ASETS – An Academic Trading Simulation Platform

Claudiu VINȚE, Alexandru LIXANDRU, Andrei JURUBIȚĂ, Adrian BARDAN[1]
Bucharest Academy of Economic Studies
claudiu.vinte@ie.ase.ro, alex.lixandru@gmail.com, andrei.jurubita@stud.ase.ro,
adrian.bardan@a3d.ro

This paper is intended to present the results of our academic research upon a distributed computing environment dedicated to trading simulation. Our research has been conducted with the aim of creating a trading simulation platform, that would provide both the foundation for future experiments with trading systems architectures, components, APIs, and the framework for research on trading strategies, trading algorithms design, and equity markets analysis tools.
Mathematics Subject Classification: 68M14 (distributed systems).
Keywords: *Trading Systems, Simulation, Distributed Computing, Service-Oriented Architecture (SOA), Message-Oriented Middleware (MOM), Java Message Service (JMS)*

1. Introduction

The initial idea of a trading simulator within academia came along as a necessity, under the auspices of the Master's program in Economic Informatics[2], which proposed for the first time, as part of curriculum for the university year 2009-2010, the course upon *The Informatics of the Equity Markets*. ASETS is the acronym from the Romanian version of the Academy of Economic Studies Trading System. Our research has been directed toward designing a trading environment that would create the opportunity for students to study in details the investor's needs from an electronic trading platform, the components of a trading system and their functionality in a *straight through processing* approach, and the trading strategies that can be implemented to corroborate models for automatic (program) trading.
In our view, ASETS platform lays the foundations for multiple directions of research, concerning electronic transactions on the equity markets [1].
Today's stock exchanges are high-tech organizations. The main parts of an electronic trading platform are [2] [3]:

- the user *front end*, consisting of a trading system and a trading interface (API);
- the *network*, consisting of access points to a wide area backbone network and interfaces that are provided to members;
- the *back end*, which handles the major functions of trading and trade management, market supervision and control, information dissemination, and provides industry standard interfaces to settlement organizations, information vendors and market data vendors.

Through its components, ASETS simulation platform supplies functionality in all these three areas. Apart from settlement and clearing activities, ASETS covers the entire transactional process of ordering, order matching, execution generation and capture, trade disseminated and client portfolio management [4]. In addition, within the simulation environment, there are real world delayed-prices disseminated to the investors, who can also access the simulated-market depth for each tradable financial instrument, and the *market map*.

[1] ASETS project involved, in a modularly fashion of design and development, the entire Master's program series of students
[2] The Master's program organized by the Department of Economic Informatics, within the Faculty of Cybernetics, Statistics and Economic Informatics

2. The approach to trading simulation

The sole traders in a pure order-driven market are the investors who are seeking to buy or sell shares for their own portfolio purposes. They are sometimes referred to as "the naturals". Two basic order types used by naturals in a trading environment are:

- *limit orders* – a maximum price limit is placed on a buy order, and a minimum price limit is places on a sell order;
- *market orders* – the instruction on a market order is simply to buy or sell "at market", which means that the order will be matched with the best available position (depending of the side) existent in the marketplace.

The limit orders, which are entered into a limit order book, establish the prices at which the market orders will execute. The market is order-driven precisely because the limit orders placed by some participants set the values at which others can trade by placing market orders. In such an environment, the limit order placers are the liquidity suppliers, and the market order traders are the liquidity takers. In an order-driven market, the liquidity builds as limit orders are entered in the book, and liquidity is drawn down as market orders trigger trades that eliminate limit orders from the book.

Some participants are motivated to be liquidity providers because, whenever a trade is made, the transaction price typically favors the limit order placer. For instance, assume that the best bid set by a limit order placer seeking to buy is 10 monetary units, and that the best offer set by a limit order placer seeking to sell is 10.5. If a market order to buy arrives, it will execute at 10.5, or 0.5 monetary units more than the limit order buyer would pay if his or her limit order were to execute. Similarly, if a market order to sell arrives, it will execute at 10, or 0.5 monetary units less than the limit order seller would receive if his or her order were to be executed [5]. On the other hand, while the market order trader pays more for a purchase or receives less for a sale, he or she benefits from trading with certainty and immediacy [6].

Two conditions must be met for an order-driven market to function:

- some participants must be looking to buy at a time when others are looking to sell;
- on each side of the market, some participants must choose to place limit orders while others must select the market order strategy.

Taking into to account the above-mentioned aspects, we came to the realization that having only live participants on the marketplace would not create enough liquidity in the simulation environment and, therefore, we created a specially tailored module for pouring limit orders into the market, for each tradable financial instrument. This component, Pseudo-Random Order Generator (PROG), plays in these circumstances the role of a liquidity supplier, while the live participants, who are allowed to place market orders along with limit orders, benefit from such a simulation environment as liquidity takers.

ASETS has been designed to reflect a real world, order-driven market structure, aiming to capture the dynamic properties of price formation. The simulation environment is electronic, and it currently accepts only two basic types of orders (market and limit).

A trading simulator can be based on various approaches:

- *canned data* – where quotes, orders, prices and trades are taken from a historic transaction record, and the live participants trades are done against the historic prices;
- *computer-generated data* – where the simulation itself creates the entire market environment, the quotes, and the transactions record; the live participants trades are done against the simulated prices;
- *computer-generated data with real world price seeds* – where the computer-generated orders are placed at price levels departed from the real market prices, although randomly created within predefined spreads.

The canned data approach is limited in two respects. First, a live player's own orders cannot affect the record of past prices – they continue to be the past stored prices. In the real world, the live trader's orders can affect the evolution of prices in the marketplace. Second, with canned data, it is not possible to rerun a simulation using different parameter settings and/or trading strategies, because the transaction record is the product of the specific market that produced it. On the other hand, with solely computer-generated data, the trading simulation would lack the connection with the real world prices and market evolution.
ASETS was conceived as a trading simulation environment where live participants' orders coexists with computer-generated orders, created departing from delayed prices captured from the Bucharest Stock Exchange (BSE) [7]. The assumption was that only the orders placed by the live participants would not be enough for sustaining the liquidity of the simulation environment [8]. Consequently, a special component has been design (Pseudo-Random Order Generator – PROG) to play the role of a *market maker*, by placing regularly buy and sell orders, for each tradable financial instrument, at prices confined within predefined (parameterized) spreads from the delayed prices received from BSE. Following this approach, ASETS offers the ability of conducting simulation-trading against a marketplace very similar to the one supplied by BSE. Based on the delayed market-data[1], PROG module is designed to take into account, when it comes to order generation, the evolution of the real market in terms of price variation, order volume, and number of trades already completed. Furthermore, by reproducing the key parameters of a real marketplace, ASETS platform can offer a virtually even more active and liquid trading environment, creating the fundaments for our intended future research in trading algorithms and market analysis tools.

3. A service-oriented architecture with an underlying messaging middleware

One of desiderates that we embraced from the initial phase of this research was the commitment to employing open source technologies throughout the trading simulation environment. Our goal has been to provide to the users of the trading simulation system a convenient way of accessing the platform from the internet, through the means of employing a servlet responsible for HTTP tunneling [9] [10]. Essentially, ASETS *graphical user interface* (Trading GUI) has been implemented as a Java applet, which can be launched from a web browser. The potential user (investor) has first to register on the ASETS web site, choosing an user name and obtaining an automatically generated password (which may be changed later on). Trading GUI applet is the only component of the system that the participants at the trading activity come in contact with. All the other components of the simulation platform are transparent to the end-user, and create an environment that replicates the perspective of being connected to a live trading marketplace. The architectural design is one of *service-orientation* (SOA). Each component of the system exposes its functionality, as a service provider, to the other components [11]. The requests for services and the supplies of replies are flowed through a *message-oriented middleware* (MOM). The intention and the format of this paper do not afford us the necessary space to go into all the details of ASETS message-oriented trading API. For further references, can be consulted our website: www.iem.ase.ro. A MOM makes use of a message provider (broker) to mediate the messaging operations. In this parading, the elements of a MOM-based system are client applications, messages, and MOM messaging provider. Under the broad umbrella of client applications, can be in fact identified cetain applications that play functionally the role of a client, and others that have the functional role of a server. All the system applications are perceived as clients of the MOM messaging provider [12]. Using a MOM system, a client

[1] Along with prices, Delayed market-Data Feed (DDF) supplies information upon trading volume, value, number of trades etc.

makes an API call to send a message to a destination managed by the provider. The call invokes provider services to route and deliver the message to the consumer. Once it has sent the message, the producer can continue the processing flow, relying on the fact that the messaging provider retains the message until a receiving client retrieves it. In this manner, the MOM-based model, in connection with the message provider, open the possibility of creating a system of loosely coupled components. Such a system can continue to function reliably, without downtime, even when individual components or connections fail. The client applications are consequently effectively relieved of every communication issue, except that of sending, receiving and processing messages. Through an administrative tool coupled with the messaging provider the user can monitor and tune the performance of the communication flows. Fig. 1 shows the architecture of ASETS simulation platform. The functional services (modules) are interconnected using ASETS API, described in [13].

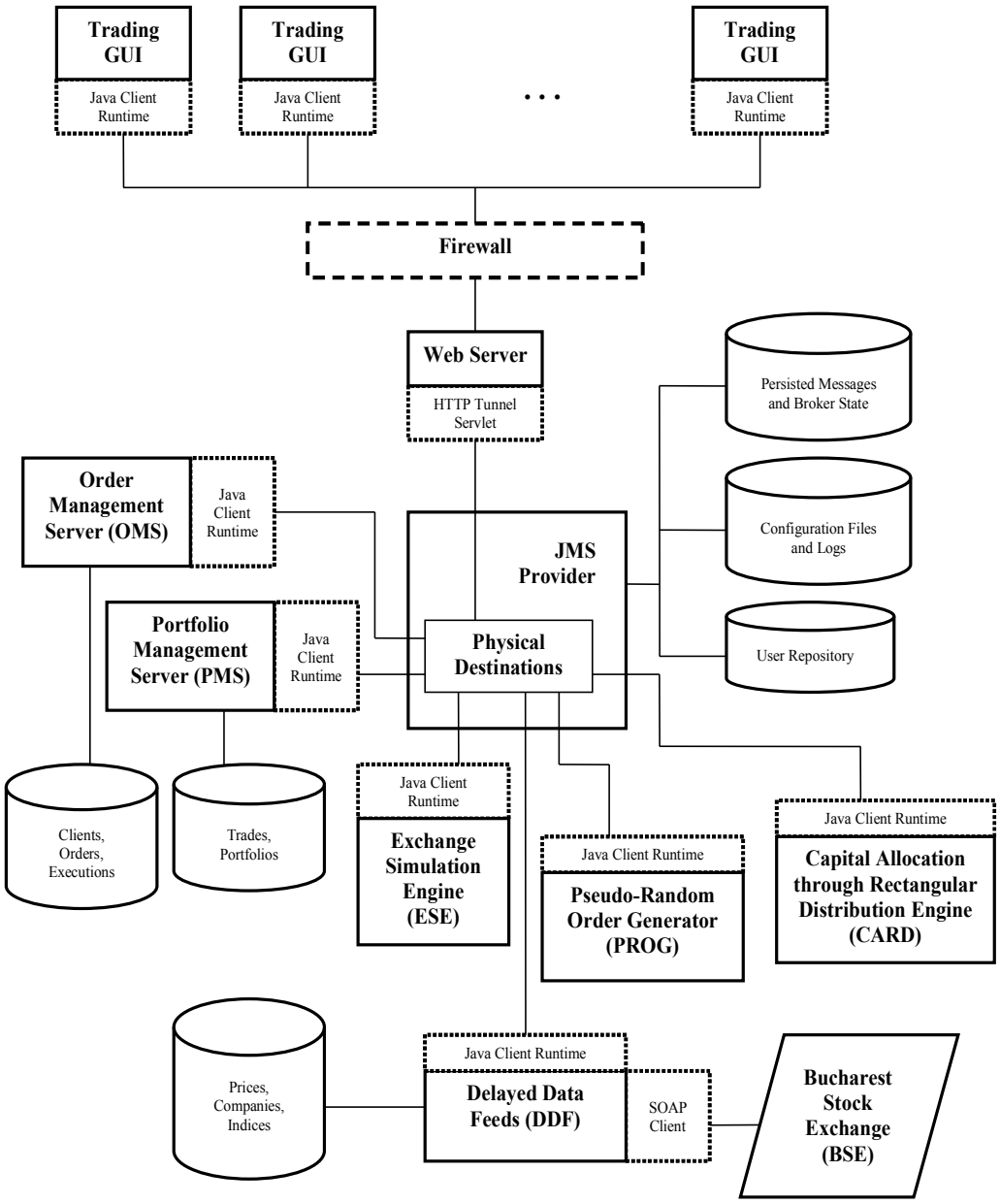

Fig. 1 – The architecture of ASETS trading simulation platform

ASETS API has been designed and implemented in conjunction with Java Message Service (JMS) API. JMS specification captured, from its conception, the essential elements of a generic messaging systems, namely:

- the concept of a messaging provider that routes and deliver messages;
- distinct messaging patterns, or domains such point-to-point messaging and publish/subscribe messaging;
- facilities for synchronous and asynchronous message receipt;
- support for reliable message delivery;
- common message formats such as text, byte and stream.

Summarizing, messaging is a very effective means of building the abstraction layer within SOA, needed to fully abstract a business service (functionality) from its underlying implementation. Through business messaging, the business service (say, the order booking) does not need to be concerned about where the corresponding implementation service is located, what language it is written in, what platform it is deployed on, or even the name of the implementation service. All the above-mentioned elements have equally constituted the reasons why we turned to Open Message Queue (OpenMQ), as the open source MOM implementation of JMS, for designing ASETS architecture based on it.

4. System components and functionality

ASETS trading simulation platform has a modular service-oriented design. Each module plays a precise role within the system, the simulation environment being conceived to deliver a real-world-like trading experience to a potential investor. The main modules and their functionality are described below.

Order Management Server (OMS) is chiefly responsible for processing orders placed by the market participants, managing user connections, and channeling the executions generated by the order-matching engine back to the users.

Portfolio Management Server (PMS) delivers the trade generation processing, and client portfolio maintenance.

Exchange Simulation Engine (ESE) is the module where the order-matching takes place. It implements the matching algorithm, and plays the role of a stock exchange.

Delayed-Data Feed (DDF) consists of a collection of web-clients that connect to corresponding web services, intended to capture delayed market-data disseminated by The Bucharest Stock Exchange (BSE). The feed gathers data regarding the financial instruments traded on BSE, listed companies and their status, prices, volumes, exchange indices etc. The captured data is stored in the system database. Delayed prices are also published to a specific topic within the messaging provider, topic at which the system components interested in them can subscribe.

Pseudo-Random Order Generator (PROG) is the module that creates buy and sell limit orders, departing from the real market data captured by DDF from BSE, and place them on the simulated marketplace offered by ESE. As we mentioned earlier, this module provides the needed liquidity to an environment with an expectedly modest trading activity coming from live participants, when compared to a real stock market.

Capital Allocation through Rectangular Distribution Engine (CARD) is the component that generates the rectangular distribution for creating the map of the market. The *market map* is a GUI feature that provides to the investor a tridimensional perspective of the stock market [14]. The area of a rectangle corresponds to the market capitalization of a listed company. The color of a rectangle is based on the price variation of the stock, within the considered interval of time:

- nuances of red for loses;

- black for stagnation of stock price;
- nuances of green for price gains on the market.

Trading GUI is the graphical interface offered to the users of ASETS platform. The GUI can be launched from a web browser and it runs as a Java applet within browser's window. Once the investor created a valid user name from ASETS website, and that user name was *enabled*, then the GUI applet can be launched and the investor may be able to connect to the ASETS simulation platform (Fig. 2).

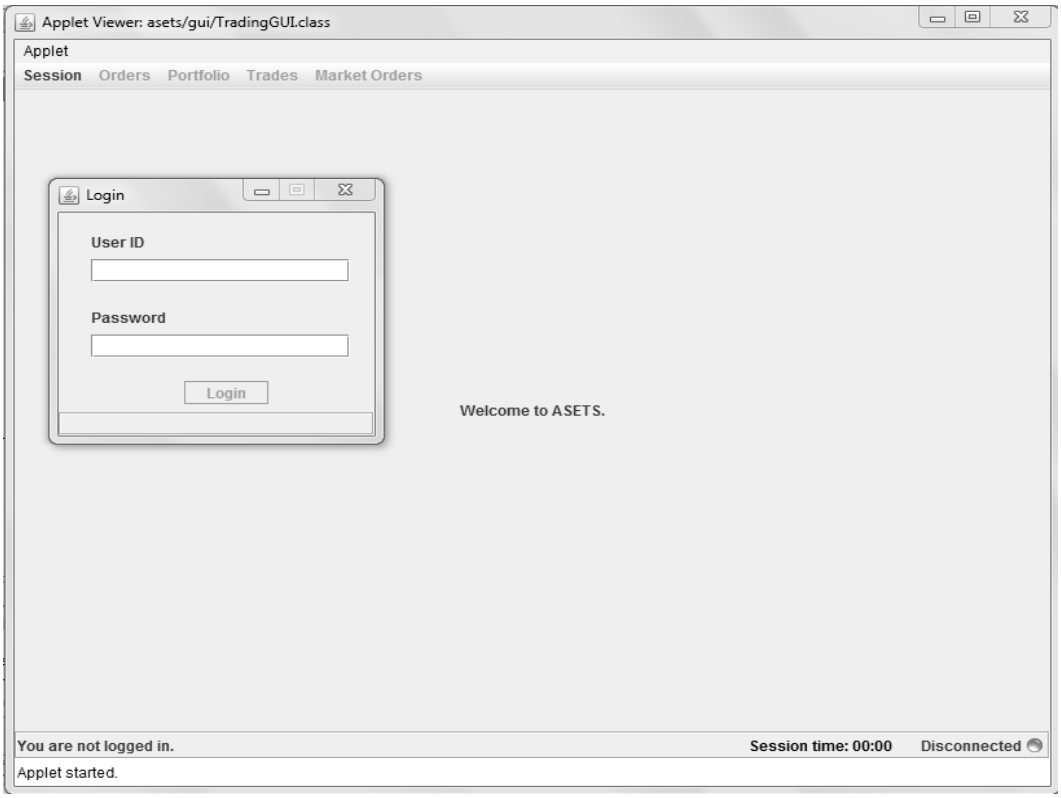

Fig. 2 – The main workbench of ASETS GUI, containing the login window

Once the user name and the password are authenticated by the system, the investor will be able to retrieve his or her trading activity realized during the current trading session. ASETS platform provides daily trading sessions, meaning that the investor orders are valid within the trading session they were placed in. At the end the day, all the orders that were not executed, or not fully executed, are considered as canceled, and they are not carry on to the next trading session. The system does not handle *good-till-canceled* (GTC) type of orders.
The workbench of ASETS GUI is organized in two panels, for *Orders* and *Executions*. The *Orders* panel contains all the orders the user placed during the current trading session. Each order has assigned to it an order (system) status and a market status. From the trading system perspective, an order may be in one of the following states:

- **`ADD`**, newly created order, which has not been accepted by the market (ESE) yet;
- **`ADDED`**, newly created order accepted by the market;
- **`ADD_FAILED`**, newly created order, which was not accepted by the market;
- **`UPDATE`**, order update, which has not been accepted by the market yet;
- **`UPDATED`**, order update accepted by the market;
- **`UPDATE_FAILED`**, order update, which was not accepted by the market;
- **`CANCEL`**, order cancel, which has not been accepted by the market yet;

- **CANCELED**, order cancel accepted by the market;
- **CANCEL_FAILED**, order cancel, which was not accepted by the market;
- **EXECUTED**, order executed, either fully or partially.

From the market perspective, a trading order may fall in one of the below categories:

- **PENDING**, if there was not received any acknowledgement from the market (ESE);
- **ON_MARKET**, when the order was acknowledge to have reached the market;
- **PARTIALLY_EXECUTED**, when the ordered quantity was partially filled;
- **FULLY_EXECUTED**, when the ordered quantity was entirely satisfied;
- **CLOSED**, when the order was successfully canceled and removed from the market.

The *Executions* panel contains the market-generated executions, corresponding to the orders placed by the user, and which were captured by OMS and supplied to GUI (Fig. 3).

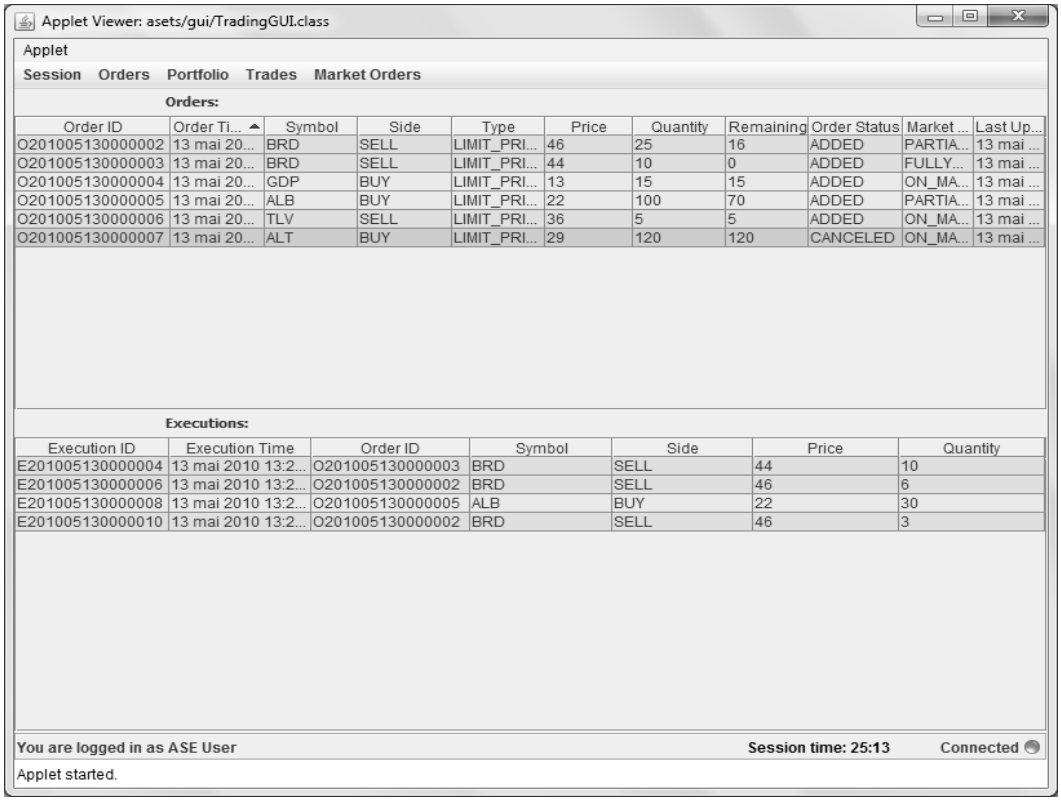

Fig. 3 – ASETS GUI with *Orders* and *Executions* panels

From the GUI, the investor has the ability to place new buy and sell orders. The total value of the buy orders has to be within the limits of a system defined cash amount, available to each investor once he or she registered into the system. As for the sell orders, the investor may only sell a financial instrument that possesses in his o her portfolio. ASETS simulation platform currently does not support short selling, but this aspect may be subject to change in the future. The user interface for order entry was designed to help investor make convenient choices, in terms of symbol for the desired financial instrument to be ordered, and the type of order that he or she intends to place (market or limit). This approach also reduces the chance of potential input errors from the investor's part. The principle followed here was to ensure a robust input-data validation locally, in order to minimize the probability of action failure, and reduce the overall data flow among component systems. A successfully placed order can be subsequently changed (updated) or canceled.

The order update operation allows only for:

- changing a limit order into a market order, but not the vice versa;
- decreasing the ordered quantity, for both market and limit orders.

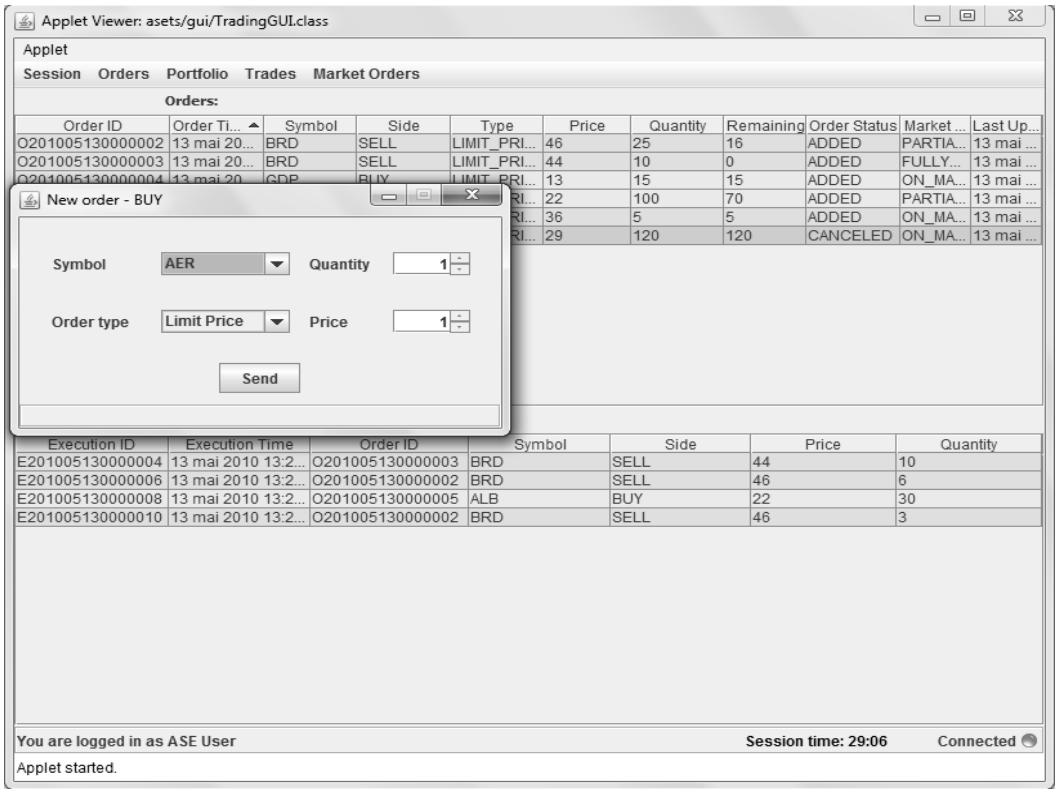

Fig. 4 – The input window for placing a new order (buy side)

The order placement window is shown in Fig. 4. To place in a sell order, the investor would need to know exactly what financial instruments he or she possesses, and in what quantity. ASETS GUI offers to the investor the ability to view and save his or her portfolio of acquired financial instruments, keeping its value up to date through a price feed supplied by DDF, as Fig. 5 illustrates.

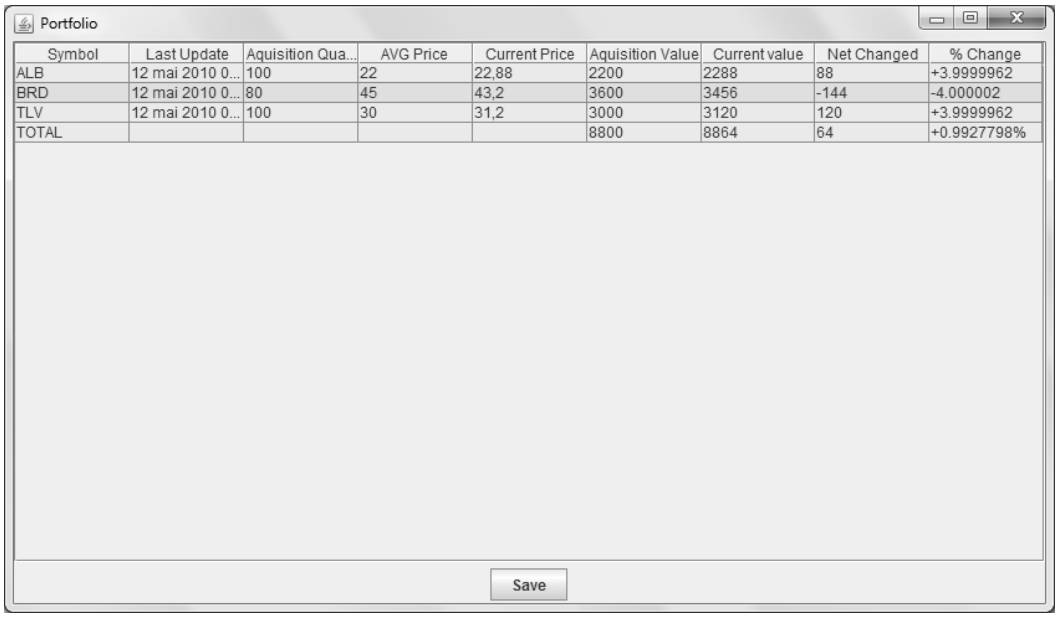

Fig. 5 – Investor's portfolio window

Through the GUI, the live participant to the simulation environment is also able to consult the market depth for a given trading symbol, as Fig. 6 shows. Furthermore, the investor can

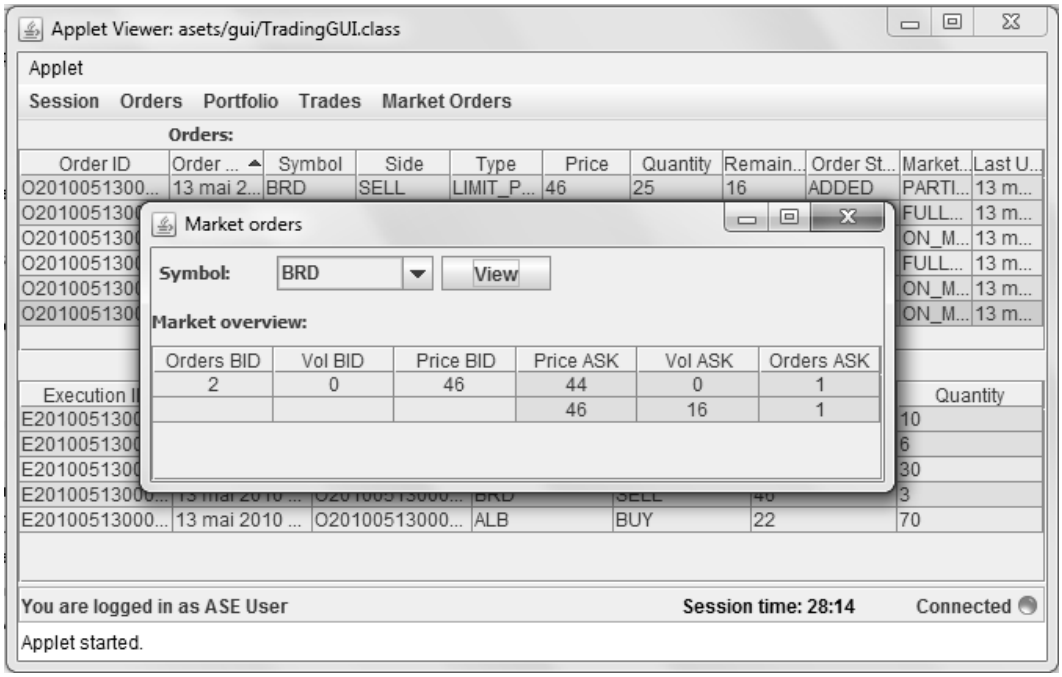

Fig. 6 – Market depth window, showing the current orders on the exchange for symbol BRD

benefit from the *market map* tridimensional perspective (Fig. 7).

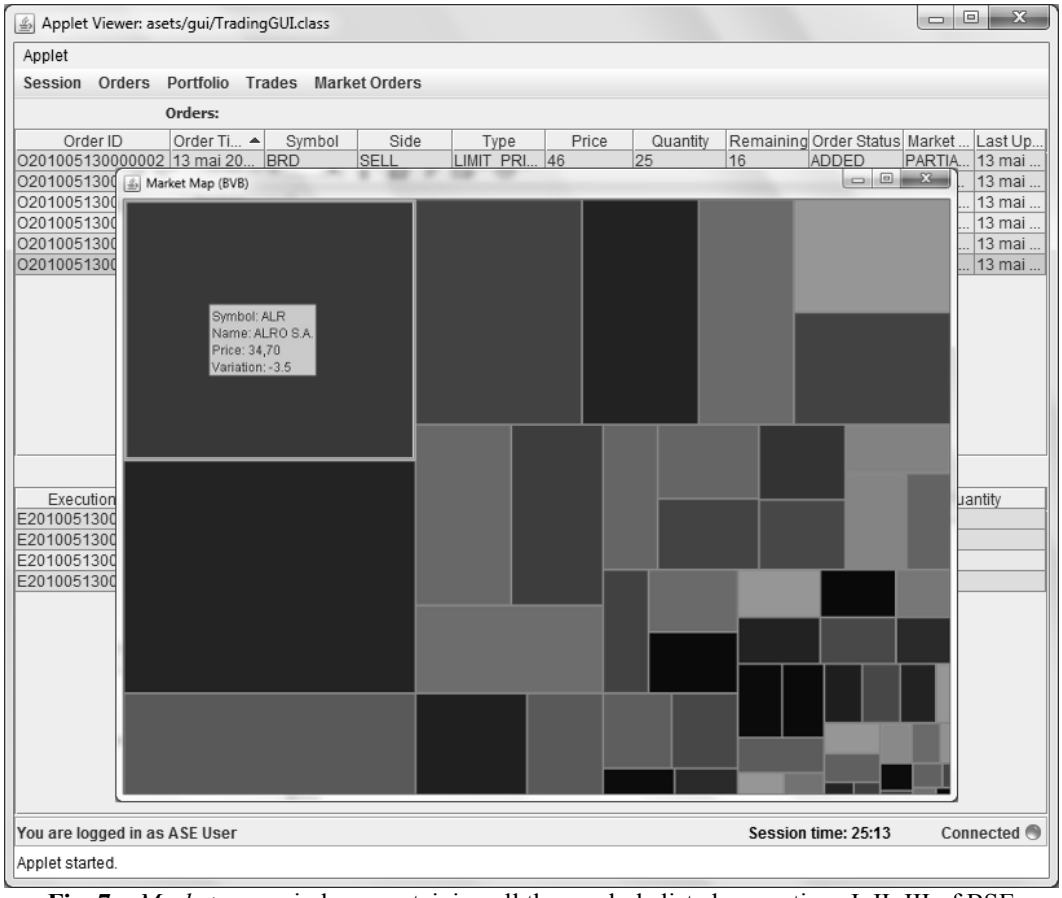

Fig. 7 – *Market map* window, containing all the symbols listed on sections I, II, III of BSE

5. Conclusions and future directions of research

Our research has been directed toward designing a trading environment that would create the opportunity for the students to study in details the investor's needs from an electronic trading platform, the components of a trading system and their functionality in a *straight through processing* approach, and the trading strategies that can be implemented to corroborate models for automatic (program) trading. In a simulation-trading environment, human agents compete on resources created by computer algorithms, within a scenario-driven market place. The components that create these scenarios have to *sense* the trading patterns of the human investors, and act accordingly. By designing a trading API for ASETS platform based on a message-oriented middleware, we achieved a fine balance, concerning the overall system response, availability, reliability, and flexibility in accepting future changes and extensions.

The use of messaging, as part of the overall service-oriented trading simulation, allows for greater architectural flexibility and agility. These qualities are achieved through the use of abstraction and decoupling. With messaging, subsystems, components, and even services can be abstracted to the point where they can be replaced with little or no knowledge by the client components. Architectural agility is the ability to respond quickly to constantly changing environment.

In our view, ASETS platform lays the foundations for multiple directions of research upon electronic transactions on the equity markets. Based on the current results, we intend to focus our research on two primary domains: market analysis (trend identification) and algorithmic trading. These two domains are cascading style interconnected: an efficient model for algorithmic trading cannot be conceived without the ability to identify or predict, in a timely fashion, the market directions of evolution.

Acknowledgements

We would like to extend our thanks and appreciations to the entire 2009-2010 Master's program series of students, who contributed to ASETS project, in a modularly fashion of design and development. Special thanks have to go the following students, who willingly devoted their time and energy in the final integration phase of the project: Alexandru SIROMASCENCO (DDF, PROG), Cristina PAUNĂ (GUI), Andreea MARCU (PMS), Irina MARCU (PMS), Mădălina APOSTOL (ASETS website), Gheorghe SOROCEAN (GUI), Sabina Monica POPESCU (API), Irina MANEA (ESE), Ştefan DRAGOMIR (PROG).

References:

[1] McIntyre Hal (editor) – *Straight Through Processing* - The Summit Group Publishing, Inc., New York, 2004

[2] McIntyre Hal (editor) - *How the U.S. Securities Industry Works - Updated and Expanded in 2004* - The Summit Group Press, New York, 2004

[3] Schwartz A. Robert, Francioni Reto - *Equity Markets in Action (The Fundamentals of* Liquidity, Market Structure & Trading) - John Wiley & Sons, Inc., 2004

[4] Harris Larry – *Trading and Exchanges* – Oxford University Press, Oxford, 2003

[5] Vinţe Claudiu - *The Informatics of the Equity Markets - A Collaborative Approach* – in Informatica Economică vol. 13, no. 2/2009, INFOREC, Bucharest, 2009

[6] Katz Jeffrey Owen, McCormick L. Donna – *The Encyclopedia of Trading Strategies* – McGraw-Hill, New York, 2000

[7] Tanenbaum S. Andrew, Maarten van Steen - *Distributed Systems - Principles and Paradigm* - Vrije Universiteit Amsterdam, The Netherlands, Prentice Hall, New Jersey, 2002

[8] Vinţe Claudiu - *Upon a Trading System Architecture based on OpenMQ Middleware* – in Open Source Scientific Journal, Vol.1, no.1, 2009 - http://www.opensourcejournal.ro/

[9] Sun Microsystems, Inc. – *Java Message Service* - http://java.sun.com/products/jms/
[10] Sun Microsystems, Inc. - *Open Message Queue: Open Source Java Message Service (JMS)* - https://mq.dev.java.net/
[11] Erl Thomas (with additional contributors) - *SOA Design Patterns* – Prentice Hall by SOA Systems Inc., New Jersey, 2009
[12] Richards Mark, Monson-Haefel Richard, Chappell A. David - *Java Message Service (Second Edition)* – O'Reilly Media Inc., Sebastopol, California, 2009
[13] Vințe Claudiu – *Upon a Message-Oriented Trading API* – in Informatica Economica Journal vol. 14, no. 1/2010
[14] Vințe Claudiu – *Upon a Tridimensional Perspective of the Stock Market* – proceedings of The Ninth International Conference on Informatics in Economy, May 7-8 2009

Claudiu VINȚE has over thirteen years experience in the design and implementation of software for equity trading systems and automatic trade processing. He is currently CEO and co-founder of Opteamsys Solutions, a software provider in the field of securities trading technology and equity markets analysis tools. Previously he was for over six years with Goldman Sachs in Tokyo, Japan, as Senior Analyst Developer in the Trading Technology Department. Claudiu graduated in 1994 The Faculty of Cybernetics, Statistics and Economic Informatics, Department of Economic Informatics, within The Bucharest Academy of Economic Studies. He holds a PhD in Economics from The Bucharest Academy of Economic Studies. Claudiu has also been given lectures and coordinated the course and seminars upon *The Informatics of the Equity Markets*, within the Master's program organized by the Department of Economic Informatics. His domains of interest and research include combinatorial algorithms, middleware components, and web technologies for equity markets analysis.

Ionuț-Alexandru LIXANDRU graduated The Bucharest Academy of Economic Studies in 2008. He is a M. SC student in the field of Economic Informatics within The Faculty of Cybernetics, Statistics and Economic Informatics, with the M. SC thesis *Distributed System for Supporting Stock Exchange Transactions in a Simulation Environment*. Alexandru has been for over 3 years with TechTeam Global, within the Global Business Applications Department. His main areas of interest are systems integrations, web technologies, software maintainability, and knowledge management.

Andrei JURUBIȚĂ is student of The Bucharest Academy of Economic Studies, and he works as a web programmer for Globalsys Solutions. Andrei graduated in 2008 The Faculty of Cybernetics, Statistics and Economic Informatics, Department of Economic Informatics, and he is currently following the Master's program in Economic Informatics. He was awarded with a prize at the Informatics Olympiad in High School, and he was awarded 3rd place at Infomatrix 2004 with "C++ for Kids", working within a team of four. His domains of interest and research include compression, cryptography, distributed systems, operating systems.

Adrian-Ion BARDAN graduated in 2008, and has a Bachelor Degree in Informatics from the University of Bucharest. He will graduate in June 2010 the Master's program in the field of Economic Informatics within the Bucharest Academy of Economic Studies, with the final thesis upon *Applying OCR algorithms using a distributed system*. His main area of expertise includes the design and development of web applications, while being also interested in software architectures, new web technologies, Java EE programming, and mobile applications development.