using symbol-market summary pairs and secondly, during the session, as *CommonTickersPack* structures embedded inside *TickersPack* messages.

We emphasize once again that the Level 1 data will only be available to subscribed market-symbol pairs.

When a single order transaction occurs at different prices, ADP will receive a number of *TickersPack* messages which is equal to the number of distinct transaction prices.

*The processing of Level 2 data*

In order to get a complete picture of a pair symbol ball-market, ADP must first obtain an initial picture of the order book which will be updated after any change through integrated structures within *ActionTickerPack*.

The initial image is obtained by a *GetMarketByOrderCmd* command launched when the session starts. The answer to this command will be a *MboDto* structure containing the order book as a pair of lists *OrdDto* structures.

Even if for a symbol, *MboDto* contains only one list of orders on both sides (buying and selling), the elements of this list are sorted according to the trading priority for each part.

ADP client lists must build two orders: one for buying and another one for selling, without changing the original sorting order. This is required because changes in the order book which arrive as ATP objects must be applied to the initial list.

Own orders can be identified in their order book by testing the field own which will then have the value *true*. For reasons of confidentiality, orders entered by other participants will have certain fields set to irrelevant values (e.g., for an order released by another participant, the field retaining the account of the user that gave the order will always be set to 0) .

*Arena Gateway Cache*

Arena Gateway maintains its own cache in the online order book in order to respond more quickly to requests from *GetMarketByOrder*. After *UserEnvDto* order is processed, ADP will begin to receive a series of unsolicited messages which will contain orders from the order book. The series begins with a start message followed by messages which describe the order registry for each subscribed market-symbol pair and ends with a message announcing the end of the initialization operation

After the end of the initialization, the *GetMarketByOrder* command will get the answer built in the cache if the user is subscribed to the requested symbol-market and the order book is in the cache. If the user is not subscribed to the symbol-market, the *GetMarketByOrder* order will receive an error message. If the user has recently subscribed to a market-symbol pair and the order register has not yet been fully loaded in the cache, as a reply to the *GetMarketByOrder* command, the ADP will get an error message which signals that the update has not yet been finished.

| **Command Type** | **Command Name** | **Description** |
|---|---|---|
| System | LoginCmd | Login command |
| | LogoutCmd | Logout command |
| | HeartBeatCmd | Test the connection between the client and the gateway |
| Business | LoadSymbolCmd | Load a symbol in the client's virtual environment |
| | AddOrderBuyCmd | Buy order command |
| | AddOrderSellCmd | Sell order command |
| | AddCrossOrdersCmd | Add cross orders for immediate execution |
| | UpdateMMOrdersCmd | Add or change a pair of orders ( a quote ) |
| | CancelMMOrdersCmd | Delete a pair of orders ( a quote ) |
| | AddSubscribeCmd | Subscribe to a symbol-market |
| | CancelOrderCmd | Cancel an order |
| | ChgOrderCmd | Change an order |
| | DelSubscribeCmd | Unsubscribe from a symbol-market |
| | GetMarketByOrderCmd | Get order book snapshot |
| | ReleaseOrderBuyCmd | Resume a buy order |
| | ReleaseOrderSellCmd | Resume a sell order |
| | SuspendOrderCmd | Suspend an order |

| | | |
|---|---|---|
| | UnloadSymbolCmd | Unload a symbol from the client's virtual environment |
| | MailCmd | Send message to another user |
| Report Request | Find2AccountCmd | Find details about two accounts in a single request |
| | FindAccountCmd | Find details about an account |
| | FindFreshOrderReportCmd | Find an order ( by order number) |
| | GetGenericOrderAuditCmd | Get all records related to an order ( order audit) |
| | GetOrdersDailyLogCmd | Get all records related to all orders ( for current member) |
| | GetOutstandingOrdersCmd | Get outstanding orders ( for current member) |
| | GetDailyTradesCmd | Get trades from current date ( for current member) |
| | GetUsersByNameCmd | Get users ( connected or by member ) |
| | GetSymbolsByNameCmd | Get list of symbols (based on load status) |
| | GetDailyPublicTradesCmd | Get all public trades from current date |
| | GetDailyIndicesCmd | Get the value of indices from current date |
| | GetStepsCmd | Get all the lists of price steps |
| | GetPublicParametersCmd | Get the parameters for all markets and symbol-markets |

- *The Multi Agent System*

The Multi Agent System architecture consists of four types of interacting agents. The system enables easy scaling and adding of agents for real-time data analysis (Sandita, 2014). Its architecture is shown in Figure 2.

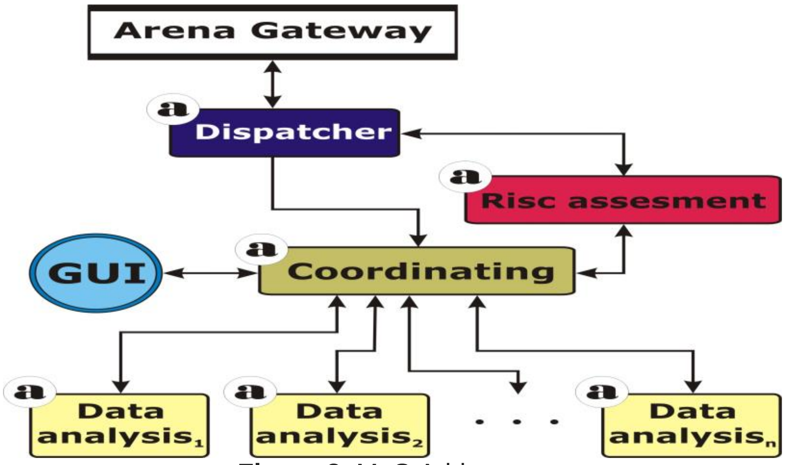

**Figura 2.** MaS Arhitecture

The main components of multi-agent system are:

*The coordinating agent* is the most complex component of MAS. Equipped with s graphical interface, it provides the user the necessary information about the system status and the market developments. At the same time, it enables the user to intervene in the trading process in real time and to modify the parameters of the agent system.

When deemed appropriate, responding to market conditions, it enables or disables data analyzing agents.

*The risk assesment agent* is placed on either one of the local area network computer and has the task of monitoring transations done by the coordonating agents. It receives transation data from the dispecer agent and according to the preset risk threshold and the market conditions it sends a stop ongoing actions signal to the other system agents. Before the stop signal, the risk evaluator signals via system message its entry into action. After the interruption caused by the agent, the system must be started manually by the user.

*Data analysis agents* are specialized agents that receive market information via the coordinating agent, analyse quotations and volume of transactions in real time and transmit signals to it. They are equipped with an internal evaluation mechanism that determines the workload assessment data and quantifies the time necessary to accomplish the tasks. When exceeding a preset threshold, the data analysing agents emit to the coordinator an overload signal. In this situation, the coordinator activates another analysis agent, dormant on another workstation and relieves the overloaded agent of one of the current tasks by transferring it to the new agent. Shares and reliability degree given for each agent are predefined and stored in the database assigned to the agent system.

*The dispatcher agent*, the subject of this paper, provides the data transfer from the Gateway Arena to the MAS. In the following we will present the way it works and its accomplished tasks.

## 4. Implementation and Results

The dispatcher agent, liaises with the Gateway assigned to the application and can be placed on any of the computers on the internal network.

At system initialisation, through *LoginCmd* command the dispatcher agent autheticates into the BSE system and ensures the consistency of the data stream flowing between trading server and system agents. It is always active and directs the flow of unsolicited data to the coordinator agent, which in turn transfers them to the agent dedicated to data analysis.

The dispatcher agent periodically generates to the Gateway and to the coordinator heartbeat signals to ensure the continuity of data links.

Now, the frequency of heartbeat signals is one every two seconds.

If the Heartbeat message sent to the Gateway doesn`t receive a reply, the dispatcher agent transmits to the coordinator an error message and attempts to reestablish communication after an interval of 5 seconds. If it manages to reestablish the connection with the Arena Gateway, it sends the current system status data to the coordinating agent and flow resumes.

Upon receiving the error message on the lack of connection with the Arena Gateway, the coordinating agent in turn emits an error message to the graphic interface; this way the user is informed about the interruption of data flow.

If the Heartbeat message sent to the coordinating agent receives no response, the dispatcher sends an error signal to the broker via *MailCmd* command, routed through the Arena Gateway.

As it is possible for the message to not reach the recipient if both connections are down simultaneously, the coordinator agent has its own mechanism for handling communication errors. While the coordinator agent does not send Heartbeat messages to the dispatcher agent, it notifies the lack of messages from it, and after a time frame set at 4 seconds, it sends to the GUI a warning message.

Because communication failures within the MaS can leave the system in an unstable state, the occurrence of such errors involves initiating a manual reset by the user.

In this case, after the Login and restoring of the current state of the system, all market orders placed earlier are automatically canceled and GUI is forwarded a report on operations carried out in the current trading session BSE.

In this way, it eliminates the possibility of the system losing control over previously submitted orders, before the loss of connection, or to ignore transactions made while being disconnected.

The dispatcher agent has a data transfer synchronizing mechanism and a tracking of potential time lags mechanism in terms of data received from the Gateway and that transferred to the MAS.

To this point, there was no record of deferral of such data.

The system presented above is realized in Delphi and it is in a real envinroment testing period since early 2015. So far it suffered major transformations only regarding the mechanism of data analysis agents administration.

## 5. Conclusions

Mobile agent systems can be successfully used in practice. The scalability and reliability of such a system has been demonstrated by implementing it in a real and strongly competitive environment such as that of the BSE. The distribution of tasks across multiple calculus systems facilitates the delivery of performance due to the global uniform loading of such a system.

The challenge that we will have to face is the implementation at data analysis agents level of more efficient change trend of quotations detection techniques.

Greater attention will be paid to the agent coordinating the implementation of an automated system of weights that allow quantifying and assessing the reliability of data provided may be granted one or other analytical agency. Currently, static weights are given by the user, based on previous results.

Acknowledgements. This work was cofinaced from the European Social Fund through Sectoral Operational Programme Human Resources Development 2007-2013, project number POSDRU/159/1.5/S/140863, Competitive Researchers in Europe in the Field of Humanities and Socio-Economic Sciences. A Multi-regional Research Network.

## References

Bellifemine F., Caire G., Greenwood D. (2007), *Developing Multi-Agent Systems with JADE*, John Wiley & Sons, Ltd.

Hendershott T., Riordan R. (2013), Algorithmic Trading and the Market for Liquidity, Journal Of Financial And Quantitative Analysis.

Odell, J., & Burkhart, R. (1998). *Beyond objects: Unleashing the power of adaptive agents*. Tutorial presented at OOPSLA, Vancouver, B.C.

Posland S (2007), Specifying Protocols for Multi-Agent Systems Interaction, ACM Trans. Autonom. Adapt. Syst.

Sandita A. (2014), Automated transaction based on multi agent systems, Conferinţa „Provocările discursului academic actual: teme, tendinţe, metode”, Brașov.

Stoll H.(2010), Electronic Trading in Stock Markets, Journal of Economic Perspectives

* * *, Arena® Gateway 3.0.0, User Manual, Bucharest Stock Exchange, 2014.