Paulo Manuel B. R. Santos

A.I. Stock Exchange

Departamento de Ciências de Computadores
Faculdade de Ciências da Universidade do Porto
Janeiro / 2007

Paulo Manuel B. R. Santos

A.I. Stock Exchange

Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Informática (Ramo de Sistemas e Redes)

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Janeiro / 2007

Agradecimentos

À minha esposa Lilian, aos meus pais Manuel e Elmira, ao meu irmão Ricardo, e à minha avó Alice, por todo o apoio, força e compreensão, que me deram ao longo desta caminhada.

Ao meu orientador e professor Eugénio, pela sua competência, experiência e com o qual aprendi bastante, apesar de não ter podido aproveitar ao máximo a sua disponibilidade.

A todos os meus professores e colegas que, de alguma forma, contribuíram para que este projecto visse a luz do dia.

Finalmente, a todos os meus amigos, pela sua amizade e incentivo nas horas mais difíceis.

Resumo

Numa época em que a informação corre a uma velocidade estonteante, onde a cada segundo são debitadas para a rede quantidades imensas de novos dados e existem inúmeras fontes de informação, torna-se imperioso o uso do poder computacional existente nos dias de hoje para que, com essa informação, nós possamos construir matrizes de apoio à decisão, decisão essa que se quer atempada e munida de todos os factores relevantes e actuais.

Se existe uma temática onde essa afirmação se torna mais preponderante e verdadeira, é precisamente a dos mercados das Bolsas de Valores e toda a sua envolvente, onde o ritmo a que a informação chega é tal, que não existe capacidade de absorção humana capaz de a processar, e onde qualquer posição que seja tomada, nem que seja um segundo mais tarde, pode significar a perda de milhões.

Daí surgiu a ideia para este trabalho, que atendendo às teorias existentes de análise financeira e fazendo uso das tecnologias disponíveis, e na tentativa de conseguir combiná-las da melhor forma possível aproveitando o melhor de cada uma, pretende assim dotar a comunidade de investidores de uma ferramenta, que se imagina ser há muito esperada e lhes permita tomar decisões de investimento no momento certo maximizando os seus rendimentos.

Ao longo das próximas páginas apresentam-se então, todos os passos dados na análise, desenho e construção de dito sistema, desde a pesquisa sobre as ferramentas já existentes na mesma área, passando pela temática dos mercados de títulos e os métodos existentes para a avaliação do potencial de cada acção, nomeadamente os da análise técnica, e finalmente descrevendo todo o processo de desenvolvimento e experimentação do mesmo, detalhando os algoritmos matemáticos implementados.

Abstract

At a time where information runs at an astonishing pace, where each second incredible amounts of new data are debited to the net and there are numerous information sources, it becomes imperious the use of nowadays existing computational power so that, with that information, we can build decision support matrices, decision that we want timely and in possession of all relevant and actual factors.

If there exists a thematic where this statement becomes more preponderant and true, is precisely that of the Stock Markets and all it's enfolding, where the rate at which the information arrives is such, that a human absorption capability capable of processing it doesn't exist, and where any position that is taken, even if just a second latter, can mean the loss of millions.

From there the ideia for this work appeared, that attending at the already existing financial analysis theories and making use of the available technologies, and in the attempt to successfully combine them in the best possible way taking advantage of the best each one has to offer, aims in providing the investors community of a tool, one imagines to be long waited and that allows them to make investment decisions at the right moment maximizing their income.

For the following pages are presented then, all the steps given in the analysis, drawing and construction of said system, from the research on the already existing tools in the same area, passing through the thematic of the bonds markets and existing methods for the evaluation of the potential of each share, nominated those of the technical analysis, and finally describing all the process of development and experimentation of the same, detailing the implemented mathematical algorithms.

Abstrait

Dans nos temps où l'information circule à une vitesse étourdissante, où à chaque seconde sont débitées pour le réseau d'immenses quantités de nouvelles données et existent innombrables sources d'informations, se révèle impérieuse l'utilisation du pouvoir informatique existant de nos jours pour que, avec ces informations, nous peuvent construire des matrices d'aide à la décision, décision qui se veut rapide et pourvue de tous les facteurs importants et actuels.

S'il existe une thématique où cette affirmation devient plus prépondérant et vraie, c'est précisément celle des marchés des Bourses de Valeurs et toute son enveloppant, où le rythme auquel les informations arrivent est tel, qui n'existe capacité d'absorption humaine capable de la traiter, et où quel que soit la position que soit prise, ni qu'est une seconde plus tard, peut signifier la perte de millions.

Est à partir de là apparue l'idée pour ce travail, que en faisant attention des théories existantes d'analyse financière et en faisant utilisation des technologies disponibles, et dans la tentative de réussir à les combiner de la meilleure forme possible en profitant du meilleur de chacune d'elles, prétend ainsi doter la communauté d'investisseurs d'un outil, que s'imagine être beaucoup attendue et leur permette prendre des décisions d'investissement au moment exact en maximisant leurs revenus.

Au long des proches pages se présente alors, tous les étapes données dans l'analyse, le dessin et la construction de dite système, depuis la recherche sur les outils déjà existants dans le même secteur, passant par les thématique des marchés de titres et méthodes existantes pour le évaluation du potentiel de chaque action, notamment de l'analyse technique, et finalement en décrivant tout le processus de développement et l'expérimentation du même, détaillant les algorithmes mathématiques mis en oeuvre.

Conteúdo

Resumo **5**

Abstract **6**

Abstrait **7**

Índice de Tabelas **14**

Índice de Figuras **16**

1 Introdução **17**

1.1 Objectivo 17

1.2 Motivação 18

1.3 Estrutura do documento 18

2 Estado da arte **20**

2.1 Introdução 20

2.2 *Trendax - "The Money Making Machine"* 20

2.3 *Eclipse Trader* 22

2.4 *Merchant of Venice* 24

2.5 Conclusão 27

3 Mercado de títulos e métodos de avaliação **28**

3.1 Introdução 28

3.2 Mercado secundário 29

3.2.1 As bolsas de valores 29
3.2.2 Operações de bolsa 30
3.2.2.1 Operações no mercado à vista 30
3.2.2.2 As ordens de bolsa 30
3.2.2.3 A efectivação das cotações 32
3.3 As acções 33
3.3.1 Definição 33
3.3.1.1 Direitos estatutários 33
3.3.1.2 Direitos económicos 34
3.3.2 Tipos de acções quanto à sua forma de transmissão 34
3.3.3 Formato das acções 35
3.4 Avaliação das acções 35
3.4.1 Introdução 35
3.4.2 Risco e rentabilidade 36
3.4.2.1 O princípio e os limites da diversificação 36
3.4.2.2 As componentes do risco de uma acção 36
3.4.3 Métodos de avaliação de acções 38
3.4.3.1 O valor contabilístico 38
3.4.3.2 Óptica de dividendos 38
3.4.3.3 Rácios da análise fundamental PER, PCF e EPS . . . 39
3.4.3.4 Óptica de património e rendimento 41
3.4.3.5 Análise técnica 42
3.4.3.6 Indicadores da análise técnica RSI, SI e MACD 43
3.5 Conclusão 49

4 A plataforma *JATLite* 50

4.1 Introdução 50
4.2 Infraestrutura 51
4.3 O *Router JATLite* (AMR) 52

4.4 A linguagem KQML 55
4.5 Agentes de software 56
4.5.1 Introdução 56
4.5.2 Definições e atributos de agentes 57
4.5.3 Categorias de agentes 59
4.5.4 Aplicações de agentes inteligentes 60
4.6 Conclusão 61

5 Plataforma e desenvolvimento do sistema **62**
5.1 Plataforma de desenvolvimento 62
5.1.1 Introdução 62
5.1.2 Java Development Kit 5.0 62
5.1.2.1 Principais características da linguagem Java 63
5.1.2.2 Orientação ao objecto 64
5.1.2.3 Máquina Virtual Java 64
5.1.2.4 Independência da plataforma 65
5.1.2.5 Segurança 66
5.1.3 NetBeans IDE 5.0 66
5.1.4 JATLite Beta 0.4 67
5.1.5 Java Financial Library 1.6.1 67
5.1.6 JFreeChart 1.0.2 67
5.1.7 PostgreSQL 8.1 70
5.2 Desenvolvimento do sistema 71
5.2.1 Introdução 71
5.2.2 Modelação da base de dados 72
5.2.3 O agente Bolsa e os seus componentes 73
5.2.3.1 Interface gráfica e outras funções 73
5.2.3.2 *Thread* para gestão da comunicação com os agentes . . 74
5.2.3.3 *Thread* para gestão das tabelas de cotações 74

5.2.3.4 *Thread* para manutenção da base de dados 75
5.2.4 O agente Investidor e os seus componentes 75
5.2.4.1 Interface gráfica e outras funções 75
5.2.4.2 *Thread* para gestão da comunicação com o agente Bolsa 76
5.2.4.3 *Thread* para actualização das tabelas de acções e respectivas cotações 76
5.2.4.4 *Thread* para cálculo de oportunidades de investimento 77
5.2.5 Algoritmos e fórmulas matemáticas implementadas 77
5.2.5.1 Algoritmo RSI (*Relative Strenght Index*) 77
5.2.5.2 Algoritmo SI (*Stochastic Indicator*) 78
5.2.5.3 Algoritmo MACD (*Moving Average Convergence / Divergence*) 78

6 Experimentação **80**
6.1 Introdução 80
6.2 Algoritmo RSI (*Relative Strenght Index*) 80
6.3 Algoritmo SI (*Stochastic Indicator*) 81
6.4 Algoritmo MACD (*Moving Average Convergence / Divergence*) 84
6.5 Conclusão e resultados 88

7 Conclusões **90**

A Código SQL para criação das tabelas da BD **92**

B Descrição das classes **95**
B.1 Class **Bolsa** 95
B.1.1 Declaration 95
B.1.2 Serializable Fields 95
B.1.3 Constructors 96
B.1.4 Methods 97
B.2 Class **ThreadBolsaAG** 102

B.2.1 Declaration . . . 102
B.2.2 Constructors . . . 102
B.2.3 Methods . . . 102
B.3 Class **ThreadBolsaDB** . . . 107
B.3.1 Declaration . . . 107
B.3.2 Constructors . . . 107
B.3.3 Methods . . . 107
B.4 Class **ThreadCleanDB** . . . 112
B.4.1 Declaration . . . 112
B.4.2 Constructors . . . 112
B.4.3 Methods . . . 113
B.5 Class **Trader** . . . 113
B.5.1 Declaration . . . 113
B.5.2 Serializable Fields . . . 113
B.5.3 Methods . . . 114
B.6 Class **ThreadTraderAG** . . . 123
B.6.1 Declaration . . . 123
B.6.2 Constructors . . . 123
B.6.3 Methods . . . 124
B.7 Class **ThreadTraderST** . . . 134
B.7.1 Declaration . . . 134
B.7.2 Constructors . . . 134
B.7.3 Methods . . . 134
B.8 Class **ThreadTraderINV** . . . 135
B.8.1 Declaration . . . 136
B.8.2 Constructors . . . 136
B.8.3 Methods . . . 136

C Interface gráfica **138**

Lista de Abreviaturas **145**

Referências **147**

Lista de Tabelas

6.1 Resultados experimentais do algoritmo RSI 81
6.2 Resultados experimentais do algoritmo SI 81
6.3 Resultados experimentais do algoritmo MACD 84

Lista de Figuras

2.1 Funcionamento do processo de análise de notícias do *Trendax* 22

2.2 Ecrã de acompanhamento de mercados do *Eclipse Trader* 23

2.3 Ecrã de análise de gráficos do *Eclipse Trader* 23

2.4 Ecrã de histórico de cotações do *Merchant of Venice* 24

2.5 Ecrã de investimento em títulos do *Merchant of Venice* 25

2.6 Ecrã de gestão de *portfolios* do *Merchant of Venice* 25

2.7 Ecrã de análise de gráficos do *Merchant of Venice* 26

3.1 Gráfico RSI com divergências explícitas 44

3.2 Gráfico SI com as linhas K e D 45

3.3 Gráfico SI com divergências explícitas 46

3.4 Gráfico MACD com o respectivo Signal 47

3.5 Gráfico MACD com divergências explícitas 48

4.1 O *JATLite* é constituído por uma série de camadas especializadas . . . 51

4.2 Infraestrutura dos agentes e o papel do *router* (facilitador) 52

5.1 Exemplo da informação que pode ser obtida a partir do *site Yahoo Finance* 68

5.2 Gráfico do tipo *Price Volume* gerado pelo JFreeChart 69

5.3 Gráfico do tipo *Candlestick* gerado pelo JFreeChart 69

5.4 Gráfico do tipo *High Low with Moving Average* gerado pelo JFreeChart 69

5.5 Esquema da base de dados e a interdependência entre tabelas 73

6.1 Performance SI vs MACD 89

C.1 Ecrã de consulta de mensagens enviadas/recebidas 138
C.2 Ecrã de consulta, inserção e eliminação de registos da tabela mercado . 139
C.3 Ecrã de consulta, inserção e eliminação de registos da tabela acções . . 139
C.4 Ecrã de consulta e eliminação de registos da tabela de cotações *intraday* 140
C.5 Ecrã de consulta e eliminação de registos da tabela de cotações diárias 140
C.6 Ecrã de consulta de cotações com gráfico *intraday* 141
C.7 Ecrã de consulta de cotações com gráfico *price volume* 141
C.8 Ecrã de consulta de cotações com gráfico *candlestick* 142
C.9 Ecrã de consulta de cotações com gráfico *high low with moving average* 142
C.10 Ecrã de consulta de cotações com gráfico do indicador RSI 143
C.11 Ecrã de consulta de cotações com gráfico do indicador SI 143
C.12 Ecrã de consulta de cotações com gráfico do indicador MACD 144
C.13 Ecrã exemplo do *portfolio* de um investidor 144

Capítulo 1

Introdução

1.1 Objectivo

Esta dissertação foi realizada no âmbito do Mestrado em Informática da Faculdade de Ciências da Universidade do Porto e vem de encontro às matérias leccionadas, nomeadamente no que diz respeito à aplicação de Sistemas Multi-Agentes[1] e de conceitos de Inteligência Artificial em domínios e problemas bem reais da nossa sociedade.

É objectivo deste trabalho elaborar um Sistema Multi-Agente que simule tão próximo da realidade quanto possível a dinâmica de uma Bolsa de Valores e de todas as transacções que lá se efectuem. Parece então, haver aqui lugar para duas vertentes, a vertente da Bolsa em si como uma entidade única onde se centram todas as trocas e movimentações de capital, e de onde se obtêm todas as informações necessárias às transacções; e a vertente dos Investidores, num número mais alargado, que com mais ou menos conhecimentos nos domínios associados às conjunturas políticas, económicas e sociais, efectuarão pedidos de compra e venda de títulos, construindo assim os seus *portfolios*, e procurando obter mais valias das suas transacções.

Em termos práticos e no que diz respeito ao agente Bolsa, pretende-se usar informação real dos mercados, nomeadamente cotações, volumes transaccionados, variações de cotação, etc., recorrendo a serviços de cotações *online* com um diferimento mínimo no caso de não ser possível a sua obtenção em tempo real, e fornecer essa mesma informação aos outros agentes presentes no mercado. Quanto aos agentes Investidores, estes deverão deliberar e reagir sobre as informações veiculadas pelo agente Bolsa, e fazendo uso de algoritmos e técnicas modernas da análise técnica[2], deverão decidir quando comprar e quando vender, sempre com o objectivo de obterem o máximo rendimento de suas acções.

[1] Uma explicação mais detalhada sobre agentes de software será dada na secção 4.5

[2] Estas técnicas e algoritmos serão alvo de análise nas secções 3.4.3.5 e 3.4.3.6

1.2 Motivação

Este tema foi proposto pelo aluno, pois combina duas áreas de seu particular interesse, nomeadamente a área da Inteligência Artificial e de toda a sua envolvente, e a área da Análise e Gestão Financeira e de todas as temáticas com ela relacionada.

Num contexto como é o de um mercado de valores deste género, com imensas variáveis que poderão definir o sucesso ou o insucesso dos investimentos, a quantidade de dados e informação é quase absurda tornando-se impossível para um investidor humano assimilar tanta informação e ao ritmo que lhe é exigido no sentido de agir prontamente no meio ambiente em que está inserido. Justifica-se então o aparecimento de uma entidade computacional que faça uso desse poder de computação e juntamente com os conceitos e teorias das áreas económica e financeira que lhe forem facilitados, aja em conformidade com o perfil do investidor que representa, ainda mais, tendo em conta a conjuntura actual e o ambiente de instabilidade financeira que se vive, cada segundo de reacção que se perca pode levar ao insucesso.

Assim sendo, e dado que o próprio aluno é um pequeno investidor que nem sempre teve a melhor sorte nos seus investimentos, surgiu então a ideia de desenvolver este sistema, que não pretende enriquecer ninguém, mas que dará concerteza conselhos úteis e alertará para algumas oportunidades de investimento que sem esta ferramenta poderiam passar despercebidas.

1.3 Estrutura do documento

Este documento encontra-se dividido em sete capítulos e três apêndices, sendo que o primeiro dos capítulos corresponde ao actual, *Introdução*, onde além da estrutura do documento se apresentam o objectivo e a motivação para a realização deste trabalho.

No segundo capítulo, *Estado da arte*, são enumeradas e analisadas algumas das ferramentas existentes no mercado, que actuam na mesma área de acção a que esta dissertação se refere, focando-se sobretudo nos aspectos mais interessantes e diferenciadores de cada uma das soluções apresentadas.

No terceiro capítulo, *Mercado de títulos e métodos de avaliação*, trata-se de toda a teoria associada ao funcionamento dos mercados de títulos bem como dos métodos existentes para avaliação das acções, dando-se maior relevância aos métodos de análise técnica, que serão utilizados na vertente prática deste trabalho.

De seguida, no quarto capítulo, fala-se sobre *A plataforma JATLite*, onde se pretende não só abordar esta plataforma de comunicação entre agentes de software (pois será a usada no desenvolvimento da aplicação já mencionada), mas também apresentar diversas definições e aplicações para esses mesmos agentes.

No capítulo seguinte, o quinto, que trata da *Plataforma e desenvolvimento do sistema*,

fala-se numa primeira fase de todas as ferramentas necessárias ao desenvolvimento do sistema justificando em alguns casos essas escolhas. Na segunda parte, expõem-se todos os passos dados no desenvolvimento da aplicação, como por exemplo a modelação da base de dados, as estruturas dos agentes Bolsa e Investidor, bem como os algoritmos matemáticos implementados.

No sexto capítulo expõem-se todos os resultados obtidos na fase de *Experimentação* do sistema desenvolvido, comparando-se as *performances* dos diferentes algoritmos implementados e chegando por fim às conclusões apresentadas.

No último capítulo, o sétimo, correspondente às *Conclusões*, faz-se uma reflexão sobre o resultado final deste trabalho, nomeadamente sobre quais os objectivos que foram atingidos e quais não o foram.

Finalmente, e no que diz respeito aos apêndices, de referir apenas que tratam do código SQL para a criação das tabelas da base de dados, da descrição das classes Java desenvolvidas e da demonstração dos ecrãs criados para a *interface* gráfica, por esta ordem.

Capítulo 2

Estado da arte

2.1 Introdução

Nesta área de investigação e desenvolvimento não são muitas as ferramentas que existem, e aquelas que existem não passam de tentativas tímidas de criarem esse sistema de investimento automático, das quais veremos alguns exemplos mais adiante. Contudo, existe uma que interessa aqui realçar, e que afirma ser o supra-sumo das novas tecnologias de suporte ao investimento, congregando uma série de técnicas e estratégias inovadoras, que a serem verdadeiras, estaríamos perante a ferramenta por que muitos esperavam, esse sistema chama-se *Trendax*.

2.2 *Trendax - "The Money Making Machine"*

O *Trendax* nasceu em 2003, das mãos de um conhecido *hacker* e já condenado fraudador financeiro, de seu nome Kim Schmitz, mais conhecido como Kimble. Schmitz foi o fundador de um grupo de *hackers* chamados YIHAT (*Young Intelligent Hackers Against Terrorism*), que a certa altura afirmaram ter penetrado nas contas de membros da Al Qaeda, incluindo Osama Bin Laden, e que estavam muito próximos de o localizar, mas até hoje falharam em apresentar qualquer prova.

Pensa-se então, que este será mais um dos seus truques, desta vez com um pacote de impostura tecnológica, para satisfazer os sonhos de uma fortuna instantânea daqueles que acreditam em contos de fadas. Assim é descrito esse sistema no seu *site* oficial[1]:

> *"O Trendax é um sistema automático de investimento em Bolsa que opera simultaneamente em múltiplos mercados. O seu sistema de decisão baseado na em conceitos de inteligência artificial selecciona a combinação óptima*

[1] http://www.trendax.com

> *das estratégias de investimento tendo em conta as condições actuais do mercado. Trendax especializa-se em todos os mercados de futuros: acções, obrigações, moedas ou commodities. Apesar de se focar principalmente nos mercados de futuros, o Trendax também opera noutros mercados com propósitos de optimização e maximização de rendimentos.*
>
> *O Trendax tem capacidades cognitivas e é capaz de se aperfeiçoar a ele próprio, pois está continuamente a testar múltiplos cenários variando os parâmetros estratégicos sobre situações já passadas. Usa uma combinação complexa de análise técnica sofisticada, análise em tempo real de notícias, análise estatística multi-dimensional e técnicas proprietárias avançadas de matemática. O seu sofisticado sistema de gestão de investimentos assegura estratégias óptimas quer de entrada, quer de saída de um investimento, pois implementa uma gestão de risco estrita, limitando assim os riscos de uma descida maximizando os lucros. Além disso o Trendax é um sistema distribuído tolerante a falhas com múltiplos nodos localizados em pontos estratégicos de todo o mundo, assegurando que todos os investimentos são executados com sucesso, mesmo quando acontece alguma falha no sistema."*
>
> (Tradução de autor)

De entre as principais estratégias de processamento propostas pelo *Trendax* destacam-se as seguintes:

- redes neuronais para análise de movimentos de mercado com análise multi-dimensional das suas condições usando indicadores proprietários;
- análise léxica, um processador de linguagem natural e aprendizagem de máquina;
- análise inteligente do conteúdo das notícias usando combinação de padrões, *data mining* avançado e representação do conhecimento;
- optimização regressiva usando variados algoritmos matemáticos, lógica imprecisa, algoritmos genéticos e computação evolucionária.

Mas a característica que mais se evidencia, e aquela que é também a mais credível, é precisamente o seu **sistema inteligente de análise de notícias** (ver Figura 2.1) que, segundo eles próprios, se encontra continuamente a processar todas as notícias financeiras que lhe vão chegando e analisa sintacticamente o seu conteúdo em busca de palavras chave que em determinados contextos podem dar indicações precisas de oportunidades de investimento / desinvestimento.

Apesar de tudo isto parecer mais uma fraude de Kimble, temos que admitir que são propostas uma série de técnicas bastante interessantes e válidas também, ideias essas que a serem bem desenvolvidas dariam uma boa base para esse sistema de investimento automático que se deseja perfeito.

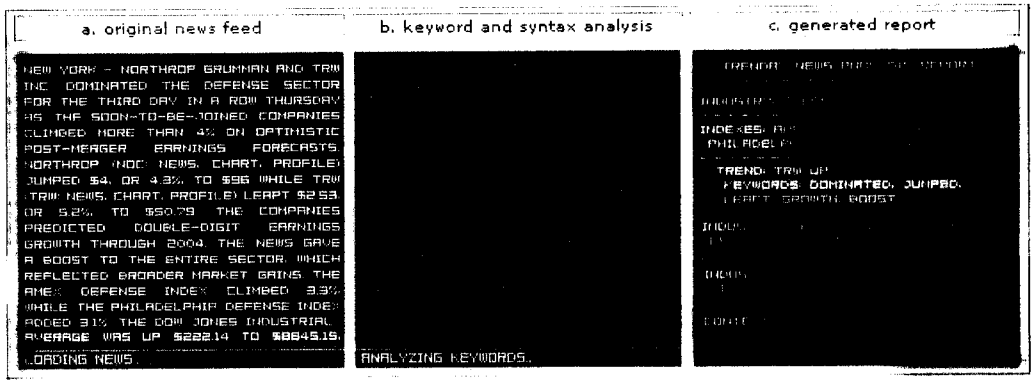


Figura 2.1: Funcionamento do processo de análise de notícias do *Trendax*

2.3 *Eclipse Trader*

O *Eclipse Trader*[2] é uma aplicação *open source*, cujo objectivo fundamental é proporcionar uma plataforma para a construção de um sistema de investimento *online*.

Actualmente conta com as seguintes características:

- cotações em tempo real;
- gráficos *intraday*;
- gráficos com histórico;
- indicadores de análise técnica;
- detecção de padrões de preços;
- notícias financeiras;
- dados de mercado de nível II;
- gestão de contas de investimento;
- investimento integrado.

A sua arquitectura permite-lhe ainda, através do uso de *plugins*[3], estender as suas funcionalidades para incluir por exemplo indicadores proprietários, acesso a fontes de dados sujeitas a subscrição paga, envio de ordens, etc...

Tal como se pode constatar nas Figuras 2.2 e 2.3 que se seguem, outros pontos fortes desta aplicação são o seu *interface*, bastante amigável e repleto de informações, e também a qualidade dos gráficos que gera e a quantidade de indicadores que nos são fornecidos.

[2] http://eclipsetrader.sourceforge.net/

[3] *softwares* externos, que se acoplam a um outro principal, estendendo as funcionalidades deste

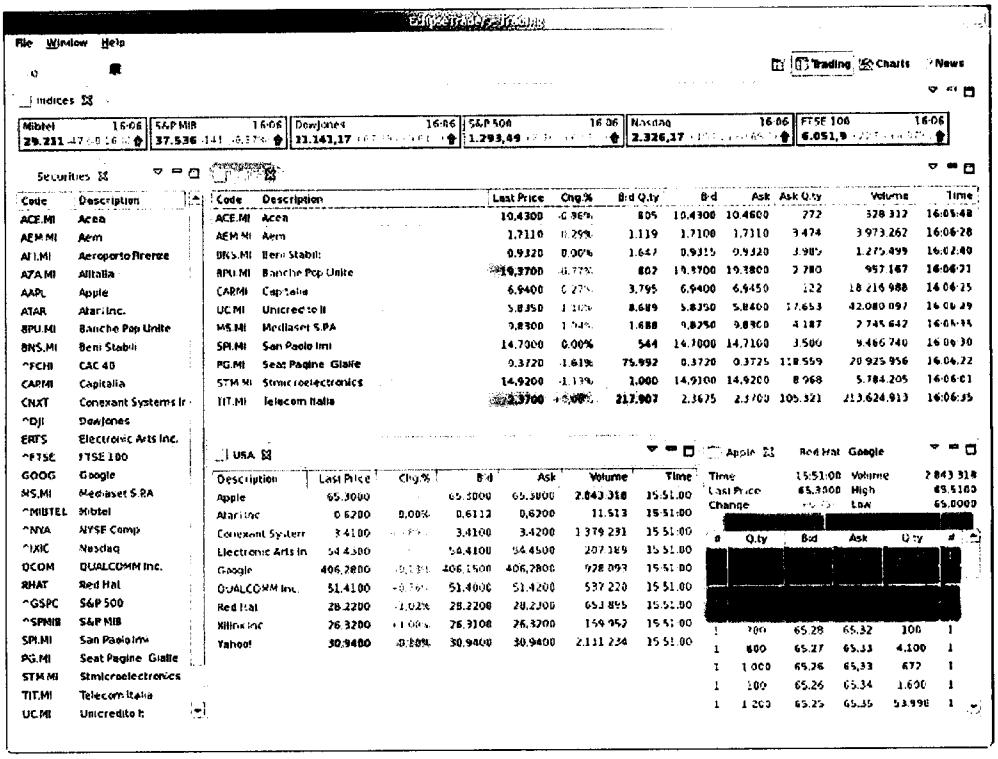


Figura 2.2: Ecrã de acompanhamento de mercados do *Eclipse Trader*

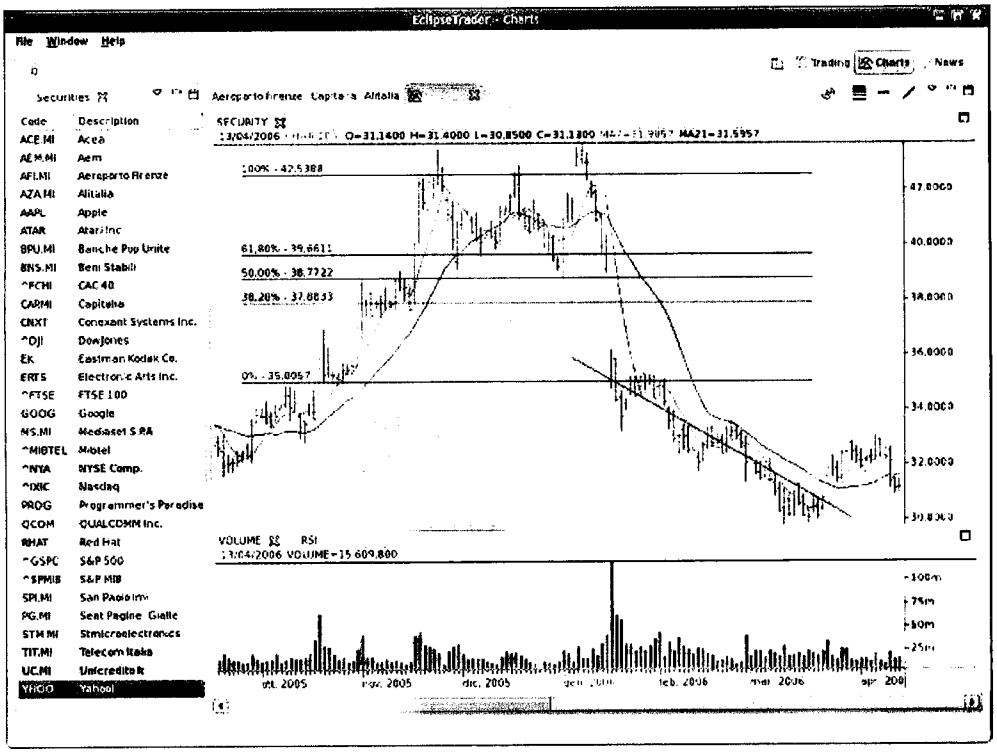


Figura 2.3: Ecrã de análise de gráficos do *Eclipse Trader*

2.4 *Merchant of Venice*

O *Merchant of Venice*[4] é um programa de investimento nos mercados das bolsas de valores, que suporta, por exemplo:

- gestão de *portfolios*;
- geração de gráficos;
- análise técnica;
- investimento automático em títulos;
- métodos experimentais, tais como programação genética.

O seu *interface* também está bem conseguido (ver Figuras 2.4, 2.5, 2.6 e 2.7), se bem que não chegando ao nível dos exemplos anteriores; a informação disponibilizada parece ser suficiente, embora os indicadores existentes não sejam numerosos. Seguem-se então alguns exemplos de ecrãs da aplicação.

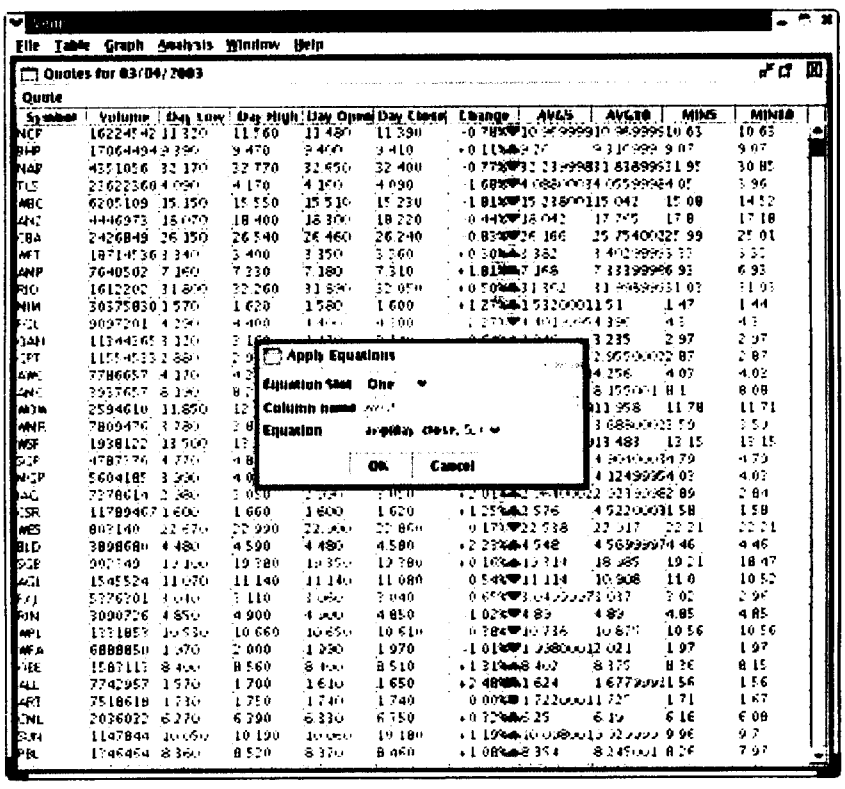


Figura 2.4: Ecrã de histórico de cotações do *Merchant of Venice*

[4]http://mov.sourceforge.net/

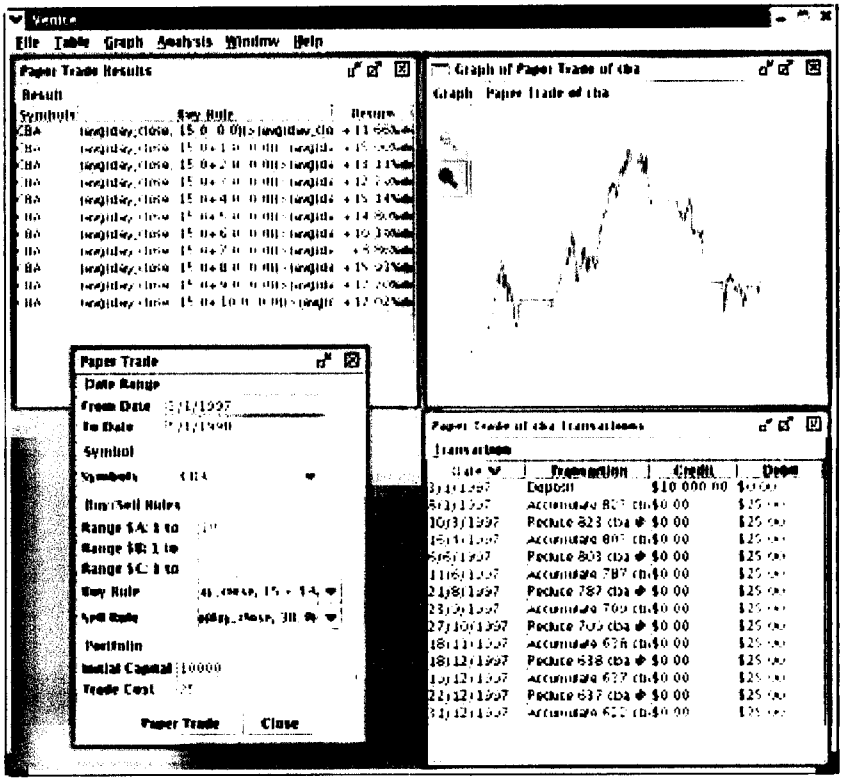


Figura 2.5: Ecrã de investimento em títulos do *Merchant of Venice*

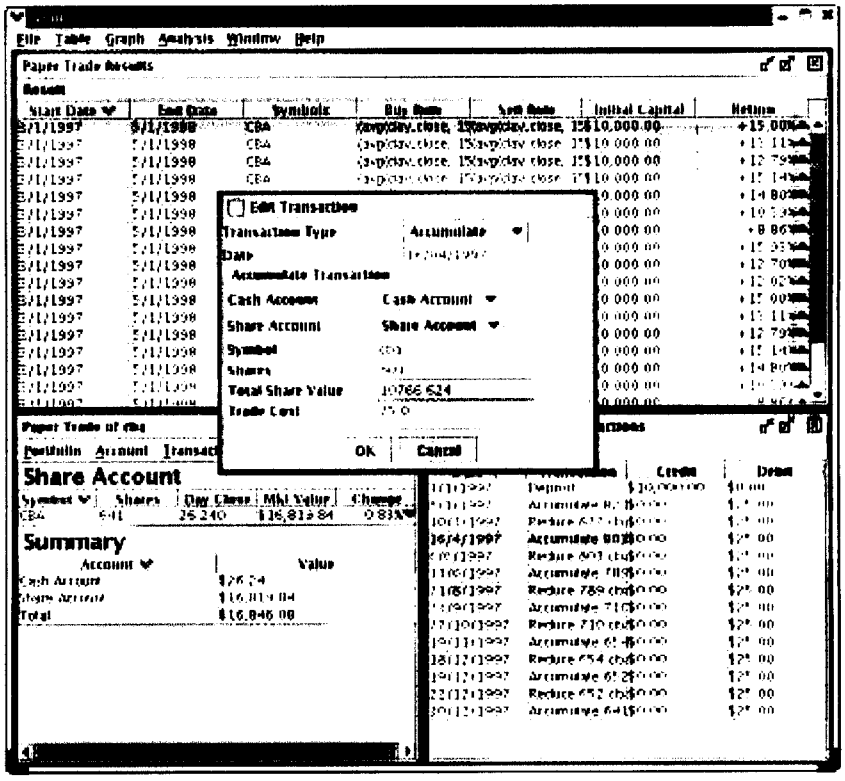


Figura 2.6: Ecrã de gestão de *portfolios* do *Merchant of Venice*

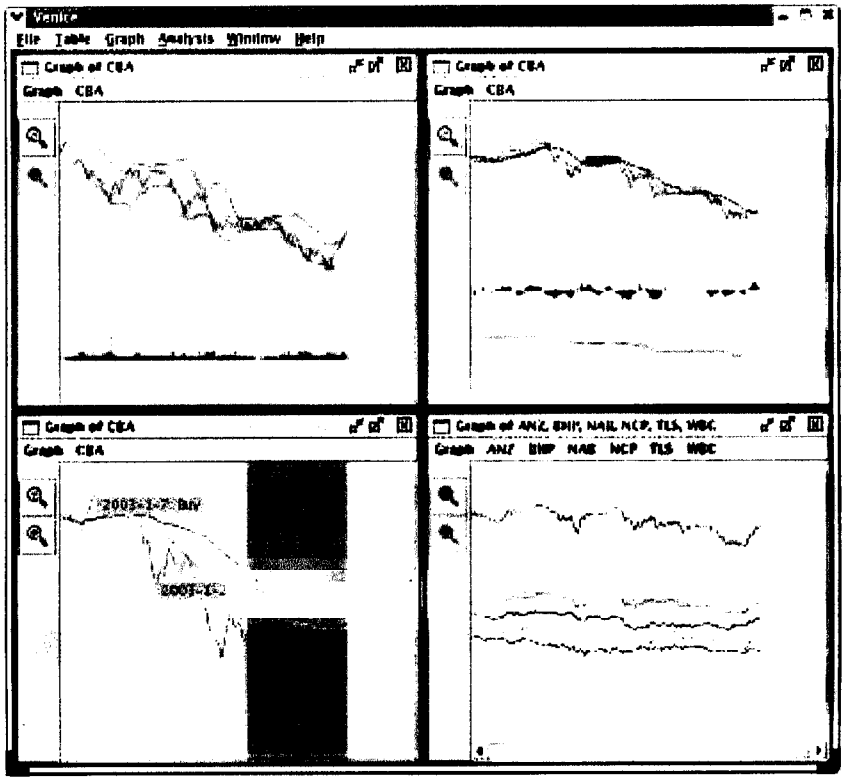


Figura 2.7: Ecrã de análise de gráficos do *Merchant of Venice*

2.5 Conclusão

Com este capítulo pretendeu-se dar a conhecer algumas das aplicações existentes e que tentam dar respostas à problemática levantada por esta dissertação. Refira-se no entanto, que apesar de a ideia da criação de um tal sistema não ser claramente inovadora, os exemplos dignos de referência não eram muitos e na minha opinião resumiam-se aos que foram apresentados.

Não foi possível também avaliar devidamente a *performance* de cada uma das soluções apresentadas, nalguns casos porque as mesmas não estavam facilmente acessíveis ou porque o seu custo era proibitivo, mas mesmo assim justifica-se plenamente pelo menos a sua referência, mesmo que depois não seja possível comparar resultados, com os obtidos na fase de experimentação do sistema desenvolvido.

Assim sendo, o estudo realizado sobre essas ferramentas e sobre as técnicas usadas por cada uma delas, foi sempre feito com base em comentários de outros utilizadores, ou mesmo dos próprios criadores, principalmente no caso do *Trendax*, sendo que por isso, qualquer análise realizada nunca poderia ser considerada conclusiva.

Capítulo 3

Mercado de títulos e métodos de avaliação

3.1 Introdução

O modo de funcionamento e a organização deste mercado, serão abordados, pelo menos de uma forma introdutória, neste capítulo. Podemos começar por definir o mercado de títulos como um componente do sistema financeiro que trabalha, sobretudo, com instrumentos de médio e longo prazo, envolvendo portanto, as acções, obrigações, os títulos de participação, os títulos do tesouro, etc. Por isso, costuma ser identificado como um mercado de valores mobiliários.

O mercado de títulos tem como objectivos principais, facilitar a transferência de recursos e a conversão de activos líquidos em investimento, ou seja, a canalização das poupanças para o sector produtivo da economia, com a vantagem de não gerar massa monetária através de qualquer efeito multiplicador, evitando desta forma tensões inflacionistas. O mercado de títulos, também designado de capitais, deve ser subdividido, por sua vez, em dois mercados ou componentes:

O mercado primário

Compreende os activos financeiros que iniciam a sua circulação no mercado através de emissões (públicas e privadas). É importante não só para permitir a abertura do capital das empresas ao grande público, como também para fornecer ao mercado novos títulos, fazendo, deste modo, a absorção dos capitais gerados na componente secundária e o equilíbrio entre a oferta e a procura.

O mercado secundário

Compreende os activos financeiros que já estão em circulação no mercado e que são objecto de transacções nas Bolsas de Valores ou fora delas (caso dos balcões bancários e dos corretores), procurando assim, assegurar a liquidez dos valores mobiliários. Será

este o mercado alvo de estudo no âmbito deste trabalho.

3.2 Mercado secundário

3.2.1 As bolsas de valores

O mercado secundário está sobretudo organizado em Bolsas de Valores e pelo chamado mercado de balcão.

As Bolsas de Valores são mercados estruturados e dotados de instrumentos jurídicos adequados que permitem que se efectuem transacções ou operações de compra e venda de títulos, com transparência e segurança.

As Bolsas de Valores têm fundamentalmente os seguintes objectivos ou funções:

- a formação dos preços dos valores mobiliários no mercado;
- a canalização da poupança para o mercado primário;
- a avaliação dos valores mobiliários;
- assegurar e proporcionar liquidez aos títulos.

Os títulos admissíveis à negociação nas Bolsas de Valores são:

- as acções e obrigações, incluindo obrigações de caixa, emitidas por sociedades e outras entidades nacionais ou estrangeiras;
- os fundos nacionais e estrangeiros e os valores mobiliários a eles equiparados;
- os títulos de participação;
- as unidades de participação de fundos de investimentos fechados.

De acordo com a legislação em vigor, em cada Bolsa de Valores existirá um mercado de cotações oficiais e um segundo mercado, podendo ainda haver um mercado sem cotações. Os títulos cotados no mercado oficial de uma das Bolsas nacionais podem ser transaccionados em todas as Bolsas existentes no mercado nacional, enquanto que os títulos admitidos no segundo mercado ou no mercado sem cotações só podem ser negociados na Bolsa de Valores em que forem admitidos.

3.2.2 Operações de bolsa

As operações de Bolsa podem ser efectuadas no mercado à vista ou a contado e/ou no mercado a prazo, para além de haver as operações sobre opções, as de futuros financeiros e as operações em conta margem e de contrapartida.

3.2.2.1 Operações no mercado à vista

As operações à vista são aquelas em que o vendedor e o comprador, se obrigam, respectivamente, a entregar os títulos e a pagar o preço da transacção no próprio dia da operação ou num prazo mínimo após a sua realização. Estas operações fazem-se sempre de acordo com as normas estabelecidas pela CMVM e tendo em atenção os sistemas de negociação e liquidação vigentes (por exemplo, a liquidação financeira da operação pode ser três ou cinco dias úteis após a data da efectivação da operação).

Ainda a este propósito, e como é fácil depreender, o corretor não pode dar execução a qualquer ordem de venda à vista sem que o ordenador faça a entrega dos valores a vender, no caso de se tratarem de valores mobiliários titulados e não se encontrarem depositados em qualquer intermediário financeiro autorizado. Por outro lado, o corretor também não é obrigado a dar execução a qualquer ordem de compra sem que o ordenador lhe entregue o montante financeiro necessário ao pagamento da transacção.

As transacções no mercado à vista podem ser sobre:

- os valores mobiliários, onde se incluem as acções;
- os direitos associados aos valores mobiliários, susceptíveis de negociação autónoma, tais como os direitos de acções resultantes das incorporações em reservas, os direitos de acção das reservas de preferência e os direitos de subscrever acções associadas às obrigações com *warrants*;
- as operações sobre opções;
- as operações em conta margem;
- as operações de contrapartida no âmbito dos contratos de liquidez.

Também se pode adiantar que as transacções sobre direitos, as operações em conta margem e as operações de contrapartida só podem ser realizadas no mercado à vista.

3.2.2.2 As ordens de bolsa

Os investidores que pretendem comprar títulos nas Bolsas de Valores, ou os detentores de títulos que os querem vender, têm de realizar uma operação de Bolsa e para isso têm

de se dirigir a uma instituição financeira ou aos corretores para darem uma **ordem de Bolsa**.

Os corretores ou as sociedades financeiras de corretagem são entidades legalmente autorizadas a efectuar a venda e a compra dos títulos, satisfazendo, assim, as diferentes ordens de Bolsa que recebem directamente das pessoas ou através das instituições financeiras.

As ordens de Bolsa contêm os seguintes elementos:

- a identificação do ordenador;
- a identificação dos títulos a transaccionar indicando a espécie, quantidade e as condições de preço;
- a data e o prazo de validade da ordem de Bolsa;
- o tipo de operação: a contrato ou a prazo;
- a Bolsa de Valores em que deva ser executada, quando os valores não tenham de ser transaccionados no sistema de negociação de âmbito nacional e estiverem cotados em mais de uma Bolsa.

Uma ordem de Bolsa pode ter as seguintes condições de preço:

- ao melhor preço;
- com limite de preço;
- com limite de preço e a menção stop;
- casadas.

As **ordens ao melhor preço** caracterizam-se por não terem limites de preço. Neste caso, o corretor, quando recebe uma ordem de compra ao melhor preço, procura satisfazê-la ao preço mais baixo possível e, no caso de se tratar de uma ordem de venda ao melhor preço, o corretor tentará sempre vender ao preço mais alto.

As ordens ao melhor preço, como veremos de seguida, têm mais possibilidades de se efectuarem do que as com limite de preço.

Com **limite de preço** verifica-se quando o vendedor fixa o *preço mínimo* a que está disposto a vender, ou o comprador determina o *preço máximo* a que pretende comprar.

As ordens com limite de preço com menção stop permitem aos emitentes a transformação das ordens inicialmente dadas nesta modalidade em ordens com limite de preço, no caso de terem indicado um segundo limite de preço, ou em ordens ao melhor. Esta alteração verificar-se-á, no caso do ordenador/comprador, quando a cotação na

Bolsa atingir ou ultrapassar o limite de preço máximo antes fixado, ou, no caso do emitente/vendedor, quando a cotação em Bolsa igualar ou descer abaixo do limite mínimo fixado inicialmente.

As ordens casadas são também uma inovação e têm lugar quando o emitente dá simultaneamente uma ordem de compra e uma ordem de venda relativas a títulos diferentes, condicionando a execução de uma à execução da outra.

Ainda em matéria de tipos de ordens, a lei também determina que as ordens relativas a uma quantidade de valores mobiliários inferior ao lote mínimo, têm de ser ordens ao melhor.

Relativamente à duração ou prazo, as ordens podem ser dadas com as seguintes indicações:

- por prazo determinado, e, neste caso, o emitente pode limitar a ordem à sessão da Bolsa do dia em que a ordem é transmitida ou fixar um prazo posterior. No entanto, de acordo com a lei, as ordens não podem exceder um período de 45 dias;

- sem limite de data, e, neste caso, a ordem é válida até à última sessão do mês em que tenha sido recebida.

As ordens podem, no entanto, ser anuladas, suspensas ou modificadas sempre que os emitentes assim o entenderem.

3.2.2.3 A efectivação das cotações

Segundo o diploma e o regulamento que legisla a organização e funcionamento das Bolsas de Valores, a cotação é o preço por que os valores são transaccionados. É estabelecida em sistema de mercado e deverá fixar-se de modo a que seja satisfeita a maior quantidade de títulos.

Para se efectivar a cotação de um dado tipo de valores, devem observar-se as seguintes regras:

1. a cotação deve ser obtida a partir da última efectuada;

2. verificar se na base dessa cotação é satisfeito o maior volume de compras ou de vendas, considerando a totalidade das ordens ao melhor e das ordens com limite de preço;

3. encontrando-se o equilíbrio entre os volumes das compras e das vendas, a cotação considera-se determinada;

4. não existindo esse equilíbrio, e conforme se verifique maior volume de ordens de compra e venda, ensaia-se a cotação com base no escalão imediatamente superior ou inferior, e assim sucessivamente, até se encontrar a cotação ou o preço que satisfaça a maior quantidade de valores possível;

5. se a formação do preço, em função das regras anteriores, conduzir à variação máxima admitida, a cotação só se fixa se as ordens assim efectuadas representarem uma determinada percentagem;

6. quando, por força das regras precedentes, não se puder satisfazer todas as ordens existentes para um determinado preço, devem observar-se as seguintes regras:

   - a execução das ordens ao melhor têm prioridade sobre a das com preço;
   - a execução das ordens de compra superiores à cotação logo a seguir, bem como as ordens de venda a valores inferiores;
   - as ordens de compra e venda à cotação fixada são as últimas a serem satisfeitas;
   - o rateio efectua-se considerando apenas os lotes mínimos e os seus múltiplos, na proporção das ordens existentes;

7. não podem fixar-se cotações com base na negociação de lotes inferiores aos mínimos que se encontram estabelecidos. Quando numa sessão de Bolsa se efectuem apenas, em relação a uma determinada espécie, operações com lotes inferiores aos mínimos, indicar-se-á no boletim das cotações a quantidade transaccionada e o preço efectuado, para que haja uma clara distinção relativamente aos restantes preços realizados.

3.3 As acções

3.3.1 Definição

As acções são títulos representativos do capital social das sociedades anónimas e das sociedades em comandita por acções. Cada acção confere então ao seu titular a qualidade de accionista, proporcionando-lhe um conjunto de direitos que se podem classificar da seguinte forma: direitos estatutários e direitos económicos.

3.3.1.1 Direitos estatutários

O direito de voto - os estatutos da sociedade ditam o número de acções que o accionista deve possuir para poder assistir à Assembleia Geral e exercer o seu voto. Por vezes, basta possuir uma acção, mas, podem também, os estatutos exigirem dez, vinte e mais acções. Se, no entanto, não possuir o número mínimo exigível de acções pode reunir-se com outros pequenos accionistas.

O direito de ser informado - por escrito, antes da Assembleia Geral, ou verbalmente, durante a reunião, pode igualmente pedir informações sobre os assuntos que estejam na ordem do dia.

O Conselho de Administração, por seu lado, está obrigado a pôr à disposição dos accionistas o relatório, balanço e contas de cada exercício.

3.3.1.2 Direitos económicos

Dentro dos direitos económicos, podemos considerar os seguintes:

- **O direito de receber dividendos,** no caso de a sociedade obter lucros e de os distribuir. Como a remuneração proporcionada pelas acções está dependente do apuramento dos resultados líquidos da sociedade, bem como da proposta de aplicação desses lucros por parte do Conselho de Administração e da deliberação da Assembleia Geral (accionistas), as acções também são designadas por títulos de rendimento variável. Os dividendos serão pagos contra a entrega de um cupão (anexo ao título) ou por carimbagem das acções. Deverá tomar-se atenção ao anúncio do pagamento do dividendo, porque nele, por vezes, se indica o número do cupão que tem de se apresentar para receber a parte dos lucros obtidos pela sociedade, que serão distribuídos pelos accionistas;

- **O direito de exercer a partilha do fundo social,** no caso de incorporação de reservas no capital social. O direito de atribuição pode não ser consubstanciado através de entrega de novas acções, mas sim por alteração do valor nominal das antigas, mediante o processo de carimbagem dos títulos;

- Os accionistas têm também **o direito de preferência** na subscrição de novas acções, salvo disposição contrária nos estatutos. Quando a sociedade aumenta o seu capital em numerário, através da emissão de novas acções, os accionistas têm direito a subscrever novas acções na proporção que lhes corresponder. Se, no entanto, não lhes interessa participar no aumento de capital social, podem vender o seu direito de subscrição das novas acções. E tratando-se de acções cotadas nas Bolsas de Valores, pode vender os direitos de subscrição nas Bolsas.

3.3.2 Tipos de acções quanto à sua forma de transmissão

As acções nominativas

São aquelas em que a titularidade dos seus possuidores resulta da inscrição do nome do titular no livro de registos da sociedade. Toda a mudança de titularidade exige certas formalidades que se encontram estabelecidas legalmente. A propriedade e a transmissão das acções nominativas apenas produzirá efeitos para com a sociedade e

para com terceiros a partir da data do respectivo averbamento no livro de registos de acções. Em contrapartida aos inconvenientes de tal formalismo, os títulos nominativos podem oferecer algumas vantagens para as sociedades, uma vez que conhecem exactamente os seus accionistas.

As acções ao portador

São aquelas cuja titularidade resulta da posse material do próprio título. As acções ao portador devem obedecer a normas no que diz respeito às técnicas de impressão, cores, papel e numeração, a fim de evitar a sua falsificação.

3.3.3 Formato das acções

As acções apresentam-se, por enquanto, materialmente sob a forma de um documento com determinado formato que comporta duas zonas: de um lado o *corpo do título*, que contém as indicações essenciais sobre a sociedade (denominação, sede social, montante de capital, quantidade das acções que representa, etc.); do outro lado uma *folha de cupões* destinados a serem destacados na altura do pagamento do dividendo e utilizados no exercício dos direitos de aumento de capital.

Os possuidores de acções têm sempre a faculdade de as terem à sua guarda. Nesta hipótese, terão que destacar os cupões do dividendo e apresentá-los junto da sociedade ou nas instituições de crédito pagadoras, aquando da distribuição de dividendos.

3.4 Avaliação das acções

3.4.1 Introdução

A avaliação de acções é substancialmente mais complexa que a de títulos de rendimento fixo, particularmente pelas seguintes razões:

a) em primeiro lugar, porque as suas principais fontes de rendimento (dividendos e mais-valias de capital), são de muito mais difícil antecipação do que os juros e amortizações de capital inerentes às obrigações. Deste facto resulta uma muito maior incerteza em termos de rentabilidade do investimento;

b) em segundo lugar, porque a diferenciação em termos desse risco de rendimento, de acção para acção, envolve considerações de maior complexidade em oposição à matriz comum de que dispõem as obrigações – o *rating* – à facilidade do seu reflexo no rendimento das mesmas (via taxa de juro).

Neste secção procurar-se-á encontrar respostas para estes dois problemas. Num primeiro ponto analisar-se-á a relação entre risco e rentabilidade, aspecto crucial na

avaliação de títulos de rendimento variável, tendo em vista a definição de um modelo operacional de quantificação de tal relação. Serão também sublinhadas algumas das ideias essenciais da teoria financeira moderna a que subjazem aqueles modelos (a diversificação como método de diminuição do risco, etc.).

Um segundo ponto dirá respeito, uma vez encontrada uma forma satisfatória de análise risco/rentabilidade, à apresentação de diferentes métodos de determinação do preço de uma acção (óptica contabilística, de dividendos, PER, PCF e EPS, entre outras).

3.4.2 Risco e rentabilidade

3.4.2.1 O princípio e os limites da diversificação

A quantificação entre risco e rentabilidade deu os seus primeiros passos quando em 1952 Harry Markowitz apresentou uma teoria original [Mar52] que nos seus subsequentes desenvolvimentos veio a ser conhecida pela teoria do *portfolio* e que constitui um dos pilares essenciais da gestão financeira moderna. A ideia fundamental da teoria do *portfolio* é a de que os investidores não têm sempre como único objectivo a maximização do seu rendimento. A maioria dos investidores dispersa o seu capital por diferentes acções, na convicção de que diversificando consegue reduzir o risco do seu investimento; a optimização do rendimento (em termos absolutos) não é a sua única preocupação. Se assim não fosse, todos os investidores tenderiam a colocar todo o seu capital disponível na acção que lhes parecesse oferecer a melhor perspectiva de rendimento. Na realidade são poucos os que actuam deste modo. A forma como a diversificação actua sobre o risco (v.s. efeito sobre rendimento) representa o primeiro conceito essencial da teoria do *portfolio*, valendo a pena explorar melhor o seu significado e alcance.

3.4.2.2 As componentes do risco de uma acção

O risco pode ser dividido em duas componentes: uma, possível de progressiva eliminação através da diversificação da carteira; outra, imutável e completamente resistente a qualquer composição do investimento.

Por exemplo, para o primeiro caso, uma situação na qual uma determinada empresa, cotada em Bolsa, é obrigada a paralisar por tempo indeterminado a sua produção e a realizar vultuosos investimentos por incumprimento de disposições legais antipoluição. Nesta situação, não surpreenderá que a cotação das suas acções sofra uma imediata descida, em consequência daquele novo elemento que certamente conduzirá os investidores a uma reavaliação das perspectivas da empresa. Aquele acontecimento, no entanto, apenas afectou a rentabilidade dos títulos de uma empresa, não tendo qualquer influência na das restantes. Trata-se, pois, de um factor de risco específico. Imagine-se agora uma outra situação, na qual se estava perante e eminência de um conflito internacional com nefastas consequências na economia mundial. Neste quadro,

igualmente não surpreenderá que se assista a uma baixa generalizada das cotações, afectando a rentabilidade de todas as acções (ainda que, eventualmente, com intensidades diferenciadas). Neste caso, qualquer que seja(m) o(s) título(s) detido(s), o rendimento dos investidores virá diminuído. Trata-se de uma situação em que o fenómeno da diversificação nada pode auxiliar, estando-se assim perante um factor de risco não eliminável.

Os exemplos anteriores ilustraram dois grandes grupos de factores que afectam a rentabilidade das acções e que importa agora precisar um pouco mais.

Risco específico

Genericamente poderá ser associado a todos os factores que não afectam o rendimento da totalidade das acções mas apenas o de uma ou de um conjunto identificável de acções. Este nível de risco pode ser sub-categorizado em três classes:

Risco da empresa - Engloba todos os factores que somente afectam positiva ou negativamente a rentabilidade de uma empresa, como sejam, entre muitos outros exemplos, a alteração nos órgãos de gestão, o lançamento mal sucedido de um novo produto, uma política de financiamento excessivamente endividada e/ou onerosa, uma campanha de promoção muito bem concebida e uma inovação no processo de fabrico.

Risco do sector - Inclui todos os factores susceptíveis de modificar a rentabilidade das empresas de um determinado sector, como por exemplo a eliminação de restrições à importação (nesse sector), a criação de um sistema de incentivos financeiros e/ou fiscais de índole sectorial ou uma inovação tecnológica que cria um sucedâneo mais competitivo.

Risco comum não sectorial - Compreende o conjunto de factores que afectam a rentabilidade de mais de uma empresa (mas não a totalidade do mercado) e que têm como matriz comum uma base não sectorial. Como exemplos citem-se a introdução de severas disposições anti-poluição em determinada zona ou o encerramento de um aeroporto ou terminal ferroviário/marítimo.

Qualquer uma destas três classes que compõem o risco específico é susceptível de eliminação através de uma adequada política de diversificação, exactamente porque em nenhum caso afectam a totalidade das acções.

Risco de mercado (ou sistemático)

Integra os factores que de forma geral têm capacidade de alterar (ainda que com maior ou menor intensidade) a rentabilidade de todas as acções do mercado, como sejam, entre outros exemplos, a inflação, os níveis das taxas de juro, a política económica, financeira ou fiscal do Governo e a conjuntura política nacional ou internacional.

3.4.3 Métodos de avaliação de acções

3.4.3.1 O valor contabilístico

A identificação do valor das acções com a situação líquida constante do balanço da empresa, constitui a forma mais tradicional de valorização destes títulos. O método consiste simplesmente em dividir o valor da situação líquida pelo número de acções existentes.

Este método, cujo fundamento assenta no facto de a situação líquida, ao representar a porção do activo que excede as dívidas da empresa a terceiros (passivo), constituir o valor que os seus proprietários efectivamente detêm, apresenta vários e inultrapassáveis inconvenientes. Na verdade, a situação líquida representa principalmente:

a) os fundos que os accionistas investiram directamente na empresa, ou seja, o capital social líquido de incorporação de reservas mais eventuais prémios de emissão;

b) os resultados anuais gerados (e não distribuídos) ao longo da vida da empresa e que se encontram disseminados pelas várias rubricas de reservas ou incorporados no capital social;

c) a reavaliação dos activos imobilizados (na verdade, trata-se de uma uma mera revalorização, tentando repor a desvalorização monetária);

d) os resultados líquidos do ano.

Assim, a situação líquida traduz, no máximo, o investimento dos accionistas na empresa, mais a rentabilidade não distribuída e por ele gerada no passado. Contudo, o valor actual de um título ou de qualquer outro activo não depende, como é compreensível, nem do rendimento que gerou no passado nem do investimento que o mesmo representou, mas exclusivamente do nível de remuneração que irá proporcionar no futuro. Assim, o conceito de situação líquida numa óptica contabilística, ao resultar somente da actividade passada da empresa, não pode expressar de forma adequada o valor das suas acções.

3.4.3.2 Óptica de dividendos

Referiu-se no ponto anterior que o valor de um activo, para qualquer investidor racional, depende exclusivamente do rendimento futuro que o mesmo irá proporcionar. Nesta perspectiva, a avaliação de uma acção deverá, então, centrar-se somente em dois aspectos: nos dividendos a distribuir anualmente por acção e nas mais ou menos-valias (variação de cotação ao longo do tempo) que o título venha a registar.

Assim, num determinado ano, a taxa de rentabilidade de uma acção será igual a:

$$r = \frac{d_1 + (c_1 - c_0)}{c_0}$$

Onde:
d_1 – dividendo a distribuir no ano;
c_0 – cotação do título no início do ano;
c_1 – cotação no final do ano.

Esta fórmula traduz o que se afirmou: a taxa de rentabilidade anual da acção decorre da comparação entre os rendimentos a obter, ou seja, do dividendo que eventualmente se receba(d_1), das mais ou menos-valias que se obtenham no capital durante o ano ($c_1 - c_0$) e do investimento realizado (c_0).

3.4.3.3 Rácios da análise fundamental PER, PCF e EPS

O PER (*Price Earning Ratio*) - é o resultado do quociente entre a cotação da acção e o resultado líquido unitário da mesma, tal como se apresenta na equação seguinte:

$$PER = \frac{Ca}{Rlu}$$

Onde:
Ca – Cotação da acção;
Rlu – Resultado líquido unitário.

Trata-se de um indicador de mercado tradicionalmente bastante utilizado nas análises bolsistas enquanto critério de compra ou venda de acções, pelo que plenamente se justifica alguma reflexão sobre a sua validade e limitações.

Importa referir que o PER é tendencialmente utilizado de uma forma marcadamente qualitativa, servindo para análises comparativas entre o PER de uma empresa e o PER médio do sector ou da Bolsa. Destas análises resulta a detecção de acções sub ou sobre-avaliadas e, consequentemente, a tomada de determinadas decisões de investimento. Convirá a este propósito sublinhar alguns aspectos que podem auxiliar a uma melhor avaliação deste tipo de comparações.

A detecção de uma empresa cujo PER é inferior ao do sector ou ao da Bolsa (o primeiro é preferível, por se tratar de uma análise mais homogénea), não significa, necessariamente, que se tenha identificado uma acção subavaliada. Vejamos alguns factores que racionalmente podem explicar essa situação:

a) em primeiro lugar, a empresa em causa, pelas mais diversas razões, pode perspectivar um futuro nível de crescimento dos seus resultados, inferior ao dos seus concorrentes no sector. Neste caso, faz sentido que o seu PER seja menor;

b) em segundo lugar, e mesmo admitindo que o enunciado no ponto anterior não se verifica, a empresa em análise pode apresentar um risco mais elevado que o dos seus concorrentes (seja por uma política de endividamento mais agressiva, por um grau de obsolescência dos equipamentos mais elevado, a exigir vultuosos investimentos futuros, seja pelo seu posicionamento em termos de segmentos de mercado ou ainda por qualquer outra razão). Nesta perspectiva, o seu PER deverá ser inferior ao do sector;

c) em terceiro lugar, é importante sublinhar que o PER apenas utiliza os resultados líquidos de um ano (geralmente os do ano em curso ou os do próximo ano, quando se aproxima o final do exercício). Caso se esteja na presença de um mercado com um mínimo de eficiência é natural que o PER reflicta uma análise temporal superior a um ano. Assim, e supondo que a empresa em causa irá apresentar um resultado que não manterá (por qualquer razão) nos anos seguintes, é de esperar que o seu PER seja menor que o do sector (assumindo que tal tendência não se verificará no mesmo), na medida em que já incorporará a expectativa da evolução da empresa referida. Situação inversa será o caso de uma empresa aparentemente com um PER demasiadamente elevado mas que, afinal, não reflectirá mais do que uma expectativa de grande crescimento de resultados no(s) ano(s) seguinte(s) ao em análise.

Em síntese, a utilização qualitativa do PER pressupõe sempre uma avaliação cuidada da evolução futura das empresas, sectores e economia, a enquadrar convenientemente numa efectiva comparação dos PER. Em todo o caso, e esse será sempre o eterno problema de qualquer análise qualitativa, nunca se possuirá um critério de decisão inequívoco.

O PCF (*Price Cash Flow*) - é um indicador de construção semelhante ao PER (relacionando agora a cotação da acção e o *cash flow* do exercício). Tendo em conta o que se referiu na análise do PER, haverá que ter em consideração os seguintes aspectos:

a) o PCF tem sempre um valor inferior ao do PER , na medida em que o cash flow de qualquer empresa é superior aos seus resultados, ao não incluir no seu cálculo as amortizações e provisões líquidas do exercício;

b) muitos analistas defendem uma maior utilização do PCF em detrimento do PER, atendendo à sua diferença de cálculo; na verdade, ao não se incluírem as provisões e amortizações está-se por um lado a eliminar uma das tradicionais áreas de manipulação contabilística (e fiscal) com repercussão nos resultados apurados e, por outro, o conceito de *cash flow* (assim calculado) evidencia de forma mais apropriada o potencial de rentabilidade financeira da empresa;

c) a modelização do PCF é bastante mais complicada que a do PER, na medida em que há que, adicionalmente, estimar valores de provisões e amortizações.

No entanto, os aspectos a cuidar na realização de análises comparativas do PER, atrás mencionadas, mantêm-se igualmente válidos neste caso; a esmagadora maioria dos factores que influenciam a evolução dos resultados líquidos afectam no mesmo sentido e com intensidade similar, o *cash flow.*

O EPS (*Earnings Per Share*) - não é mais que uma variante da rentabilidade do capital próprio em que, no denominador, em vez de aparecer um valor financeiro, aparece o número de acções. É por vezes preferido em relação a outros rácios, pois é um indicador mais directo da valorização de um título, não estando sujeito a critérios contabilísticos de avaliação.

O resultado por acção é um dos rácios mais utilizados pelos analistas financeiros no mercado em conjunção com o PER, é necessário, no entanto, ter em atenção o seu modo de cálculo, utilidade e limitações. Mas não deixa de ser bastante útil quando conjugado com outras informações para avaliar o desempenho económico da gestão e estimar o seu potencial.

O resultado por acção, em termos puramente conceptuais, deve representar o enriquecimento absoluto de cada acção ordinária. Assim, em estruturas de capital simples, em que o capital social é constituído apenas por acções ordinárias, o EPS resulta do seguinte cálculo:

$$EPS = \frac{Rle}{Nma}$$

Onde:
Rle – Resultado líquido do exercício;
Nma – Nº médio ponderado de acções ordinárias.

As acções em circulação são incluídas na média ponderada, desde a data em que adquirem o direito ao dividendo. O factor de ponderação é a quantidade de dias em que as acções específicas estiveram em circulação.

3.4.3.4 Óptica de património e rendimento

Os métodos analisados anteriormente incidem fundamentalmente sobre o rendimento das empresas e/ou investidores (tónica nos resultados, *cash flow* ou dividendos futuros). Ao longo das últimas décadas tem-se desenvolvido em paralelo uma metodologia de avaliação de empresas que combina os conceitos de património e rendimento.

De uma forma geral, tais métodos assentam que o valor de uma empresa (Ve) é o resultado de duas componentes:

$$\text{Ve} = \text{Activo Real Líquido (Arl)} + \text{Goodwill (Gw)}$$

A primeira parcela, o Arl, representa o activo da empresa (valorizado a preços de mercado), deduzido do passivo efectivo da empresa. Constitui, pois, o seu valor patrimonial; aquele que em teoria seria obtido caso se procedesse à sua liquidação, em condições normais de mercado.

A segunda parcela, o Gw, traduz a valorização da própria actividade específica da empresa (imagem, *know-how*, qualidade de gestão, marcas, localização, etc.) que conduz à obtenção de um resultado diferente daquele que seria gerado, caso se aplicasse o valor do seu património num activo financeiro sem risco (obrigações do Estado, por exemplo). É precisamente a diferença entre o resultado desta última aplicação e o que a empresa obtém na sua actividade, que constitui o *goodwill* (ou *badwill*, se negativo), que acaba por se identificar como um lucro extra gerado por todos aqueles factores atrás referidos inerentes à actividade de cada empresa e que não podem ser encontrados num activo financeiro sem risco (é, aliás, a possibilidade da obtenção desse lucro extra que justifica o investimento, naturalmente mais arriscado, no capital de uma empresa).

3.4.3.5 Análise técnica

Os métodos de avaliação anteriormente apresentados têm subjacente uma análise detalhada da empresa (posição de mercado, política de investimentos, rentabilidade, estrutura financeira, características da gestão, etc.) que em conjunto com uma análise da envolvente da mesma (economia em geral e de sector) possibilita a projecção dos valores futuros das variáveis relevantes do(s) modelo(s), sejam os resultados líquidos, *cash flows*, dividendos ou quaisquer outras. Esta forma de avaliação é geralmente designada por **análise fundamental**, porque, como o próprio nome o indica, tem por base o estudo fundamentado de determinada realidade e respectiva evolução futura.

A análise técnica situa-se num campo totalmente oposto. Este método assume que todo o tipo de considerações de natureza política, económica, empresarial, psicológica ou outra, que afectem a perspectiva dos investidores sobre cada acção, se encontram integralmente reflectidas nas cotações para se identificarem as melhores decisões de investimento e desinvestimento.

Deste modo, o fulcro da análise centra-se na evolução das cotações que, no entanto, e de *per si*, pouco auxiliam na tomada de decisões, a menos que as empresas disponham de uma base instrumental que lhes permita antecipar o futuro curso das cotações. A análise técnica, neste âmbito, assenta em quatro ideias-chave:

a) as cotações são determinadas pela procura e oferta no quadro de um mercado concorrencial;

b) a todo o momento, a procura e a oferta reflectem centenas de considerações racionais e irracionais: factos, opiniões, tendências e previsões acerca do futuro;

c) apesar da existência de algumas flutuações menores, as cotações movem-se segundo tendências que persistem durante um considerável período de tempo;

d) alterações na tendência representam uma mudança no peso relativo entre a procura e a oferta. Independentemente das causas de tais mudanças, estas são sempre detectáveis, mais cedo ou mais tarde, na evolução dos preços do mercado.

Em síntese, a análise técnica assenta em grande medida na projecção da evolução futura das cotações através do estudo de uma ou várias séries de cotações passadas. Compreende-se assim facilmente que o essencial dos instrumentos de trabalho tenham por base a utilização de técnicas oriundas da estatística, nomeadamente gráficos, séries, médias (em particular móveis), etc., que, através da sua aplicação à análise das cotações e aos volumes de transacções, possibilitaram a criação de um quase infindável número de métodos e critérios de decisão, tendo em vista a selecção do *timing* ideal para a compra e venda de títulos.

Os pressupostos da análise técnica acerca do funcionamento do mercado não estão empírica e cientificamente demonstrados, no entanto existem duas boas razões para o seu estudo:

a) em primeiro lugar, porque apura a capacidade de tratamento dos dados de mercado e uma maior reflexão sobre o seu funcionamento e insuficiências;

b) em segundo lugar, porque basta que existam operadores no mercado que utilizem tal análise, para que os investidores mais atentos não a menosprezem por uma simples razão: sempre que se verificar a existência de um ponto de compra ou venda (segundo os padrões da análise técnica), pode-se estar certo que alguma faixa do mercado irá actuar e que o conhecimento antecipado desse tipo de reacções pode, naturalmente, revelar-se da maior utilidade na gestão de uma carteira de títulos.

3.4.3.6 Indicadores da análise técnica RSI, SI e MACD

O RSI (*Relative Strength Index*) - é um oscilador que foi criado por Welles Wilder em Junho de 1978. Este oscilador é muito popular entre os analistas técnicos devido aos seus bons resultados. Em termos de cálculo, a fórmula é bastante simples e é esta que se apresenta de seguida:

$$RSI = 100 - \frac{100}{1 + \frac{u}{d}}$$

Onde:
u – média das cotações no período n em que a cotação subiu;
d – média das cotações no período n em que a cotação desceu;

O criador deste indicador recomenda o cálculo de um RSI de 14 dias mas é habitual calcular um RSI de 9 ou 25 dias. O valor do RSI pode variar entre 0 e 100.

Da observação do gráfico RSI, existem basicamente 3 análises que se podem efectuar:

a) uma das interpretações mais simplistas que se podem retirar de um gráfico do RSI é o que concerne à saída de uma zona de *oversold/overbought*. Sempre que o RSI caia abaixo dos 70 pontos depois de ter estado na zona de *overbought*, é gerado um sinal de venda do título. Sempre que o RSI sai de uma zona de *oversold*, isto é, o seu valor passe a estar acima dos 30, é dada uma indicação de compra do título;

b) outra intepretação gráfica que se pode retirar do RSI são as divergências (ver Figura 3.1). É neste ponto que talvez se encontre a maior virtude deste oscilador. Sempre que a cotação atinja novos máximos e o gráfico do RSI esteja a cair, é provável que a cotação do título corrija através da queda. Raciocínio análogo pode ser feito para os mínimos. Sempre que a cotação teste novos mínimos e o gráfico do RSI não acompanhe, é muito provável que a cotação do título suba;

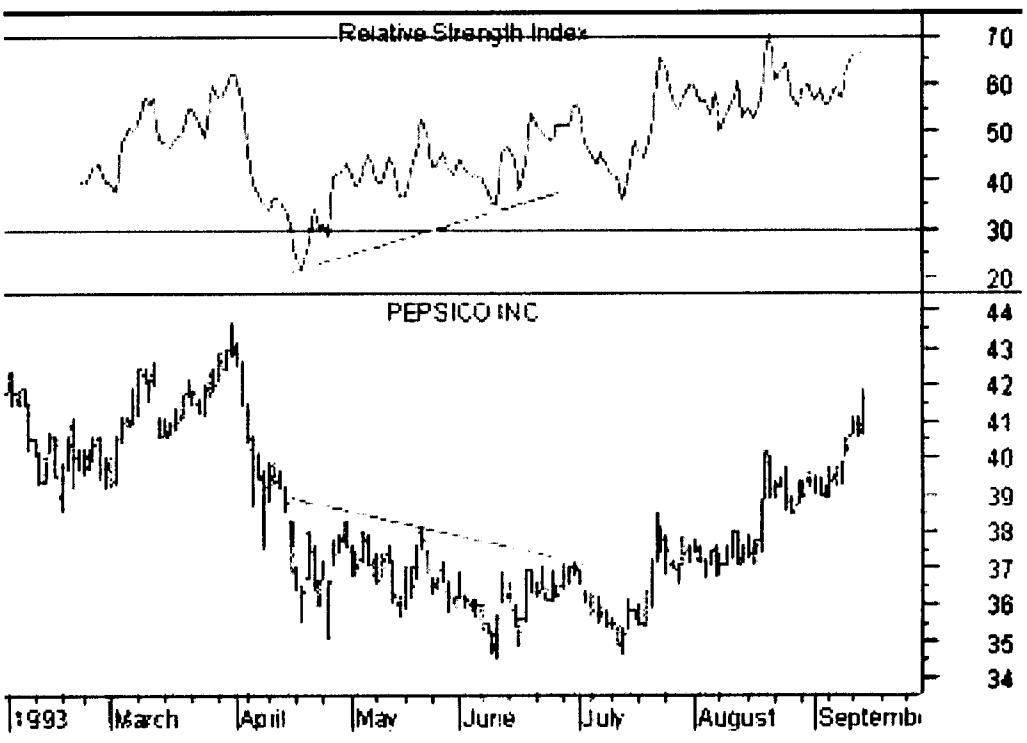


Figura 3.1: Gráfico RSI com divergências explícitas

c) o gráfico do RSI é também excelente para traçar linhas de resistência / suporte / tendência da mesma forma que são traçadas num gráfico de cotações.

O SI (*Stochastic Indicator*) - é importante sob o ponto de vista técnico, pois permite-nos determinar qual a relação entre a cotação de um título e o intervalo de cotações onde esse título de manteve nos últimos N dias. A importância desta análise

reside na tendência que as cotações têm em fechar perto quer do máximo do intervalo quer do valor mínimo, no caso de uma subida de cotação ou descida respectivamente.

No caso de uma tendência de descida de cotação, o preço tende a bater no limite mínimo do intervalo para depois inverter a tendência e começar a subir afastando-se do valor mínimo do intervalo. Raciocínio análogo pode ser feito para uma tendência de subida de cotação.

Em termos gráficos, o indicador estocástico é representado por duas linhas (ver Figura 3.2). A linha principal denomina-se de %K. A segunda linha que é usualmente impressa a tracejado, denomina-se %D e é uma média móvel da linha %K.

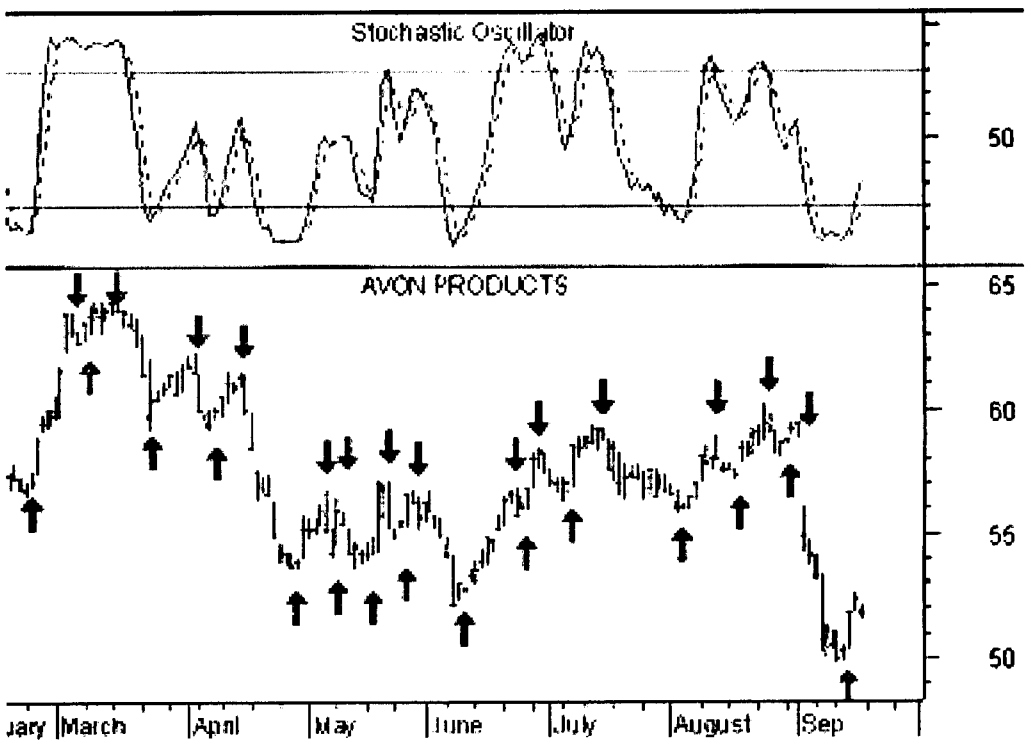


Figura 3.2: Gráfico SI com as linhas K e D

As fórmulas aplicadas para cálculo de ambos os valores é a seguinte:

$$\%K = \frac{Ca - Lo_{(n)}}{Hi_{(n)} - Lo_{(n)}} * 100$$

Onde:
Ca – Cotação da acção;
$Lo_{(n)}$ – Cotação mínima no período n;
$Hi_{(n)}$ – Cotação máxima no período n;

%D = média movel de %K em determinado período, normalmente 3 dias

n = número de períodos usados no cálculo, normalmente 14 dias é o indicado

O valor de %K pode variar entre os 0% e os 100%. Se o valor for 0% quer dizer que a cotação de fecho é igual ao mínimo valor da cotação nos últimos n períodos. Se o valor for 100%, quer dizer que a cotação de fecho bateu no máximo dos últimos n períodos.

Tendo calculado o %K e a correspondente média móvel %D, existem várias interpretações que se podem retirar da comparação entre as linhas %K e %D e que são as seguintes:

a) um sinal de compra é gerado quando a linha %K ou a linha %D desce abaixo dos 20% para depois subir acima dos 20%. Um sinal de venda é gerado quando a linha %K ou a linha %D sobe acima dos 80% para depois descer abaixo desse nível;

b) um sinal de compra é gerado no momento em que a linha %K sobe acima da linha %D. Um sinal de venda é gerado quando a linha %K desce abaixo da linha %D;

c) tal como no indicador RSI, um dos sinais mais fortes dados por este indicador é dado pela divergência que surja entre a cotação de fecho e o indicador (ver Figura 3.3). Assim, se a cotação do título continuar a testar novos máximos e a linha %K tem máximos relativos cada vez menores, é provável que a tendência de subida do título se inverta o que pode sugerir que se venda o título em questão. Raciocínio análogo pode ser aplicado às quedas.

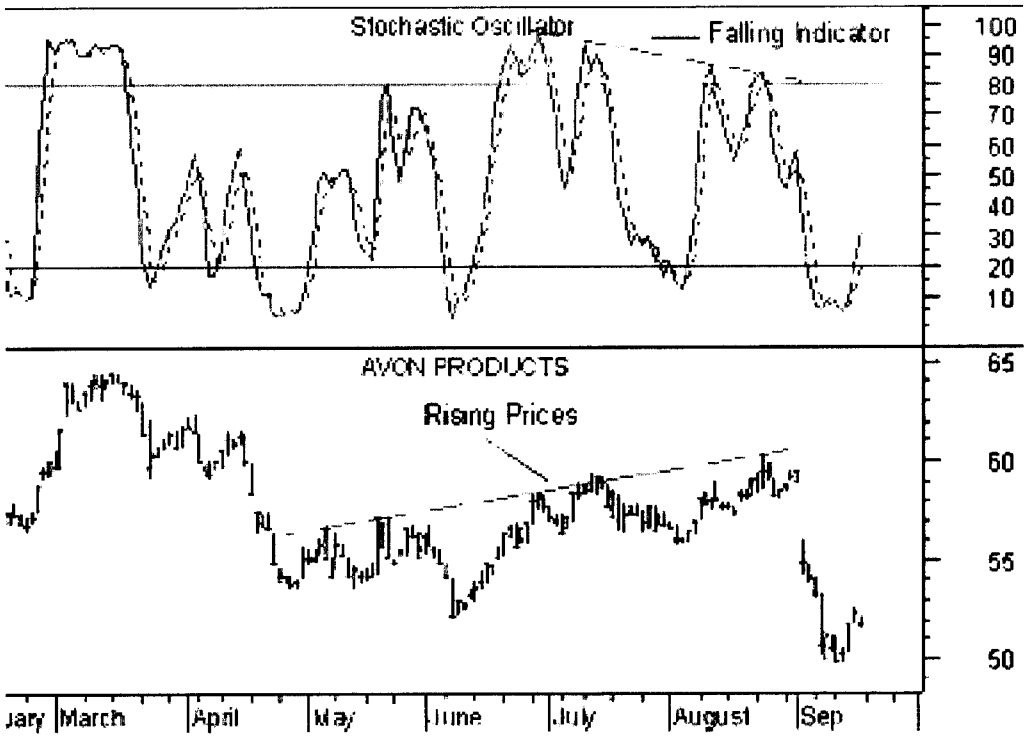


Figura 3.3: Gráfico SI com divergências explícitas

O MACD (*Moving Average Convergence / Divergence*) - é um indicador de tendência que mostra a relação entre duas médias móveis. É calculado subtraindo à média móvel exponencial de 26 dias dos valores de fecho da cotação da acção, a média móvel exponencial de 12 dias dos mesmos valores. O gráfico que daí se obtém é comparado com o gráfico da média móvel exponencial de 9 dias do próprio indicador MACD, denominado de linha de sinal ou *trigger* que geralmente é um gráfico a picotado (ver Figura 3.4).

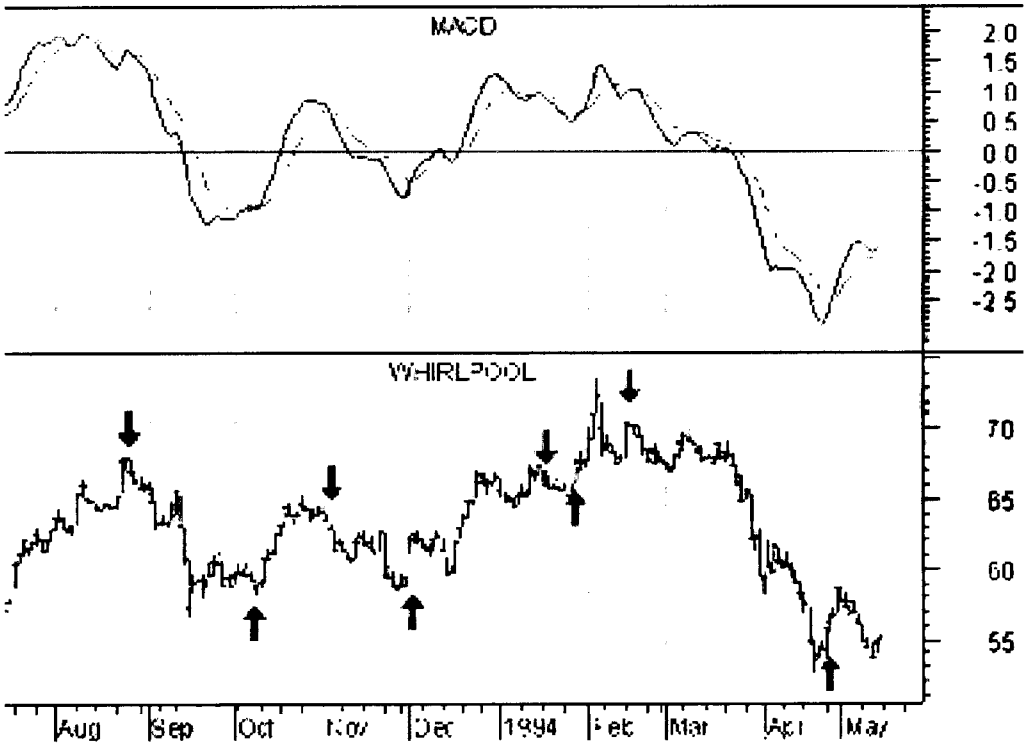


Figura 3.4: Gráfico MACD com o respectivo Signal

Para podermos calcular este indicador primeiro definimos os períodos:

$$n_1 = 26; n_2 = 12; n_3 = 9$$

Calculámos os factores de ponderação:

$$k_1 = \frac{2}{n_1 + 1}; k_2 = \frac{2}{n_2 + 1}; k_3 = \frac{2}{n_3 + 1}$$

De seguida calculámos as médias móveis exponenciais de 26 e 12 dias:

$$mme_i^{(1)} = (ca - mme_{i-1}^{(1)}) * k_1 + mme_{i-1}^{(1)}$$

$$mme_i^{(2)} = (ca - mme_{i-1}^{(2)}) * k_1 + mme_{i-1}^{(2)}$$

Onde:
ca – cotação da acção

Finalmente para calcular o indicador MACD subtraímos à média móvel exponencial de 26 dias ($\text{mme}^{(1)}$) a média móvel exponencial de 12 dias ($\text{mme}^{(2)}$), e ao mesmo tempo calculámos a média móvel exponencial de 9 dias ($\text{mme}^{(3)}$) do MACD:

$$MACD_i = mme_i^{(1)} - mme_i^{(2)}$$

$$mme_i^{(3)} = (MACD_i - MACD_{i-1}) * k_3 + MACD_{i-1}$$

Existem três tipos distintos de interpretação gráfica do MACD:

a) intersecções dos gráficos: uma regra do MACD é vender sempre que o seu gráfico passe para baixo do gráfico da sua linha de *trigger*. Da mesma forma, um sinal de compra é emitido sempre que o seu gráfico passa para cima da sua linha de *triger*;

b) zonas de *overbought/oversold*: quando o valor do MACD aumenta (na prática isto significa que o valor da média móvel exponencial de mais curto prazo diminui face ao valor da média de 26 dias), é provável que a cotação do título esteja *oversold*. Como consequência poderá haver uma queda na sua cotação pelo que é dado um sinal de venda;

c) divergências: outra interpretação valiosa retirada do MACD é a detecção do fim de uma tendência. Sempre que a evolução gráfica do MACD de um título diverge da evolução gráfica das suas cotações, então está detectada uma divergência (ver Figura 3.5). Se o MACD está a atingir mínimos sucessivos e a sua cotação não, estamos perante uma divergência *Bearish*[1], sendo provável que as cotações venham a cair. Se o MACD está a atingir máximos sucessivos enquanto a sua cotação não atinge novos máximos, estamos perante uma divergência *Bullish*[2] sendo provável que a cotação do título venha a subir.

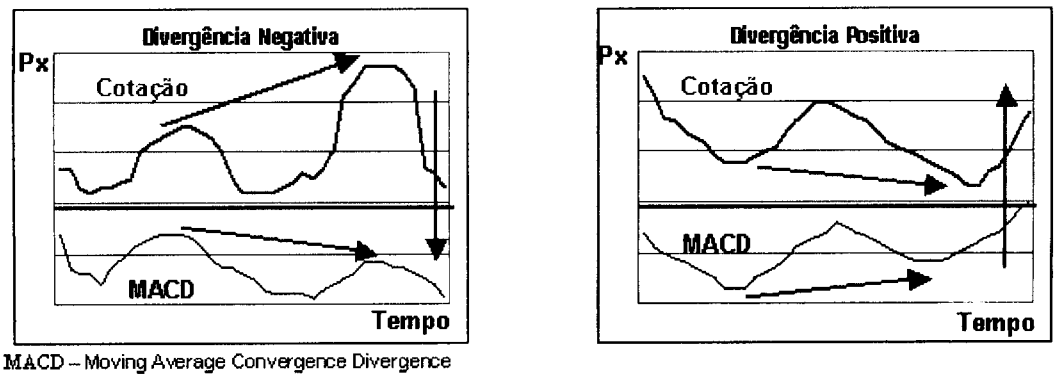


Figura 3.5: Gráfico MACD com divergências explícitas

[1]Termo inglês para designar uma tendência de desvalorização dos preços num determinado mercado financeiro.

[2]Termo inglês para designar uma tendência de valorização dos preços num determinado mercado financeiro.

De referir no entanto que o MACD não é um indicador que antecipa mudanças no mercado. O MACD é na realidade um indicador que segue a tendência do mercado.

3.5 Conclusão

Sobre os indicadores da análise técnica, mencionados na secção anterior, serão as suas fórmulas matemáticas as que serão usadas na implementação dos agentes investidores e que vão suportar as suas decisões de investimento. Isto porque, sendo os únicos que se baseiam no histórico das cotações para fazerem o cálculo das tendências de mercado, os dados de que necessitam são os mais fáceis de obter, nas fontes disponíveis na *internet*. Além disso, os outros métodos da análise fundamental baseiam-se em variáveis por vezes difíceis de analisar e quantificar.

Capítulo 4

A plataforma *JATLite*

4.1 Introdução

As raízes do *JATLite* (*Java Agent Template, Lite*) datam de 1993, quando no *Center for Design Research* (CDR) da Universidade de Stanford, começaram a usar a linguagem KQML[1] para comunicação entre agentes de software. Cedo aperceberam-se de que seria necessário encontrar um standard que proporcionasse os meios básicos de comunicação e as funções de interacção que os agentes necessitavam.

Houve uma primeira aproximação em 1996 que teve algum sucesso e gerou bastante interesse na comunidade académica, denominado de *Java Agent Template*, que evolucionou rapidamente, tendo sido desenvolvidas versões cada vez mais sofisticadas. Mas acabou por ter uma vida curta, pois com a saída do projecto do seu principal "mentor", os restantes decidiram-se pela sua descontinuação já que se tinha tornado demasiado complexo e de difícil suporte. Além disso, incorporava teorias de pesquisa específicas que não seriam necessárias numa infraestrutura que se queria genérica, e o sistema desenvolvido originalmente não incluía um reencaminhador de mensagens que logo verificaram ser necessário para correr agentes como *applets*. Assim, nesse mesmo ano de 1996, surgiu o *JATLite* que é um pacote de software mais pequeno, mais rápido, mais robusto, e direccionado apenas para fornecer a ligação e a infraestrutura de comunicações que os agentes necessitam.

O *JATLite* é pois, um pacote de programas escritos em linguagem Java, que permite aos seus utilizadores criarem rapidamente novos agentes de software que comunicam de forma robusta através da Internet, ele proporciona a infraestrutura básica na qual os agentes se registam com um *Agent Message Router* usando um nome e uma password, ligam/desligam da Internet, enviam e recebem mensagens, transferem ficheiros, e mais genericamente trocam informações com outros agentes nos vários computadores onde eles estejam a correr.

[1] *Knowledge Query and Manipulation Language*

4.2 Infraestrutura

A arquitectura do *JATLite* está organizada hierarquicamente por camadas cada vez mais especializadas (ver Figura 4.1), permitindo assim aos programadores poder escolher qual a camada mais apropriada a partir da qual devem começar a implementar os seus sistemas. Assim, um programador que queira utilizar comunicações TCP/IP mas não queira usar KQML pode usar apenas as camadas *Abstract* e *Base*.

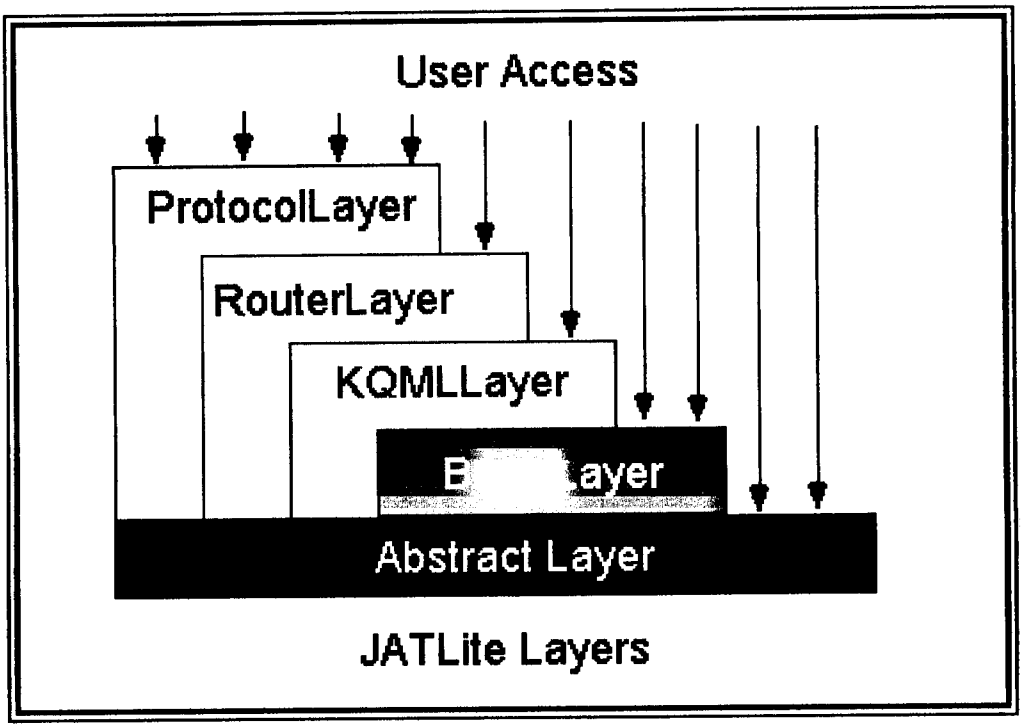


Figura 4.1: O *JATLite* é constituído por uma série de camadas especializadas

Estas camadas são as que se descrevem de seguida:

- a camada *Abstract* proporciona uma colecção de classes abstractas necessárias para a implementação do *JATLite*. Apesar de o *JATlite* assumir que todas as ligações são feitas usando o TCP/IP, poder-se-ia, por exemplo, implementar outros protocolos como o UDP estendendo a camada *Abstract*;

- a camada *Base* possibilita a comunicação básica através do TCP/IP e da camada *Abstract*. Não existe restrição quanto à linguagem da mensagem e ao seu protocolo. A camada *Base* pode ser estendida, por exemplo, para permitir *input* através de *sockets* e *output* para ficheiros. A camada *Base* pode também ser estendida para proporcionar aos agentes múltiplas portas para mensagens, etc.;

- a camada KQML possibilita o armazenamento e o processamento das mensagens KQML. Extensões ao standard KQML, propostas pelo CDR, foram implementa-

das para proporcionar um protocolo standard para registo, conexão, desconexão, etc., dos agentes;

- a camada *Router* fornece os serviços de registo de nomes e de gestão de filas de espera / reencaminhamento de mensagens para os agentes. Todos os agentes enviam e recebem mensagens através do *Router*, que depois as reenvia para os seus destinatários. Quando um agente se desliga intencionalmente, ou cai por acidente, o *Router* guarda as mensagens recebidas até que o agente se volte a ligar. O *Router* é particularmente importante para os agentes *applet* que, devido a restrições de segurança da WWW e do Java, podem apenas iniciar conexões *socket* com o servidor que os lançou;

- no topo da camada *Router* surge a camada *Protocol* que nesta versão *beta* suporta serviços internet standard como o SMTP e o FTP, ambos para aplicações e *applets*, mas outros protocolos podem ser facilmente estendidos a partir da camada *Protocol*. Se é esperado que os agentes vão transferir grande quantidade de informação, ou mesmo dados em formato binário, ou se os agentes necessitam enviar mensagens KQML através de e-mail, a camada *Protocol* será um bom ponto de partida.

4.3 O *Router JATLite* (AMR)

O *Router* é uma aplicação especializada que recebe a mensagem de um agente registado e reencaminha a mensagem para o destinatário correcto (ver Figura 4.2). As mensagens que forem sendo recebidas serão guardadas em fila de espera no sistema de ficheiros local.

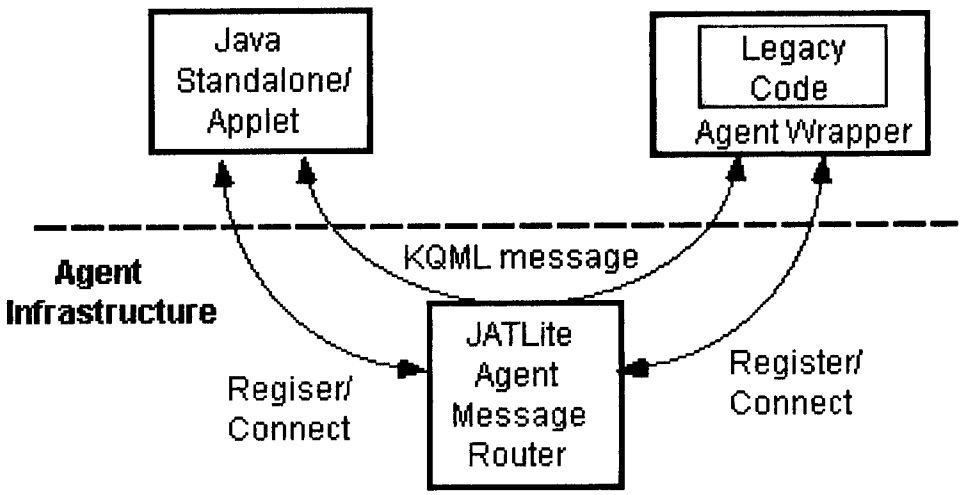


Figura 4.2: Infraestrutura dos agentes e o papel do *router* (facilitador)

Os agentes podem eliminar as mensagens que lhes são dirigidas enviando uma mensagem com o conteúdo "delete-message" para o *Router*.

No que concerne à implementação de um "*router*", neste caso em particular o AMR, as preocupações principais que se devem ter em atenção são as seguintes:

- Conexão por tentativas vs. colocação das mensagens em fila de espera: no caso da primeira, quando um agente pretende enviar uma mensagem para outro agente, primeiro verificará se existe já uma ligação disponível, se não existir, então tentará ligar-se ao agente receptor e se for bem sucedido, só então enviará a mensagem. Este tipo de aproximação levanta alguns problemas no caso de a conexão com o destinatário não poder ser iniciada (ou porque o receptor não está a correr por alguma razão, ou porque é um agente "*applet*"), assim a mensagem não consegue ser entregue directamente, surge então a necessidade de um depósito de mensagens (fila) onde as gravar e de um mecanismo de reencaminhamento para enviar a mensagem assim que se estabeleça ligação com o destinatário. Ora é precisamente isto que proporciona o *router* AMR, entre outras vantagens:

  - o *router* pode funcionar como um serviço de directórios e disponibilizar um mecanismo de comunicação com outros tipos de agentes (não Java ou não *applet*);
  - o *router* proporciona robustez colocando em fila de espera as mensagens, para que quando receba muitas mensagens de uma só vez ou quando um agente vá abaixo, as mensagens não se percam;
  - centralizando o depósito de mensagens, a tarefa de seguir o rasto às mensagens e depurar algum problema que ocorra torna-se mais fácil;

- Comunicação com agentes *applet*: como já referimos anteriormente, os agentes lançados como *applets* num *web browser*, têm determinadas limitações de segurança que só lhes permite comunicar com o servidor HTTP que os lançou, isto quer dizer que, na prática, os agentes deverão enviar todas as mensagens para alguma aplicação de reencaminhamento de mensagens que esteja a correr nesse servidor, que esta logo se encarregará de as reenviar para o seu destinatário, daí a necessidade da existência do *router* AMR;

- *Agent Name Service* (ANS): este é um serviço disponibilizado para a comunicação entre agentes, sendo que cada agente não precisa manter uma lista de endereços de todos os possíveis destinatários, já que esses podem estar constantemente a mudar. O agente perguntará pelo endereço de um outro agente, informação essa que o *router* mantém numa tabela dinâmica e disponibiliza para os agentes que a solicitem. Normalmente, para o envio e recepção de mensagens, esta funcionalidade não é necessária, mas os agentes podem utilizá-la para outros fins;

- Problema de escalonamento: pode-se esperar que se em determinado momento um elevado número de agentes está ligado e pretende enviar mensagens, isto possa causar problemas de performance ou mesmo perda de mensagens. Mas a estrutura do *router* foi pensada para permitir o escalonamento em situações como esta, podendo ser invocadas múltiplas instâncias do agente reencaminhador.

No que diz respeito às funcionalidades do *router* AMR podemos destacar as seguintes:

Fila de espera de mensagens: todas as mensagens que o *router recebe* serão colocadas em fila de espera no sistema de ficheiros local, sendo depois reenviadas ou apagadas segundo pedido do agente a que se destinam;

Reencaminhamento de mensagens: as mensagens serão reencaminhadas para o destinatário correcto logo que ele se ligue e possa recebê-las;

Serviço de nomes de agentes (ANS): o *router* proporciona o serviço de nomes para os agentes que o solicitem, apesar de não necessitarem dele para o simples envio de mensagens;

Registo: é disponibilizada a funcionalidade de registo de agentes. Se um agente quer usar os serviços do *router* deve registar-se junto dele;

Segurança: o *router* proporciona meios de garantir a segurança verificando as credenciais de determinado agente usando o seu nome e a password com que se registou. Quando se inicia uma ligação este protocolo de conexão deve ser respeitado;

Lista de agentes: o agente pode enviar para o *router* uma mensagem do tipo "*list-agents*"se quiser conhecer a lista dos agentes que estão registados e qual o seu estado;

Desconexão: o agente pode enviar uma mensagem do tipo "*disconnect-agent*"se não desejar receber mais mensagens até que se volte a ligar. Se um agente não envia essa mensagem, ou por opção ou porque caiu, o *router* tentará dentro de um certo período de tempo e de acordo com a sua disponibilidade, voltar a enviar as mensagens que lhe são destinadas, se essas tentativas não forem sucedidas, o registo do agente será automaticamente eliminado;

Reserva de mensagens: algumas mensagens podem ser reservadas para serem entregues num momento e local específico. O receptor pode ser por exemplo o próprio agente ou outro agente qualquer.

Quando falámos no *router* AMR do *JATLite* existem algumas premissas que são dadas como certas:

- a comunicação é baseada no protocolo TCP/IP;

- é permitida apenas uma ligação para cada agente registado (um agente não pode abrir mais do que um *socket* para o router ao mesmo tempo);
- as mensagens deverão ser mensagens KQML processáveis e o destinatário deverá ser especificado;
- todas as mensagens recebidas serão guardadas no sistema de ficheiros local. Quando um agente pretender apagar alguma mensagem deverá enviar uma mensagem do tipo "*delete-message*";
- um agente deve registar-se junto do *router* se pretende usar os seus serviços, logo deverá respeitar o protocolo de conexão enviando as mensagens correctas para estabelecimento da ligação.

4.4 A linguagem KQML

O *KQML* ou *Knowledge Query and Manipulation Language*, é uma linguagem e ao mesmo tempo um protocolo de comunicação de alto nível, para troca de mensagens, independente de conteúdo e da ontologia aplicável. Foi desenvolvida pelo *KSE* (*Knowledge Sharing Effort*) para servir ao mesmo tempo como um formato de mensagem e um protocolo de gestão de mensagens. Esta linguagem não se preocupa com o conteúdo da mensagem, mas sim com a especificação da informação necessária à compreensão do conteúdo.

Baseia-se num conjunto de performativas que permitem a realização de muitos tipos de protocolos, desde a simples interacção pergunta/resposta de um cliente/servidor, até à subscrição de um serviço e posterior recepção de respostas de forma assíncrona.

A linguagem KQML é dividida em três camadas[FLM96]:

- **conteúdo** - inclui a informação a ser efectivamente transmitida, numa linguagem de representação escolhida pelo agente emissor. A linguagem KQML suporta qualquer linguagem de representação, pois o conteúdo da mensagem é ignorado durante o processo de transmissão (com excepção da determinação do seu início e fim), cabe apenas ao agente receptor a tarefa de interpretação deste conteúdo;
- **mensagem** - é a parte central da linguagem KQML, e codifica a totalidade da informação que um agente deseja transmitir a outro. A mensagem inclui não apenas o conteúdo, mas também a "intenção"do emissor da mensagem, essa "intenção"especifica qual o tipo de interacção entre os agentes emissor e receptor, e obedece a um conjunto pré-estabelecido. Pode ainda incluir outras características descritivas opcionais, como por exemplo, a identidade do receptor, o formato da informação ou a ontologia[Gru93][Gru95][2] adoptada;

[2]Designa-se por ontologia o conjunto de conhecimentos específicos sobre determinado domínio de aplicação[Sil99]

- **comunicação** - codifica um conjunto de características para descrição de parâmetros de comunicação de baixo nível (por exemplo a identidade do emissor e do receptor, ou um identificador único associado à comunicação).

Embora o KQML tenha um conjunto de performativas reservadas, este não está minimizado nem fechado, e um agente que use KQML, pode escolher apenas algumas performativas para comunicar. O conjunto é extensível; uma comunidade de agentes pode escolher utilizar performativas adicionais, se a comunidade concordar, no entanto, se uma performativa é reservada, ele deve ser implementada de forma padrão.

Segundo alguns autores, o significado de performativas reservadas e padrões no KQML é pouco claro, normalmente está associado à intuição. Outros problemas e dificuldades normalmente associados ao KQML são:

- ambiguidade e termos vagos;
- performativas com nomes inadequados - algumas performativas têm nomes que não correspondem directamente ao acto comunicativo a ela associado;
- falta de performativas - alguns actos comunicativos não estão representados entre as performativas disponíveis.

Devido às questões acima levantadas, o KQML viria a ser substituído, como *standard*, pelo FIPA-ACL[3].

4.5 Agentes de software

4.5.1 Introdução

Definir agentes como um conceito único ou padrão é muito difícil, pois existem várias abordagens e pontos de vista de diferentes autores. Além disso, devido às suas mais diversas aplicações, uma definição precisa torna-se cada vez mais complicada e variada.

Mesmo assim, uma definição comum é que os agentes são componentes de aplicações que actuam em nome dos seus utilizadores[Nwa96], apresentando um conjunto básico de atributos reconhecidos, tais como: autonomia, persistência e sociabilidade. Adicionalmente podem apresentar características como: inteligência, aprendizagem ou mobilidade. Os agentes de software servem para facilitar a vida dos seus utilizadores; através do modelo de delegação, os utilizadores atribuem aos seus agentes tarefas tipicamente rotineiras, complexas, demoradas, ou tarefas dificilmente realizáveis, em tempo útil, por seres humanos.

[3] *Foundation for Intelligent Physical Agents - Agent Communication Language*

A FIPA (*Foundation for Intelligent Physical Agents*), define agente como *"uma entidade que reside num ambiente onde interpreta dados recebidos através de sensores que reflectem eventos no ambiente e executam acções que produzem efeitos no ambiente. Um agente pode ser software ou hardware puro..."*.

No geral, os agentes são entidades de *software* autónomas que actuam em determinados ambientes de forma a interagir com outros agentes, ao mesmo tempo que produzem acções e percepções sem requererem intervenções humanas constantes. Na perspectiva da Inteligência Artificial, um agente ideal teria que ser capaz de funcionar continuamente e adquirir experiências e conhecimentos acerca do ambiente com que está a interagir, ou seja, ser capaz de aprender e tomar decisões a partir de situações diferentes.

4.5.2 Definições e atributos de agentes

Segundo Ferber[Fer99], os agentes podem ser entidades físicas, ou seja, algo concreto que actue no mundo real; ou entidades virtuais, onde seriam representados por entidades abstractas tais como componentes de *software*, que é o objecto de estudo deste trabalho.

Algumas comparações entre objectos e agentes são inevitáveis, porém, apesar de existirem semelhanças, as diferenças são evidentes. Segundo Gerard Weiss[Wei99], o facto de os agentes terem uma noção mais forte de autonomia em relação aos objectos, serem capazes de um comportamento flexível e a característica de que um Sistema Multi-Agente é inerentemente *multi-thread*, espelham bem essas diferenças.

Mas a definição geralmente aceite é a adoptada por alguns pesquisadores, como é o caso de Michael Wooldridge e Nick Jennings[WJ95], onde adoptaram duas noções gerais: noção "fraca"e noção "forte"de agentes. Na definição ou noção "fraca"de agentes, eles são caracterizados como sistemas computacionais, sendo *hardware* ou *software*, com certas propriedades tais como autonomia, pró-actividade, sociabilidade, reactividade, persistência e robustez. Na noção "forte"de agentes, mais adoptada pelos pesquisadores ligados à área da Inteligência Artificial, possui, além das propriedades acima citadas, noções relacionadas com o comportamento humano, tais como o conhecimento, a crença, a intenção e a obrigação.

Assim sendo, e no que diz respeito à noção "fraca", podemos considerar o seguinte conjunto mínimo de características que um agente deve possuir:

- **autonomia** - um agente deve poder tomar decisões e acções importantes para a conclusão de uma tarefa ou objectivo, sem a necessidade de interferência do ser humano ou de qualquer outra entidade. Deve ser capaz de agir de forma independente com o seu ambiente através dos seus próprios sensores ou com as suas próprias percepções, com o objectivo de realizarem alguma tarefa, seja ela atribuída externamente ou gerada por ele próprio. Resumindo, operam sem a

intervenção humana e têm algum tipo de controlo sobre as suas acções e o seu estado interno;

- **pró-actividade** - relacionada com a autonomia está a pró-actividade, que não é mais que a capacidade que os agentes devem ter de tomar iniciativa, ou seja, não actuarem apenas em resposta a alterações no meio, eles devem ter a capacidade de encetar comportamentos conduzidos por objectivos;

- **sociabilidade** - os agentes interagem com outros agentes e possivelmente com os seus utilizadores. Esta função requer a existência de um meio de comunicação que lhes permita informar os outros sobre quais são os seus requisitos; e de um mecanismo interno de decisão sobre como e quando as interacções com os outros são apropriadas. Deve ser capaz de comunicar também com repositórios de informações. Sejam apenas simples repositórios de dados, o próprio ambiente ou os outros agentes, é fundamental uma constante troca de informações;

- **reactividade** - os agentes analisam o ambiente, e esta é a capacidade de reagir rapidamente a alterações no mesmo, ou seja, percebem o meio e respondem de modo oportuno. Para isso baseiam-se em três componentes principais: percepção, acção e comunicação;

- **persistência** - os agentes mantêm o seu estado interno ao longo da sua existência;

- **robustez** - um agente deve ter também robustez, para ser capaz de tomar decisões baseando-se em informações escassas ou incompletas, lidar com erros e ter a capacidade de adaptação ou aprendizagem através da experiência.

Para Wooldridge em [WJ95] e [Wei99], a existência de uma combinação de atributos tais como autonomia, reactividade, pró-actividade e sociabilidade, serão suficientes para classificar um agente como inteligente.

Todavia, e no que diz respeito à comunidade de Inteligência Artificial, a noção de agente apresenta um significado mais forte do que a noção "fraca"até agora apresentada. Para esta comunidade, um agente é um sistema computacional, que para além das propriedades referidas, possui também características normalmente associadas à qualidade humana, tais como mentais ou mesmo emocionais. Adicionalmente é-lhes exigida uma representação simbólica do ambiente envolvente, capacidades cognitivas e capacidades de aprendizagem. Seguem-se alguns dos atributos geralmente considerados na determinação de um agente, no que diz respeito a essa noção "forte":

- **aprendizagem** - a capacidade de aprendizagem está intrinsecamente associada à capacidade de manipulação e geração de conhecimento. Os agentes com capacidade de aprendizagem vão, ao longo da sua existência, reconhecendo padrões de comportamentos, padrões de preferências, etc., e vão actualizando a sua base de conhecimentos;

- **intencionalidade** - capacidade de representação explícita dos objectivos de um agente. Estes agentes, ditos intencionais ou cognitivos, apresentam quatro componentes; para além da percepção, acção e comunicação, apresentam ainda, capacidade de raciocínio sobre uma base de conhecimento;

- **mobilidade** - a capacidade de um agente se mover numa rede de modo a realizar as suas tarefas e atingir os seus objectivos. Diz-se agente móvel caso tenha essa capacidade, caso contrário diz-se estático ou estacionário;

- **racionalidade** - a característica de um agente não aceitar objectivos impossíveis de concretizar, contraditórios com os seus, ou ainda que não sejam compensatórios quer em termos de custo, risco ou esforço envolvido;

- **benevolência** - na premissa de que um agente não assume objectivos contraditórios e que procura realizar todas as tarefas que lhe forem solicitadas, diz-se benevolente quando adopta os objectivos dos outros, caso lhe seja solicitado, e desde que não sejam incompatíveis com os seus;

- **veracidade** - a característica de um agente não comunicar, pelo menos deliberadamente, informação falsa; quer seja com o seu utilizador, quer seja com outros agentes com os quais interaja;

- **características mentais** - a definição de um modelo de agente baseado em noções mentais, tais como conhecimento, crenças, intenções ou desejos, é umas das áreas de investigação mais importantes na disciplina de Inteligência Artificial, normalmente dividida em duas correntes de pensamento, a de Rao e Georgeff[RG95] baseada em agentes BDI (agentes com crenças, desejos e intenções) e a de Wooldridge e Jennings[WJ95] baseada em agentes com atitudes e pró-atitudes.

Contudo, um agente não precisa de ter todas estas características ao mesmo tempo, existem agentes que possuem apenas algumas, outros que possuem todas, o que é certo é que actualmente existe pouca concordância sobre a importância dessas propriedades e sobre se poderão ser ou não consideradas obrigatórias para a caracterização de um agente. O consenso é que essas características tornam, em muito, um agente diferente de simples programas e objectos.

4.5.3 Categorias de agentes

Para uma melhor compreensão da aplicabilidade da tecnologia de agentes, torna-se interessante falar sobre os diversos tipos de agentes e sobre as características que os diferenciam, para que tenhamos uma melhor noção de utilidade no emprego de agentes. É possível fazer uma classificação dos agentes de acordo com vários aspectos, como quanto à mobilidade, quanto ao relacionamento entre agentes e quanto à capacidade de raciocínio:

- **agentes móveis** - são agentes que têm a mobilidade como característica principal, isto é, capacidade para se moverem por uma rede interna local (*intranet*) ou até mesmo pela *internet*, transportando-se pelas plataformas, levando dados e código. O seu uso tem crescido devido a alguns factores, tais como uma heterogeneidade cada vez maior das redes e o seu grande auxílio em tomadas de decisões baseadas em grandes quantidades de informação;
- **agentes estáticos ou estacionários** - são aqueles opostos aos móveis, isto é, são fixos num mesmo ambiente ou plataforma e não se movimentam em qualquer rede, muito menos na *internet*;
- **agentes competitivos** - são agentes que competem entre si para a realização dos seus objectivos ou tarefas, ou seja, não existe colaboração entre os agentes;
- **agentes coordenados ou cooperantes** - são agentes com a finalidade de alcançarem um objectivo maior, para isso realizam tarefas específicas, porém coordenando-as entre si de forma a que as suas actividades se completem;
- **agentes reactivos** - são agentes que reagem a estímulos sem terem memória do que já foi realizado no passado, nem previsão da acção a ser tomada no futuro. Não têm representação do seu ambiente ou de outros agentes e são incapazes de preverem e anteciparem acções. Baseiam-se principalmente na "teoria do caos", na qual se afirma que, até mesmo no caos existe uma "certa organização". Geralmente actuam em sociedades, como uma colónia de formigas por exemplo, onde uma única formiga não apresenta muita inteligência, mas quando age em grupo, comporta-se o todo como uma única entidade com uma certa inteligência. Assim, a força de um agente reactivo vem da capacidade de formar um grupo e construir colónias capazes de se adaptarem a um ambiente;
- **agentes cognitivos** - estes, ao contrário dos agentes reactivos, podem raciocinar sobre as acções tomadas no passado e planearem acções a serem tomadas no futuro, ou seja, um agente cognitivo é capaz de resolver problemas por ele próprio. Ele tem objectivos e planos explícitos, os quais lhe permitem atingir o seu objectivo final. Ferber[Fer99] afirma que, para que isso se concretize, cada agente deve ter uma base de conhecimento disponível, que compreende todos os dados e todo o *know-how* para realizar as suas tarefas e interagir com os outros agentes e com o próprio ambiente. A sua representação interna e os seus mecanismos de inferência, permitem-lhe actuar independentemente dos outros agentes, e dão-lhe uma grande flexibilidade na forma de expressão do seu comportamento. Além disso, e devido à sua capacidade de raciocínio baseado nas representações do mundo, são capazes de ao mesmo tempo memorizar situações, analisá-las e prever possíveis reacções para as suas acções.

4.5.4 Aplicações de agentes inteligentes

Seguem-se alguns exemplos de aplicações de agentes de *software* inteligentes:

- **agentes no processo de ensino e aprendizagem** - utilização de agentes no processo de aprendizagem e no processo de formação de utilizadores;
- **agentes na indústria** - a grande vantagem da aplicação de agentes na produção industrial é o facto de que muitos dos sistemas que actuam neste ambiente são complexos e isolados, o papel dos agentes seria integrá-los e compartilhar informações entre eles;
- **agentes em simulações** - nestas aplicações o agente actua como um participante que auxilia e monitoriza certas actividades dos utilizadores, assistindo-os ou ajudando-os quando necessário, principalmente em ambientes virtuais muito complexos;
- **agentes na prestação de serviços** - nesse contexto os agentes irão adicionar valor a certos serviços, gerindo a informação com o fim de satisfazer as necessidades dos clientes. Exemplos desses serviços seriam:
  - manutenção e actualização de bases de dados, explorando a informação (*data mining*);
  - comércio electrónico, onde os agentes procuram os preços mais convenientes ou ofertas e produtos com as especificações desejadas pelo utilizador;
  - gestão de correio electrónico e filtragem de mensagens, funcionando como um assistente pessoal;
  - gestão de produção e muitas outras aplicações.
- **agentes em redes de computadores** - as suas funções nesta área varia muito, varia desde à definição de melhores rotas, controlo de manutenção de equipamentos, gestão de dados até à monitorização de actividades e gestão da própria rede.

4.6 Conclusão

Com este capítulo pretendeu-se fazer uma introdução teórica a toda a envolvente da Inteligência Artificial em que se enquadra este projecto, como são os Sistemas Multi-Agentes e os respectivos Agentes de software, permitindo assim a sua compreensão por um público mais abrangente. Começa-se então por definir e de alguma forma justificar a escolha da infraestrutura de comunicação entre os agentes, depois faz-se uma breve referência à linguagem usada por essa plataforma e finalmente aborda-se a teoria dos agentes de software apresentando-se diversas definições de alguns dos estudiosos nesta área, entre outras informações consideradas relevantes para a percepção desta problemática.

Capítulo 5

Plataforma e desenvolvimento do sistema

5.1 Plataforma de desenvolvimento

5.1.1 Introdução

Nas próximas secções, onde se referem as ferramentas que compõem a plataforma de desenvolvimento, pretende-se apenas enumerá-las e apresentar as vantagens de cada uma justificando assim a sua escolha e utilização.

5.1.2 Java Development Kit 5.0

O Java é uma linguagem de programação orientada ao objecto, desenvolvida na década de 90 pelo programador James Gosling e sua equipa, na empresa Sun Microsystems. Diferente de todas as outras linguagens convencionais que existiam até então, que são compiladas para código nativo, a linguagem Java é compilada para um "*bytecode*" que é executado por uma máquina virtual.

Nasceu de um projecto cujo objectivo era antecipar-se à "próxima onda" do mundo digital, em que acreditavam que haveria uma convergência dos computadores com os equipamentos e electrodomésticos vulgarmente usados pelas pessoas no seu dia-a-dia, logo necessitavam de algo que lhes permitisse que dispositivos heterogéneos comunicassem entre si. Inicialmente, foi desenvolvido a pensar nas redes da TV a cabo, em que o utilizador poderia controlar as emissões de televisão que chegam a sua casa e usufruir do vídeo *on demand*, mas esta revelou-se uma ideia demasiado visionária para a altura, e acabou por não gerar muito interesse junto do mercado que não estava preparado para os chamados dispositivos electrónicos "inteligentes".

A sorte é que o *boom* da Internet aconteceu, e rapidamente a grande rede interactiva que procuravam estava a crescer, a tecnologia Java que tinha sido projectada para se mover através de redes de dispositivos heterogéneos, tinha agora na Internet a sua plataforma de lançamento. Agora as aplicações poderiam ser executadas dentro dos *browsers* nos *applets* Java e tudo seria disponibilizado pela Internet instantaneamente, foi o estático HTML dos *browsers* que promoveu a rápida disseminação da dinâmica tecnologia Java. A velocidade dos acontecimentos que se seguiram foi assustadora, o número de utilizadores cresceu rapidamente e grandes "*players*"como a IBM anunciaram o seu suporte para a tecnologia Java.

5.1.2.1 Principais características da linguagem Java

A linguagem Java foi projectada tendo em vista os seguintes objectivos:

- orientação ao objecto - baseado no modelo de Smalltalk e Simula67;
- portabilidade - independência de plataforma - "*write once run anywhere*";
- recursos de rede - possui uma extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP;
- segurança - pode executar programas via rede com restrições de execução.

Além disso, podem-se destacar outras vantagens apresentadas pela linguagem:

- sintaxe similar à linguagem C/C++;
- facilidades de internacionalização - suporta nativamente caracteres *Unicode*;
- simplicidade na especificação, tanto da linguagem como do "ambiente"de execução (JVM);
- é distribuída com um vasto conjunto de bibliotecas (ou APIs);
- possui facilidades para criação de programas distribuídos e multi-tarefa (múltiplas linhas de execução num mesmo programa);
- "*desalocação*"de memória automática por processo de colector de lixo;
- carregamento dinâmico de código - programas em Java são formados por uma colecção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

5.1.2.2 Orientação ao objecto

Umas das principais características do Java, a orientação ao objecto ("OO"), refere-se a um método de programação e arquitectura da linguagem. Apesar de existirem muitas interpretações da OO, a ideia principal é desenhar *software* de forma a que os diferentes tipos de dados que sejam manipulados estejam combinados com as operações relevantes a que eles estejam sujeitos. Assim, os dados e o código são combinados em entidades denominadas objectos. Um objecto pode ser visto como um conjunto fechado e hermético de comportamentos (código) e de estados (dados). Esta separação em objectos coerentes proporciona uma base mais estável para o desenvolvimento de sistemas de informação. O objectivo é tornar grandes projectos de *software* mais fáceis de manejar, melhorando assim a sua qualidade e reduzindo o número de projectos falhados ou que ultrapassam os tempos previstos de conclusão.

Outro dos objectivos principais da programação OO é desenvolver objectos genéricos para que parte do software possa ser reutilizado noutros projectos. Neste sentido, espera-se que os objectos sejam vistos mais como componentes *plug-and-play*, ajudando a indústria a desenvolver aplicações principalmente a partir de fragmentos de código "compilado"já testados, resultando em reduções na duração dos seus desenvolvimentos. A reutilização do *software* encontrou-se na prática com duas dificuldades principais: o desenho de objectos verdadeiramente genéricos não é bem entendido, e não existe uma metodologia para uma comunicação abrangente e efectiva de oportunidades de reutilização. Algumas comunidades *open source*[1] tentam ajudar a diminuir este problema da reusabilidade do *software*, fornecendo aos autores formas e ferramentas para disponibilizarem e disseminarem informação acerca dos objectos genéricos reutilizáveis e das livrarias que desenvolveram.

5.1.2.3 Máquina Virtual Java

Os programas Java não são traduzidos para a linguagem de máquina como outras linguagens estaticamente compiladas e sim para uma representação intermédia, chamada de *bytecodes*, esses *bytecodes* são depois interpretados pela máquina virtual Java (JVM - Java Virtual Machine).

Muitas pessoas acreditam que por causa desse processo, o código interpretado Java tem baixo desempenho. Durante muito tempo essa foi uma afirmação verdadeira, porém novos avanços têm tornado o compilador dinâmico (a JVM), em muitos casos, mais eficiente que o compilador estático. Hoje o Java já possui uma performance próxima do C++. Isso é possível graças a optimizações como a compilação especulativa, que aproveita o tempo inactivo do processador para pré-compilar *bytecode* para código nativo.

No entanto essa implementação tem alguns inconvenientes, a pré-compilação exige

[1] Veja-se o caso por exemplo das comunidade *sourceforge.net* e *freshmeat.net*

tempo, o que faz com que programas Java demorem significativamente mais para começarem a funcionar, somando a isso ainda o tempo de carregamento da máquina virtual. Ora, isso não é um grande problema para programas que correm em servidores e que deveriam ser arrancados apenas uma vez, no entanto isso pode ser bastante indesejável para computadores pessoais onde o utilizador deseja que o programa execute logo depois de o lançar.

O Java possui ainda uma outra desvantagem considerável em programas que usam bastante processamento numérico. O padrão Java tem uma especificação rígida de como devem funcionar os tipos numéricos, e essa especificação não condiz com a implementação de números de vírgula flutuante na maioria dos processadores o que faz com que o Java seja significativamente mais lento para estas aplicações quando comparado a outras linguagens.

Os *bytecodes* produzidos pelos compiladores Java permitem ainda a aplicação de técnicas de engenharia reversa para a recuperação do programa fonte original, sendo que esta é uma característica que atinge em menor grau todas as linguagens compiladas. No entanto já existem hoje tecnologias que "baralham"e até mesmo encriptam os *bytecodes* impedindo praticamente qualquer tentativa de engenharia reversa.

5.1.2.4 Independência da plataforma

Uma das características fundamentais do Java é a sua portabilidade, ou seja, um programa Java deverá correr independentemente da plataforma computacional de suporte. Os programas são compilados para um ficheiro de um formato intermédio, o qual é executado pela máquina virtual Java.

A independência da plataforma é conseguida conjuntamente através dos seguintes mecanismos:

- definição de *bytecodes* independentes da máquina;
- tipos de dados primitivos (caracter, inteiro, real, etc.) de tamanho fixo;
- representação de código independente da rede (através da representação/ordenação de *bytes* segundo o formato *big-endian*);
- existência de uma biblioteca standard de classes.

O principal benefício desta característica, é que por esta forma o mesmo programa pode ser executado, salvaguardando a existência da máquina virtual de suporte, em diferentes plataformas computacionais e em diferentes domínios de aplicações, entre outros: em computadores de secretária, servidores, microprocessadores para electrodomésticos, telefones móveis, etc.

5.1.2.5 Segurança

A segurança do sistema Java coloca-se a diferentes níveis conforme de seguida se apresenta:

- ao nível da linguagem - o Java é uma linguagem segura, isto é, a questão não é se o código é malicioso, mas sim se é robusto, ou seja, se a definição da linguagem não permite a construção de programas incorrectos (verificação ao nível do compilador). Como exemplo da segurança na definição da linguagem, os seguintes pontos podem ser enumerados:
  - não permite a manipulação de apontadores;
  - as conversões de tipos (*casts*) são controladas de forma a não serem violadas as regras da linguagem;
  - a gestão da memória é automática.
- ao nível da execução/interpretação de código - além dos testes realizados na fase de compilação, descritos no ponto anterior, a máquina virtual executa adicionalmente um outro conjunto de testes, agora sobre o *bytecode* gerado, tanto na fase de carregamento do código, como durante a sua execução, tais como:
  - garantir que o acesso a um elemento de um *array* não sai fora dos seus limites definidos;
  - garantir a compatibilidade de tipos de objectos;
  - etc.
- ao nível da aplicação e da implementação das bibliotecas de classes - um conjunto significativo das classes standard do Java, designadamente aquelas que permitem o acesso a recursos de sistema (ficheiros, conexões de rede, janelas gráficas, etc.), são desenhadas de forma que sejam sempre realizados testes de segurança antes do seu acesso e da execução dos seus métodos. Adicionalmente, as classes de aplicações finais, designadamente aquelas que providenciam acesso a recursos novos e eventualmente críticos, podem também seguir o modelo de segurança proposto.

5.1.3 NetBeans IDE 5.0

NetBeans [Myc00] é um projecto *open source* de sucesso, com uma grande base de utilizadores, uma crescente comunidade e perto de 100 parceiros mundiais, que continuam a aumentar. A *Sun Microsystems* fundou o projecto *NetBeans* em Junho de 2000 e continua a ser o seu principal patrocinador.

Hoje, existem dois produtos: o IDE *NetBeans* (*NetBeans IDE*) e a Plataforma *NetBeans* (*NetBeans Platform*).

O *NetBeans IDE* é um ambiente de desenvolvimento - uma ferramenta para programadores, que permite escrever, compilar, depurar e instalar programas. A IDE é completamente escrita em Java, mas pode suportar qualquer linguagem de programação. Existem também um grande número de módulos que tornam o IDE *NetBeans* particularmente extensível. O *NetBeans* IDE é um produto livre e sem restrições de como pode ser usado.

Também está disponível a *NetBeans Platform*; uma base modular e extensível que pode ser usada como infraestrutura para se criar grandes aplicações de *desktop*. Inúmeros parceiros fornecem *plug-ins* que podem ser facilmente integrados na Plataforma, e que podem ser utilizados para desenvolver ferramentas e soluções próprias.

Ambos os produtos são *open source* e livres para uso comercial e não comercial. O código fonte está disponível para ser usado através da licença *Common Development and Distribution License* (CDDL).

5.1.4 JATLite Beta 0.4

Foi usada a versão 0.4 Beta do *JATLite* disponível no servidor FTP da Universidade de Stanford [Uni96]. A abordagem desta plataforma já foi feita num capítulo anterior pelo que não cabe aqui fazê-lo novamente.

5.1.5 Java Financial Library 1.6.1

A *Java Financial Library* [Ree03] é uma livraria para aplicações Java que converte valores monetários para diferentes moedas nacionais e obtém informação sobre as acções cotadas em bolsa através da Internet. Usa fontes de dados externas tais como a *Yahoo* e a *Oanda* para obter essas informações, tais como cotação actual, valor de abertura, máximos, mínimos e volume no caso das acções, e factores de conversão no caso das moedas.

No âmbito deste trabalho será usada como fonte de dados a *Yahoo*, nomeadamente através do site *http://finance.yahoo.com* (ver Figura 5.1), que disponibiliza as cotações actuais das acções nos principais mercados bolsistas, com um pequeno atraso de 15 minutos no caso do *Nasdaq* e 20 minutos para o *NYSE* (*New York Stock Exchange*).

5.1.6 JFreeChart 1.0.2

A livraria *JFreeChart* [Gil00] resulta de um projecto fundado há 6 anos atrás, em Fevereiro de 2000 por David Gilbert. É uma livraria 100% Java, totalmente grátis, para tornar mais fácil aos programadores, a inclusão de gráficos com aspecto e qualidade profissional nas suas aplicações.

Figura 5.1: Exemplo da informação que pode ser obtida a partir do *site Yahoo Finance*

De entre as características desta livraria destacam-se as seguintes:

- uma API consistente e bem documentada, que suporta um vasto leque de tipos de gráfico;
- desenho flexível que é facilmente extensível, e serve tanto para aplicações que operam do lado do servidor como também do lado do cliente;
- exporta para inúmeros formatos, incluindo componentes Swing, formatos gráficos (incluindo PNG e JPEG), e formatos gráficos vectoriais (incluindo PDF, EPS e SVG);
- é *open source* ou, mais especificamente, *software* livre. É distribuída segundo os termos da *GNU Lesser General Public Licence* (LGPL).

Além disso, e com o objectivo de financiar e suportar o projecto, e mantê-lo grátis acima de tudo, David Gilbert criou uma empresa onde vende documentação de apoio para esta livraria e serviços de consultadoria.

Exemplos de alguns gráficos gerados por esta livraria são:

- *Price Volume* que combina a variação diária do preço com o volume de transacção de uma determinada acção, tal como se pode ver na Figura 5.2;
- *Candlestick* que combina informação sobre o preço de abertura, máximos e mínimos diários, o preço de fecho de uma determinada acção e o seu volume de transacção. Ao mesmo tempo é possível analisar a diferença entre o preço de abertura e o preço de fecho, sendo representada pelo sólido rectangular que aparece em cada período em análise. A cor que apresenta esse sólido também tem um significado, se for verde será porque houve uma evolução positiva do preço, vermelho significará o contrário, tal como se pode ver na Figura 5.3;
- *High Low with Moving Average* em que tal como no gráfico anterior, apresenta todos os valores que pode assumir a cotação de uma determinada acção no

período em análise, acrescentando uma linha com a média móvel dos valores assumidos, tal como se pode ver na Figura 5.4.

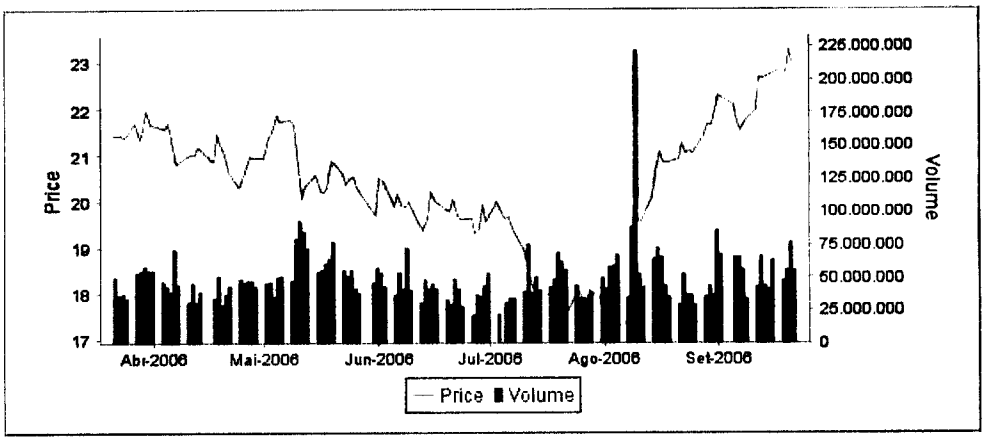


Figura 5.2: Gráfico do tipo *Price Volume* gerado pelo JFreeChart

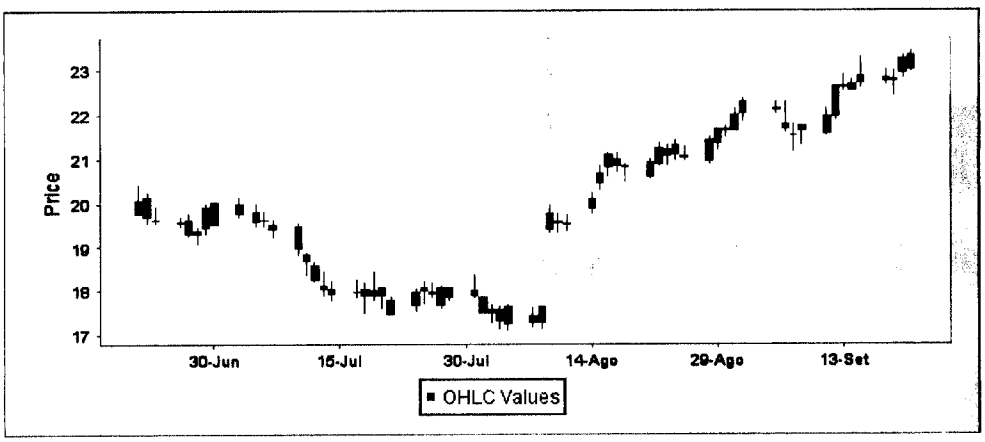


Figura 5.3: Gráfico do tipo *Candlestick* gerado pelo JFreeChart

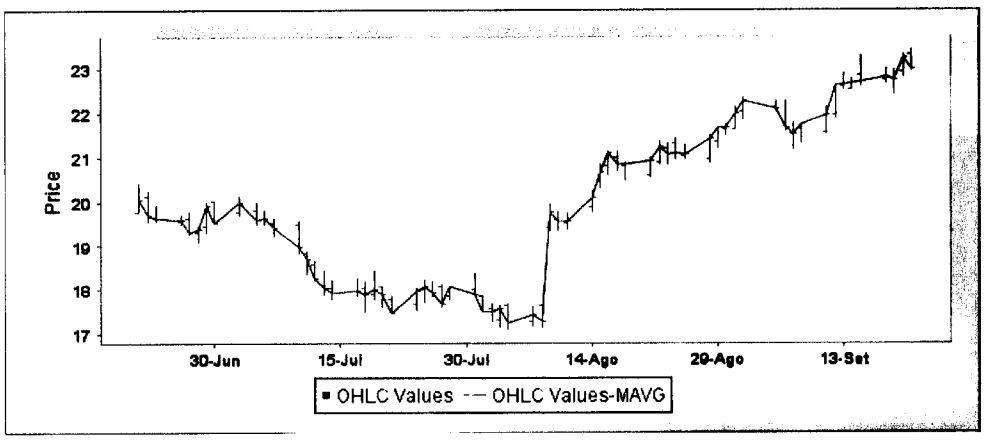


Figura 5.4: Gráfico do tipo *High Low with Moving Average* gerado pelo JFreeChart

5.1.7 PostgreSQL 8.1

PostgreSQL [Pos95] é um sistema de gestão de bases de dados objecto-relacional (ORDBMS) baseado no *POSTGRES*, versão 4.2, desenvolvido na Universidade da Califórnia, Departamento de Ciência dos Computadores de Berkeley. O *POSTGRES* foi pioneiro em muitos conceitos, que só muito tarde ficaram também disponíveis em alguns sistemas de bases de dados comerciais.

PostgreSQL é o descendente *open source* do código original de Berkeley, que renasceu durante o ano de 1995. Suporta os standards SQL92 e SQL99 e oferece, entre outras, algumas funcionalidades mais modernas:

- consultas complexas;
- chaves estrangeiras;
- integridade transaccional;
- controle de concorrência multi-versão;
- suporte ao modelo híbrido objecto-relacional;
- *triggers*;
- *views*;
- *stored procedures* em várias linguagens.

Além disso o *PostgreSQL* pode ser estendido pelo utilizador de muitas formas, por exemplo acrescentando:

- tipos de dados;
- funções;
- operadores;
- funções agregadas;
- métodos de indexação;
- linguagens processuais.

E devido à sua licença liberal, o *PostgreSQL* pode ser usado, modificado, e distribuído livre de encargos para qualquer propósito, quer seja ele privado, comercial, ou académico.

Usando um modelo cliente/servidor, uma sessão *PostgreSQL* consiste num processo servidor e em aplicações clientes. O processo servidor, que maneja os ficheiros da base de dados, aceita ligações das aplicações cliente, e executa as acções a pedido dos clientes. O programa servidor da base de dados é chamado *postmaster*.

5.2 Desenvolvimento do sistema

5.2.1 Introdução

Tal como já foi indicado anteriormente, pretendia-se o desenvolvimento de um Sistema Multi-Agente que simulasse, tão próximo do real quanto possível (apenas as transacções monetárias não seriam reais), um ambiente de uma Bolsa de Valores. Este sistema seria composto por um agente Bolsa e por vários agentes Investidores. O agente Bolsa, seria a entidade responsável, entre outras coisas, por obter e fornecer aos agentes Investidores toda a informação necessária, para que pudessem tomar as suas decisões de investimento com o objectivo de maximizarem os seus rendimentos.

Ainda numa tentativa de aproximar mais esta simulação da realidade, as acções a serem transaccionadas seriam as de um mercado real e as suas cotações deveriam ser obtidas em tempo real a partir de uma qualquer fonte disponível na Internet. Ora com o uso da *Java Financial Library* esta tornou-se uma tarefa mais simples, já que se tornava possível obter as cotações a partir do *site* financeiro da *Yahoo*[2] apenas com 15 minutos de atraso no caso do *Nasdaq* e 20 minutos no caso do *New York Stock Exchange*, tempos considerados aceitáveis para um ambiente de simulação como este.

Quanto aos agentes Investidores, estes usariam das mais modernas teorias da análise técnica, tais como os indicadores Índice de Força Relativa (RSI), o Estocástico (SI) e o da Média Móvel Convergência / Divergência (MACD), sobre os dados que obtêm do agente Bolsa para assim calcularem oportunidades de investimento.

Finalmente, a aplicação deveria dispor de um ambiente gráfico agradável e de fácil utilização, tanto no caso do agente Bolsa como no caso dos agentes Investidores, onde toda a informação necessária deveria ser apresentada de uma forma clara e sucinta. E tratando-se de uma aplicação com uma forte componente financeira, o uso de gráficos seria desejável, tarefa também simplificada com o uso da livraria *JFreeChart*.

Assim, e depois de ter absorvido toda a teoria que já foi exposta nos capítulos anteriores era chegada a hora de colocar todo esse conhecimento em funcionamento e passar à prática. Depois de ultrapassadas as fases de configuração da plataforma de desenvolvimento que incluíram:

- Instalação e configuração do servidor da base de dados *PostgreSQL*;
- Instalação e configuração da plataforma *JATLite*;
- Instalação e configuração do ambiente de desenvolvimento *NetBeans*;
- Instalação e colocação no *CLASSPATH*:
  - da livraria do *JATLite*;

[2] http://finance.yahoo.com/

- da *Java Financial Library*;
- da livraria do *JFreeChart*.

passar-se-ia à fase seguinte, ou seja, à modelação da base de dados.

5.2.2 Modelação da base de dados

No que diz respeito à modelação da base de dados e tendo em conta as necessidades de um sistema como este, obviamente teríamos necessidade das seguintes tabelas:

- tabela para os **mercados**, onde registaríamos um código identificativo, a sua designação, a hora de abertura e a hora de fecho (aqui, para simplificação não vamos considerar zonas horárias, considerámos sempre a zona horária do servidor em que corre a aplicação);
- tabela para as **acções**, onde registaríamos um código identificativo, a sua designação, o código do mercado a que pertence, o mínimo das últimas 52 semanas, o máximo das 52 últimas semanas, o indicador RSI, o indicador SI e o indicador MACD;
- tabela para o histórico das cotações ***intraday***, onde registaríamos o código único da acção, um *Timestamp*[3] identificativo do momento da leitura, o volume transaccionado nesse momento e a sua cotação;
- tabela para o histórico das cotações **diárias**, onde registaríamos o código identificativo da acção, uma data representando o dia da leitura, o volume transaccionado, o valor de abertura, o valor mínimo, o valor máximo e o valor de fecho;
- tabela para o histórico diário dos **indicadores**, onde registaríamos o código identificativo da acção, uma data representando o dia de cálculo do indicador, o indicador RSI, o indicador SI, o *sinal SI*, o histograma SI, o indicador MACD, o *signal MACD* e o histograma MACD;
- tabela para os **investidores**, onde registaríamos um código identificativo, uma palavra passe e o saldo da conta do investidor;
- tabela para os ***portfolio*** dos investidores, onde registaríamos o código identificativo do investidor, o código único da acção, a quantidade possuída e o preço de compra.

[3]Um Timestamp identifica um momento exacto no tempo, normalmente contém o número de milisegundos desde 1 de Janeiro de 1970

As instruções SQL para criação das tabelas acima descritas podem ser consultadas no apêndice A.

O esquema da base de dados e a interdependência entre as tabelas é aquele que se apresenta na Figura 5.5 que se segue:

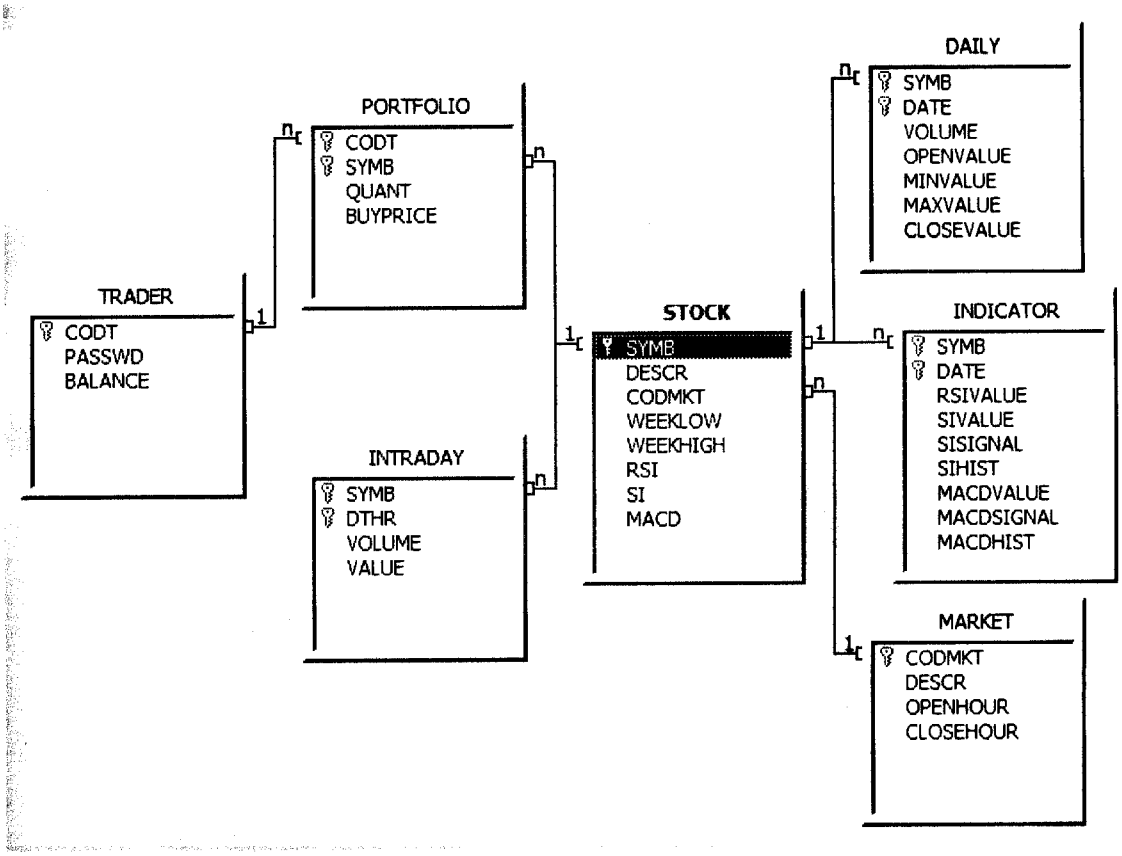


Figura 5.5: Esquema da base de dados e a interdependência entre tabelas

5.2.3 O agente Bolsa e os seus componentes

Tal como já foi referido, pretende-se o desenvolvimento de um Sistema Multi-Agente, sendo que este primeiro componente corresponde ao agente Bolsa que serve de intermediário entre os agentes Investidores e o mundo real das Bolsas de Valores.

A descrição das classes do agente Bolsa e dos seus componentes pode ser consultada no apêndice B.

5.2.3.1 Interface gráfica e outras funções

A interface gráfica, tal como já foi mencionado, foi totalmente desenvolvida fazendo uso do ambiente integrado de desenvolvimento da *NetBeans*.

Esta interface permite, entre outras coisas, as seguintes funções:

- controlar o funcionamento das *threads*, isto é, permite pausar, resumir, parar e arrancar as *threads* de comunicação com os agentes e de gestão das tabelas de cotações, além de permitir também mudar os seus tempos de espera;
- consultar as mensagens enviadas e recebidas, para e dos agentes respectivamente (consultar fig.C.1 no apêndice C);
- manter as tabelas de acções, mercados, cotações *intraday* e diárias, manutenção essa que inclui a consulta, inserção e eliminação de registos (consultar figuras C.2, C.3, C.4 e C.5 no apêndice C);
- visualizar de uma forma gráfica as cotações das acções em tempo real e os gráficos a elas associados (consultar figuras C.6, C.7, C.8, C.9, C.10, C.11 e C.12 no apêndice C);

A descrição da classe que implementa a interface gráfica, entre outras funções, pode ser consultada no apêndice B.1.

5.2.3.2 *Thread* para gestão da comunicação com os agentes

A *thread* que faz a gestão da comunicação com os agentes, funciona por defeito de meio em meio segundo e tem como principais funções:

- registar-se junto do *router* do *JATLite*;
- enviar uma mensagem de abertura de mercado e aguardar os pedidos dos agentes investidores;
- responder às performativas recebidas dos agentes investidores, quer sejam de gestão de utilizadores, quer sejam de pedido de cotações, quer sejam de compra e venda de acções.

A descrição da classe que implementa a *thread* para gestão da comunicação com os agentes, entre outras funções, pode ser consultada no apêndice B.2.

5.2.3.3 *Thread* para gestão das tabelas de cotações

A *thread* que faz a gestão das tabelas de cotações, funciona por defeito em períodos de 5 em 5 minutos, e tem como principais funções:

- funcionar segundo o horário dos mercados e respeitar os dias de descanso;

- actualizar as cotações das acções presentes na base de dados, nomeadamente as tabelas *intraday* e diária, recolhendo informação de uma fonte real, disponível na Internet;
- actualizar os valores dos máximos e mínimos anuais (*52 week high and low*);
- calcular e actualizar os valores dos indicadores diários de análise técnica (RSI, SI e MACD);
- actualizar o quadro onde são mostradas as cotações em tempo real.

A descrição da classe que implementa a *thread* para gestão das tabelas de cotações, entre outras funções, pode ser consultada no apêndice B.3.

5.2.3.4 *Thread* para manutenção da base de dados

A *thread* que faz a manutenção da base de dados é responsável e tem como única função manter a base de dados "saudável"e na sua óptima forma, para isso lança uma série de processos de limpeza e de reindexação de meia em meia hora.

A descrição da classe que implementa a *thread* para manutenção da base de dados pode ser consultada no apêndice B.4.

5.2.4 O agente Investidor e os seus componentes

Este componente corresponde ao agente Investidor que, em representação do seu utilizador se comunica com o agente Bolsa, obtendo a informação necessária para calcular oportunidades de investimento e realizar as respectivas operações de compra e venda.

A descrição das classes do agente Investidor e dos seus componentes pode ser consultada no apêndice B.

5.2.4.1 Interface gráfica e outras funções

A interface gráfica, tal como no caso do agente Bolsa, foi totalmente desenvolvida fazendo uso do ambiente integrado de desenvolvimento da *NetBeans*.

Esta interface permite, entre outras coisas, as seguintes funções:

- controlar o funcionamento das *threads*, isto é, permite pausar, resumir, parar e arrancar as *threads* de comunicação com o agente Bolsa, de actualização das tabelas de acções e respectivas cotações e da *thread* de cálculo de oportunidades de investimento, além de permitir também mudar os seus tempos de espera;

- consultar as mensagens enviadas e recebidas, para e do agente Bolsa respectivamente (consultar fig.C.1 no apêndice C);
- visualizar de uma forma gráfica as cotações das acções em tempo real e os gráficos a elas associados (consultar figuras C.6, C.7, C.8, C.9, C.10, C.11 e C.12 no apêndice C);
- visualizar de uma forma gráfica a evolução das cotações das acções pertencentes ao *porfolio* do investidor (consultar fig.C.13 no apêndice C), bem como realizar operações manuais de investimento. É aqui também que se definem os parâmetros que influenciam o comportamento do agente.

A descrição da classe que implementa a interface gráfica, entre outras funções, pode ser consultada no apêndice B.5.

5.2.4.2 *Thread* para gestão da comunicação com o agente Bolsa

A *thread* que faz a gestão da comunicação com o agente Bolsa, funciona por defeito de meio em meio segundo e tem como principais funções:

- registar-se junto do *router* do *JATLite*;
- enviar as mensagens KQML para gestão de utilizadores, requisição de cotações diárias e *intraday*, requisição de *portfolios* e envio de ordens de compra e venda;
- receber e actualizar as cotações das acções presentes nas tabelas internas do investidor, nomeadamente a das cotações *intraday*, a das cotações diária, a tabela das acções e a tabela do *portfolio* do investidor;
- calcular os valores dos indicadores diários de análise técnica (RSI, SI e MACD) para as cotações recebidas.

A descrição da classe que implementa a *thread* para gestão da comunicação com o agente Bolsa, entre outras funções, pode ser consultada no apêndice B.6.

5.2.4.3 *Thread* para actualização das tabelas de acções e respectivas cotações

A *thread* que faz o pedido de actualização das tabelas de acções e respectivas cotações ao agente Bolsa, funciona por defeito em períodos de 5 em 5 minutos, e tem como principais funções:

- requisitar ao agente Bolsa a actualização das cotações das acções presentes nas tabelas internas do investidor, nomeadamente a das cotações *intraday*, a das cotações diária e a tabela das acções;

- requisitar ao agente Bolsa a actualização da evolução das acções pertencentes ao *portfolio* do investidor e suas cotações diárias;
- actualizar o quadro onde são mostradas as cotações em tempo real.

A descrição da classe que implementa a *thread* para actualização das tabelas de cotações, entre outras funções, pode ser consultada no apêndice B.7.

5.2.4.4 *Thread* para cálculo de oportunidades de investimento

A *thread* que faz o cálculo de oportunidades de investimento, funciona por defeito em períodos de 5 em 5 minutos, e tem como principais funções:

- calcular para as acções existentes no *portfolio* do investidor, quais as que, de acordo com os parâmetros definidos se devem vender e para essas enviar ordens de venda ao agente Bolsa;
- calcular para as acções existentes no mercado, quais as que, de acordo com os parâmetros definidos se devem comprar, e para essas enviar ordens de compra ao agente Bolsa.

A descrição da classe que implementa a *thread* para cálculo de oportunidades de investimento pode ser consultada no apêndice B.8.

5.2.5 Algoritmos e fórmulas matemáticas implementadas

5.2.5.1 Algoritmo RSI (*Relative Strenght Index*)

Seguem-se os passos necessários para o cálculo do indicador RSI, que será um dos algoritmos disponíveis para selecção na aplicação e que permite depois identificar oportunidades de investimento.

$$Se\ RSI\ passar\ acima\ dos\ 30 \Rightarrow IRSI^4 = 100 - RSI$$

$$Se\ RSI\ descer\ abaixo\ dos\ 70 \Rightarrow IRSI = (-1) * RSI$$

$$Depois\ IRSI = IRSI * \frac{10}{7}$$

$$Finalmente\ temos\ que\ o\ IRSI = IRSI * \frac{1}{n}$$

Em que n corresponde ao número de dias em que ocorreu qualquer um dos *triggers* acima indicados (ou a passagem acima dos 30, ou descida abaixo dos 70).

[4]Indicador RSI

5.2.5.2 Algoritmo SI (*Stochastic Indicator*)

Seguem-se os passos necessários para o cálculo do indicador SI, que será um dos algoritmos disponíveis para selecção na aplicação e que permite depois identificar oportunidades de investimento.

$$Se\ SI\ passar\ acima\ dos\ 20\%\ pela\ 2^a\ vez \Rightarrow ISI_{(1)} = (100 - SI) * 0.5$$

$$Se\ SI\ passar\ abaixo\ dos\ 80\%\ pela\ 2^a\ vez \Rightarrow ISI_{(1)} = (-1 * SI) * 0.5$$

$$ISI_{(1)} = ISI_{(1)} * \frac{1}{n}$$

Em que n corresponde ao número de dias em que ocorreu qualquer um dos *triggers* acima indicados (ou passar acima dos 20% pela 2^a vez, ou descer abaixo dos 80% pela 2^a vez).

$$Agora,\ considerando\ HistSI^5 = SI(\%K) - TriggerSI(\%D)$$

$$Se\ HistSI\ passar\ para\ valores\ positivos \Rightarrow ISI_{(2)} = (HistSI/MaxHistSI_{(n)}) * 0.5$$

$$Se\ HistSI\ passar\ para\ valores\ negativos \Rightarrow ISI_{(2)} = (HistSI/MinHistSI_{(n)}) * 0.5$$

$$ISI_{(2)} = ISI_{(2)} * \frac{1}{n+1}$$

Em que n corresponde à quantos dias ocorreram, ou o valor mínimo, ou valor máximo, acima indicados.

$$Finalmente\ temos\ que\ o\ ISI^6 = ISI_{(1)} + ISI_{(2)}$$

5.2.5.3 Algoritmo MACD (*Moving Average Convergence / Divergence*)

Seguem-se os passos necessários para o cálculo do indicador MACD, que será um dos algoritmos disponíveis para selecção na aplicação e que permite depois identificar oportunidades de investimento.

Se MACD passar para valores positivos $\Rightarrow$

$$IMACD_{(1)} = (MACD/MaxMACD_{(n)}) * 0.5$$

Se MACD passar para valores negativos $\Rightarrow$

$$IMACD_{(1)} = (MACD/MinMACD_{(n)}) * 0.5$$

$$IMACD_{(1)} = IMACD_{(1)} * \frac{1}{n+1}$$

[5]Histograma SI
[6]Indicador SI

Em que n corresponde ao número de dias em que ocorreu qualquer um dos *triggers* acima indicados (ou passar acima dos 20% pela 2^a vez, ou descer abaixo dos 80% pela 2^a vez).

$$Agora,\ considerando\ HistMACD^7 = MACD - TriggerMACD$$

Se HistMACD passar para valores positivos $\Rightarrow$

$$MACD_{(2)} = (HistMACD/MaxHistMACD_{(n)}) * 0.5$$

Se HistMACD passar para valores negativos $\Rightarrow$

$$MACD_{(2)} = (HistMACD/MinHistMACD_{(n)}) * 0.5$$

$$MACD_{(2)} = MACD_{(2)} * \frac{1}{n+1}$$

Em que n corresponde à quantos dias ocorreram, ou o valor mínimo, ou valor máximo, acima indicados .

$$Finalmente\ temos\ que\ o\ IMACD^8 = MACD_{(1)} + MACD_{(2)}$$

[7]Histograma MACD
[8]Indicador MACD

Capítulo 6

Experimentação

6.1 Introdução

Neste capítulo pretende-se avaliar o desempenho e comportamento, num ambiente de experimentação de uma bolsa virtual, dos três algoritmos fundamentais de análise técnica, implementados na aplicação desenvolvida no âmbito deste trabalho. O período de estudo a que dizem respeito os valores apresentados nas próximas secções, está compreendido entre o dia 31/10/2006 e o dia 29/12/2006, num total de 42 dias úteis de observação.

Nesta simulação foram incluídos três agentes investidores, cada um implementando um algoritmo de cálculo de oportunidades de investimento diferente, nomeadamente o *Relative Strenght Index* (RSI), o *Stochastic Indicator* (SI), e o *Moving Average Convergence / Divergence* (MACD). Cada um inicia com uma carteira de títulos vazia e com um capital inicial de €3000.

6.2 Algoritmo RSI (*Relative Strenght Index*)

De seguida é apresentada a tabela com os resultados do agente baseado no algoritmo RSI, bem como as operações que este efectuou ao longo do período em estudo.

Nessa tabela apresenta-se então, para cada dia, a composição da carteira com identificação do título, quantidade possuída, cotação unitária e valor total; são apresentadas também as operações efectuadas diariamente e a variação percentual da rentabilidade da carteira.

Tabela 6.1: Resultados experimentais do algoritmo RSI

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
31-10-06	Depósito	Capital			3000,00	3000,00	

Tal como se pode verificar na tabela, no período de observação não foi efectuado nenhum investimento pelo agente baseado no algoritmo RSI, o que indica que esse algoritmo será talvez um pouco conservador, dando poucos sinais de compra ao agente investidor.

6.3 Algoritmo SI (*Stochastic Indicator*)

De seguida é apresentada a tabela com os resultados do agente baseado no algoritmo SI, bem como as operações que este efectuou ao longo do período em estudo.

Nessa tabela apresenta-se então, para cada dia, a composição da carteira com identificação do título, quantidade possuída, cotação unitária e valor total; são apresentadas também as operações efectuadas diariamente e a variação percentual da rentabilidade da carteira.

Tabela 6.2: Resultados experimentais do algoritmo SI

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
31-10-06	Depósito	Capital			3000,00	3000,00	
01-11-06		Capital			2014,20		
	Compra	RHT	60	16,43	985,80	3000,00	0,00
02-11-06		Capital			1032,59		
		RHT	60	16,10	966,00		
	Compra	YHOO	37	26,53	981,61	2980,20	-0,66
03-11-06		Capital			2001,25		
		RHT	60	16,67	1000,20		
	Venda	YHOO	37	26,18	(968,66)	3001,45	0,05
06-11-06		Capital			1017,42		
		RHT	60	16,75	1005,00		
	Compra	YHOO	37	26,59	983,83	3006,25	0,21
07-11-06		Capital			2001,99		
		RHT	60	16,90	1014,00		
	Venda	YHOO	37	26,61	(984,57)	3015,99	0,53

Continua na página seguinte...

Tabela 6.2 – Continuação

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
08-11-06		Capital			1006,69		
		RHT	60	17,20	1032,00		
	Compra	YHOO	37	26,90	995,30	3033,99	1,13
09-11-06		Capital			1006,69		
		RHT	60	17,16	1029,60		
		YHOO	37	27,45	1015,65	3051,94	1,73
10-11-06		Capital			1006,69		
		RHT	60	17,85	1071,00		
		YHOO	37	27,39	1013,43	3091,12	3,01
13-11-06		Capital			19,69		
	Compra	INTC	47	21,00	987,00		
		RHT	60	17,53	1051,80		
		YHOO	37	27,40	1013,80	3072,29	2,41
14-11-06		Capital			1027,57		
		INTC	47	21,88	1028,36		
		RHT	60	16,88	1012,80		
	Venda	YHOO	37	27,24	(1007,88)	3068,73	2,29
15-11-06		Capital			1027,57		
		INTC	47	22,32	1049,04		
		RHT	60	16,93	1015,80	3092,41	3,08
16-11-06		Capital			2034,97		
		INTC	47	22,33	1049,51		
	Venda	RHT	60	16,79	(1007,40)	3084,48	2,82
17-11-06		Capital			2034,97		
		INTC	47	22,10	1038,70	3073,67	2,46
20-11-06		Capital			3081,66		
	Venda	INTC	47	22,27	(1046,69)	3081,66	2,72
21-11-06		Capital			2104,62		
	Compra	YHOO	36	27,14	977,04	3081,66	2,72
22-11-06		Capital			1119,58		
	Compra	RHT	56	17,59	985,04		
		YHOO	36	28,49	1025,64	3130,26	4,34
24-11-06		Capital			1119,58		
		RHT	56	17,48	978,88		
		YHOO	36	28,03	1009,08	3107,54	3,58
27-11-06		Capital			3052,18		
	Venda	RHT	56	16,98	(950,88)		
	Venda	YHOO	36	27,27	(981,72)	3052,18	1,74
28-11-06		Capital			3052,18	3052,18	1,74
29-11-06		Capital			3052,18	3052,18	1,74

Continua na página seguinte...

Tabela 6.2 – Continuação

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
30-11-06		Capital			1075,98		
	Compra	INTC	46	21,40	984,40		
	Compra	RHT	57	17,40	991,80	3052,18	1,74
01-12-06		Capital			2038,76		
	Venda	INTC	46	20,93	(962,78)		
		RHT	57	17,18	979,26	3018,02	0,60
04-12-06		Capital			2038,76		
		RHT	57	17,13	976,41	3015,17	0,51
05-12-06		Capital			3010,04		
	Venda	RHT	57	17,04	971,28	3010,04	0,33
06-12-06		Capital			3010,04	3010,04	0,33
07-12-06		Capital			3010,04	3010,04	0,33
08-12-06		Capital			3010,04	3010,04	0,33
11-12-06		Capital			3010,04	3010,04	0,33
12-12-06		Capital			1028,60		
	Compra	INTC	48	20,73	995,04		
	Compra	RHT	60	16,44	986,40	3010,04	0,33
13-12-06		Capital			1028,60		
		INTC	48	20,70	993,60		
		RHT	60	16,58	994,80	3017,00	0,57
14-12-06		Capital			1028,60		
		INTC	48	20,77	996,96		
		RHT	60	17,30	1038,00	3063,56	2,12
15-12-06		Capital			1028,60		
		INTC	48	20,96	1006,08		
		RHT	60	17,62	1057,20	3091,88	3,06
18-12-06		Capital			1028,60		
		INTC	48	20,84	1000,32		
		RHT	60	17,94	1076,40	3105,32	3,51
19-12-06		Capital			1028,60		
		INTC	48	20,66	991,68		
		RHT	60	18,37	1102,20	3122,48	4,08
20-12-06		Capital			3141,80		
	Venda	INTC	48	20,60	(988,80)		
	Venda	RHT	60	18,74	(1124,40)	3141,80	4,73
21-12-06		Capital			3141,80	3141,80	4,73
22-12-06		Capital			3141,80	3141,80	4,73
26-12-06		Capital			2161,83		
	Compra	RHT	43	22,79	979,97	3141,80	1,73
27-12-06		Capital			1162,23		

Continua na página seguinte...

Tabela 6.2 – Continuação

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
	Compra	INTC	49	20,40	999,60		
		RHT	43	22,70	976,10	3137,93	1.60
28-12-06		Capital			1162,23		
		INTC	49	20,42	1000,58		
		RHT	43	22,96	987,28	3150,09	5.00
29-12-06		Capital			1162,23		
		INTC	49	20,25	992,25		
		RHT	43	23,00	989,00	3143,48	1.78

Tal como se pode verificar na tabela, no período de observação, o algoritmo SI originou bastantes movimentos, dando indicações quer de compra, quer de venda, podendo-se considerar bastante volátil neste aspecto. Mesmo assim conseguiu manter a rentabilidade da carteira, sempre em terreno positivo, tendo na recta final atingido valores próximos dos 5%. Recomenda-se, como um bom indicador, para um investidor que queira apostar a curto prazo.

6.4 Algoritmo MACD (*Moving Average Convergence / Divergence*)

De seguida é apresentada a tabela com os resultados do agente baseado no algoritmo MACD, bem como as operações que este efectuou ao longo do período em estudo.

Nessa tabela apresenta-se então, para cada dia, a composição da carteira com identificação do título, quantidade possuída, cotação unitária e valor total; são apresentadas também as operações efectuadas diariamente e a variação percentual da rentabilidade da carteira.

Tabela 6.3: Resultados experimentais do algoritmo MACD

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
31-10-06	Depósito	Capital			3000,00	3000,00	
01-11-06		Capital			3000,00	3000,00	0,00
02-11-06		Capital			1026,39		
	Compra	DELL	40	24,80	992,00		

Continua na página seguinte...

Tabela 6.3 – Continuação

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
	Compra	YHOO	37	26,53	981,61	3000,00	0,00
03-11-06		Capital			1026,39		
		DELL	40	24,24	969,60		
		YHOO	37	26,18	968,66	2964,65	-1,18
06-11-06		Capital			1026,39		
		DELL	40	24,47	978,80		
		YHOO	37	26,59	983,83	2989,02	-0,37
07-11-06		Capital			1026,39		
		DELL	40	24,56	982,40		
		YHOO	37	26,61	984,57	2993,36	-0,22
08-11-06		Capital			47,49		
	Compra	CSCO	39	25,10	978,90		
		DELL	40	24,18	967,20		
		YHOO	37	26,90	995,30	2988,89	0,37
09-11-06		Capital			47,49		
		CSCO	39	26,71	1041,69		
		DELL	40	24,98	999,20		
		YHOO	37	27,45	1015,65	3104,03	3,47
10-11-06		Capital			47,49		
		CSCO	39	26,74	1042,86		
		DELL	40	24,89	995,60		
		YHOO	37	27,39	1013,43	3099,38	3,31
13-11-06		Capital			47,49		
		CSCO	39	26,68	1040,52		
		DELL	40	25,49	1019,60		
		YHOO	37	27,40	1013,80	3121,41	4,05
14-11-06		Capital			47,49		
		CSCO	39	26,64	1038,96		
		DELL	40	25,59	1023,60		
		YHOO	37	27,24	1007,88	3117,93	3,93
15-11-06		Capital			47,49		
		CSCO	39	26,60	1037,40		
		DELL	40	25,75	1030,00		
		YHOO	37	27,15	1004,55	3119,44	3,98
16-11-06		Capital			47,49		
		CSCO	39	27,15	1058,85		
		DELL	40	25,10	1004,00		
		YHOO	37	26,64	985,68	3096,02	3,20
17-11-06		Capital			47,49		
		CSCO	39	26,93	1050,27		

Continua na página seguinte...

Tabela 6.3 – Continuação

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
		DELL	40	25,02	1000,80		
		YHOO	37	26,91	995,67	3094,23	3,14
20-11-06		Capital			53,61		
		CSCO	39	27,11	1057,29		
	Venda	DELL	40	24,65	(986,00)		
	Compra	INTC	44	22,27	979,88		
		YHOO	37	26,72	988,64	3079,42	2,65
21-11-06		Capital			53,61		
		CSCO	39	26,80	1045,20		
		INTC	44	21,57	949,08		
		YHOO	37	27,14	1004,18	3052,07	1,74
22-11-06		Capital			53,61		
		CSCO	39	26,91	1049,49		
		INTC	44	21,73	956,12		
		YHOO	37	28,49	1054,13	3113,35	3,78
24-11-06		Capital			53,61		
		CSCO	39	26,84	1046,76		
		INTC	44	21,59	949,96		
		YHOO	37	28,03	1037,11	3078,44	2,91
27-11-06		Capital			1015,85		
	Venda	CSCO	39	25,80	(1006,20)		
	Venda	INTC	44	21,10	(928,40)		
	Compra	DELL	36	27,01	972,36		
		YHOO	37	27,27	1008,99	2997,20	-0,09
28-11-06		Capital			1015,85		
		DELL	36	26,95	970,20		
		YHOO	37	27,00	999,00	2985,05	-0,50
29-11-06		Capital			1015,85		
		DELL	36	27,62	994,32		
		YHOO	37	27,04	1000,48	3010,65	0,36
30-11-06		Capital			2015,22		
		DELL	36	27,24	980,64		
	Venda	YHOO	37	27,01	(999,37)	2995,86	-0,14
01-12-06		Capital			2015,22		
		DELL	36	27,27	981,72	2996,94	-0,10
04-12-06		Capital			2015,22		
		DELL	36	26,84	966,24	2981,46	-0,62
05-12-06		Capital			2015,22		
		DELL	36	26,90	968,40	2983,62	-0,55
06-12-06		Capital			2004,50		

Continua na página seguinte...

Tabela 6.3 – Continuação

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
	Venda	DELL	36	26,58	(956,88)		
	Compra	KO	20	48,38	967,60	2972,10	0,93
07-12-06		Capital			2004,50		
		KO	20	48,72	974,40	2978,90	-0,70
08-12-06		Capital			2004,50		
		KO	20	48,91	978,20	2982,70	-0,58
11-12-06		Capital			2004,50		
		KO	20	48,81	976,20	2980,70	-0,64
12-12-06		Capital			2004,50		
		KO	20	48,90	978,00	2982,50	-0,58
13-12-06		Capital			2004,50		
		KO	20	48,84	976,80	2981,30	-0,62
14-12-06		Capital			1050,90		
	Compra	IBM	10	95,36	953,60		
		KO	20	49,00	980,00	2984,50	-0,52
15-12-06		Capital			1050,90		
		IBM	10	95,30	953,00		
		KO	20	48,93	978,60	2982,50	-0,58
18-12-06		Capital			1050,90		
		IBM	10	95,44	954,40		
		KO	20	48,90	978,00	2983,30	-0,56
19-12-06		Capital			1050,90		
		IBM	10	96,00	960,00		
		KO	20	48,77	975,40	2986,30	-0,46
20-12-06		Capital			57,68		
		IBM	10	96,00	960,00		
		KO	20	48,75	975,00		
	Compra	RHT	53	18,74	993,22	2985,90	-0,47
21-12-06		Capital			1030,88		
		IBM	10	95,91	959,10		
	Venda	KO	20	48,66	(973,20)		
		RHT	53	17,96	951,88	2941,86	-1,94
22-12-06		Capital			1030,88		
		IBM	10	95,25	952,50		
		RHT	53	22,46	1190,38	3173,76	5,79
26-12-06		Capital			1987,48		
	Venda	IBM	10	95,66	(956,60)		
		RHT	53	22,79	1207,87	3195,35	6,51
27-12-06		Capital			1015,48		
	Compra	IBM	10	97,20	972,00		

Continua na página seguinte...

Tabela 6.3 – Continuação

Data	Composição da carteira					Total da carteira	Var. %
	Operação	Título	Qtd.	Valor	Total		
		RHT	53	22,70	1203,10	3190,58	6,35
28-12-06		Capital			1015,48		
		IBM	10	96,67	969,70		
		RHT	53	22,96	1216,88	3202,06	6,74
29-12-06		Capital			1015,48		
		IBM	10	97,15	971,50		
		RHT	53	23,00	1219,00	3205,98	6,87

Tal como se pode verificar na tabela, no período de observação, o algoritmo MACD não originou muitos movimentos, quer de compra, quer de venda, podendo-se considerar um pouco moderado neste aspecto. A sua *performance* ao longo do período em estudo variou bastante, tendo a rentabilidade da carteira oscilado entre valores positivos, depois negativos, tendo finalmente recuperado na recta final para valores positivos próximos dos 7%. Recomenda-se, como um bom indicador, para um investidor que queira apostar a médio/longo prazo.

6.5 Conclusão e resultados

No gráfico da Figura 6.1 apresentada mais adiante compara-se a evolução do valor da carteira, entre os agentes que usaram os algoritmos SI e MACD. Dado que não houve variação no valor da carteira do agente RSI, em virtude deste não ter realizado nenhum investimento tal como se pode observar pelos valores registados na fase de experimentação, ele não será tido em conta.

Da leitura do gráfico podemos concluir que no final do período de observação, o algoritmo MACD obteve melhores prestações face ao SI, se bem que nem sempre foi assim, e só a sua recuperação final permitiu esse resultado. De acordo com esta observação conclui-se então que o SI será um bom indicador para o curto prazo, enquanto que o MACD estará mais virado para o médio/longo prazo.

De notar também que ambos os algoritmos obtiveram resultados positivos, concluindo-se que a implementação dos algoritmos de cálculo de oportunidades de investimento foi bem conseguida.

Também teria sido interessante poder comparar os resultados obtidos nesta fase com os de outras aplicações existentes no mercado, mas como já foi referido num capítulo anterior, tal não foi possível, limitando-se os testes e comparações de resultados, aos obtidos pelas implementações dos diferentes algoritmos.

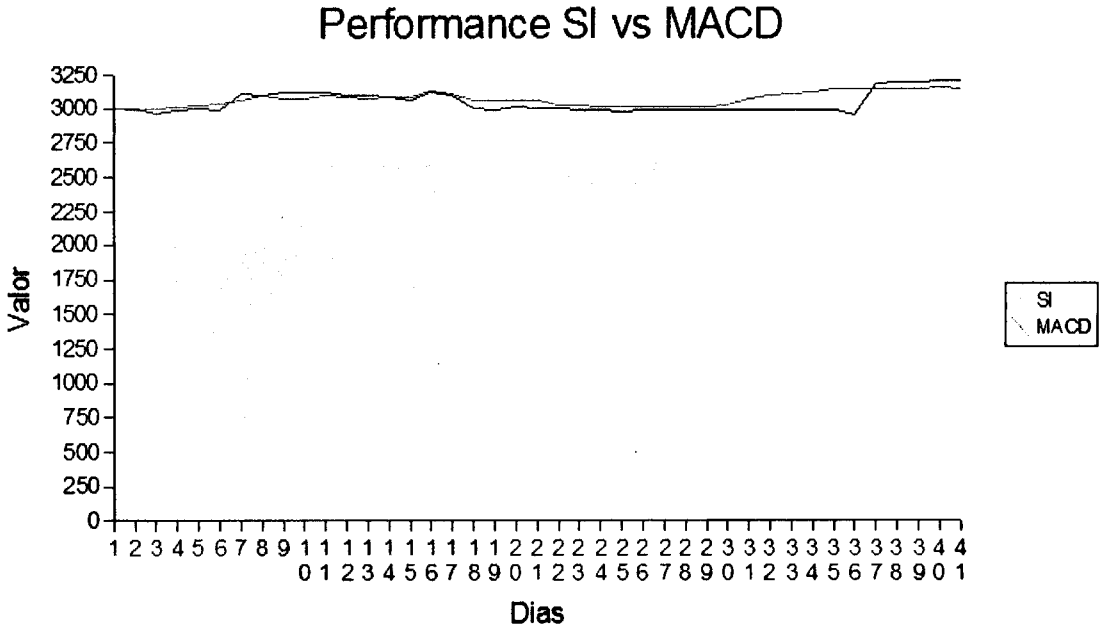


Figura 6.1: Performance SI vs MACD

Capítulo 7

Conclusões

Chegado ao final desta longa caminhada, como assim já me referi a esta dissertação, resta então fazer uma avaliação dos objectivos atingidos e também especular um pouco sobre quais as possibilidades de evolução de um projecto desta natureza. Sendo da natureza humana nunca estar satisfeita, e buscar e querer sempre mais, eu não sou excepção, mas no entanto julgo que os objectivos principais foram atingidos, nomeadamente no que diz respeito à aplicação dos conceitos e teorias dos Sistemas Multi-Agentes, fazendo uso de uma infraestrutura de desenvolvimento bastante conhecida a nível académico nesta área, a já referida *JATLite*.

A elaboração deste trabalho deu-me também a possibilidade de me aperceber de qual a potencialidade da aplicação destas matérias de Inteligência Artificial em contextos e problemas bem reais, e que muitas vezes os resultados obtidos são, muitas das vezes, largamente satisfatórios. Neste caso em concreto, e no que diz respeito ao problema em estudo, verifica-se que este é um exemplo de uma das áreas em que um Sistema Multi-Agente devidamente projectado e com capacidades cognitivas e, quem sabe, características emocionais, poderia ter imenso sucesso no seio de uma comunidade de investidores, que operem num mercado real e não virtual.

A minha intenção era precisamente dotar essa comunidade de uma ferramenta desse género, que pudesse auxiliá-los na tomada de decisões de investimento, se possível na posse de dados reais e actuais, que depois de processados lhes dariam os tão desejados indicadores económicos. Nesse sentido, considero atingidos os objectivos que concernem à obtenção de cotações reais a partir de uma fonte na Internet[1] e à aplicação de algoritmos e técnicas modernas de análise técnica para avaliação de acções.

Mas muito mais poderia ser feito no sentido de aproximar esta aplicação de um ambiente real de uma Bolsa de Valores, fornecendo-lhe não só as cotações das acções, mas também uma série de outras informações que são veiculadas por diversas fontes e que não dizem respeito directamente às cotações mas que influenciam o seu compor-

[1] http://finance.yahoo.com/

tamento.

Falo nomeadamente da análise sintáctica de diversas notícias financeiras que a cada segundo são disponibilizadas na Internet, e da busca no seu conteúdo de palavras chave que em determinados contextos poderiam dar indicações precisas de oportunidades de investimento. Essas notícias e as palavras chave encontradas poderiam fazer prever tendências de mercado, influenciado as estratégias de investimento dos agentes.

Falo por exemplo da inclusão de outros indicadores económicos, onde além das estratégias de investimento com base na análise técnica já referidas, fossem também incluídos indicadores da análise fundamental. Poderiam ser tidos em conta diversos factores envolventes dessa análise fundamental tais como: o estado da economia, o estado do sector de actividade a que pertence a empresa, análise dos rácios das empresas, etc. Seria de facto interessante ter em conta factores económicos como por exemplo a inflação, taxas de juro, o preço do petróleo, etc. Seria então, também importante, estudar quais os sectores de actividade mais promissores, influenciando assim a matriz de decisão dos agentes. Por último, e analisando os rácios das empresas, poder identificar se determinado investimento, é um investimento sólido e com futuro.

Seria também interessante ter em conta determinados factores sociais, existindo para isso uma espécie de indicador do "sentimento"da sociedade, indicador esse que seria influenciado por factores, como por exemplo, flutuações de preços (neste caso não como factor económico, mas como um factor que pode influenciar o modo de vida da sociedade, nomeadamente a sua apetência para o consumo), guerras, ataques terroristas, resultados desportivos, etc.

Outro aspecto de que seria importante poder dotar os agentes, seria de uma certa capacidade de "auto-aprendizagem", em que estes por exemplo, pudessem testar variações de estratégias de investimento sobre situações passadas, calculando qual seriam os seus resultados no presente ou num futuro próximo, identificando assim se as decisões que tomaram no passado foram as mais correctas, no fundo, aprendendo com os seus próprios "erros".

Para terminar, e no que diz respeito à infraestrutura de desenvolvimento usada, importa referir que se este projecto se estivesse a iniciar agora, a minha escolha iria para o *JADE*[2] (*Java Agent Development Framework*), que é também uma plataforma para desenvolvimento de Sistemas Multi-Agentes, cujo projecto tem evoluído bastante, mantendo-se bastante activo e com constantes actualizações, ao contrário do *JATLite* que ficou parado no tempo. O *JADE* cumpre na totalidade as especificações da *FIPA*[3] (*Foundation for Intelligent Physical Agents*) e possui uma série de ferramentas gráficas que suportam as fases de desenvolvimento e depuração. A migração de plataforma seria portanto, outra das possíveis e lógicas evoluções.

Visto que ainda existem muitas ideias que ficaram no ar e objectivos por atingir, será este mesmo o fim da caminhada, ou apenas uma pausa no percurso?

[2]http://jade.tilab.com/
[3]http://www.fipa.org/

Apêndice A

Código SQL para criação das tabelas da BD

```
CREATE TABLE Market (
   codMkt VARCHAR(8),
   descr VARCHAR(25) NOT NULL,
   openHour TIME,
   closeHour TIME,
   CONSTRAINT Market_codMkt_PK PRIMARY KEY (codMkt),
   CONSTRAINT Market_descr_UN UNIQUE (descr)
);

CREATE TABLE Stock (
   symb VARCHAR(8),
   descr VARCHAR(25) NOT NULL,
   codMkt VARCHAR(8) NOT NULL,
   weekLow NUMERIC(6,2) DEFAULT 0,
   weekHigh NUMERIC(6,2) DEFAULT 0,
   rsi NUMERIC(6,2) DEFAULT 0,
   si NUMERIC(6,2) DEFAULT 0,
   macd NUMERIC(6,2) DEFAULT 0,
   CONSTRAINT Stock_symb_PK PRIMARY KEY (symb),
   CONSTRAINT Stock_descr_UN UNIQUE (descr),
   CONSTRAINT Stock_codMkt_FK FOREIGN KEY (codMkt)
      REFERENCES Market (codMkt)
      ON UPDATE CASCADE
);

CREATE TABLE Intraday (
   symb VARCHAR(8),
   dthr TIMESTAMP,
```

```
    volume INTEGER DEFAULT 0,
    value NUMERIC(6,2) DEFAULT 0,
    CONSTRAINT Intraday_SymbDthr_PK PRIMARY KEY (symb, dthr),
    CONSTRAINT Intraday_symb_FK FOREIGN KEY (symb)
        REFERENCES Stock (symb)
        ON UPDATE CASCADE
);

CREATE TABLE Daily (
    symb VARCHAR(8),
    date DATE,
    volume INTEGER DEFAULT 0,
    openValue NUMERIC(6,2) DEFAULT 0,
    minValue NUMERIC(6,2) DEFAULT 0,
    maxValue NUMERIC(6,2) DEFAULT 0,
    closeValue NUMERIC(6,2) DEFAULT 0,
    CONSTRAINT Daily_SymbDate_PK PRIMARY KEY (symb, date),
    CONSTRAINT Daily_symb_FK FOREIGN KEY (symb)
        REFERENCES Stock (symb)
        ON UPDATE CASCADE
);

CREATE TABLE Indicator (
    symb VARCHAR(8),
    date DATE,
    rsiValue NUMERIC(13,10) DEFAULT 0,
    siValue NUMERIC(13,10) DEFAULT 0,
    siSignal NUMERIC(13,10) DEFAULT 0,
    siHist NUMERIC(13,10) DEFAULT 0,
    macdValue NUMERIC(13,10) DEFAULT 0,
    macdSignal NUMERIC(13,10) DEFAULT 0,
    macdHist NUMERIC(13,10) DEFAULT 0,
    CONSTRAINT Indicators_SymbDate_PK PRIMARY KEY (symb, date),
    CONSTRAINT Indicators_symb_FK FOREIGN KEY (symb)
        REFERENCES Stock (symb)
        ON UPDATE CASCADE
);

CREATE TABLE Trader (
    codt VARCHAR(25),
    passwd VARCHAR(8) NOT NULL,
    balance NUMERIC(8,2) DEFAULT 0,
    CONSTRAINT Trader_codt_PK PRIMARY KEY (codt)
);
```

```
CREATE TABLE Portfolio (
   codt VARCHAR(25),
   symb VARCHAR(8),
   quant INTEGER DEFAULT 0,
   buyprice NUMERIC(6,2) DEFAULT 0,
   CONSTRAINT Portfolio_CodtSymb_PK PRIMARY KEY (codt, symb),
   CONSTRAINT Portfolio_codt_FK FOREIGN KEY (codt)
      REFERENCES Trader (codt)
      ON UPDATE CASCADE,
   CONSTRAINT Portfolio_symb_FK FOREIGN KEY (symb)
      REFERENCES Stock (symb)
      ON UPDATE CASCADE
);
```

Apêndice B

Descrição das classes

B.1 Class Bolsa

Main class for project AIStockEx Builds GUI and launches Threads

B.1.1 Declaration

public class Bolsa **extends** javax.swing.JFrame

B.1.2 Serializable Fields

- private Connection db

  –

- private Statement stStk

  –

- private Statement stMkt

  –

- private ResultSet rsStk

–

- private ResultSet rsMkt

  –

- private ResultSet rsDaily

  –

- private ResultSet rsIntra

  –

- private PreparedStatement sqlDaily

  –

- private PreparedStatement sqlIntra

  –

- private Vector stk

  –

- private Vector mkt

  –

- private ThreadBolsaAG threadBolsaAG

  –

- private ThreadBolsaDB threadBolsaDB

  –

- private ThreadCleanDB threadCleanDB

  –

- private boolean isDrawing

  –

B.1.3 CONSTRUCTORS

- *Bolsa*
  `protected` **Bolsa**`( )`

  - **Usage**
    * Creates new form Bolsa

B.1.4 METHODS

- *addRcv*
  `protected void` **addRcv**`( java.lang.String` **msg** `)`
  - **Usage**
    - Adds message received from agent to the text area
  - **Parameters**
    - `msg` - String, representing received message

- *addSent*
  `protected void` **addSent**`( java.lang.String` **msg** `)`
  - **Usage**
    - Adds message sent to agent to the text area
  - **Parameters**
    - `msg` - String, representing sent message

- *centerParent*
  `protected void` **centerParent**`( java.awt.Component` **parent**`, java.awt.Component` **child** `)`
  - **Usage**
    - Centers child component relative to parent component on screen
  - **Parameters**
    - `parent` - Parent component
    - `child` - Child component

- *changeAspect*
  `protected void` **changeAspect**`( java.awt.Component` **component** `)`
  - **Usage**
    - Changes application aspect to that of the operating system it's run on
  - **Parameters**
    - `component` - Component to change aspect

- *changeMenuAG*
  `protected void` **changeMenuAG**`( boolean` **enable** `)`
  - **Usage**
    - Enables/Disables menu that controls Thread AG
  - **Parameters**
    - `enable` - Boolean, true to enable menu, false to disable

- *changeMenuDB*
  `protected void` **changeMenuDB(** `boolean` **enable)**
  - **Usage**
    - Enables/Disables menu that controls Thread DB
  - **Parameters**
    - `enable` - Boolean, true to enable menu, false to disable

- *clearMnuMkt*
  `protected void` **clearMnuMkt( )**
  - **Usage**
    - Clears all fields on market menu

- *clearMnuStk*
  `protected void` **clearMnuStk(** `boolean` **cleanLst)**
  - **Usage**
    - Clears all fields on stock menu
  - **Parameters**
    - `cleanLst` - Boolean, true if it's to clean search results list

---

- *createPrice*
  `protected XYDataset` **createPrice( )**
  - **Usage**
    - Creates Price XYDataset for drawing graphics
  - **Returns** - XYDataset for drawing graphics

- *createTblStk*
  `protected void` **createTblStk(** `java.util.Vector` **data)**
  - **Usage**
    - Creates stock table
  - **Parameters**
    - `data` - Vector, representing stock data

- *createValue*
  `protected XYDataset` **createValue(** `java.lang.String` **type)**
  - **Usage**
    - Creates XYDataset for drawing graphics
  - **Parameters**
    - `type` - String, representing the type of value to create

- **Returns** - XYDataset for drawing graphics

- *createValues*
  `protected OHLCDataset` **createValues**`( )`
  - **Usage**
    - Creates OHLCDataset for drawing graphics
  - **Returns** - OHLCDataset for drawing graphics

- *createVolume*
  `protected XYDataset` **createVolume**`( )`
  - **Usage**
    - Creates Volume XYDataset for drawing graphics
  - **Returns** - XYDataset for drawing graphics

- *drawGraph*
  `protected void` **drawGraph**`( )`
  - **Usage**
    - Draws Graph according to selected parameters

- *exitApp*
  `protected void` **exitApp**`( )`
  - **Usage**
    - Stops all threads and closes all ResultSets, Statements and PreparedStatements

---

- *getMktHour*
  `protected String` **getMktHour**`( java.lang.String` **type** `)`
  - **Usage**
    - Gets the open/close hour for the market
  - **Parameters**
    - `type` - String representing the type of hour, open or close
  - **Returns** - String, representing the open/close hour for the market

---

- *getPctChg*
  `protected String` **getPctChg**`( double` **openvalue**`, double` **closevalue** `)`
  - **Usage**
    - Returns the formatted stock percentage variation
  - **Parameters**

    * `openvalue` - double, representing the stock open value
    * `closevalue` - double, representing the stock close value
  - **Returns** - String, representing the formatted stock percentage variation

---

- *getPctInd*
  `protected String` **getPctInd**`( double` **rsi**`, double` **si**`, double` **macd** `)`
  - **Usage**
    * Returns the formatted stock indicator percentage
  - **Parameters**
    * `rsi` - double, representing the stock rsi indicator
    * `si` - double, representing the stock si indicator
    * `macd` - double, representing the stock macd indicator
  - **Returns** - String, representing the formatted stock indicator percentage

---

- *isNegChg*
  `protected String` **isNegChg**`( int` **rowindex**`, int` **colindex** `)`
  - **Usage**
    * Returns a string representing a stock variation or indicator sign
  - **Parameters**
    * `rowindex` - Int, representing the selected row of the stock table
    * `colindex` - Int, representing the selected column of the stock table
  - **Returns** - String, representing a stock variation or indicator sign

---

- *main*
  `public static void` **main**`( java.lang.String [] args )`
  - **Usage**
    * Main class to launch JFrame Bolsa
  - **Parameters**
    * `args` - the command line arguments

- *readResultSet*
  `protected void` **readResultSet**`( )`
  - **Usage**
    * Reads stock ResultSet

- *readResultSetM*
  `protected void` **readResultSetM**`( )`
  - **Usage**
    * Reads market ResultSet

- *refreshCmbStk*
  `protected void` **refreshCmbStk**`( )`
  - **Usage**
    - Refreshes information in stocks combo-box

- *refreshDaily*
  `protected void` **refreshDaily**`( )`
  - **Usage**
    - Refreshes information in daily table

- *refreshIntraday*
  `protected void` **refreshIntraday**`( )`
  - **Usage**
    - Refreshes information in intraday table

- *refreshTblStk*
  `protected void` **refreshTblStk**`( )`
  - **Usage**
    - Refreshes information in stocks table

- *selResult*
  `protected void` **selResult**`( )`
  - **Usage**
    - Tries to find select symbol on the stock RecordSet

- *setStatus*
  `protected void` **setStatus**`( java.lang.String` **msg** `)`
  - **Usage**
    - Sets status line to received message
  - **Parameters**
    - `msg` - String, representing message

- *showMsg*
  `protected void` **showMsg**`( java.lang.String` **msg**`, java.lang.String` **title**`, int` **type** `)`
  - **Usage**
    - Shows JOptionPane with received message to the user
  - **Parameters**

  * `msg` - String, representing message
  * `title` - String, representing title
  * `type` - Int, representing type

- *startAll*
  `protected void startAll( )`
  - **Usage**
    * Verifies if connection to database is available and starts all threads on startup

B.2 CLASS ThreadBolsaAG

Thread who sends and receives messages from the other JATLite Agents

B.2.1 DECLARATION

public class ThreadBolsaAG **extends** RouterLayer.AgentClient.RouterClientAction

B.2.2 CONSTRUCTORS

- *ThreadBolsaAG*
  `public` **ThreadBolsaAG**`( Bolsa` **application**`, java.sql.Connection` **db** `)`
  - **Usage**
    * Creates a new instance of ThreadBolsaAG
  - **Parameters**
    * `application` - Calling parent JFrame
    * `db` - Database Connection provided by parent

B.2.3 METHODS

- *Act*
  `public boolean` **Act(** `java.lang.Object` **obj)**
  - **Usage**
    - JATLite extended method to process received messages
  - **Parameters**
    - `obj` - KQMLmail message received from other agents
  - **Returns** - Boolean, true if message correctly processed, false otherwise

---

- *actBalance*
  `protected boolean` **actBalance(** `java.lang.String` **trader,** `double` **amount)**
  - **Usage**
    - Adds to the current balance of the received trader, the received amount
  - **Parameters**
    - `trader` - String, representing the trader
    - `amount` - Double, representing the amount
  - **Returns** - Boolean, true if current balance set, false otherwise

---

- *actPortfolio*
  `protected boolean` **actPortfolio(** `java.lang.String` **action,** `java.lang.String` **trader,** `java.lang.String` **symb,** `int` **quant,** `double` **price)**
  - **Usage**
    - Adds/Deletes stocks from the received trader portfolio
  - **Parameters**
    - `action` - String, representing the action: buy or sell
    - `trader` - String, representing the trader
    - `symb` - String, representing the stock
    - `quant` - Integer, representing the quantity
    - `price` - Double, representing the price
  - **Returns** - Boolean, true if action suceeded, false otherwise

---

- *getBalance*
  `protected double` **getBalance(** `java.lang.String` **trader)**
  - **Usage**
    - Gets the current balance of the received trader
  - **Parameters**
    - `trader` - String, representing the trader
  - **Returns** - Double, representing the current balance

- *getPctChg*
  `protected String` **getPctChg(** `java.lang.Double` **openvalue,** `java.lang.Double` **closevalue)**
  - **Usage**
    - Returns the formatted stock percentage variation
  - **Parameters**
    - `openvalue` - Double, representing the stock open value
    - `closevalue` - Double, representing the stock close value
  - **Returns** - String, representing the formatted stock percentage variation

---

- *getSleep*
  `protected long` **getSleep( )**
  - **Usage**
    - Returns ThreadBolsaAG sleep time in milliseconds
  - **Returns** - Sleep time in milliseconds

- *getStop*
  `protected boolean` **getStop( )**
  - **Usage**
    - Gets stop state of ThreadBolsaAG
  - **Returns** - Boolean, true if thread is stopped, false otherwise

---

- *getValue*
  `protected String` **getValue(** `java.lang.String` **symb)**
  - **Usage**
    - Returns current value of received stock
  - **Parameters**
    - `symb` - String of the stock symbol
  - **Returns** - String, representing current value of received stock

---

- *getWait*
  `protected boolean` **getWait( )**
  - **Usage**
    - Gets wait state of ThreadBolsaAG
  - **Returns** - Boolean, true if thread is waiting, false otherwise

---

- *isMktClosed*
  `protected Boolean` **isMktClosed(** `java.lang.String` **symb)**
  - **Usage**

    * Returns if market is already closed
  - **Parameters**
    * `symb` - String of the stock symbol
  - **Returns** - Boolean, true if market closed, false otherwise

---

- *processMessage*
  `public void` **processMessage**`( java.lang.String` **msg**`, java.lang.Object` **obj** `)`
  - **Usage**
    * JATLite extended method to process received messages (not used)
  - **Parameters**
    * `msg` - String message received from other agents
    * `obj` - KQMLmail message received from other agents

- *rcvAsk*
  `protected void` **rcvAsk**`( KQMLLayer.KQMLmessage` **kqml** `)`
  - **Usage**
    * Processes received Ask messages
  - **Parameters**
    * `kqml` - Ask KQMLmessage received from other agents

- *rcvTell*
  `protected void` **rcvTell**`( KQMLLayer.KQMLmessage` **kqml** `)`
  - **Usage**
    * Processes received Tell messages
  - **Parameters**
    * `kqml` - Tell KQMLmessage received from other agents

- *run*
  `public void` **run**`( )`
  - **Usage**
    * Run method of ThreadBolsaAG

- *sendMsg*
  `protected void` **sendMsg**`( java.lang.String` **sender**`, java.lang.String` **receiver**`, java.lang.String` **perform**`, java.lang.String` **msg** `)`
  - **Usage**
    * Sends KQMLmessage to specified receiver
  - **Parameters**

    * `sender` - String, representing sender agent
    * `receiver` - String, representing receiver agent
    * `perform` - String, representing performative
    * `msg` - String, representing text message

- *sendPortfolio*
  `protected void` **sendPortfolio(** `java.lang.String` **trader)**
  - **Usage**
    * Sends KQML messages to inform of the trader's portfolio
  - **Parameters**
    * `trader` - String, representing the trader

- *setSleep*
  `protected void` **setSleep(** `long` **sleep)**
  - **Usage**
    * Sets sleep time of ThreadBolsaAG in milliseconds
  - **Parameters**
    * `sleep` - Sleep time in milliseconds

- *setStop*
  `protected void` **setStop( )**
  - **Usage**
    * Sets stop state of ThreadBolsaAG

- *setWait*
  `protected void` **setWait( )**
  - **Usage**
    * Sets state of ThreadBolsaAG to wait

- *start*
  `public void` **start( )**
  - **Usage**
    * Start method of ThreadBolsaAG

- *stopAG*
  `protected void` **stopAG( )**
  - **Usage**
    * Stops TreadBolsaAG, closes Statements, PreparedStatements and RecordSets that might be open, and disconnects/unregisters JATLite Agent

B.3 CLASS ThreadBolsaDB

Thread who connects to financial source in internet to get stock quotes, and writes them to the database; is also responsible for other database functions, such as calculating the 52 week high and low of stocks

B.3.1 DECLARATION

public class ThreadBolsaDB

extends java.lang.Thread

B.3.2 CONSTRUCTORS

- *ThreadBolsaDB*
  `public` **ThreadBolsaDB**`( Bolsa` **application**`, java.sql.Connection` **db** `)`
  - **Usage**
    - Creates a new instance of ThreadBolsaDB
  - **Parameters**
    - `application` - Calling parent JFrame
    - `db` - Database Connection provided by parent

B.3.3 METHODS

- *act52Week*
  `protected void` **act52Week**`( java.lang.String` **symbol** `)`
  - **Usage**
    - Calculates 52 week high and low of received stock symbol parameter
  - **Parameters**
    - `symbol` - String of the stock symbol

- *actIndicators*
  `protected void` **actIndicators**`( java.lang.String` **symbol** `)`
  - **Usage**
    - Calculates the technical analysis indicators of received stock symbol parameter
  - **Parameters**
    - `symbol` - String of the stock symbol

- *calcEma*
  `protected Vector` **calcEma**`( java.util.Vector` **vector**`, float` **period** `)`
  - **Usage**
    - Calculates the exponential moving average of a received vector of values for a received period of days
  - **Parameters**
    - `vector` - Vector<String>representing the values on which to calculate the EMA
    - `period` - Float representing the period to calculate the EMA
  - **Returns** - Vector<String>representing the calculated values of the EMA

- *calcHist*
  `protected Vector` **calcHist**`( java.util.Vector` **indicator**`, java.util.Vector` **signal** `)`
  - **Usage**
    - Calculates the histogram values representing the diferences between some indicator and it's signal
  - **Parameters**
    - `indicator` - Vector<String>representing the values of the indicator
    - `signal` - Vector<String>representing the values of the indicator's signal
  - **Returns** - Vector<String>representing the values of the calculated histogram

---

- *calcIndicators*
  `protected void` **calcIndicators**`( java.util.Vector` **vector** `)`
  - **Usage**
    - Calculates all the components for the MACD indicator, and the RSI indicator, of a vector of values
  - **Parameters**
    - `vector` - vector from which to get the vector of values

- *calcIndMacd*
  `protected double` **calcIndMacd**`( java.lang.String` **symbol** `)`
  - **Usage**
    - Process to calculate the MACD percentage indicator of a selected stock
  - **Parameters**
    - `symbol` - String, representing the symbol of the stock
  - **Returns** - Double, representing the calculated MACD percentage

---

- *calcIndRsi*
  `protected double` **calcIndRsi**`( java.lang.String` **symbol** `)`
  - **Usage**
    - Process to calculate the RSI percentage indicator of a selected stock
  - **Parameters**
    - `symbol` - String, representing the symbol of the stock
  - **Returns** - Double, representing the calculated RSI percentage

---

- *calcIndSi*
  `protected double` **calcIndSi**`( java.lang.String` **symbol** `)`
  - **Usage**
    - Process to calculate the SI percentage indicator of a selected stock
  - **Parameters**
    - `symbol` - String, representing the symbol of the stock
  - **Returns** - Double, representing the calculated SI percentage

---

- *calcMacd*
  `protected Vector` **calcMacd**`( java.util.Vector` **vector** `)`
  - **Usage**
    - Calculates the MACD indicator for a vector of values
  - **Parameters**
    - `vector` - Vector<String>of the values
  - **Returns** - Vector<String>of the calculated MACD indicator

---

- *calcRsi*
  `protected Vector` **calcRsi**`( java.util.Vector` **vector** `)`
  - **Usage**
    - Calculates the RSI indicator for a vector of values
  - **Parameters**
    - `vector` - Vector<String>of the values

- **Returns** - Vector<String>of the calculated RSI indicator

---

- *calcSi*
  `protected Vector` **calcSi(** `java.util.Vector` **maxVector,** `java.util.Vector` **minVector,** `java.util.Vector` **priceVector)**
  - **Usage**
    - Calculates the SI indicator for a vector of values
  - **Parameters**
    - `maxVector` - vector from which to get the vector of max values
    - `minVector` - vector from which to get the vector of min values
    - `priceVector` - vector from which to get the vector of price values
  - **Returns** - Vector<String>of the calculated SI indicator

---

- *calcSiIndicator*
  `protected void` **calcSiIndicator(** `java.util.Vector` **maxVector,** `java.util.Vector` **minVector,** `java.util.Vector` **priceVector)**
  - **Usage**
    - Calculates all the components of the SI indicator for a vector of values
  - **Parameters**
    - `maxVector` - vector from which to get the vector of max values
    - `minVector` - vector from which to get the vector of min values
    - `priceVector` - vector from which to get the vector of price values

---

- *calcSma*
  `protected Vector` **calcSma(** `java.util.Vector` **vector,** `float` **period)**
  - **Usage**
    - Calculates the simple moving average of a received vector of values for a received period of days
  - **Parameters**
    - `vector` - Vector<String>representing the values on which to calculate the SMA
    - `period` - Float representing the period to calculate the SMA
  - **Returns** - Vector<String>representing the calculated values of the SMA

- *cleanUp*
  `protected void` **cleanUp(** **)**
  - **Usage**
    - Closes Statements, PreparedStatements and RecordSets that might be open

---

- *getSleep*
  `protected long` **getSleep( )**
  - **Usage**
    - Returns ThreadBolsaDB sleep time in milliseconds
  - **Returns** - Sleep time in milliseconds

- *getStop*
  `protected boolean` **getStop( )**
  - **Usage**
    - Gets stop state of ThreadBolsaDB
  - **Returns** - Boolean, true if thread is stopped, false otherwise

---

- *getWait*
  `protected boolean` **getWait( )**
  - **Usage**
    - Gets wait state of ThreadBolsaDB
  - **Returns** - Boolean, true if thread is waiting, false otherwise

---

- *insertDB*
  `protected void` **insertDB( )**
  - **Usage**
    - Connects to financial source in internet, gets stock quotes and writes them to the database

---

- *run*
  `public void` **run( )**
  - **Usage**
    - Run method of ThreadBolsaDB

- *setSleep*
  `protected void` **setSleep(** `long` **sleep)**
  - **Usage**
    - Sets sleep time of ThreadBolsaDB in milliseconds
  - **Parameters**
    - `sleep` - Sleep time in milliseconds

- *setStop*
  `protected void` **setStop( )**
  - **Usage**

* Sets stop state of ThreadCleanDB

- *setWait*
  `protected void` **setWait( )**
  - **Usage**
    * Sets state of ThreadBolsaDB to wait

- *stopDB*
  `protected void` **stopDB( )**
  - **Usage**
    * Stops ThreadBolsaDB

B.4 CLASS ThreadCleanDB

Thread who maintains database integrity

B.4.1 DECLARATION

public class ThreadCleanDB
extends java.lang.Thread

B.4.2 CONSTRUCTORS

- *ThreadCleanDB*
  `public` **ThreadCleanDB(** `Bolsa` **application,** `java.sql.Connection` **db)**
  - **Usage**
    * Creates a new instance of ThreadCleanDB
  - **Parameters**
    * `application` - Calling parent JFrame
    * `db` - Database Connection provided by parent

B.4.3 METHODS

- *getStop*
  `protected boolean` **getStop( )**
  - **Usage**
    - Gets stop state of ThreadCleanDB
  - **Returns** - Boolean, true if thread is stopped, false if not

- *run*
  `public void` **run( )**
  - **Usage**
    - Run method of ThreadCleanDB

- *setStop*
  `protected void` **setStop( )**
  - **Usage**
    - Sets state of ThreadCleanDB to stopped

B.5 CLASS Trader

Main class for project aitraderex Builds GUI and launches Threads

B.5.1 DECLARATION

public class Trader
extends javax.swing.JFrame

B.5.2 SERIALIZABLE FIELDS

- private ThreadTraderAG threadTraderAG

–

- private ThreadTraderST threadTraderST

  –

- private ThreadTraderINV threadTraderINV

  –

- private String server

  –

- private String login

  –

- private String passwd

  –

- private int cursor

  –

- private long timeout

  –

- private boolean isDrawing

  –

B.5.3 METHODS

- *addCursor*
  `protected void` **addCursor**`( )`

  – **Usage**
    * Adds to the variable cursor and sets the wait state

- *addRcv*
  `protected void` **addRcv**`( java.lang.String` **msg** `)`

  – **Usage**
    * Adds message received from agent to text area

  – **Parameters**
    * `msg` - String, representing received message

- *addSent*
  `protected void` **addSent(** `java.lang.String` **msg)**
  - **Usage**
    - Adds message sento to agent to text area
  - **Parameters**
    - `msg` - String, representing sent message

- *askMarket*
  `protected void` **askMarket( )**
  - **Usage**
    - Asks the server for the market information

- *askPort*
  `protected void` **askPort( )**
  - **Usage**
    - Asks the server for the portfolio information

- *askStock*
  `protected void` **askStock( )**
  - **Usage**
    - Asks the server for the stock information

- *buyStock*
  `protected void` **buyStock(** `java.lang.String` **symbol,** `int` **quant)**
  - **Usage**
    - Sends KQML message to buy selected stock in stock market
  - **Parameters**
    - `symbol` - String, representing the symbol of the stock to buy
    - quant - Int, representing the quantity to buy

- *calcTotal*
  `protected double` **calcTotal( )**
  - **Usage**
    - Calculates the total of the buy/sell operation in the portfolio
  - **Returns** - Double, representing the total of the buy/sell operation in the portfolio

---

- *centerParent*
  `protected void` **centerParent(** `java.awt.Component` **parent,**
  `java.awt.Component` **child)**

- **Usage**
  - Centers child component relative to parent component on screen
- **Parameters**
  - `parent` - Parent component
  - `child` - Child component

- *changeAspect*
  `protected void` **changeAspect**`( java.awt.Component` **component** `)`
  - **Usage**
    - Changes application aspect to that of the operating system it's run on
  - **Parameters**
    - `component` - Component to change aspect

- *changeMenuAG*
  `protected void` **changeMenuAG**`( boolean` **enable** `)`
  - **Usage**
    - Enables/Disables menu that controls Thread AG
  - **Parameters**
    - `enable` - Boolean, true to enable menu, false to disable

- *changeMenuINV*
  `protected void` **changeMenuINV**`( boolean` **enable** `)`
  - **Usage**
    - Enables/Disables menu that controls Thread INV
  - **Parameters**
    - `enable` - Boolean, true to enable menu, false to disable

- *changeMenuST*
  `protected void` **changeMenuST**`( boolean` **enable** `)`
  - **Usage**
    - Enables/Disables menu that controls Thread ST
  - **Parameters**
    - `enable` - Boolean, true to enable menu, false to disable

- *convPercDouble*
  `protected double` **convPercDouble**`( java.lang.String` **perc** `)`
  - **Usage**
    - Converts a percentage string to a double
  - **Parameters**

    * `perc` - String, representing the percentage to convert
  - **Returns** - Double, representing the value of the converted percentage

---

- *createTblPort*
  `protected void` **createTblPort**`( java.util.Vector` **data** `)`
  - **Usage**
    * Creates portfolio table
  - **Parameters**
    * `data` - Vector, representing stock data

- *createTblStk*
  `protected void` **createTblStk**`( java.util.Vector` **data** `)`
  - **Usage**
    * Creates stock table
  - **Parameters**
    * `data` - Vector, representing stock data

- *createValue*
  `protected XYDataset` **createValue**`( java.lang.String` **type** `)`
  - **Usage**
    * Creates XYDataset for drawing graphics
  - **Parameters**
    * `type` - String, representing the type of value to create
  - **Returns** - XYDataset for drawing graphics

- *createValues*
  `protected OHLCDataset` **createValues**`( java.lang.String` **name** `)`
  - **Usage**
    * Creates OHLCDataset for drawing graphics
  - **Parameters**
    * `name` - String, representing the value to fetch
  - **Returns** - OHLCDataset for drawing graphics

- *drawGraph*
  `protected void` **drawGraph**`( )`
  - **Usage**
    * Draws Graph according to selected parameters

- *exitApp*
  `protected void` **exitApp**`( )`

- **Usage**
  - Stops all threads and closes all ResultSets, Statements and PreparedStatements

---

- *getAmount*
`protected String` **getAmount**`( double` **amount** `)`
  - **Usage**
    - Returns the formatted currency amount
  - **Parameters**
    - `amount` - Double, representing the amount
  - **Returns** - String, representing the formatted currency amount

---

- *getBalance*
`protected double` **getBalance**`( )`
  - **Usage**
    - Gets the current balance amount for the trader
  - **Returns** - Double representing the current balance amount

---

- *getIndBuy*
`protected int` **getIndBuy**`( )`
  - **Usage**
    - Gets the minimum indicator percentage to permit buying
  - **Returns** - Double, representing the minimum indicator percentage to permit buying

---

- *getIndSell*
`protected int` **getIndSell**`( )`
  - **Usage**
    - Gets the minimum indicator percentage below which selling is obrigatory
  - **Returns** - Double, representing the minimum indicator percentage below which selling is obrigatory

---

- *getMaxAmount*
`protected double` **getMaxAmount**`( )`
  - **Usage**
    - Gets the maximum amount to invest
  - **Returns** - Double, representing the maximum amount to invest

---

- *getMaxStk*
  `protected int` **getMaxStk( )**
  - **Usage**
    - Gets max number of stocks in the portfolio
  - **Returns** - Int, representing the max number of stocks in the portfolio

---

- *getMinAmount*
  `protected double` **getMinAmount( )**
  - **Usage**
    - Gets the minimum amount to invest
  - **Returns** - Double, representing the minimum amount to invest

---

- *getMktHour*
  `protected String` **getMktHour(** `java.lang.String` **type)**
  - **Usage**
    - Gets the open/close hour for the market
  - **Parameters**
    - `type` - String, representing the type of hour we want to get
  - **Returns** - String, representing the open/close hour for the market

---

- *getPctChg*
  `protected String` **getPctChg(** `double` **openvalue,** `double` **closevalue)**
  - **Usage**
    - Returns the formatted stock percentage variation
  - **Parameters**
    - `openvalue` - Double, representing the stock open value
    - `closevalue` - Double, representing the stock close value
  - **Returns** - String, representing the formatted stock percentage variation

---

- *getPctInd*
  `protected String` **getPctInd(** `double` **rsi,** `double` **si,** `double` **macd)**
  - **Usage**
    - Returns the formatted stock indicator percentage
  - **Parameters**
    - `rsi` - Double, representing the stock rsi indicator
    - `si` - Double, representing the stock si indicator
    - `macd` - Double, representing the stock macd indicator
  - **Returns** - String, representing the formatted stock indicator percentage

---

- *getPortfolio*
  `protected Vector` **getPortfolio( )**
  - **Usage**
    - Returns a vector with the stocks in the portfolio and their associated values
  - **Returns** - Vector<object>with the stocks in the portfolio and their associated values

---

- *getStocks*
  `protected Vector` **getStocks( )**
  - **Usage**
    - Returns a vector with the stocks available in the stock market and their associated values
  - **Returns** - Vector<object>with the stocks available in the stock market and their associated values

---

- *getStopGain*
  `protected int` **getStopGain( )**
  - **Usage**
    - Gets the percentage of the maximum stop gain, to prevent posible losses after achieving some level of profits
  - **Returns** - Double, representing the percentage of the stop gain

---

- *getStopLoss*
  `protected int` **getStopLoss( )**
  - **Usage**
    - Gets the percentage of the minimum stop loss, to prevent greater losses
  - **Returns** - Double, representing the percentage of the stop loss

---

- *getTimeout*
  `protected long` **getTimeout( )**
  - **Usage**
    - Gets the thread timeout in milliseconds
  - **Returns** - Long, representing the timeout in milliseconds

---

- *isNegChg*
  `protected String` **isNegChg(** `int` **rowindex,** `int` **colindex)**
  - **Usage**

    * Returns a string representing a stock variation for a table line
  - **Parameters**
    * `colindex` - Int, representing the selected column of the stock table
    * `rowindex` - Int, representing the selected row of the stock table
  - **Returns** - String, representing a stock variation

- *isPNegChg*
  `protected String` **isPNegChg(** `int` **rowindex,** `int` **colindex)**
  - **Usage**
    * Returns a string representing a stock variation for a table cell
  - **Parameters**
    * `rowindex` - Int, representing the selected row of the portfolio table
    * `colindex` - Int, representing the selected column of the portfolio table
  - **Returns** - String, representing a stock variation

- *main*
  `public static void` **main(** `java.lang.String []` **args)**
  - **Usage**
    * Main class to launch JFrame Trader
  - **Parameters**
    * `args` - the command line arguments

- *refreshCmbStk*
  `protected void` **refreshCmbStk( )**
  - **Usage**
    * Refreshes information in stocks combo

- *refreshPortStk*
  `protected void` **refreshPortStk( )**
  - **Usage**
    * Refreshes information in portfolio stocks combo

- *refreshTblPort*
  `protected void` **refreshTblPort( )**
  - **Usage**
    * Refreshes information in portfolio table

- *refreshTblStk*
  `protected void` **refreshTblStk( )**
  - **Usage**

* Refreshes information in stocks table

- *sellStock*
  `protected void` **sellStock(** `java.lang.String` **symbol,** `int` **quant)**
  - **Usage**
    * Sends KQML message to sell selected stock in portfolio
  - **Parameters**
    * `symbol` - String, representing the symbol of the stock to sell
    * `quant` - Int, representing the quantity to sell

- *setBalance*
  `protected void` **setBalance(**)
  - **Usage**
    * Sets the current balance amount in the portfolio tab

- *setBalance*
  `protected void` **setBalance(** `double` **balance)**
  - **Usage**
    * Sets the current balance amount in the portfolio tab
  - **Parameters**
    * `balance` - Double, representing the balance amount

- *setStatus*
  `protected void` **setStatus(** `java.lang.String` **msg)**
  - **Usage**
    * Sets status line to received message
  - **Parameters**
    * `msg` - String, representing message

- *showMsg*
  `protected void` **showMsg(** `java.lang.String` **msg,** `java.lang.String` **title,** `int` **type)**
  - **Usage**
    * Shows JOptionPane with received message to the user
  - **Parameters**
    * `msg` - String, representing message
    * `title` - String, representing title
    * `type` - Int, representing type

- *startAG*
  `protected void` **startAG(**)

- **Usage**
  - Starts thread AG

- *startINV*
  `protected void` **startINV**`( )`

  - **Usage**
    - Starts thread INV

- *startST*
  `protected void` **startST**`( )`

  - **Usage**
    - Starts thread ST

- *subCursor*
  `protected void` **subCursor**`( )`

  - **Usage**
    - Subtracts from the variable cursor and sets the default state

B.6 CLASS ThreadTraderAG

Thread who sends and receives messages from the other JATLite Agents

B.6.1 DECLARATION

public class ThreadTraderAG

extends RouterLayer.AgentClient.RouterClientAction

B.6.2 CONSTRUCTORS

- *ThreadTraderAG*
  `public` **ThreadTraderAG**`( Trader` **application**`, java.lang.String` **server**`, java.lang.String` **login**`, java.lang.String` **passwd** `)`

- **Usage**
  - Creates a new instance of ThreadTraderAG
- **Parameters**
  - `server` - String representing the server to connect
  - `login` - String representing the trader login
  - `passwd` - String representing the trader password
  - `application` - Calling parent JFrame

B.6.3 METHODS

- *Act*
  `public boolean` **Act(** `java.lang.Object` **obj)**
  - **Usage**
    - JATLite extended method to process received messages
  - **Parameters**
    - `obj` - KQMLmail message received from other agents
  - **Returns** - Boolean, true if message correctly processed, false otherwise

---

- *askLogin*
  `protected void` **askLogin( )**
  - **Usage**
    - Asks/Validates login from the server

- *askMarket*
  `protected void` **askMarket( )**
  - **Usage**
    - Ask market information from the server

- *askPort*
  `protected void` **askPort( )**
  - **Usage**
    - Ask portfolio information from the server

- *askStock*
  `protected void` **askStock( )**
  - **Usage**
    - Ask stock information from the server

- *buy*
  `protected void` **buy(** `java.lang.String` **symb,** `java.lang.String` **quant)**
  - **Usage**
    - Sends message to the server ordering a buy operation of some quant of some stock
  - **Parameters**
    - `symb` - String representing the stock to buy
    - `quant` - Integer representing the quantity to buy

- *calcEma*
  `protected Vector` **calcEma(** `java.util.Vector` **vector,** `float` **period)**
  - **Usage**
    - Calculates the exponential moving average of a received vector of values for a received period of days
  - **Parameters**
    - `vector` - Vector<String>representing the values on which to calculate the EMA
    - `period` - Float representing the period to calculate the EMA
  - **Returns** - Vector<String>representing the calculated values of the EMA

- *calcHist*
  `protected Vector` **calcHist(** `java.util.Vector` **indicator,** `java.util.Vector` **signal)**
  - **Usage**
    - Calculates the histogram values representing the diferences between some indicator and it's signal
  - **Parameters**
    - `indicator` - Vector<String>representing the values of the indicator
    - `signal` - Vector<String>representing the values of the indicator's signal
  - **Returns** - Vector<String>representing the values of the calculated histogram

---

- *calcIndicators*
  `protected void` **calcIndicators(** `Daily` **daily,** `java.util.Vector` **vector)**
  - **Usage**
    - Calculates all the components for the MACD indicator, and the RSI indicator, of a vector of values

- **Parameters**
  - `daily` - vector from which to get the vector of values and where to keep the calculated ones
  - `vector` - vector from which to get the vector of values

- *calcMacd*
  `protected Vector` **calcMacd**`( java.util.Vector` **vector** `)`
  - **Usage**
    - Calculates the MACD indicator for a vector of values
  - **Parameters**
    - `vector` - Vector<String>of the values
  - **Returns** - Vector<String>of the calculated MACD indicator

---

- *calcRsi*
  `protected Vector` **calcRsi**`( java.util.Vector` **vector** `)`
  - **Usage**
    - Calculates the RSI indicator for a vector of values
  - **Parameters**
    - `vector` - Vector<String>of the values
  - **Returns** - Vector<String>of the calculated RSI indicator

---

- *calcSi*
  `protected Vector` **calcSi**`( Daily` **daily** `)`
  - **Usage**
    - Calculates the SI indicator for a vector of values
  - **Parameters**
    - `daily` - vector from which to get the vector of values
  - **Returns** - Vector<String>of the calculated SI indicator

---

- *calcSiIndicator*
  `protected void` **calcSiIndicator**`( Daily` **daily** `)`
  - **Usage**
    - Calculates all the components of the SI indicator for a vector of values
  - **Parameters**
    - `daily` - vector from which to get the vector of values and where to keep the calculated ones

---

- *calcSma*
  `protected Vector` **calcSma**`( java.util.Vector` **vector**`, float` **period** `)`

- **Usage**
  - Calculates the simple moving average of a received vector of values for a received period of days
- **Parameters**
  - `vector` - Vector<String>representing the values on which to calculate the SMA
  - `period` - Float representing the period to calculate the SMA
- **Returns** - Vector<String>representing the calculated values of the SMA

- *createLogin*
  `protected void` **createLogin**`( )`
  - **Usage**
    - Asks to create login in the server

- *deleteLogin*
  `protected void` **deleteLogin**`( )`
  - **Usage**
    - Asks to delete login from the server

- *deposit*
  `protected void` **deposit**`( double` **amount** `)`
  - **Usage**
    - Asks to deposit some amount to the trader's account in the server
  - **Parameters**
    - `amount` - Double representing the amount to deposit

- *getBalance*
  `protected double` **getBalance**`( )`
  - **Usage**
    - Gets the current balance
  - **Returns** - Double with the current balance

- *getDaily*
  `protected Vector` **getDaily**`( java.lang.String` **symb**`, java.lang.String` **data** `)`
  - **Usage**
    - Gets selected daily data for a stock symbol
  - **Parameters**
    - `symb` - String representing stock to get
    - `data` - String representing the type of data to get

- **Returns** - Vector<String>with the daily data

- *getInternetIP*
  `protected String` **getInternetIP**`( )`
  - **Usage**
    - Gets the internet address, or returns localhost if not connected
  - **Returns** - String representing the internet address

- *getIntra*
  `protected Vector` **getIntra**`( java.lang.String` **symb**`, java.lang.String` **data** `)`
  - **Usage**
    - Gets intraday price and volume data for a stock
  - **Parameters**
    - `symb` - String representing stock to get
    - **data** - String representing the data to get, Price or Volume
  - **Returns** - Vector<String>with the intraday data

- *getIntraH*
  `protected String` **getIntraH**`( java.lang.String` **symb**`, java.lang.String` **type** `)`
  - **Usage**
    - Gets intraday information of the stock, and returns open or close hour for that stock
  - **Parameters**
    - `symb` - String representing stock to get
    - `type` - String representing the type of hour to get
  - **Returns** - String representation of the open or close hour for the stock

---

- *getLogin*
  `protected String` **getLogin**`( )`
  - **Usage**
    - Gets the login information
  - **Returns** - String with the login

- *getMarket*
  `protected Vector` **getMarket**`( )`
  - **Usage**
    - Gets the market data
  - **Returns** - Vector<Object>with the market data

- *getMktHour*
  `protected String` **getMktHour(** `java.lang.String` **symb,** `java.lang.String` **type)**
  - **Usage**
    - Gets the open/close hour for the market
  - **Parameters**
    - `symb` - String of the stock symbol
    - `type` - String of the type of hour we want (open or close)
  - **Returns** - String, representing the open/close hour for the market

---

- *getPort*
  `protected Vector` **getPort( )**
  - **Usage**
    - Gets the portfolio data for the trader
  - **Returns** - Vector<Object>with the portfolio data

- *getSleep*
  `protected long` **getSleep( )**
  - **Usage**
    - Returns ThreadTraderAG sleep time in milliseconds
  - **Returns** - Sleep time in milliseconds

- *getStock*
  `protected Vector` **getStock( )**
  - **Usage**
    - Gets the stock data
  - **Returns** - Vector<Object>with the stock data

- *getStop*
  `protected boolean` **getStop( )**
  - **Usage**
    - Gets stop state of ThreadTraderAG
  - **Returns** - Boolean, true if thread is stopped, false otherwise

---

- *getWait*
  `protected boolean` **getWait( )**
  - **Usage**
    - Gets wait state of ThreadTraderAG

  - **Returns** - Boolean, true if thread is waiting, false otherwise

---

- *isMktClosed*
  `protected Boolean` **isMktClosed**`( java.lang.String` **symb** `)`
  - **Usage**
    - Returns if market is already closed
  - **Parameters**
    - `symb` - String of the stock symbol
  - **Returns** - Boolean, true if market closed, false otherwise

---

- *isValid*
  `protected boolean` **isValid**`( )`
  - **Usage**
    - Returns if the login is valid
  - **Returns** - Boolean, true if the login is valid, false otherwise

---

- *processMessage*
  `public void` **processMessage**`( java.lang.String` **msg**`,`
  `java.lang.Object` **obj** `)`
  - **Usage**
    - JATLite extended method to process received messages (not used)
  - **Parameters**
    - `msg` - String message received from other agents
    - `obj` - KQMLmail message received from other agents

- *rcvTell*
  `protected void` **rcvTell**`( KQMLLayer.KQMLmessage` **kqml** `)`
  - **Usage**
    - Processes received Tell messages
  - **Parameters**
    - `kqml` - Tell KQMLmessage received from other agents

- *readBalance*
  `protected boolean` **readBalance**`( )`
  - **Usage**
    - Returns if the system is waiting to read the balance information
  - **Returns** - Boolean, true if waiting to read balance, false otherwise

---

- *readDaily*
  `protected boolean` **readDaily**`( )`

- **Usage**
  - * Returns if the system is waiting to read the daily information
- **Returns** - Boolean, true if waiting to read daily, false otherwise

---

- *readIntra*
  `protected boolean` **readIntra( )**
  - **Usage**
    - * Returns if the system is waiting to read the intraday information
  - **Returns** - Boolean, true if waiting to read intraday, false otherwise

---

- *readLogin*
  `protected boolean` **readLogin( )**
  - **Usage**
    - * Returns if the system is waiting to read the login information
  - **Returns** - Boolean, true if waiting to read login, false otherwise

---

- *readMkt*
  `protected boolean` **readMkt( )**
  - **Usage**
    - * Returns if the system is waiting to read the market information
  - **Returns** - Boolean, true if waiting to read market, false otherwise

---

- *readPort*
  `protected boolean` **readPort( )**
  - **Usage**
    - * Returns if the system is waiting to read the portfolio information
  - **Returns** - Boolean, true if waiting to read portfolio, false otherwise

---

- *readStk*
  `protected boolean` **readStk( )**
  - **Usage**
    - * Returns if the system is waiting to read the stock information
  - **Returns** - Boolean, true if waiting to read stock, false otherwise

---

- *run*
  `public void` **run( )**
  - **Usage**
    - * Run method of ThreadTraderAG

- *sell*
  `protected void` **sell(** `java.lang.String` **symb,** `java.lang.String` **quant)**
  - **Usage**
    - Sends message to the server ordering a sell operation of some quant of some stock
  - **Parameters**
    - `symb` - String representing the stock to sell
    - `quant` - Integer representing the quantity to sell

- *sendMsg*
  `protected void` **sendMsg(** `java.lang.String` **sender,** `java.lang.String` **receiver,** `java.lang.String` **perform,** `java.lang.String` **msg)**
  - **Usage**
    - Sends KQMLmessage to specified receiver
  - **Parameters**
    - `sender` - String, representing sender agent
    - `receiver` - String, representing receiver agent
    - `perform` - String, representing performative
    - `msg` - String, representing text message

- *setSleep*
  `protected void` **setSleep(** `long` **sleep)**
  - **Usage**
    - Sets sleep time of ThreadTraderAG in milliseconds
  - **Parameters**
    - `sleep` - Sleep time in milliseconds

- *setStop*
  `protected void` **setStop( )**
  - **Usage**
    - Sets stop state of ThreadTraderAG

- *setWait*
  `protected void` **setWait( )**
  - **Usage**
    - Sets state of ThreadTraderAG to wait

- *srchDaily*
  `protected int` **srchDaily(** `java.lang.String` **symb)**

- **Usage**
  - Searches a stock in the daily vector
- **Parameters**
  - `symb` - String representing stock to search
- **Returns** - Integer index position of the stock, -1 if not found

---

- *srchIntra*
  `protected int` **srchIntra**`( java.lang.String` **symb** `)`
  - **Usage**
    - Searches a stock in the intraday vector
  - **Parameters**
    - `symb` - String representing stock to search
  - **Returns** - Integer index position of the stock, -1 if not found

---

- *srchPort*
  `protected int` **srchPort**`( java.lang.String` **symb** `)`
  - **Usage**
    - Searches a stock in the portfolio vector
  - **Parameters**
    - `symb` - String representing stock to search
  - **Returns** - Integer index position of the stock, -1 if not found

---

- *srchStock*
  `protected int` **srchStock**`( java.lang.String` **symb** `)`
  - **Usage**
    - Searches a stock in the stock vector
  - **Parameters**
    - `symb` - String representing stock to search
  - **Returns** - Integer index position of the stock, -1 if not found

---

- *start*
  `public void` **start**`( )`
  - **Usage**
    - Start method of ThreadTraderAG

- *stopAG*
  `protected void` **stopAG**`( )`
  - **Usage**

* Stops TreadBolsaAG, closes Statements, PreparedStatements and RecordSets that might be open, and disconnects/unregisters JATLite Agent

---

- *withdraw*
  `protected void` **withdraw**`( double` **amount** `)`
  - **Usage**
    * Asks to withdraw some amount from the trader's account in the server
  - **Parameters**
    * `amount` - Double representing the amount to withdraw

B.7 CLASS ThreadTraderST

Thread who asks for stock information periodicaly and refreshes the graphics

B.7.1 DECLARATION

public class ThreadTraderST **extends** java.lang.Thread

B.7.2 CONSTRUCTORS

- *ThreadTraderST*
  `public` **ThreadTraderST**`( Trader` **application** `)`
  - **Usage**
    * Creates a new instance of ThreadTraderST
  - **Parameters**
    * `application` - Calling parent JFrame

B.7.3 METHODS

- *getSleep*
  `protected long` **getSleep( )**
  - **Usage**
    - Returns ThreadTraderST sleep time in milliseconds
  - **Returns** - Sleep time in milliseconds

- *getStop*
  `protected boolean` **getStop( )**
  - **Usage**
    - Gets stop state of ThreadTraderST
  - **Returns** - Boolean, true if thread is stopped, false if not

---

- *run*
  `public void` **run( )**
  - **Usage**
    - Run method of ThreadTraderST

- *setSleep*
  `protected void` **setSleep(** `long` **sleep)**
  - **Usage**
    - Sets sleep time of ThreadTraderST in milliseconds
  - **Parameters**
    - `sleep` - Sleep time in milliseconds

- *setStop*
  `protected void` **setStop( )**
  - **Usage**
    - Sets state of ThreadTraderST to stopped

B.8 CLASS ThreadTraderINV

Thread who is responsible for buying and selling stocks automaticaly, following the pre-defined values and the percentage of the indicator

B.8.1 DECLARATION

public class ThreadTraderINV **extends** java.lang.Thread

B.8.2 CONSTRUCTORS

- *ThreadTraderINV*
  `public` **ThreadTraderINV**`( Trader` **application** `)`
  - **Usage**
    - Creates a new instance of ThreadTraderINV
  - **Parameters**
    - `application` - Calling parent JFrame

B.8.3 METHODS

- *buyStocks*
  `protected void` **buyStocks**`( )`
  - **Usage**
    - Searches all the stocks available in the stock market and verifies if any comply with the parameters to buy, if so, calls the process that sends the message to buy the stock

- *existInPort*
  `protected boolean` **existInPort**`( java.lang.String` **symbol** `)`
  - **Usage**
    - Searches in the portfolio if a stock already exists
  - **Parameters**
    - `symbol` - String, representing the symbol of the stock to search
  - **Returns** - Boolean, true if stock already exists in portfolio, false otherwise

- *getSleep*
  `protected long` **getSleep**`( )`

- **Usage**
  - Returns ThreadTraderINV sleep time in milliseconds
- **Returns** - Sleep time in milliseconds

- *getStop*
  `protected boolean` **getStop( )**
  - **Usage**
    - Gets stop state of ThreadTraderINV
  - **Returns** - Boolean, true if thread is stopped, false if not

---

- *run*
  `public void` **run( )**
  - **Usage**
    - Run method of ThreadTraderINV

- *sellStocks*
  `protected void` **sellStocks( )**
  - **Usage**
    - Searches all the stocks in the portfolio and verifies if any comply with the parameters to sell, if so, calls the process that sends the message to sell the stock

---

- *setSleep*
  `protected void` **setSleep(** `long` **sleep)**
  - **Usage**
    - Sets sleep time of ThreadTraderINV in milliseconds
  - **Parameters**
    - `sleep` - Sleep time in milliseconds

- *setStop*
  `protected void` **setStop( )**
  - **Usage**
    - Sets state of ThreadTraderINV to stopped

Apêndice C

Interface gráfica

Figura C.1: Ecrã de consulta de mensagens enviadas/recebidas

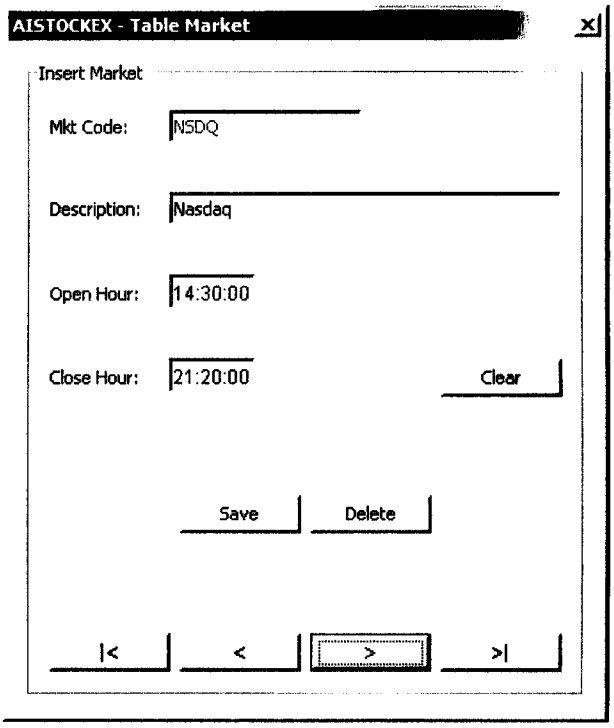


Figura C.2: Ecrã de consulta, inserção e eliminação de registos da tabela mercado

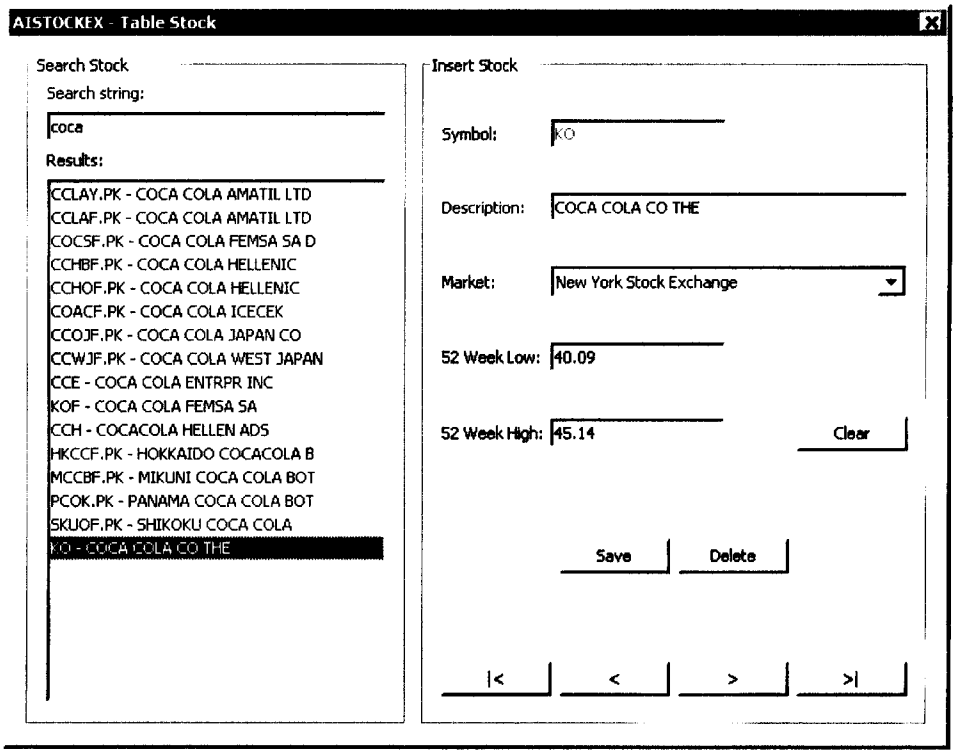
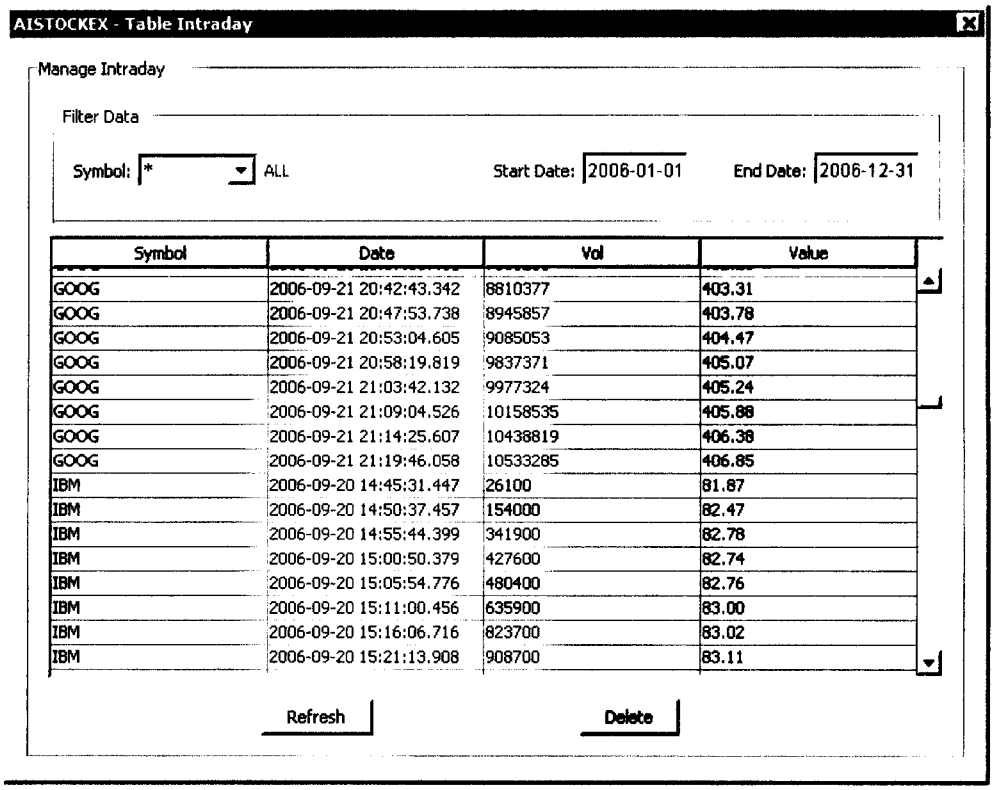
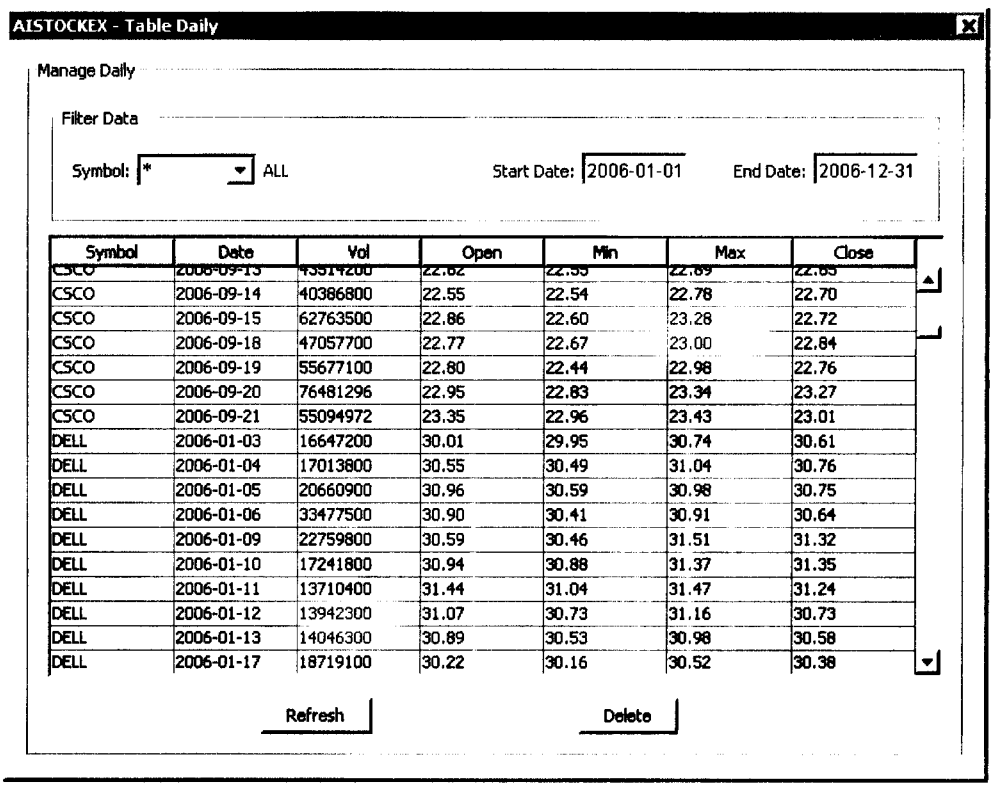


Figura C.3: Ecrã de consulta, inserção e eliminação de registos da tabela acções



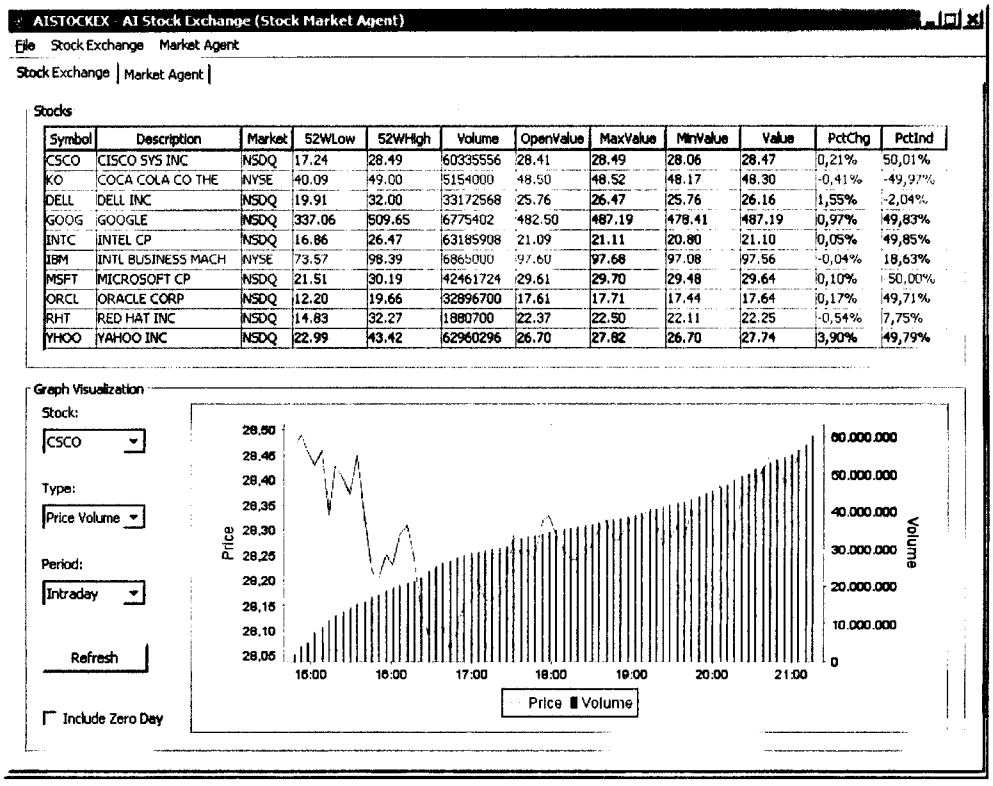
Symbol	Date	Vol	Value
GOOG	2006-09-21 20:42:43.342	8810377	403.31
GOOG	2006-09-21 20:47:53.738	8945857	403.78
GOOG	2006-09-21 20:53:04.605	9085053	404.47
GOOG	2006-09-21 20:58:19.819	9837371	405.07
GOOG	2006-09-21 21:03:42.132	9977324	405.24
GOOG	2006-09-21 21:09:04.526	10158535	405.88
GOOG	2006-09-21 21:14:25.607	10438819	406.38
GOOG	2006-09-21 21:19:46.058	10533285	406.85
IBM	2006-09-20 14:45:31.447	26100	81.87
IBM	2006-09-20 14:50:37.457	154000	82.47
IBM	2006-09-20 14:55:44.399	341900	82.78
IBM	2006-09-20 15:00:50.379	427600	82.74
IBM	2006-09-20 15:05:54.776	480400	82.76
IBM	2006-09-20 15:11:00.456	635900	83.00
IBM	2006-09-20 15:16:06.716	823700	83.02
IBM	2006-09-20 15:21:13.908	908700	83.11

Figura C.4: Ecrã de consulta e eliminação de registos da tabela de cotações *intraday*



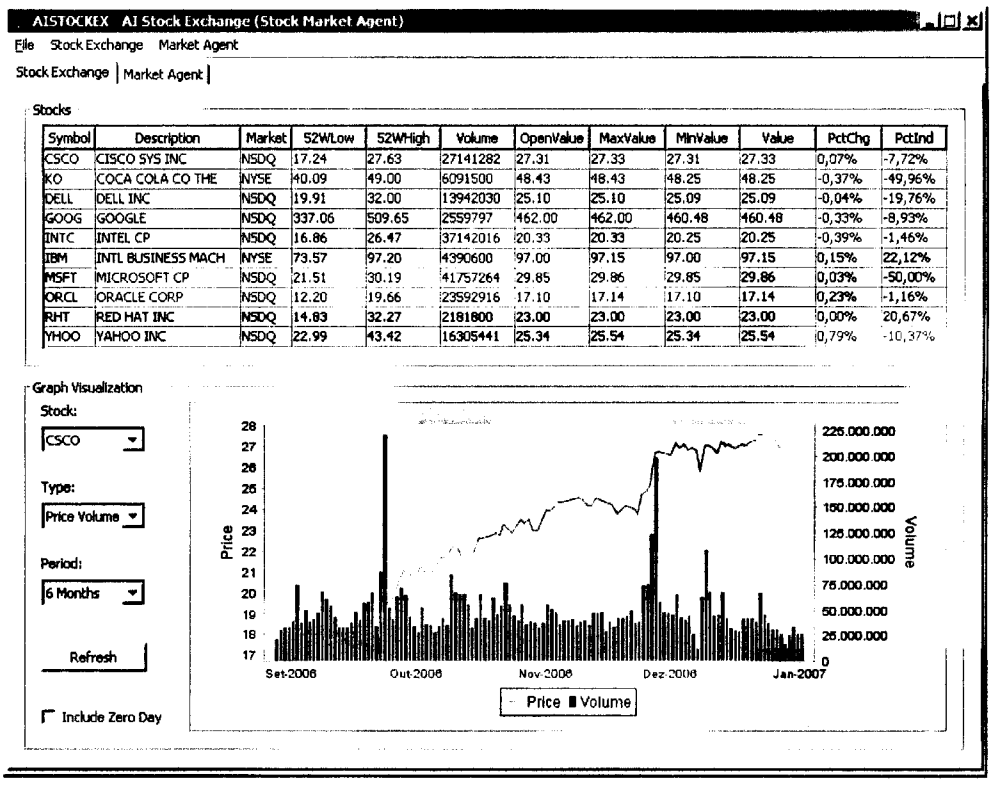
Symbol	Date	Vol	Open	Min	Max	Close
CSCO	2006-09-13	43514200	22.82	22.55	22.89	22.85
CSCO	2006-09-14	40386800	22.55	22.54	22.78	22.70
CSCO	2006-09-15	62763500	22.86	22.60	23.28	22.72
CSCO	2006-09-18	47057700	22.77	22.67	23.00	22.84
CSCO	2006-09-19	55677100	22.80	22.44	22.98	22.76
CSCO	2006-09-20	76481296	22.95	22.83	23.34	23.27
CSCO	2006-09-21	55094972	23.35	22.96	23.43	23.01
DELL	2006-01-03	16647200	30.01	29.95	30.74	30.61
DELL	2006-01-04	17013800	30.55	30.49	31.04	30.76
DELL	2006-01-05	20660900	30.96	30.59	30.98	30.75
DELL	2006-01-06	33477500	30.90	30.41	30.91	30.64
DELL	2006-01-09	22759800	30.59	30.46	31.51	31.32
DELL	2006-01-10	17241800	30.94	30.88	31.37	31.35
DELL	2006-01-11	13710400	31.44	31.04	31.47	31.24
DELL	2006-01-12	13942300	31.07	30.73	31.16	30.73
DELL	2006-01-13	14046300	30.89	30.53	30.98	30.58
DELL	2006-01-17	18719100	30.22	30.16	30.52	30.38

Figura C.5: Ecrã de consulta e eliminação de registos da tabela de cotações diárias



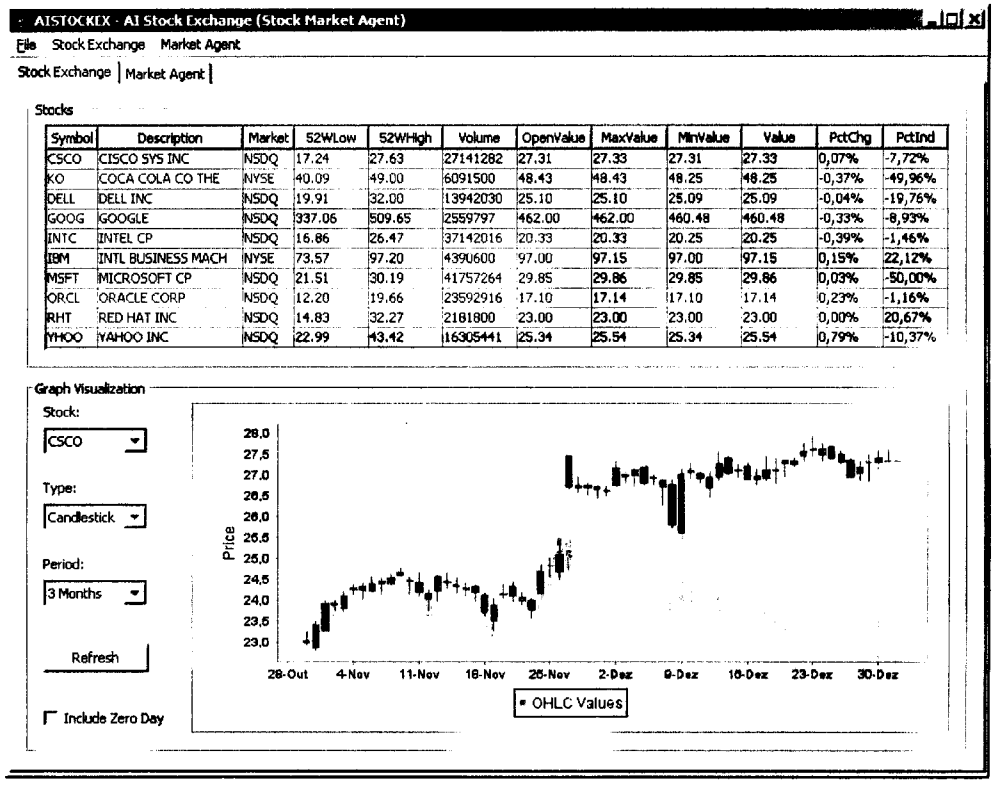
Symbol	Description	Market	52WLow	52WHigh	Volume	OpenValue	MaxValue	MinValue	Value	PctChg	PctInd
CSCO	CISCO SYS INC	NSDQ	17.24	28.49	60335556	28.41	28.49	28.06	28.47	0,21%	50,01%
KO	COCA COLA CO THE	NYSE	40.09	49.00	5154000	48.50	48.52	48.17	48.30	-0,41%	-49,97%
DELL	DELL INC	NSDQ	19.91	32.00	33172568	25.76	26.47	25.76	26.16	1,55%	-2,04%
GOOG	GOOGLE	NSDQ	337.06	509.65	6775402	482.50	487.19	478.41	487.19	0,97%	49,83%
INTC	INTEL CP	NSDQ	16.86	26.47	63185908	21.09	21.11	20.80	21.10	0,05%	49,85%
IBM	INTL BUSINESS MACH	NYSE	73.57	98.39	6865000	97.60	97.68	97.08	97.56	-0,04%	18,63%
MSFT	MICROSOFT CP	NSDQ	21.51	30.19	42461724	29.61	29.70	29.48	29.64	0,10%	-50,00%
ORCL	ORACLE CORP	NSDQ	12.20	19.66	32896700	17.61	17.71	17.44	17.64	0,17%	49,71%
RHT	RED HAT INC	NSDQ	14.83	32.27	1880700	22.37	22.50	22.11	22.25	-0,54%	7,75%
YHOO	YAHOO INC	NSDQ	22.99	43.42	62960296	26.70	27.82	26.70	27.74	3,90%	49,79%

Figura C.6: Ecrã de consulta de cotações com gráfico *intraday*



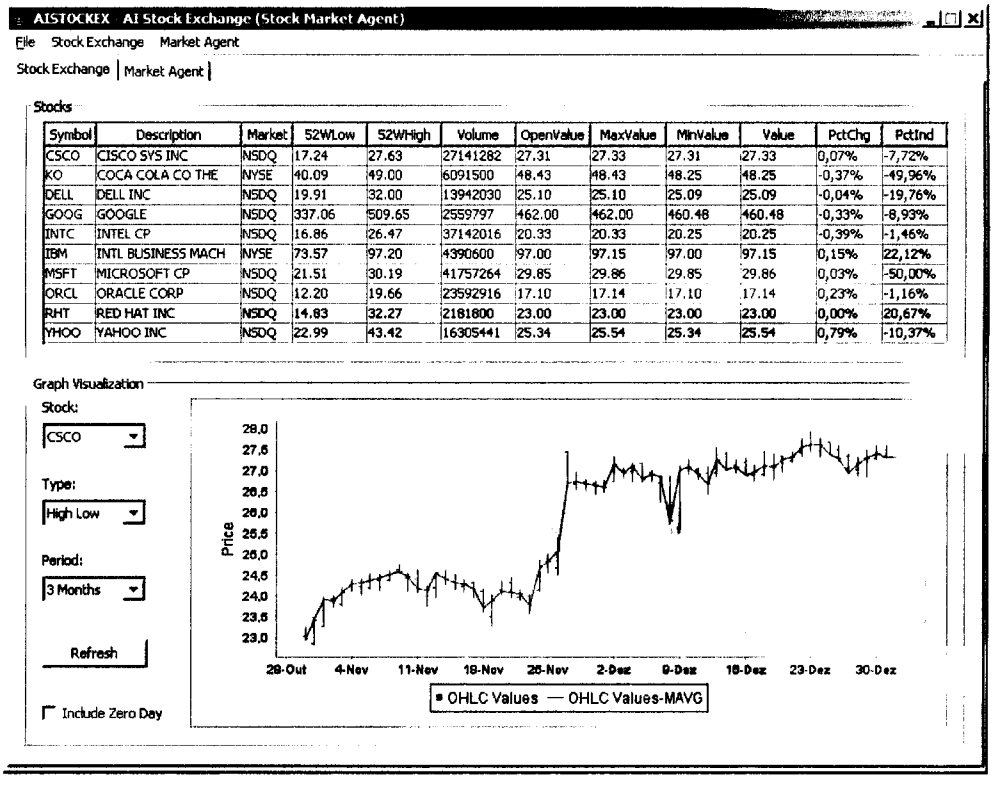
Symbol	Description	Market	52WLow	52WHigh	Volume	OpenValue	MaxValue	MinValue	Value	PctChg	PctInd
CSCO	CISCO SYS INC	NSDQ	17.24	27.63	27141282	27.31	27.33	27.31	27.33	0,07%	-7,72%
KO	COCA COLA CO THE	NYSE	40.09	49.00	6091500	48.43	48.43	48.25	48.25	-0,37%	-49,96%
DELL	DELL INC	NSDQ	19.91	32.00	13942030	25.10	25.10	25.09	25.09	-0,04%	-19,76%
GOOG	GOOGLE	NSDQ	337.06	509.65	2559797	462.00	462.00	460.48	460.48	-0,33%	-8,93%
INTC	INTEL CP	NSDQ	16.86	26.47	37142016	20.33	20.33	20.25	20.25	-0,39%	-1,46%
IBM	INTL BUSINESS MACH	NYSE	73.57	97.20	4390600	97.00	97.15	97.00	97.15	0,15%	22,12%
MSFT	MICROSOFT CP	NSDQ	21.51	30.19	41757264	29.85	29.86	29.85	29.86	0,03%	-50,00%
ORCL	ORACLE CORP	NSDQ	12.20	19.66	23592916	17.10	17.14	17.10	17.14	0,23%	-1,16%
RHT	RED HAT INC	NSDQ	14.83	32.27	2181800	23.00	23.00	23.00	23.00	0,00%	20,67%
YHOO	YAHOO INC	NSDQ	22.99	43.42	16305441	25.34	25.54	25.34	25.54	0,79%	-10,37%

Figura C.7: Ecrã de consulta de cotações com gráfico *price volume*



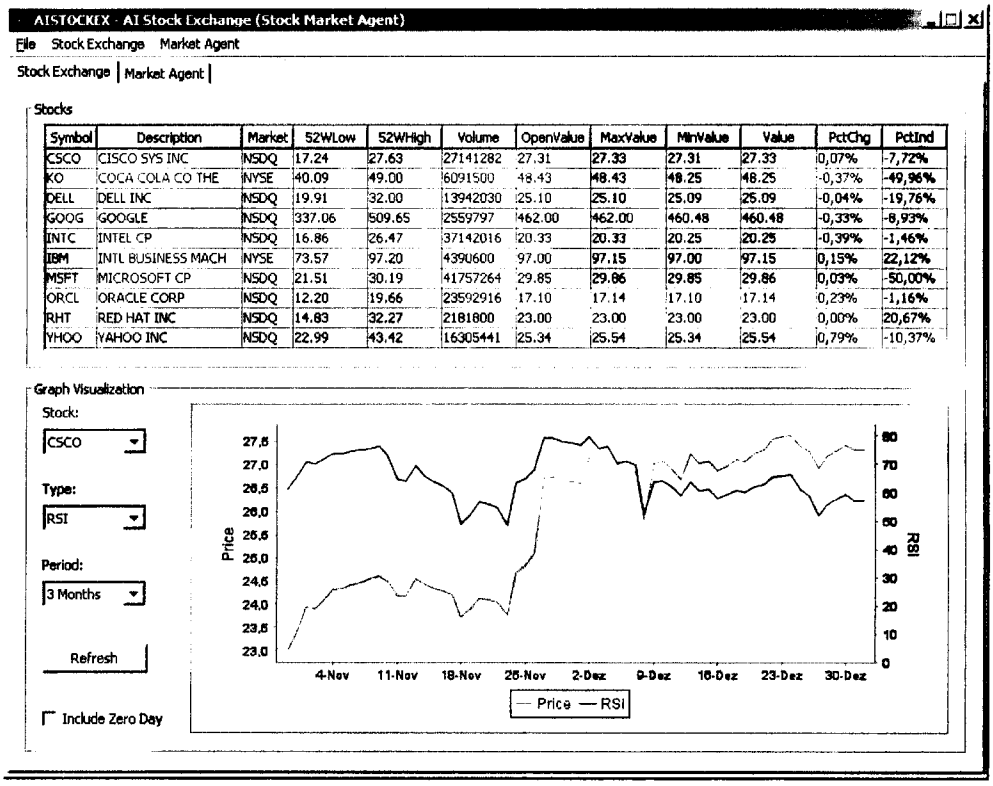
Symbol	Description	Market	52WLow	52WHigh	Volume	OpenValue	MaxValue	MinValue	Value	PctChg	PctInd
CSCO	CISCO SYS INC	NSDQ	17.24	27.63	27141282	27.31	27.33	27.31	27.33	0,07%	-7,72%
KO	COCA COLA CO THE	NYSE	40.09	49.00	6091500	48.43	48.43	48.25	48.25	-0,37%	-49,96%
DELL	DELL INC	NSDQ	19.91	32.00	13942030	25.10	25.10	25.09	25.09	-0,04%	-19,76%
GOOG	GOOGLE	NSDQ	337.06	509.65	2559797	462.00	462.00	460.48	460.48	-0,33%	-8,93%
INTC	INTEL CP	NSDQ	16.86	26.47	37142016	20.33	20.33	20.25	20.25	-0,39%	-1,46%
IBM	INTL BUSINESS MACH	NYSE	73.57	97.20	4390600	97.00	97.15	97.00	97.15	0,15%	22,12%
MSFT	MICROSOFT CP	NSDQ	21.51	30.19	41757264	29.85	29.86	29.85	29.86	0,03%	-50,00%
ORCL	ORACLE CORP	NSDQ	12.20	19.66	23592916	17.10	17.14	17.10	17.14	0,23%	-1,16%
RHT	RED HAT INC	NSDQ	14.83	32.27	2181800	23.00	23.00	23.00	23.00	0,00%	20,67%
YHOO	YAHOO INC	NSDQ	22.99	43.42	16305441	25.34	25.54	25.34	25.54	0,79%	-10,37%

Figura C.8: Ecrã de consulta de cotações com gráfico *candlestick*



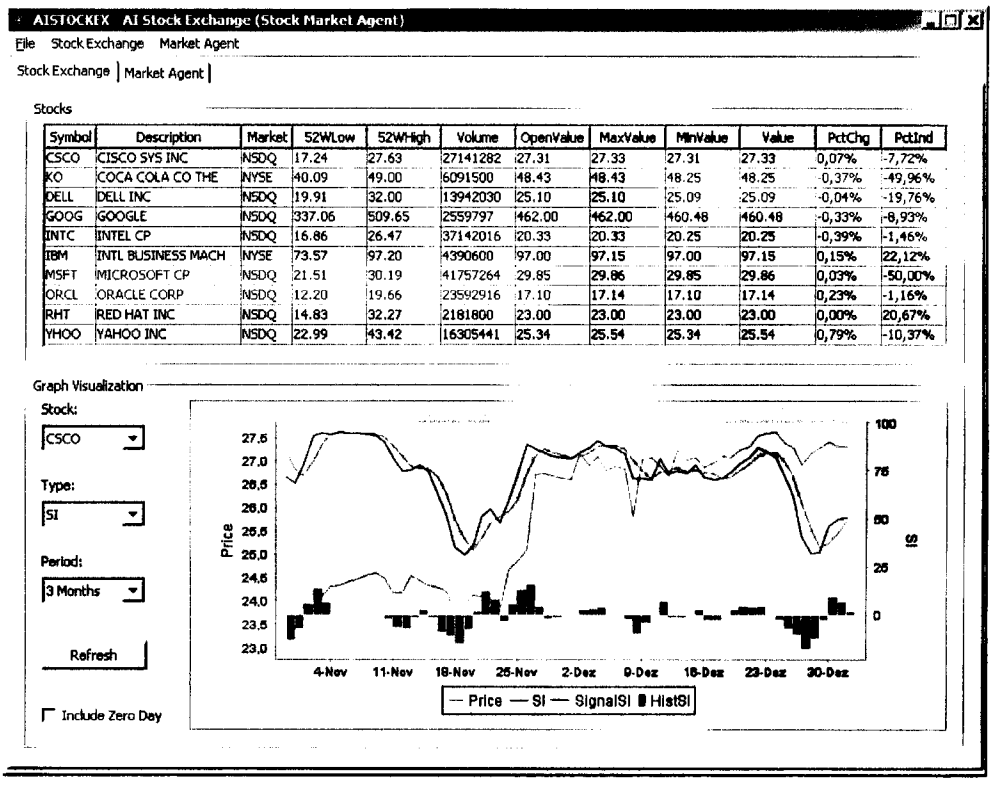
Symbol	Description	Market	52WLow	52WHigh	Volume	OpenValue	MaxValue	MinValue	Value	PctChg	PctInd
CSCO	CISCO SYS INC	NSDQ	17.24	27.63	27141282	27.31	27.33	27.31	27.33	0,07%	-7,72%
KO	COCA COLA CO THE	NYSE	40.09	49.00	6091500	48.43	48.43	48.25	48.25	-0,37%	-49,96%
DELL	DELL INC	NSDQ	19.91	32.00	13942030	25.10	25.10	25.09	25.09	-0,04%	-19,76%
GOOG	GOOGLE	NSDQ	337.06	509.65	2559797	462.00	462.00	460.48	460.48	-0,33%	-8,93%
INTC	INTEL CP	NSDQ	16.86	26.47	37142016	20.33	20.33	20.25	20.25	-0,39%	-1,46%
IBM	INTL BUSINESS MACH	NYSE	73.57	97.20	4390600	97.00	97.15	97.00	97.15	0,15%	22,12%
MSFT	MICROSOFT CP	NSDQ	21.51	30.19	41757264	29.85	29.86	29.85	29.86	0,03%	-50,00%
ORCL	ORACLE CORP	NSDQ	12.20	19.66	23592916	17.10	17.14	17.10	17.14	0,23%	-1,16%
RHT	RED HAT INC	NSDQ	14.83	32.27	2181800	23.00	23.00	23.00	23.00	0,00%	20,67%
YHOO	YAHOO INC	NSDQ	22.99	43.42	16305441	25.34	25.54	25.34	25.54	0,79%	-10,37%

Figura C.9: Ecrã de consulta de cotações com gráfico *high low with moving average*



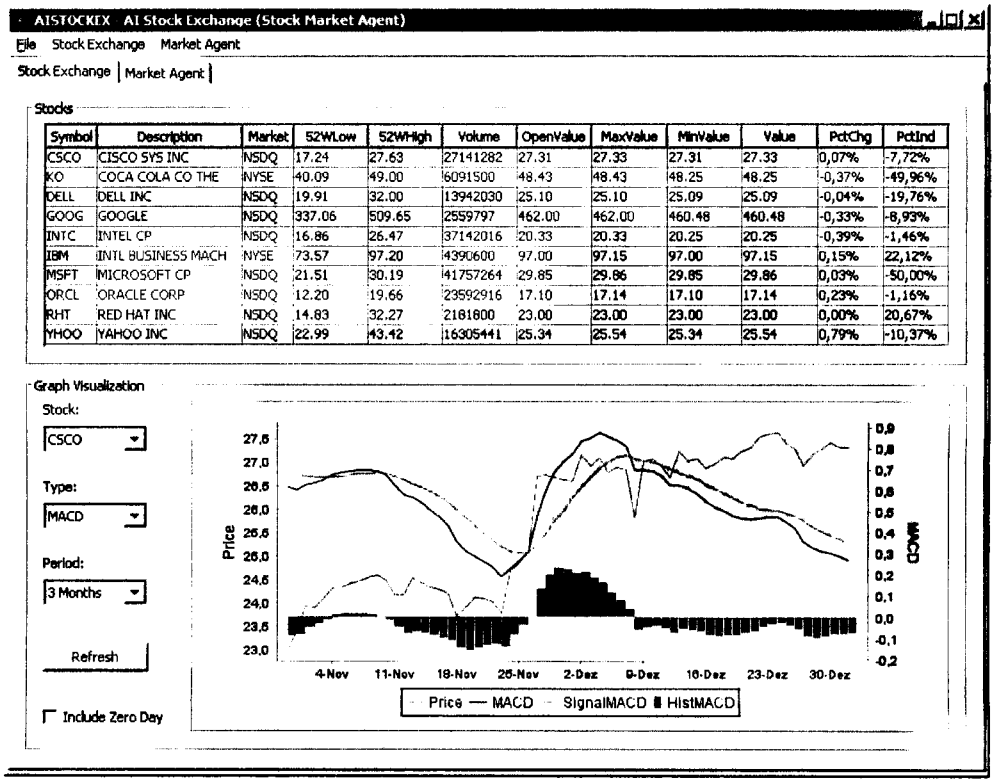
Symbol	Description	Market	52WLow	52WHigh	Volume	OpenValue	MaxValue	MinValue	Value	PctChg	PctInd
CSCO	CISCO SYS INC	NSDQ	17.24	27.63	27141282	27.31	27.33	27.31	27.33	0,07%	-7,72%
KO	COCA COLA CO THE	NYSE	40.09	49.00	6091500	48.43	48.43	48.25	48.25	-0,37%	-49,96%
DELL	DELL INC	NSDQ	19.91	32.00	13942030	25.10	25.10	25.09	25.09	-0,04%	-19,76%
GOOG	GOOGLE	NSDQ	337.06	509.65	2559797	462.00	462.00	460.48	460.48	-0,33%	-8,93%
INTC	INTEL CP	NSDQ	16.86	26.47	37142016	20.33	20.33	20.25	20.25	-0,39%	-1,46%
IBM	INTL BUSINESS MACH	NYSE	73.57	97.20	4390600	97.00	97.15	97.00	97.15	0,15%	22,12%
MSFT	MICROSOFT CP	NSDQ	21.51	30.19	41757264	29.85	29.86	29.85	29.86	0,03%	-50,00%
ORCL	ORACLE CORP	NSDQ	12.20	19.66	23592916	17.10	17.14	17.10	17.14	0,23%	-1,16%
RHT	RED HAT INC	NSDQ	14.83	32.27	2181800	23.00	23.00	23.00	23.00	0,00%	20,67%
YHOO	YAHOO INC	NSDQ	22.99	43.42	16305441	25.34	25.54	25.34	25.54	0,79%	-10,37%

Figura C.10: Ecrã de consulta de cotações com gráfico do indicador RSI



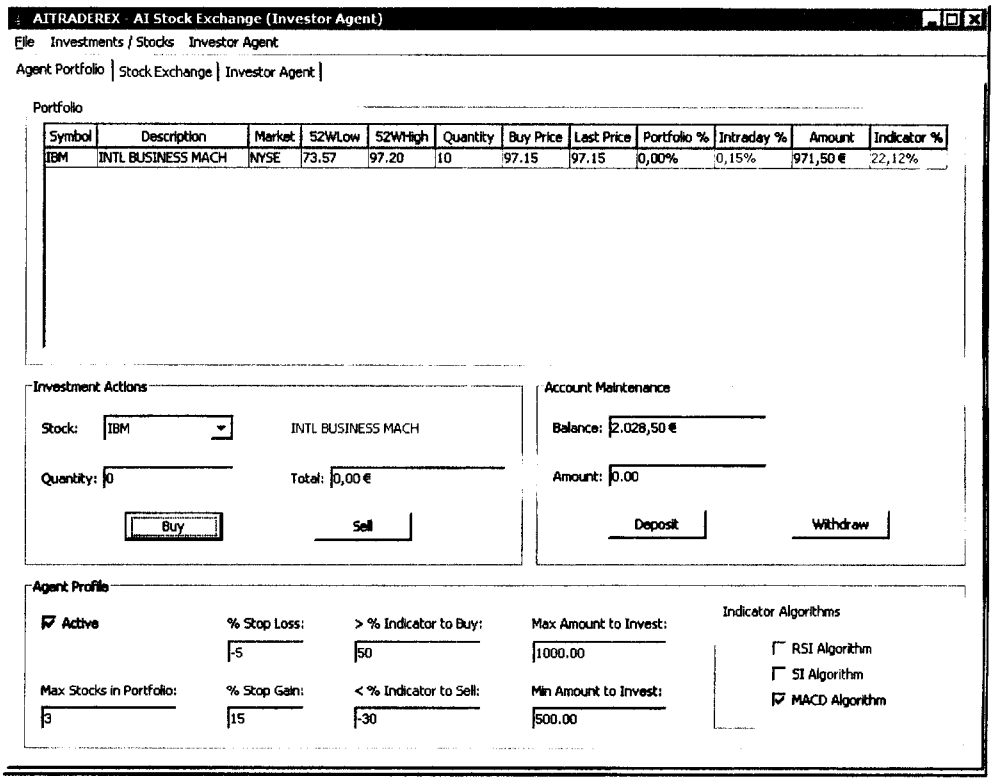
Symbol	Description	Market	52WLow	52WHigh	Volume	OpenValue	MaxValue	MinValue	Value	PctChg	PctInd
CSCO	CISCO SYS INC	NSDQ	17.24	27.63	27141282	27.31	27.33	27.31	27.33	0,07%	-7,72%
KO	COCA COLA CO THE	NYSE	40.09	49.00	6091500	48.43	48.43	48.25	48.25	-0,37%	-49,96%
DELL	DELL INC	NSDQ	19.91	32.00	13942030	25.10	25.10	25.09	25.09	-0,04%	-19,76%
GOOG	GOOGLE	NSDQ	337.06	509.65	2559797	462.00	462.00	460.48	460.48	-0,33%	-8,93%
INTC	INTEL CP	NSDQ	16.86	26.47	37142016	20.33	20.33	20.25	20.25	-0,39%	-1,46%
IBM	INTL BUSINESS MACH	NYSE	73.57	97.20	4390600	97.00	97.15	97.00	97.15	0,15%	22,12%
MSFT	MICROSOFT CP	NSDQ	21.51	30.19	41757264	29.85	29.86	29.85	29.86	0,03%	-50,00%
ORCL	ORACLE CORP	NSDQ	12.20	19.66	23592916	17.10	17.14	17.10	17.14	0,23%	-1,16%
RHT	RED HAT INC	NSDQ	14.83	32.27	2181800	23.00	23.00	23.00	23.00	0,00%	20,67%
YHOO	YAHOO INC	NSDQ	22.99	43.42	16305441	25.34	25.54	25.34	25.54	0,79%	-10,37%

Figura C.11: Ecrã de consulta de cotações com gráfico do indicador SI



Symbol	Description	Market	52WLow	52WHigh	Volume	OpenValue	MaxValue	MinValue	Value	PctChg	PctInd
CSCO	CISCO SYS INC	NSDQ	17.24	27.63	27141282	27.31	27.33	27.31	27.33	0,07%	-7,72%
KO	COCA COLA CO THE	NYSE	40.09	49.00	6091500	48.43	48.43	48.25	48.25	-0,37%	-49,96%
DELL	DELL INC	NSDQ	19.91	32.00	13942030	25.10	25.10	25.09	25.09	-0,04%	-19,76%
GOOG	GOOGLE	NSDQ	337.06	509.65	2559797	462.00	462.00	460.48	460.48	-0,33%	-8,93%
INTC	INTEL CP	NSDQ	16.86	26.47	37142016	20.33	20.33	20.25	20.25	-0,39%	-1,46%
IBM	INTL BUSINESS MACH	NYSE	73.57	97.20	4390600	97.00	97.15	97.00	97.15	0,15%	22,12%
MSFT	MICROSOFT CP	NSDQ	21.51	30.19	41757264	29.85	29.86	29.85	29.86	0,03%	-50,00%
ORCL	ORACLE CORP	NSDQ	12.20	19.66	23592916	17.10	17.14	17.10	17.14	0,23%	-1,16%
RHT	RED HAT INC	NSDQ	14.83	32.27	2181800	23.00	23.00	23.00	23.00	0,00%	20,67%
YHOO	YAHOO INC	NSDQ	22.99	43.42	16305441	25.34	25.54	25.34	25.54	0,79%	-10,37%

Figura C.12: Ecrã de consulta de cotações com gráfico do indicador MACD



Symbol	Description	Market	52WLow	52WHigh	Quantity	Buy Price	Last Price	Portfolio %	Intraday %	Amount	Indicator %
IBM	INTL BUSINESS MACH	NYSE	73.57	97.20	10	97.15	97.15	0,00%	0,15%	971,50 €	22,12%

Figura C.13: Ecrã exemplo do *portfolio* de um investidor

Lista de Abreviaturas

AMR	Agent Message Router
ANS	Agent Name Service
API	Application Programming Interface
Arl	Activo real líquido
Ca	Cotação da acção
CDDL	Common Development and Distribution License
CDR	Center for Design Research
CMVM	Comissão do Mercado de Valores Mobiliários
CSCO	Cisco Systems, Inc.
DELL	Dell Inc.
EPS	Earnings Per Share
EPS	Encapsulated Postscript
FIPA	Foundation for Intelligent Physical Agents
FIPA-ACL	Foundation for Intelligent Physical Agents - Agent Communication Language
FTP	File Transfer Protocol
Gw	Goodwill
Hi	Cotação máxima no período
HistMACD	Histograma do MACD
HistSI	Histograma do Stochastic Indicator
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corp.
IDE	Integrated Development Environment
IMACD	Indicador do MACD
INTC	Intel Corporation
IRSI	Indicador do Relative Strength Index
ISI	Indicador do Stochastic Indicator
JADE	Java Agent DEvelopment Framework
JATLite	Java Agent Template, Lite
JPEG	Joint Photographic Experts Group
JVM	Java Virtual Machine
KO	The Coca-Cola Company
KQML	Knowledge Query and Manipulation Language
KSE	Knowledge Sharing Effort

LGPL	Lesser General Public Licence
Lo	Cotação mínima no período
MACD	Moving Average Convergence / Divergence
MaxHistMACD ..	Valor Máximo do Histograma do MACD
MaxHistSI	Valor Máximo do Histograma do Stochastic Indicator
MinHistMACD ..	Valor Mínimo do Histograma do MACD
MinHistSI	Valor Mínimo do Histograma do Stochastic Indicator
Nasdaq	North American Securities Dealers Automated Quotation System
Nma	Número médio ponderado de acções ordinárias
NYSE	New York Stock Exchange
OO	Orientação ao objecto
ORDBMS	Object-Relational Database Management System
PCF	Price Cash Flow
PDF	Portable Document Format
PER	Price Earning Ratio
PNG	Portable Network Graphics
RHT	Red Hat, Inc.
Rle	Resultado líquido do exercício
Rlu	Resultado líquido unitário
RSI	Relative Strength Index
SI	Stochastic Indicator
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SVG	Scalable Vectorial Graphics
TCP/IP	Transmission Control Protocol / Internet Protocol
TriggerMACD ...	Trigger do MACD
TriggerSI	Trigger do Stochastic Indicator
UDP	User Datagram Protocol
Ve	Valor de uma empresa
WWW	World Wide Web
YHOO	Yahoo Inc.
YIHAT	Young Intelligent Hackers Against Terrorism

Referências

[Bra98] J. Bradshaw. *Handbook of Software Agents.* AAAI/MIT Press, 1998.

[DD02a] H. Deitel and P. Deitel. *Advanced Java 2 Platform - How To Program.* Prentice-Hall, 1st edition, 2002.

[DD02b] H. Deitel and P. Deitel. *Java - How To Program.* Prentice-Hall, 4th edition, 2002.

[Fer99] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1st edition, 1999.

[FLM96] T. Finin, Y. Labrou, and J. Mayfield. *KQML as an agent communication language.* Bradshaw, F., "Software Agents", MIT Press, 1996. http://www.flacp.fujitsulabs.com/ yannis/publications/mitpress96.pdf.

[Gil00] D. Gilbert. JFreeChart 1.0.2, 2000. http://www.jfree.org/jfreechart/.

[Gru93] T. Gruber. A Translation Approach to Portable Ontology Specifications, 1993. http://tomgruber.org/writing/ontolingua-kaj-1993.htm.

[Gru95] T. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal Human-Computer Studies Vol.43, Issues 5-6, 1995. http://tomgruber.org/writing/onto-design.htm.

[KD99] H. Kopka and P. Daly. *A Guide to Latex.* Addison-Wesley, 3rd edition, 1999.

[KJ98] M. Knapik and J. Johnson. *Developing Intelligent Agents for Distributed Systems: Exploring Architecture, Tecnologies and Applications.* McGraw Hill, 1998.

[MA05] M. Mendes and J. Almeida. *Preparação de Textos Científicos Usando o Latex.* Edições Sílabo, 1st edition, 2005.

[Mar52] H. Markowitz. *Portfolio Selection.* Journal of Finance, 1952.

[MT98] A. Mota and J. Tomé. *Mercado de títulos - Uma abordagem integrada.* Texto Editora, 5th edition, 1998.

[Myc00] Sun Mycrosystems. NetBeans IDE 5.0, 2000. http://www.netbeans.org/.

[Nev02] J. Neves. *Análise Financeira - Vol.I Técnicas fundamentais.* Texto Editora, 14th edition, 2002.

[Nwa96] H. Nwana. *Software Agents: An Overview.* Knowledge Engineering Review, 1996.

[Pei00] J. Peixoto. *Introdução à análise técnica.* Vida Económica, 1st edition, 2000.

[Pos95] PostgreSQL. PostgreSQL 8.1, 1995. http://www.postgresql.org/.

[Ree03] L. Reeves. Java Financial Libray 1.6.1, 2003. http://www.neuro-tech.net/2003/03/01/java-financial-library-161/.

[RG95] A. Rao and P. Georgeff. *BDI Agents: from Theory to Practice.* Proceedings of the First International Conference on Multi-Agent Systems, 1995.

[RN96] S. Russel and P. Norving. *Artificial Intelligence - A Modern Approach.* Prentice Hall, 2nd edition, 1996.

[Sil99] A. Silva. *Agentes de Software na Internet.* Edições Centro Atlântico, 1999.

[Uni96] Stanford University. JATLite Beta 0.4, 1996. ftp://java.stanford.edu/JATLite/.

[Wei99] G. Weiss. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence.* MIT Press, 1999.

[WJ95] M. Wooldridge and N. Jennings. *Intelligent Agents: Theory and Practice.* Cambridge University Press, 1995. http://www.csc.liv.ac.uk/ mjw/pubs/ker95.pdf.

[WR99] M. Wooldridge and A. Rao. *Foundations of Rational Agency.* Kluwer Academic Publishers, 1999.