FEUP Universidade do Porto
Faculdade de Engenharia

Licenciatura em Engenharia Electrotécnica e de Computadores

Projecto/Seminário/Trabalho Final de Curso

Ano lectivo 2002/03

Simulação de uma Bolsa Virtual usando Agentes

http://www.fe.up.pt/~ee95086/PSTFC

Aluno
Daniel José dos Santos Perfeito

Orientadores
Eugénio Oliveira
Ana Paula Rocha

Local de trabalho
Laboratório de Inteligência Artificial Distribuída

Data
21 de Julho de 2003

Índice

Lista de figuras

Lista de tabelas

Lista de gráficos

1. Introdução

1.1. Objectivo do trabalho

Este trabalho tem como objectivo a implementação de um Sistema Multi-Agente para simulação de uma Bolsa de Valores. Este sistema é composto por um agente Bolsa e vários agentes Investidores, cujo objectivo é a maximização do seu lucro.

Num mercado de valores, a grande quantidade de dados e informação presente a um ritmo elevado, torna útil a existência de uma entidade computacional que faça uso do seu poder de computação para gerir e assimilar toda a informação. Esta entidade computacional deve ser dotada de capacidade de raciocínio e decisão nas áreas económica e financeira para agir em proveito do investidor que representa, e em conformidade com o seu perfil.

O Sistema Multi-Agente (SMA) implementado é composto por um agente Bolsa e múltiplos agentes Investidores. É ainda disponibilizada uma interface gráfica, quer para parametrização do agente Investidor, quer para visualização e monitorização dos agentes Investidor e Bolsa. O SMA implementado usa a plataforma de comunicação JATLite.

Para simular o funcionamento real de uma Bolsa, o agente Bolsa deve usar (dentro do possível) informação real dos mercados, relativa a cotações, volumes transaccionados, variações de cotação, etc.…: esta informação pode ser retirada de diversos serviços de cotações on-line.

O agente Investidor deve deliberar sobre as informações presentes na Bolsa, e decidir quando comprar e/ou vender, sempre com o objectivo de obter o máximo rendimento para si. Este agente deve ser capaz de observar o comportamento dos outros agentes Investidores (e consequentemente da Bolsa), e decidir em que momento é mais rentável para si investir em quais acções. O agente Investidor dispõe inicialmente de um capital, constituído por um montante em dinheiro e uma carteira (conjunto de títulos). O agente investe o seu capital de acordo com o seu perfil (arriscado, prudente,...) e restrições várias. Neste trabalho foram implementadas duas estratégias para o agente investidor: com base no indicador MACD, e com base no indicador estocástico.

1.2. Motivação

A sociedade moderna depende cada vez mais de um grande volume de informação, a qual é debitada a uma grande velocidade. A necessidade de processar toda essa informação, a sua natureza dinâmica, a possibilidade e necessidade de integração de informação geograficamente dispersa, o crescimento da complexidade das aplicações e a necessidade de soluções inteligentes levam à construção de sistemas automáticos que realizem o processamento de toda a informação necessária e a organizem de forma adequada para que a tarefa do seu utilizador seja facilitada.

Assim, num ambiente tão complexo como é uma Bolsa de Valores, é também útil que hajam sistemas de investimento automáticos com o objectivo de observar as oscilações do mercado num tempo tão curto que permita aos seus utilizadores realizar os melhores

investimentos sem que percam um tempo precioso em análises e cálculos, que estes sistemas podem fazer automaticamente e em tempo real.
Deste modo, é importante para um investidor ter uma aplicação deste tipo que seja parametrizada por ele próprio, mediante as suas intenções de investimento, capaz de fazer investimentos em seu nome ou, pelo menos, alertá-lo para investimentos com boas perspectivas de sucesso, de uma forma automática e inteligente.

1.3. Âmbito do trabalho

Este trabalho foi realizado no âmbito da disciplina Projecto/Seminário/Trabalho Final de Curso, do 5º ano da Licenciatura em Engenharia Electrotécnica e de Computadores, Ramo de Sistemas Telecomunicações, Electrónica e Computadores, da Faculdade de Engenharia da Universidade do Porto, do ano lectivo de 2002/2003.

2. Áreas de contexto

2.1. Agentes de software

2.1.1. Introdução

Não existe uma definição consensual para a designação de "agente de software". Todavia, uma definição comum é que agentes são componentes de aplicações que actuam em nome dos seus donos/utilizadores, apresentando um conjunto básico de atributos reconhecidos, tais como: autonomia, persistência e sociabilidade. Adicionalmente podem apresentar características como: inteligência, aprendizagem, ou mobilidade [Sil99].
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.
Os agentes desencadeiam também alguma desconfiança e expectativa por parte dos seus potenciais utilizadores, da comunidade científica e da indústria informática em geral. Levantam problemas de ordem (1) técnica, tais como: segurança, contabilização de recursos, robustez e fiabilidade; (2) comportamental: o utilizador desconfia das reais capacidades e formas de actuação dos seus agentes; e (3) legal: por exemplo, quem é o responsável pela utilização indevida de recursos de um serviço? - o dono do agente, a empresa que produziu/construiu o agente, ou o gestor do serviço desprotegido?

A noção de "agente" é uma metáfora utilizada em inúmeras áreas do conhecimento, desde a Psicologia, Economia, Sociologia, Biologia até às Ciências da Computação. Foi a comunidade de Inteligência Artificial que introduziu pela primeira vez, nas Ciências da Computação, o conceito de agente. Esta primazia é geralmente atribuída ao sistema Actor [Hew77]. Neste sistema, o conceito de "actor" corresponde a um objecto auto-contido, interactivo e com execução concorrente. Cada objecto encapsula o seu estado interno e pode responder a mensagens de outros objectos similares.

> *"Um actor é um agente computacional que tem um endereço de correio electrónico e um comportamento. Os actores comunicam por troca de mensagens e executam as suas mensagens concorrentemente ".*

2.1.2. Definições e atributos de agentes

Na perspectiva da inteligência artificial existiu desde sempre a tentativa de definição do que são "agentes". Para esta comunidade, um agente é um sistema computacional, que para além das características básicas de autonomia, persistência, e sociabilidade, é também conotado com atributos geralmente reconhecidos ao ser humano [VB90, GK95, RG95, WJ95, Gil+95, FG96]. É normal na inteligência artificial a caracterização de agentes usando noções mentais, tais como crenças, intenções, e obrigações [Sho93]. Outras teorias vão mais longe no processo de antropomorfismo dos agentes ao ponto de os considerarem com emoções [Bat94].

Uma das mais compreensíveis definições de agentes é a de Wooldridge e Jennings [WJ95] baseada numa noção "forte" e numa noção "fraca". As duas noções, ou visões, de agentes correspondem a atributos que estes deverão possuir, os quais se dividem respectivamente em dois grupos: o de atributos essenciais e o de atributos opcionais.

A noção fraca baseia-se no conjunto mínimo de características que um agente deverá possuir, designadamente:

- **Autonomia**: Os agentes operam sem a intervenção directa dos seus ou outros utilizadores, e têm algum tipo de controlo sobre as suas acções e o seu estado interno.

- **Sociabilidade**: Os agentes interactuam com outros agentes e possivelmente com os seus utilizadores. Esta função requer a existência de um meio de comunicação que permita aos agentes informar os outros de quais são os seus requisitos; e um mecanismo interno de decisão sobre como e quando as interacções com os outros são apropriadas.

- **Reactividade**: Os agentes analisam o seu ambiente, o qual pode ser o mundo físico; um utilizador através da sua interface gráfica; uma colecção de outros agentes; a Internet; ou talvez todos eles combinados, e respondem às alterações nele ocorridas. Estes agentes são designados por reactivos. Baseiam-se sobre três componentes principais: percepção, acção e comunicação.

- **Pró-actividade** (ou orientação por objectivos): Os agentes não actuam apenas em resposta a alterações no seu ambiente mas também apresentam comportamento conduzido por objectivos e são capazes de tomar iniciativa na realização de determinadas acções.

- **Persistência**: Os agentes mantêm o seu estado interno ao longo da sua existência.

Todavia, a noção de agente, particularmente para a comunidade de inteligência artificial, apresenta um significado mais forte do que a noção (fraca) apresentada. Para esta comunidade, um agente é um sistema computacional, que para além dos atributos referidos, é também representado com atributos antropomorfos, tais como mentais ou mesmo emocionais. Adicionalmente é exigida uma representação simbólica do ambiente envolvente, capacidades cognitivas e capacidades de aprendizagem. Eis alguns dos atributos geralmente considerados adicionais à determinação de agente:

- **Mobilidade**: A capacidade de um agente se mover numa rede de modo a realizar as suas tarefas e cumprir os seus objectivos. Diz-se agente móvel caso tenha capacidade de se mover, caso contrário, diz-se estático ou estacionário.

- **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.

- **Aprendizagem**: A capacidade de aprendizagem está intrinsecamente associada com a capacidade de manipulação e geração de conhecimento [AFJM95, BSY95]. Os agentes com capacidade de aprendizagem vão, ao longo da sua existência, reconhecendo padrões de comportamentos, padrões de preferências, etc., e actualizando a sua base de conhecimentos.

- **Veracidade**: A característica de um agente não comunicar (deliberadamente) informação falsa. Quer seja com o seu utilizador, quer seja com os agentes com que interactua.

- **Racionalidade**: A característica de um agente (racional) não aceitar objectivos impossíveis de concretizar, contraditórios com os seus, ou ainda que não sejam compensadores em termos do risco/custo/esforço envolvido.

- **Benevolência**: A assunção que um agente não tem objectivos contraditórios, e que todo o agente procura realizar as tarefas que lhe foram solicitadas. Um agente é benevolente se adopta os objectivos dos outros, caso lhe seja solicitado, desde que estes não sejam incompatíveis com os seus.

- **Características mentais**: A definição de um modelo de agente baseado em noções mentais - conhecimento, crenças, intenções ou desejos - é uma área de actividade importante no seio da comunidade de inteligência artificial. De entre as teorias envolvidas destacam-se os trabalho de Rao e Georgeff baseado em agentes BDI (agentes com crenças, desejos e intenções) [RG95] e de Wooldrige e Jennings com agentes baseados em atitudes e pró-atitudes [WJ95].

Inúmeros autores têm proposto definições de agentes [FG96, Gil+95, Coe96, Nwa96]. Todas elas apresentam similaridades na identificação das características elementares (visão "fraca"), mas diferenças significativas na respectiva forma de classificação. Apresentam-se de seguida algumas dessas definições:

1. Para Franklin e Graesser [FG96],

> *"Um agente (autónomo) é um sistema situado dentro e participante de um ambiente, que percebe (sente) esse ambiente e actua nele de forma a concretizar a sua própria agenda e como consequência, de forma a afectar a sua visão sobre o futuro".*

Estes autores discutem taxiomias de agentes baseada quer em modelos biológicos (i.e., baseada em hierarquias de tipos), quer em modelos matemáticos (i.e., baseada em colecções de atributos).

2. Outra visão menos elaborada de agente, é proposta por Gilbert et al. [Gil+95]:

> *"Um agente (inteligente) é uma entidade software que executa um conjunto de tarefas em nome do seu utilizador, ou em nome de outro programa, com um determinado grau de independência e de autonomia, e consequentemente, utiliza o conhecimento (os objectivos e desejos) do seu utilizador".*

O agente é caracterizado em termos de três vectores principais: autonomia, mobilidade, e inteligência. Inteligência é entendida como o grau de gestão de preferências, raciocínio, capacidade de planeamento e de aprendizagem patenteada pelo agente.

3. Coelho [Coe96] define agente como:

> *"Uma entidade física ou abstracta capaz de agir sobre ela própria e sobre o seu ambiente, que pode comunicar com outros agentes, que persegue um objectivo individual, e cujo comportamento é uma consequência das suas observações, dos seus conhecimentos, das suas competências e das interacções que pode ter com os outros agentes".*

Nesta definição o agente possui objectivos, capacidade de comunicação com outros agentes e tem um grau de inteligência que lhe permite aprender em contacto com o ambiente que o rodeia [Mar99].

4. Para Nwana [Nwa96],

> *"Um agente é um componente de software e/ou de hardware capaz de actuar de forma a resolver tarefas em nome do seu utilizador".*

Esta é propositadamente uma definição suficientemente geral de forma a englobar inúmeros sistemas. É na particularização de agentes que Nwana os consegue tipificar e caracterizar. Propõe uma tipificação de agentes segundo um espaço multi-dimensional, em que são considerados os seguintes principais critérios de análise: (1) a mobilidade; (2) a intencionalidade (vs. reactividade); (3) a funcionalidade, i.e., segundo as suas principais especializações (e.g., agentes de pesquisa de informação; agentes de apresentações; agentes de compra de produtos); e (4) o seguinte conjunto mínimo de atributos ideais de agente: autonomia, aprendizagem e cooperação. A combinação cruzada dos referidos atributos gera, segundo Nwana, os seguintes tipos de agentes: autónomos, cooperativos, com capacidade para aprendizagem, colaborativos, de interface, colaborativos e com capacidade para aprendizagem, e espertos (quando reúnem conjuntamente os três atributos ideais). Por motivos de facilidade de apresentação e discussão, Nwana, converteu o seu espaço multi-dimensional numa lista unidimensional, com apenas sete tipos de agentes que considerou relevantes: colaborativos, de interface, móveis, de informação/Internet, reactivos, híbridos, e espertos.

2.1.3. Arquitecturas de agentes

A Inteligência Artificial classifica os agentes computacionais em três grandes categorias: agentes deliberativos, agentes reactivos e agentes híbridos [WJ94].

- **Agente deliberativo**

  Um agente deliberativo possui um modelo simbólico explícito do mundo, e realiza as suas decisões através de raciocínio lógico. O agente possui uma representação interna do mundo e a sua racionalidade pode ser modelada através de estados mentais (ver figura 2.1). Um estado mental descreve uma atitude ou posicionamento do agente relativo a uma informação.
  Estados mentais:
  - Informacionais
    - Conhecimento
    - Crença
  - Motivacionais
    - Desejos
    - Intenções
    - Expectativas
    - Planos
  - Sociais
    - Permissões
    - Obrigações

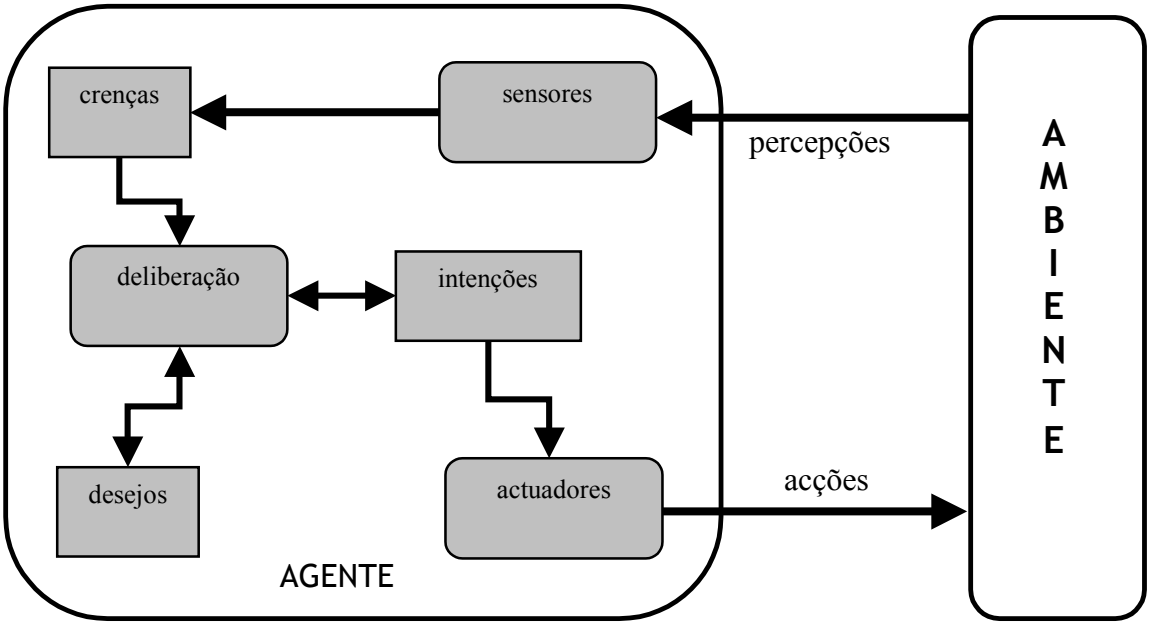


Figura 2.1 - Representação de um agente deliberativo.

- **Agente reactivo**

  Um agente reactivo é baseado em modelos de organização biológica, como, por exemplo, as sociedades de formigas. Embora uma formiga sozinha não possa ser considerada uma entidade inteligente, o formigueiro como um todo apresenta um comportamento visivelmente inteligente no sentido em que existe uma procura de alimentos e posterior armazenamento, uma organização da reprodução, etc.

O comportamento do agente é simplesmente uma resposta estereotipada a um conjunto de estímulos exteriores (semelhante ao comportamento animal). Não existe qualquer raciocínio baseado numa avaliação global do estado do mundo que justifique o conjunto de acções que devem ser desenvolvidas.

Assim, o modelo de funcionamento de um agente reactivo é o de *estímulo-resposta*. Em geral, estes agentes não apresentam memória, não planeiam sua acção futura e não comunicam com outros agentes, cada agente tomando conhecimento das acções e comportamentos dos outros agentes apenas através de modificações no ambiente (ver figura 2.2).

Um agente reactivo não inclui qualquer representação simbólica do mundo ou modelo, pois o mundo é o seu próprio melhor modelo e está sempre actualizado. O sistema está ligado ao mundo via sensores e actuadores. As vantagens deste tipo de agentes são a flexibilidade, a adaptação e tempos de resposta rápidos.

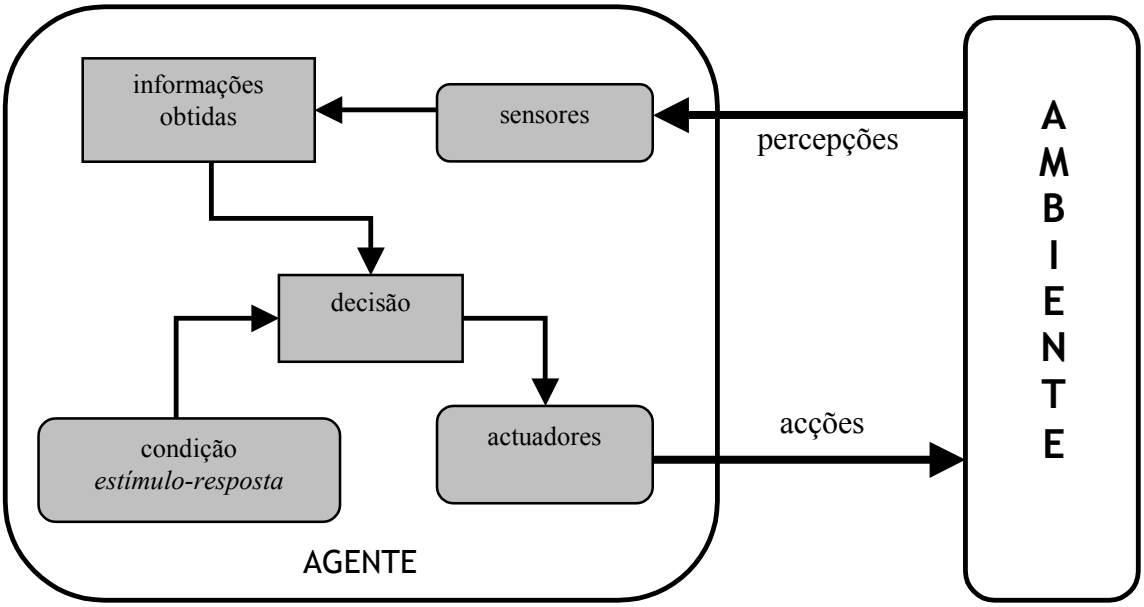


Figura 2.2 - Representação de um agente reactivo

- **Agente híbrido**

Enquanto os agentes puramente reactivos têm uma limitação muito importante, que é a dificuldade em implementar comportamento direccionado a um objectivo, os agentes puramente deliberativos são baseados em mecanismos de raciocínio muito complexos e tornam-se, por vezes, incapazes de uma reacção imediata a estímulos do exterior.

Um agente híbrido combina estas duas componentes, e caracteriza-se por uma arquitectura composta por níveis ou camadas. A possibilidade de realizar uma disposição por níveis é uma ferramenta poderosa de estruturar as funcionalidades e controlo do agente. A ideia principal é categorizar as funcionalidades do agente em camadas dispostas hierarquicamente onde geralmente a camada reactiva tem alguma

prioridade sobre a camada deliberativa, de modo a permitir uma resposta rápida aos eventos mais importantes detectados no ambiente (ver figura 2.3).

Os agentes híbridos, e porque a sua arquitectura apresenta uma disposição de componentes em níveis, podem ser incrementados dinamicamente, por adição de planos que são criados quando ocorrem eventos externos específicos, ou alterações no estado interno do agente [Roc01].

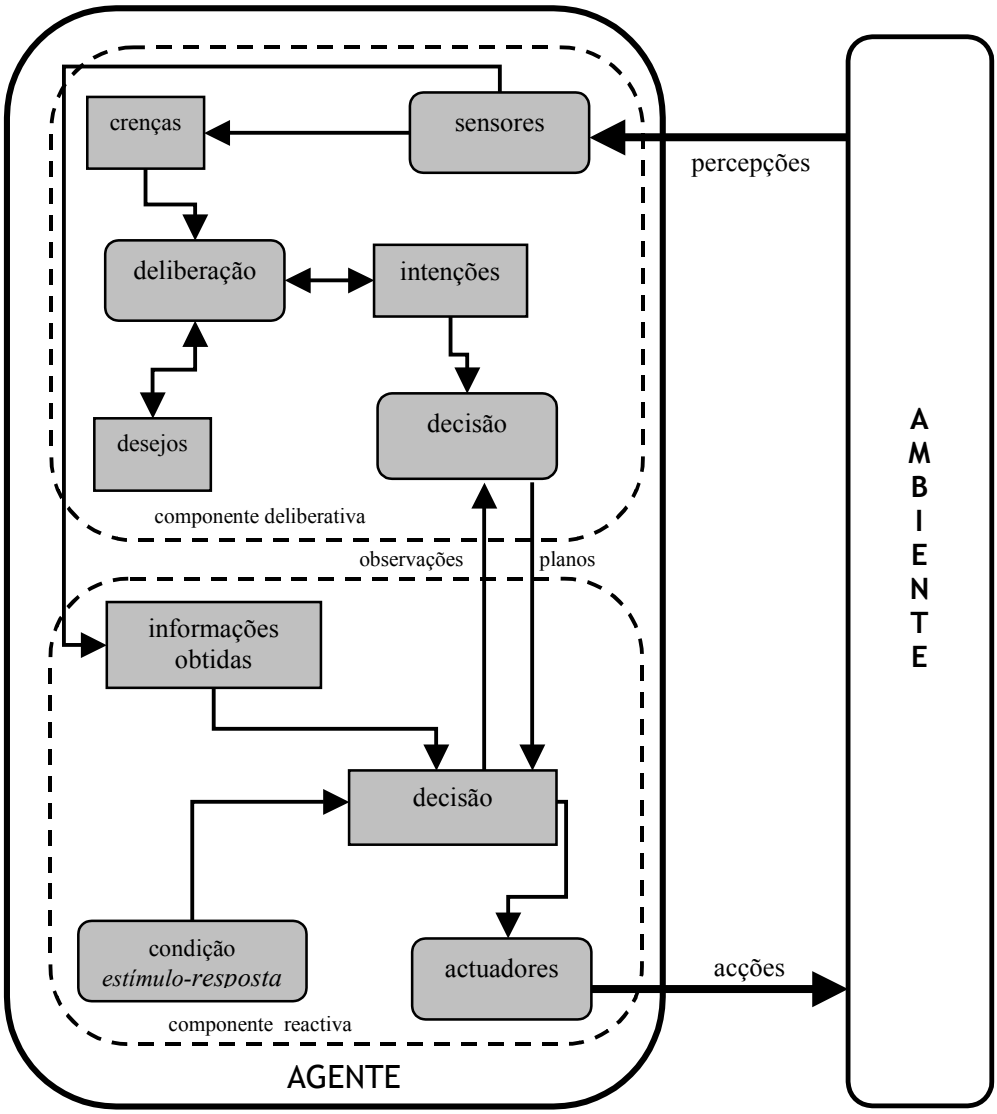


Figura 2.3 - Representação de um agente híbrido.

2.1.4. Sistema multi-agente

Um sistema multi-agente (SMA) é um sistema baseado na interacção de múltiplos agentes com um objectivo definido.

Os sistemas multi-agente permitem modelar o comportamento de um conjunto de entidades inteligentes e organizadas de acordo com leis do tipo social. Estas entidades, ou agentes, dispõem de uma certa autonomia e estão imersos num ambiente com o qual

necessitam interagir, pelo que devem possuir uma representação parcial deste ambiente e meios de percepção e comunicação.

Um *sistema multi-agente* (SMA) pode ser definido como um grupo de agentes que combinam as suas competências e cooperam com o intuito de satisfazer um objectivo comum. A adopção de mecanismos de cooperação eficiente atribuem ao sistema multi--agente uma capacidade superior à soma das capacidades individuais dos agentes, pois o desempenho emerge através das interacções dinâmicas que ocorrem naturalmente entre os agentes individuais [Roc01].

2.2. JATLite

2.2.1. Introdução

A plataforma JATLite (*Java Agent Template, Lite*) foi desenvolvida na Universidade de Stanford e é um pacote de programas escritos em linguagem Java que permitem ao utilizador uma rápida criação de agentes de software com capacidade de comunicação para Internet [W3JAT]. O JATLite possui uma infra-estrutura básica, mostrada na figura 2.4, na qual os agentes se registam num **Agent Message Router (AMR)** utilizando um nome e *password*, ligam-se e desligam-se da Internet, enviam e recebem mensagens, transferem ficheiros utilizando o protocolo FTP (*File Transfer Protocol*) e trocam geralmente a informação com outros agentes nos vários computadores onde estão a correr.

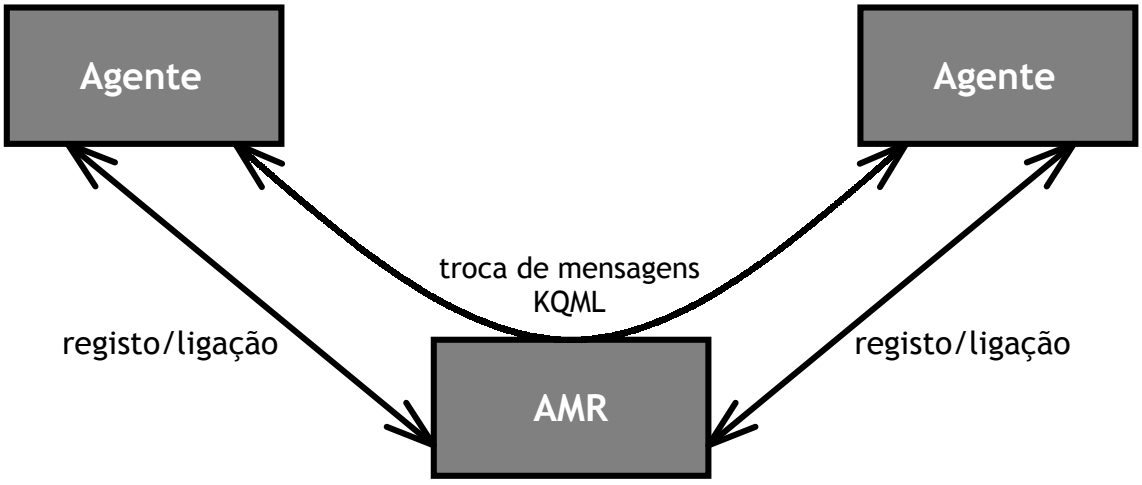


Figura 2.4 - O Agent Message Router (AMR) do JATLite.

O *Agent Message Router* é um único ponto de ligação para todos os agentes, que gere todo o processo de comunicação entre os agentes, em particular guarda todas as mensagens até que sejam explicitamente apagadas pelos agentes destinatários.

2.2.2. Arquitectura do JATLite

A arquitectura do JATLite é composta por uma estrutura hierárquica constituída pelas seguintes camadas:

- **Abstract Layer**

  Disponibiliza uma colecção de classes abstractas necessárias para a implementação do JATLite.

- **Base Layer**

  Possibilita as comunicações básicas utilizando TCP/IP e a camada abstracta.

- **KQML Layer**

  Possibilita o armazenamento e a análise (*parsing*) das mensagens KQML.

- **Router Layer**

  Permite o registo de nomes e o envio e recepção de mensagens via *router*, entre os nomes de origem e de destino. Quando um agente, acidentalmente ou devido a uma anomalia, se desliga, o *router* guarda as mensagens recebidas até que o agente se volte a ligar.

- **Protocol Layer**

  Esta camada suporta diversos serviços disponibilizados na Internet, nomeadamente, SMTP, FTP, POP3, HTTP, etc.

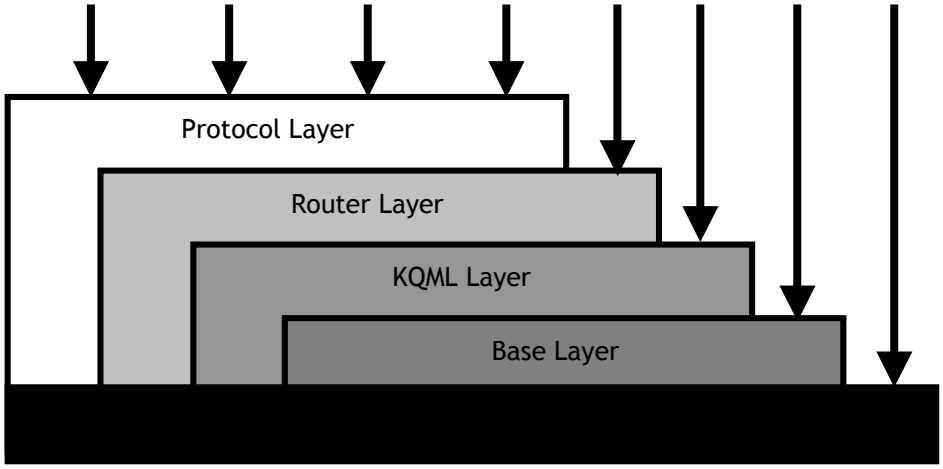


Figura 2.5 - Arquitectura de camadas do JATLite.

O JATLite fornece um *template* para construir agentes que utilizam uma linguagem de alto nível e um protocolo comuns. Este *template* fornece ao utilizador numerosas classes de Java predefinidas que facilitam a construção de agentes. Além disso, as classes são fornecidas nas camadas, de modo que o utilizador pode facilmente decidir que classes são necessárias para um dado sistema. Por exemplo, se o utilizador decidir que não quer usar KQML, as classes na camada de KQML são simplesmente omitidas. No entanto, se essa camada for incluída, as funções de análise gramatical e outras funções específicas do KQML são incluídas automaticamente.

2.2.3. Agent Message Router (AMR)

A característica mais original do JATLite é a infra-estrutura de agentes empacotada com a plataforma. Os sistemas tradicionais de agentes usam um Agent Name Server (ANS) para efectuar ligações entre agentes. Um agente usa um ANS simplesmente para procurar o endereço IP de um outro agente e então usa esse endereço para fazer uma ligação por *socket* TCP directamente a esse agente com a finalidade de trocar mensagens.

Com o ANS, se o endereço IP do agente receptor mudar, o agente emissor descobre-o quando a tentativa seguinte de emitir uma mensagem falha. Se o agente receptor deixar de funcionar de qualquer maneira, é a responsabilidade de cada agente emissor com quem este estava a comunicar conservar correctamente as mensagens falhadas para as reenviar mais tarde. É fácil ver como toda a computação distribuída poderia facilmente falhar sob tais circunstâncias.

O JATLite gere todo este processo de comunicação, resolvendo os problemas acima referidos. Todos os agentes fazem uma única ligação ao Agent Message Router (AMR). O AMR envia todas as mensagens do agente pelo nome ao último endereço IP conhecido. Mais ainda, tal como um sistema de e-mail, o AMR guarda todas as mensagens e conserva-as até que o agente de destino reconheça a sua recepção através do envio de uma mensagem de eliminação (*delete*) ao AMR.

Os agentes podem emitir mensagens para se ligar ou desligar (*connect/disconnect*) ao AMR, assim como devem enviar uma mensagem inicial do registo onde especificam o seu nome e IP. A computação distribuída é sempre preservada pelo AMR.

O Agent Message Router do JATLite é uma aplicação especializada que recebe uma mensagem dos agentes registados e a reenvia ao receptor correcto. As mensagens recebidas serão colocadas em fila de espera no sistema de ficheiros.

O uso do AMR tem diversas vantagens sobre um mais tradicional Agent Name Server (ANS):

- Um agente implementado num *applet* pode comunicar com outros agentes apesar das características da segurança do *applet* nos browsers populares da *Web* tais como o Netscape Navigator ou o Internet Explorer.

- Uma comunicação assíncrona é possível mesmo quando o agente receptor não pode receber uma mensagem.

- Um agente não necessita seguir os endereços (possivelmente em mudança) dos outros agentes ou preocupar-se sobre problemas temporários da entrega da mensagem.

Das várias funções do AMR, destacam-se as seguintes [W3JPC]:

- **Protecção de mensagens**

  Uma mensagem enviada ao AMR é guardada num sistema de ficheiros de modo que há pouca possibilidade de haver mensagens perdidas mesmo que um agente esteja

em migração ou tenha deixado de funcionar. Através do AMR, um agente pode recuperar as mensagens possivelmente perdidas quando foi desligado da rede. Um agente tem desse modo a flexibilidade para controlar seu processo e coordenar mensagens que envia. Por exemplo, se um agente prever que vai levar muito tempo para processar uma mensagem, o agente pode não querer acumular mensagens na própria memória. O agente pode desligar-se do AMR por vontade própria e o AMR guarda as mensagens recebidas enquanto o agente está desligado. Sempre que o agente se liga de novo ao AMR, o AMR enviará as mensagens guardadas ao agente. As mensagens guardadas serão eliminadas do sistema de ficheiros apenas se tal for pedido pelo agente receptor. Também, o agente pode sempre enviar uma mensagem a outros agentes, não dependendo do facto de o receptor estar ligado ou desligado.

- **Agent Name Service**

  O serviço Agent Name Service (ANS) é suportado por defeito. Um agente pode registar-se no AMR com o seu nome, endereço Internet e campo de descrição. O endereço do agente assim como a sua descrição não necessitam ser definidos estaticamente e podem ser mudados sempre que o agente se liga novamente ao AMR. O AMR segue endereços em mudança e descrições possivelmente variando para agentes registados. Simplesmente, cada agente pode inquirir sobre o endereço de um outro agente usando a performativa REQUEST-ADDRESS. O AMR responde com a performativa AGENT-ADDRESS que inclui o endereço Internet do agente pedido e o campo de descrição. Os campos de descrição do agente na performativa WHOIAM podem ser usados para estratégias diversas da coordenação. Assim, o AMR pode funcionar como um ANS simples se desejado.

- **Reencaminhamento de Mensagens**

  O agente emissor de uma mensagem envia esta para o AMR, que se encarrega de a reencaminhar para o agente receptor. Se o receptor estiver actualmente ligado ao AMR, o AMR emite a mensagem. Se o receptor tiver a funcionalidade de aceitar conexões por *sockets* (por exemplo, aplicações autónomas), o AMR tenta emitir a mensagem ao receptor iniciando uma ligação, diversas vezes até que a ligação esteja estabelecida. Se o receptor estiver intencionalmente desligado ou o AMR não puder iniciar uma ligação (por exemplo, uma ligação através de um *applet*), o AMR guarda as mensagens a ser emitidas quando o receptor se ligar ao AMR.

- **Comunicação de Agentes por Java *Applets***

  Como foi notado previamente, assim que há um AMR instalado no mesmo servidor que um Java *applet* de um agente, o agente pode usar o mecanismo de reenvio do AMR para trocar mensagens com outro agente registado. Este esquema não viola nenhum mecanismo da segurança do browser e não requer nenhuma instalação especial na parte do utilizador. O utilizador pode simplesmente "clicar" no URL para o *applet* do agente, fazer o download , ligar-se ao AMR se tal não for feito automaticamente pelo *applet*, ler as mensagens proeminentes, e responder-lhes ou enviar novas.

- **Manutenção de estado**

  As mensagens guardadas representam a parte do estado da computação distribuída. Assim, o AMR permite protecção de estado (*state-saving*) automático. Isto permite que o estado acompanhe um agente em migração, que pode ser um *applet*, ou *software* de aplicação que é executado em máquinas diferentes em horas diferentes. Por exemplo, um coordenador mecânico pode querer funcionar o mesmo programa térmico de análise em computadores diferentes em horas diferentes. Envolvendo este software com código baseado em JATLite, este programa pode ser usado com a mesma identidade em horas diferentes nos computadores diferentes que têm um papel consistente com outros programas da análise e da simulação no projecto iterativo de um artefacto.

- **Transferência de Ficheiros e E-mail**

  Além da passagem de mensagens, o AMR suporta ainda o FTP e o SMTP para agentes autónomos e *applets*. Usando o suporte do protocolo do FTP, um agente pode guardar/recuperar uma grande quantidade de dados para/de servidores FTP de qualquer parte na Internet. O suporte do protocolo SMTP torna possível que o agente emita dados para qualquer servidor SMTP. Para ambos os casos, para Java *applets*, o AMR faz o papel de uma *proxy* para criar uma ligação entre o *applet* e servidores FTP/SMTP sem colocar os dados em *cache* no AMR.

- **Protocolos administrativos**

  O AMR fornece o serviço de *password* para cada agente registado. Cada agente necessita registar-se com uma *password*, e esta *password* é verificada sempre que o agente se liga ao AMR. O serviço de *password* é independente do mecanismo específico da segurança da linguagem Java de modo que um agente implementado noutras linguagens possa utilizar o serviço. Além do serviço de *password*, o AMR executa outros protocolos administrativos que não são padrão de KQML [W3CDR]. Embora os protocolos sejam resultados da discussão aberta por e-mail, estão ainda sob discussão e necessitam um refinamento e um consenso das comunidades dos agentes de KQML.

2.2.4. Linguagem KQML

A linguagem KQML (*Knowledge Query and Manipulation Language*) é uma linguagem e um protocolo para troca de informação e conhecimento. É a linguagem de comunicação de agentes mais usual actualmente. 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: *conteúdo*, *mensagem* e *comunicação* [FLM97].

- **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 final). 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. A *intenção* de uma mensagem, especifica qual o tipo de interacção entre os agentes emissor e receptor, e obedece a um conjunto pré-estabelecido. Pode ainda incluir outras características opcionais descritivas, por exemplo, da identidade do receptor, do formato da informação, ou da ontologia[1] adoptada.

- **Comunicação**

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

Embora o KQML tenha um conjunto de performativas reservadas, este não está minimizado nem fechado. 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, ela deve ser implementada da forma padrão.

No Apêndice 2 encontra-se a descrição dos protocolos e performativas do AMR.

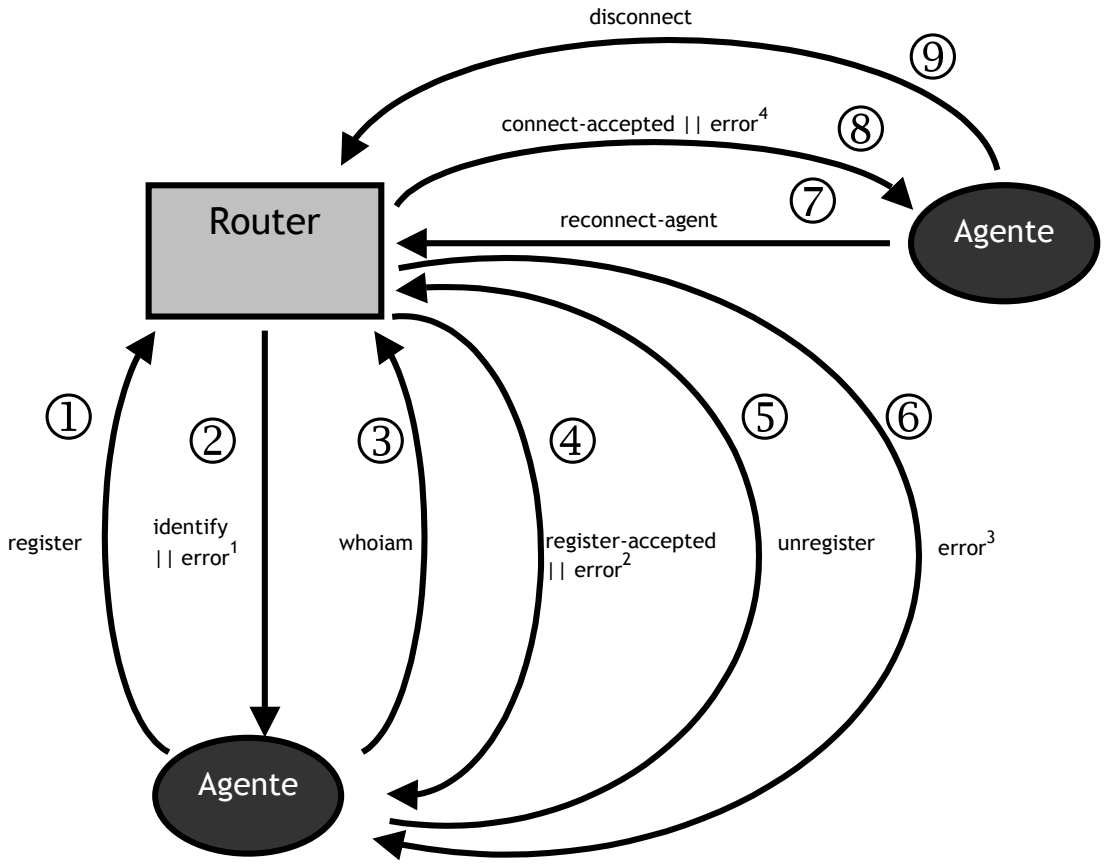
2.2.5. Troca de mensagens

A figura 2.6 mostra a sequência de mensagens trocadas para o registo de um agente. Um agente que se queira registar necessita de iniciar uma ligação ao AMR e emitir uma mensagem de registo (REGISTER) com o seu nome e a sua *password*. Se houver um agente registado com o mesmo nome, o pedido será negado e a mensagem de erro (ERROR) é emitida.

Caso contrário, o AMR emite uma mensagem de identificação (IDENTIFY) ao agente. Então, o agente necessita apresentar-se, pelo que usa a performativa WHOIAM que inclui o endereço do agente, a descrição, o endereço de e-mail (opcional), etc. Após o registo bem sucedido, o AMR emite uma mensagem de aceitação de registo (REGISTER-ACCEPTED) ao agente. Para anular o seu registo, um agente emite a performativa UNREGISTER com sua *password*. Se a *password* estiver incorrecta, a anulação do registo será negada. Uma vez registado, o agente pode ligar-se ao AMR emitindo uma mensagem de RECONNECT com a sua *password* e endereço. Se o endereço for diferente do endereço precedente, o AMR substitui a informação do

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

endereço. Se uma ligação do mesmo agente estiver ainda activa, a ligação será terminada e uma nova ligação será estabelecida. (Nota: Uma perda da ligação é detectada apenas quando uma mensagem necessita ser emitida. Uma vez detectada, a ligação é terminada pelo AMR). Um agente pode desligar-se intencionalmente do AMR emitindo uma mensagem de DISCONNECT.



① Mensagem 'register' especifica um nome e uma *password* para o agente.
error[1]: Se o nome de agente especificado já existir, será enviado 'error'.

③ Mensagem 'whoiam' especifica a informação de contacto *'contact-information'* (host, porta, enderço e--mail), método da mensagem *'message-method'*, extensões KQML *'KQML-extensions'* e descrição *'description'*.
error[2]: Se o *message-method* especificado na mensagem 'whoiam' não for suportado pelo *router*, será enviado 'error'.

⑦ Mensagem 'reconnect-agent' deve incluir o nome do agente e a sua *password*. A informação de contacto (host, porta, enderço e-mail) pode ser redefinida.
error[3]: Se a *password* na mensagem 'unregister' não for válida, será enviado 'error'.
error[4]: Se o nome de agente na mensagem 'reconnect-agent' não estiver registado, será ser enviado 'error'. Se a *password* na mensagem 'reconnect-agent' não for válida, será enviado 'error'.

Figura 2.6 - Protocolo de ligação ao AMR.

2.3. Java

2.3.1. Introdução

O Java não é apenas uma linguagem de programação. É antes de mais, um sistema completo de desenvolvimento e execução de aplicações portáveis [Sil99].

O sistema Java encontra-se integrado no designado JDK (*Java Development Kit*). O JDK providencia a generalidade de ferramentas (e.g., compilador, interpretador, depurador), documentação e bibliotecas standard de classes necessárias ao desenvolvimento de aplicações Java.
A linguagem Java foi descrita originalmente através de um conjunto de chavões/atributos, como sendo uma *"linguagem simples, familiar, orientada por objectos, distribuída, independente da plataforma, portável, interpretada, dinâmica, robusta, segura, de elevado desempenho, e com suporte a múltiplas sequências de execução (multi-threaded)"* [Sun95]. Analisa-se de seguida algumas das suas principais características [Sil99].

2.3.2. Linguagem baseada em objectos e simples

Contrariamente ao seu antecessor, C++, que é uma linguagem híbrida (por suportar simultaneamente os paradigmas da programação estruturada e baseada em objectos), a linguagem de programação Java é uma linguagem exclusivamente baseada em objectos. Excepto no suporte a tipos de dados nativos, tais como caracteres ou numéricos, todas as entidades em Java são tipos ou suas instâncias, i.e., objectos. Existem basicamente dois grupos de tipos: classes, que correspondem a tipos abstractos ou concretos, com dados e métodos; e interfaces, que correspondem a tipos abstractos com apenas especificação de métodos (de interface). Por ser uma linguagem orientada a objectos pura, providencia naturalmente capacidades de especialização de tipos (classes e interfaces), implementação de interfaces, encapsulamento e polimorfismo. Por motivos de simplicidade e de desempenho, não é suportada herança múltipla. Em substituição do mecanismo de herança múltipla, permite-se que uma classe seja uma especialização de uma outra classe mais abstracta, e que possa simultaneamente implementar uma ou mais interfaces.

A linguagem Java é simples e desenhada de forma extremamente elegante. Entre outras características o facto de não permitir a manipulação directa de memória, através de apontadores; e de esconder ao programador a gestão (alocação/libertação) de memória, ao providenciar um mecanismo automático de reciclagem de memória; eliminou consideravelmente a maioria dos problemas que os programadores de C/C++ enfrentam regularmente. Adicionalmente estas características introduzem, na própria linguagem, mecanismos de segurança significativos.

2.3.3. Bibliotecas *standard* de tipos

O Java inclui no JDK uma biblioteca considerável de tipos, que variam entre suporte básico de estruturas de dados (por exemplo tabelas de *hash*, dicionários), até tipos com elementos para construção de interfaces gráficas, passando por tipos para criação e gestão de aplicações, de processos, etc.

Por motivos de gestão, cada biblioteca de tipos encontra-se organizada num "pacote" (*package*). Cada pacote é identificado pela concatenação de um conjunto de identificadores estruturados de forma hierárquica, em que os identificadores mais à esquerda correspondem a agregações mais gerais, e os mais à direita, a agregações mais específicas.

Exemplos de identificadores de pacotes standard são: `java.lang`; `java.util`; `java.awt.applet`; `java.io`.
Apesar da linguagem Java ser simples, a maior dificuldade no desenvolvimento em Java é devida à necessidade da aprendizagem de um número elevado (mais de duas centenas) de classes e de interfaces existentes .

2.3.4. 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) formato intermédio, o qual é executado pela máquina virtual Java.

A independência da plataforma é conseguida conjuntamente através dos seguintes mecanismos: (1) definição de *bytecodes* independentes da máquina; (2) tipos de dados primitivos (caracter, inteiro, real, etc.) de tamanho fixo; (3) representação de código independente da rede (através da representação/ordenação de *bytes* segundo o formato *big-endian*); e (4) existência de uma biblioteca standard de classes.

O principal beneficio 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, computadores servidores, microprocessadores para electrodomésticos, telefones móveis, etc.

2.3.5. Segurança

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

- Ao nível da linguagem, Java é uma linguagem segura (*safe*). Neste nível 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: (1) não permite a manipulação de apontadores; (2) as conversões de tipos (*casts*) são controladas de forma a não serem violadas as regras da linguagem; (3) a gestão de 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.

- Compete ao programador de aplicações definir a política de segurança de cada aplicação em particular, a qual é estabelecida através de uma instância da classe `SecurityManager`, que providencia controlo de acessos, entre outros, aos seguintes tipos de recursos: (1) ao sistema de ficheiros locais (e.g., métodos das classes `File`, `FileStreamlnput`); (2) ao sistema (e.g., métodos das classes `System`, `EventQueue`); (3) à rede (e.g., métodos das classes `ServerSocket`, `DatagramSocket`); (4) manipulação do interpretador (e.g., métodos das classes `Thread`; `ThreadGroup`, `Runtime`); (5) manipulação da biblioteca (e.g., métodos das classes `Security`, `URL`); (6) manipulação do modelo de segurança (e.g., métodos da classe `Class`); e (7) criação de janelas (ao construtor da classe `Windows`).

2.3.6. Desempenho

O código Java compilado foi desenhado originalmente para ser compacto e não eficiente, já que o seu objectivo original era ser uma linguagem para redes de computadores, i.e., em que o código migrasse entre nós de uma rede, em vez de ser simplesmente carregado da memória secundária para memória RAM. Por essa razão, o desempenho de uma aplicação ou *applet* (mini-aplicação) Java não é comparável com o homólogo de uma aplicação compilada directamente para código nativo. Apesar de tudo, e em aplicações tipicamente de interacção homem-máquina, o desempenho de aplicações Java é geralmente tolerável.

No entanto para ultrapassar a limitação do desempenho das aplicações Java, muitos fabricantes começaram a providenciar compiladores JIT (*Just-In-Time*) conjuntamente com os interpretadores. O compilador JIT é invocado após o carregamento do código Java, convertendo esse código para código nativo, sendo seguidamente executado pela máquina real. Apesar de máquinas virtuais com compiladores JIT oferecerem tempos de execução superiores a máquinas virtuais tradicionais, esses tempos ainda são consideravelmente limitados em comparação com a execução de código nativo. A principal razão é a dificuldade de introdução de técnicas efectivas de optimizações de código.

2.4. Análise técnica da Bolsa

2.4.1. Introdução

Poucas actividades humanas têm sido tão exaustivamente estudadas durante os últimos 50 anos, sob tantos ângulos e por tão diferentes tipos de pessoas, como a actividade de comprar e vender acções de empresas.
No decorrer destes anos de estudo, dois tipos de análise distintos tomaram forma, dois métodos radicalmente diversos de dar resposta ao problema dos investidores, de dizer o que e quando comprar ou vender.
A primeira destas análises é conhecida como Fundamental; a segunda como Técnica.

2.4.2. Análise Fundamental e Análise Técnica

- **Análise Fundamental**

A Análise Fundamental (AF) é o estudo da envolvente sócio-económica e política onde a empresa se insere, com o objectivo de determinar o valor da empresa [W3AT].

Assim, a AF envolve três passos básicos:
1. Estudo da economia.
2. Estudo do sector de actividade em que a empresa está envolvida.
3. Estudo dos rácios da empresa.

Comecemos por avaliar o primeiro ponto. Não há nenhuma empresa que possa dissociar a estratégia do negócio da envolvente macro-económica. A evolução da inflação, do índice do desemprego, das taxas de juro, do mercado cambial são indicadores que influenciam de forma decisiva a actividade da empresa. E note-se que não são apenas os indicadores do país onde a empresa se insere. Com a globalização, as empresa podem ser afectadas por indicadores de outros países que se encontram a milhares de quilómetros de distância da empresa e que até podem não ter qualquer relação directa com a empresa. Por exemplo: caso as taxas de juro aumentem, a cotação das empresas em bolsa tenderá a diminuir pois o endividamento das empresas tenderá a aumentar. Essa é a reacção típica do mercado de capitais.

Segundo ponto: o sector de actividade. A empresa pode ser bem gerida, ter óptimos rácios mas se a envolvente no sector de actividade for negativa, o mercado de capitais geralmente reage e penaliza a empresa em Bolsa. Um exemplo: o sector da construção civil e obras públicas em Portugal está um pouco estagnado após o *boom* de obras a que o país assistiu no seguimento da realização da Expo. Os concursos públicos diminuíram e consequentemente as empresas deste sector ressentem-se. Outro exemplo: a indústria tabaqueira nos Estados Unidos tem sido penalizada em Bolsa pela perspectivas de que os inúmeros processos em tribunal contra empresas do sector venham a provocar a atribuição de elevadas indemnizações a pessoas que sofrem de cancro.

Outro aspecto decisivo na avaliação fundamental de uma empresa é a análise dos seus rácios. Dois dos indicadores mais estudados são o PER (*Price Earning Ratio*) e o EPS (*Earning per Share*). O primeiro é obtido dividindo a cotação do título pelo resultado líquido por acção, que é precisamente o EPS. Na análise fundamental é habitual comparar o PER da empresa com o PER de outras empresas do mesmo sector de actividade.

A maior virtude da Análise Fundamental é que permite avaliar correctamente qualquer título a longo prazo. Habitualmente este tipo de análise premeia os investidores pacientes que escolheram o sector de actividade ou a empresa através desta análise. No entanto, esta análise peca por muitas vezes estar desfasada face aos valores actuais de mercado. Uma empresa pode ter excelentes rácios e, por razões que têm a haver somente com o momento do mercado de capitais, pode ter uma cotação que difere bastante do valor que a AF antevê para o título em questão.

- **Análise Técnica**

A Análise Técnica é a análise das cotações históricas tendo em vista prever as evoluções futuras das cotações de um título [W3AT]. A Análise Técnica parte de diversas premissas que passamos a analisar:

a. A cotação de um título desconta tudo. Charles Dow é considerado o pai da Análise Técnica, razão pela qual muitas das teorias da análise técnica assentam na teoria de Dow. Talvez a mais importante seja a de que o mercado desconta tudo. Na prática, um analista técnico crê que a cotação presente de um título reflecte toda a informação que se conhece acerca do mesmo.

b. As variações das cotações não são aleatórias. A maior parte dos analistas técnicos crê que as cotações evoluem segundo tendências. No entanto, os analistas técnicos também concordam que podem existir períodos de tempo em que as cotações podem não seguir qualquer tendência definida. Ainda assim, a preocupação do analista é identificar correctamente a tendência de forma a investir correctamente.

c. O analista preocupa-se com «QUANTO?» e não com o «PORQUÊ?». O analista técnico não estuda a envolvente fundamental de uma empresa mas sim o histórico das cotações e da tendência das mesmas. A sua preocupação não é saber porque é que as cotações subiram mas sim identificar a subida antes que esta ocorra.

A análise técnica parte de uma perspectiva generalista até chegar à análise da empresa. Assim, o analista deve iniciar o seu estudo pelo sector de mercado em que a empresa se insere para depois estudar a empresa propriamente dita. A análise deve partir da análise de longo prazo para depois analisar a evolução de curto prazo quer do sector de actividade quer da empresa propriamente dita.

A ferramenta básica de análise de um analista técnico são os gráficos quer, das cotações históricas, quer de indicadores matemáticos calculados com base nos históricos de cotações. Os elementos básicos de análise são cinco: tendência, suporte, linha de resistência, momento e pressão vendedora/compradora. Estes elementos são descritos nos parágrafos seguintes.

Tendência: o objectivo primário é identificar a tendência de evolução da cotação. Numa linguagem mais simples, o objectivo é verificar se o título se encontra numa evolução de queda (*bearish*) ou numa evolução de subida (*bullish*). A figura 2.7 representa uma tendência de subida.

Suporte: áreas de congestão abaixo da cotação presente de um título são áreas que podem marcar níveis de suporte para a cotação do título. Caso essa linha seja rompida, poderemos entrar numa zona *bearish*.

Linha de Resistência: áreas de congestão acima da cotação de um título definem níveis de resistência. Se a cotação romper essa linha de resistência podemos entrar numa zona *bullish*. A figura 2.8 representa linhas de suporte e resistência.

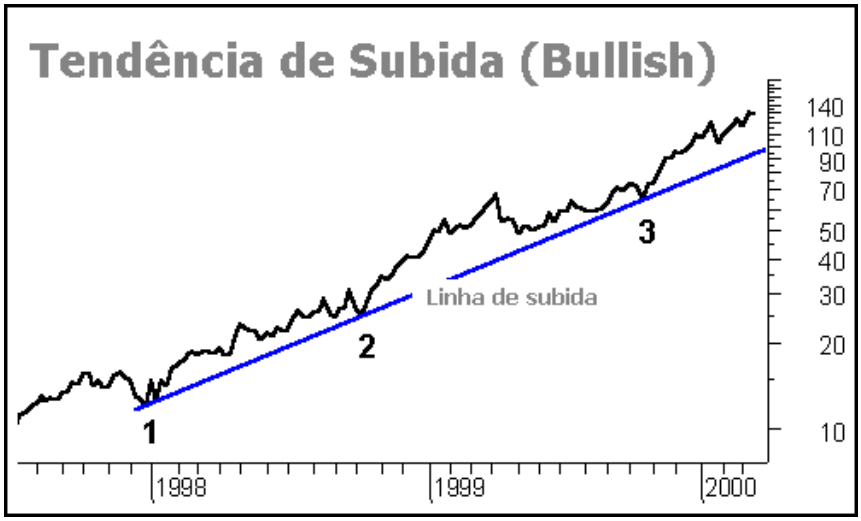


Figura 2.7 - Tendência de subida (*bullish*)

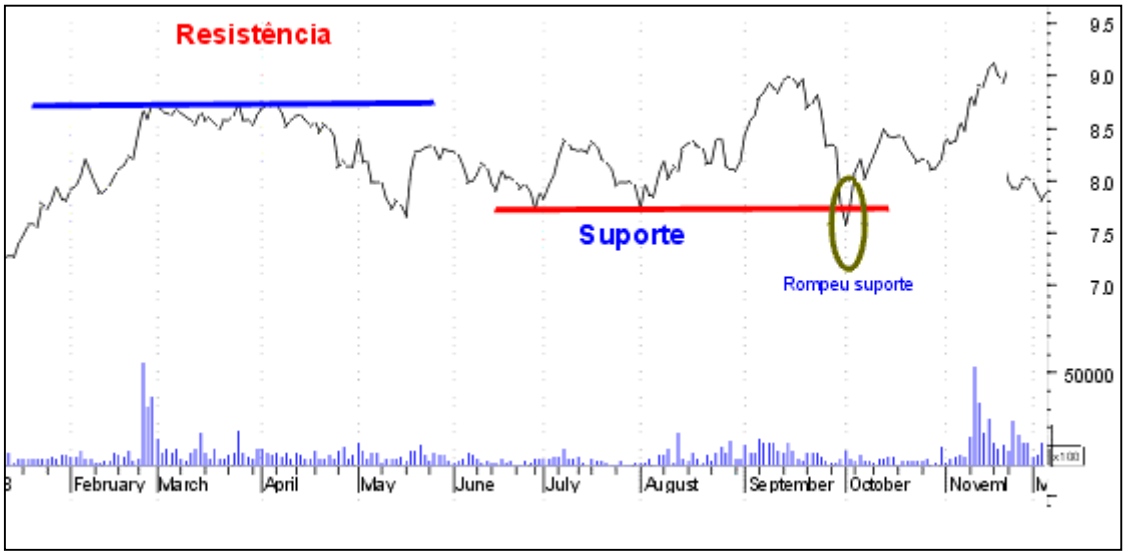


Figura 2.8 - Linhas de suporte e resistência

Momento: o momento de uma empresa é geralmente calculado com base em indicadores como o MACD[2]. O oscilador *Momento* mede a aceleração e desaceleração dos preços ao invés dos seus níveis actuais.

Pressão vendedora/compradora: para identificar se a pressão está dos lado dos *Bulls* (compradores) ou dos *Bears* (vendedores), o analista técnico socorre-se da análise do volume associado ao título.

Tal como a análise fundamental, a análise técnica encerra vantagens e desvantagens. A maior vantagem é que o analista técnico só se concentra na cotação do título. Ora se a maior preocupação é prever a evolução futura da cotação, faz todo o sentido analisar a cotação histórica. Outra vantagem prende-se com a identificação de linhas de suporte e resistência que definem limites potenciais da evolução da cotação da empresa. Finalmente, a análise técnica é óptima para decidir o momento de entrada no mercado.

[2] O indicador MACD (Média Móvel Divergente/Convergente) é um dos indicadores utilizados neste trabalho e será explicado mais à frente.

A maior desvantagem da análise 100% técnica é que é esquecida por completo a análise fundamental. As cotações, além de serem influenciadas pela sua evolução passada, são igualmente afectadas pela envolvente de mercado em que se posicionam. Outra desvantagem é que a análise técnica geralmente não antecipa as inversões de tendência antes que elas ocorram. De facto, só após a tendência se ter começado a desenhar é que é geralmente detectada.

2.4.3. As cotações e o volume

A análise das cotações de um título bem como o volume transaccionado são os elementos chave na análise técnica [W3AT]. Os pontos seguintes enumeram alguns termos que são usualmente usados em Bolsa sobre as cotações:

Cotação de Abertura: esta é a cotação do primeiro negócio do dia realizado em Bolsa sobre o título em questão.

Máximo: este é o valor máximo que a cotação de um título atinge durante o período de uma sessão de Bolsa.

Mínimo: este é o valor mínimo que a cotação de um título atinge durante o período de uma sessão de Bolsa.

Cotação de Fecho: esta é a cotação com que um dado título termina uma dada sessão de Bolsa. É este preço que é usualmente utilizado na Análise Técnica para avaliação das cotações.

Volume: é o número de acções transaccionados durante uma sessão de Bolsa. A relação entre este valor e o valor anterior são determinantes no estudo Técnico de um dado título.

Preço de Compra: este é o preço que os compradores estão dispostos a pagar por um título.

Preço de Venda: este é o preço pelo qual os vendedores estão dispostos a vender um título. Quando o preço de compra iguala o preço de venda dá-se a transacção de compra e venda em Bolsa.

Toda a análise técnica é feita em torno destes valores chave.

2.4.4. Indicador MACD

O indicador MACD é um dos dois indicadores utilizados na implementação dos agentes investidores neste projecto.

O indicador MACD (*Moving Average Convergence/Divergence indicator*) - Média Móvel Divergente/Convergente - é um indicador de tendência que mostra a relação entre duas médias móveis [W3AT]. É calculado subtraindo à média móvel exponencial de 26 dias a média móvel exponencial de 12 dias. O gráfico que daí se obtém é comparado com o gráfico da média móvel exponencial de 9 dias denominada de linha de sinal ou *trigger* que geralmente é um gráfico a picotado (ver figura 2.9).

Há três tipos distintos de interpretação gráfica do MACD:

- **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 *trigger*. (Esta é a interpretação usada neste projecto, na implementação dos agentes investidores).

- **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.

- **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. Se o MACD está a atingir mínimos sucessivos e a sua cotação não, estamos perante uma divergência *Bearish*, 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* sendo provável que a cotação do título venha a subir.

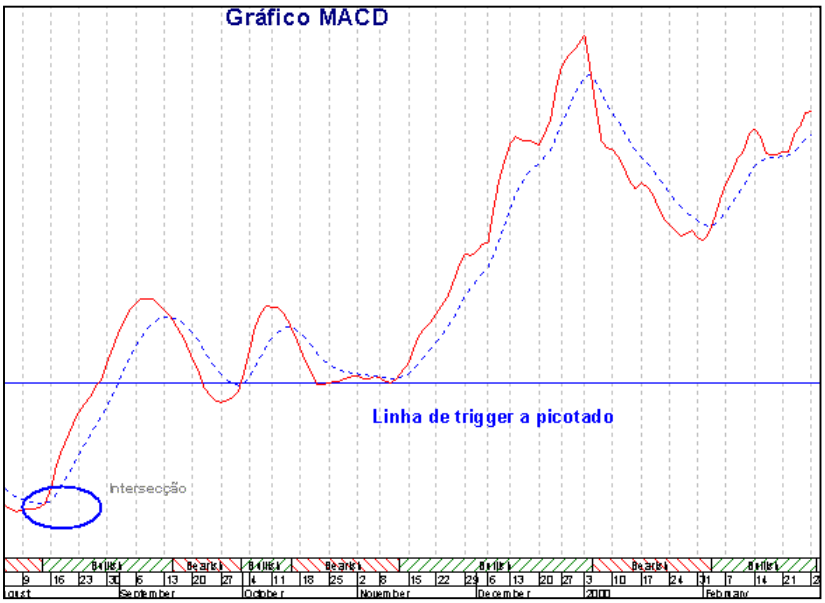


Figura 2.9 - Gráfico MACD.

2.4.5. Indicador estocástico

O indicador estocástico é o segundo indicador utilizado na implementação dos agentes investidores.

É importante sob o ponto de vista técnico determinar qual a relação entre a cotação de um título e o intervalo de cotações onde esse título se 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.

O indicador estocástico (STK), é um indicador de análise técnica que varia entre 0 e 100 (%), e que serve para medir a velocidade da reacção dos preços de títulos. Este indicador baseia-se na hipótese de que, à medida que os preços sobem, as cotações de fecho têm tendência a se posicionar mais próximos dos máximos. Se os preços caem, as cotações de fecho tendem a aproximar-se dos mínimos.

Desta forma, poderá ser um bom ponto de compra de uma acção, quando o STK estiver a aumentar em relação ao dia anterior, principalmente se seu valor for menor do que 20. Da mesma forma, poderá ser um bom ponto de venda, quando o STK estiver a diminuir em relação ao dia anterior, principalmente se seu valor for maior do que 80. A fórmula de cálculo do indicador STK é a seguinte:

$$STK\% = 100 * \frac{P - L}{H - L}$$

onde:

P: última cotação de fecho;
L: menor cotação dos últimos N períodos;
H: maior cotação dos últimos N períodos.

3. Descrição do trabalho

3.1. Introdução

Após ter aprofundado os meus conhecimentos sobre as áreas de contexto, nos primeiros dias do trabalho, comecei a planear a implementação do mesmo, sendo que o primeiro passo foi fazer o *download* de um código [W3Cor] que implementa uma bolsa virtual, semelhante àquela que eu pretendia implementar. Inicialmente analisei e executei esse código para observar o funcionamento desta bolsa virtual. Em seguida, tentei fazer uma adaptação desse mesmo código para obter a implementação tal como eu queria, mas tal revelou-se de uma extrema complexidade, devido à diferença significativa entre aquilo que era essa bolsa virtual e aquilo que eu queria que fosse o resultado final do meu trabalho (por exemplo, os valores das cotações das acções nessa bolsa virtual são gerados aleatoriamente, enquanto que no meu trabalho os valores utilizados são reais e obtidos em tempo real). Decidi então fazer a implementação do meu trabalho a partir do "zero", implementando-o de uma forma sequencial e estruturada. No entanto, utilizei no meu trabalho algumas funcionalidades obtidas a partir daquele código, tais como a interface gráfica do agente Bolsa e ainda algumas performativas utilizadas para a comunicação entre os agentes.

3.2. Funcionalidades

O sistema multi-agente é constituído por um agente Bolsa e vários agentes Investidores. O agente Bolsa tem a sua interface dividida em três áreas, uma onde são mostradas as mensagens enviadas para os Investidores, outra onde são mostradas as mensagens recebidas dos Investidores e uma terceira onde se pode observar a evolução das cotações das diversas acções. As acções apresentadas são referentes aos vinte títulos cotados no índice PSI-20. Os valores das cotações são actualizados em cada 30 segundos, sendo que os valores no topo são os mais actuais, ou seja, em cada 30 segundos, aparece uma nova linha com os valores actualizados na caixa de texto e as restantes linhas são empurradas para baixo. Após 30 linhas toda a caixa de texto é limpa e começa uma nova sequência com o aparecimento da primeira linha no topo. Os valores das cotações são obtidos pelo agente Bolsa a partir da base de dados.

A actualização da base de dados é também feita em cada 30 segundos, pelo Navegador. O Navegador é um programa que corre em *background* e faz a extracção dos valores a partir de um *site* de cotações *on-line* e armazena a informação na base de dados.

Quanto aos agentes investidores, vão interagir com o agente Bolsa, no sentido de obter as informações que são necessárias para executar as suas estratégias, como por exemplo, as cotações actuais, os históricos, etc. Cada agente Investidor tem a sua própria estratégia, sendo que esta pode ser uma sub-estratégia de uma outra, sendo obtida por parametrização da estratégia principal. Quando são iniciados, os agentes investidores, carregam as informações acerca do seu capital actual e da sua carteira de títulos. De seguida, começam a comunicação com o agente Bolsa para que possam obter as informações relativas às cotações dos títulos. Depois de obterem as informações, fazem o seu tratamento de acordo com as estratégias e caso seja favorável irão efectuar compras ou vendas. A qualquer altura podem ser consultadas as seguintes informações sobre cada um dos agentes investidores: capital inicial, capital actual, capital potencial e carteira de acções. O capital potencial é calculado a partir do valor actual da acção, que pode ser diferente do valor ao qual esta foi comprada.

Quando um agente Investidor decide comprar um título, comunica com o agente Bolsa e faz o pedido de compra, indicando qual é o título que pretende e a quantidade desejada. O agente Bolsa responde então pedindo ao Investidor que faça o pagamento referente à compra. Quando o Investidor faz o pagamento, o valor do investimento é descontado do seu capital e a sua carteira de títulos é actualizada, passando a fazer parte dela esta nova compra.

No caso de haver uma decisão de venda por parte do Investidor, ele irá comunicar ao agente Bolsa qual é o título que deseja vender e a respectiva quantidade. O agente Bolsa irá fazer o pagamento referente à venda, calculando o valor a pagar ao Investidor a partir da quantidade vendida e do valor actual do título. O Investidor recebe o pagamento referente à venda, e este é acrescentado ao seu capital.

3.3. Estrutura

A estrutura básica deste sistema é mostrada na figura 3.1.

O Navegador, codificado no ficheiro `Navegador.java`, faz uso do *package*

`extraccao`, do qual fazem parte os ficheiros `filterAndUpdate.java`, `GrabPage.java` e `Texto.java`.
O ficheiro `cotacoes.php`, serve para visualizar a informação contida na Base de Dados, e tem associado a ele o ficheiro `historico.php`, onde pode ser visualizado o histórico de cada um dos títulos.

Tanto na estrutura do agente Bolsa, assim como na estrutura dos agentes investidores, é usado o *package* `accoes`, que contém classes que definem os títulos e implementam operações sobre eles.

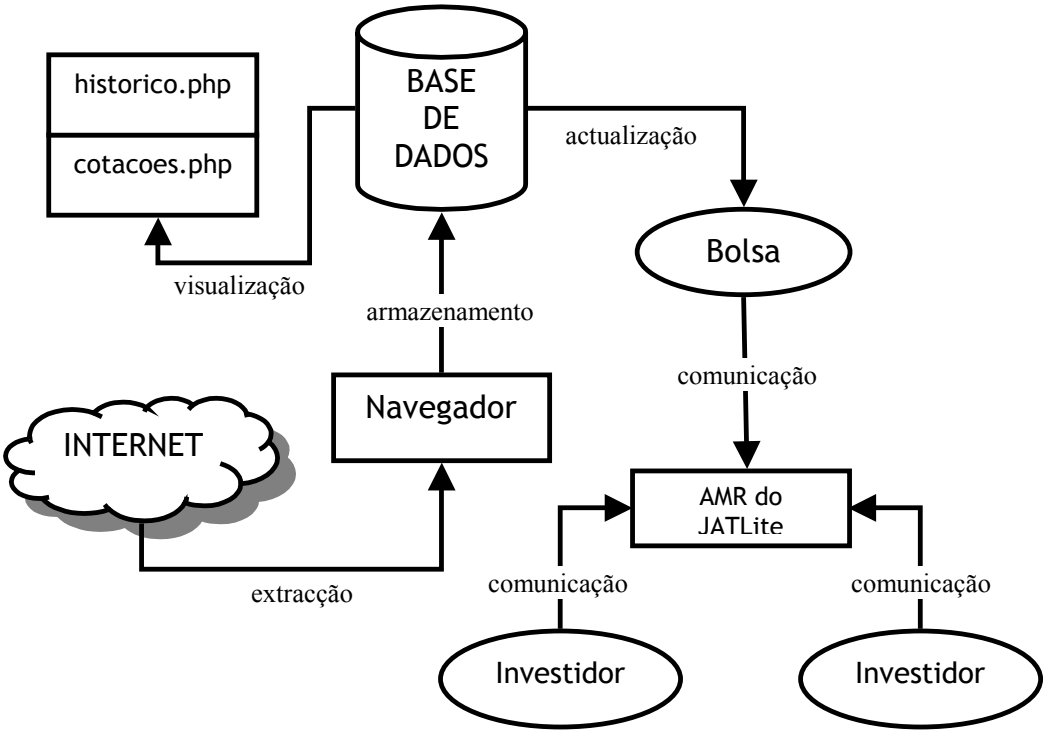


Figura 3.1 - Estrutura do sistema.

A definição dos dois tipos de agentes investidores, cada um baseado numa estratégia diferente (indicador MACD e indicador Estocástico), é feita nos ficheiros `Investidor.java` e `InvStk.java`. A parametrização dos agentes e consequente aplicação de "sub-estratégias" é feita através dos ficheiros `InvestMACD.java` e `InvestStk.java`.

No ficheiro `Wbolsa.java` são definidas as classes `Bolsa` (classe que implementa o agente Bolsa) e `Wbolsa` (classe que implementa a interface gráfica do agente Bolsa, que usa o *package* `GUITools`, o qual contém os ficheiros `PanelCotacoes.java` e `PanelMSG.java`).

3.4. Implementação

3.4.1. Base de dados

A base de dados utilizada é do tipo *MySQL* e foram construídas as seguintes tabelas:

- **PHP_BOLSA_PSI20**

  Esta tabela armazena a informação acerca do estado actual das cotações. É constituída pelas colunas: *sigla*, *titulo*, *ult_cotacao*, *hora*, *variacao*, *quantidade*, *maximo*, *minimo*, *compra* e *venda* (figura 3.2).

  À excepção das colunas *sigla* e *titulo*, todas as outras são constantemente actualizadas. O campo *sigla* é aquele que é apresentado na interface gráfica do agente Bolsa. A tabela é ordenada crescentemente pela ordem alfabética das siglas.

Database *ee95086* - table *PHP_BOLSA_PSI20* running on *localhost*

Showing records 0 - 20 (20 total)

SQL-query : [Edit]
SELECT * FROM `PHP_BOLSA_PSI20` WHERE 1 LIMIT 0, 30

Show: 30 rows starting from 0

		sigla	titulo	ult_cotacao	hora	variacao	quantidade	maximo	minimo	compra	venda
Edit	Delete	BCP .	BCP	1.49	16:30	-0.67%	4163953	1.51	1.48	1.48	1.49
Edit	Delete	BES .	BES	12.85	16:30	0.00%	33509	12.87	12.77	12.80	12.85
Edit	Delete	BPI .	BPI	2.40	16:30	-0.83%	536327	2.42	2.38	2.40	2.41
Edit	Delete	BRISA.	BRISA PRIV	4.99	16:30	1.84%	1071347	5.00	4.89	4.97	4.99
Edit	Delete	CIMPOR	CIMPOR	3.33	16:30	0.60%	173771	3.34	3.31	3.32	3.33
Edit	Delete	COFINA	COFINA	2.26	16:30	3.20%	138638	2.26	2.18	2.22	2.26
Edit	Delete	EDP .	EDP	2.10	16:30	2.94%	10401613	2.10	2.04	2.09	2.10
Edit	Delete	IBERSO	IBERSOL	3.50	15:09	-0.85%	1000	3.50	3.50	3.50	3.53
Edit	Delete	IMPRES	IMPRESA	2.37	16:30	0.85%	50999	2.38	2.33	2.37	2.38
Edit	Delete	J.MART	J.MARTINS	7.13	16:30	2.89%	171695	7.14	6.92	7.12	7.13
Edit	Delete	NOVABA	NOVABASE	6.15	16:30	0.00%	9338	6.18	6.11	6.12	6.15
Edit	Delete	PARARD	PARAREDE	0.21	15:58	5.00%	325208	0.21	0.20	0.20	0.21
Edit	Delete	PORTUC	PORTUCEL	1.30	16:30	0.78%	5859951	1.31	1.28	1.30	1.31
Edit	Delete	PT.MUL	PT.MULTI	15.20	16:30	-0.33%	26825	15.29	15.16	15.20	15.26

Figura 3.2 - Tabela PHP_BOLSA_PSI20 da Base de Dados.

- **PSI20_COTACOES**

  Nesta tabela estão armazenadas todas as cotações de todos os títulos do índice PSI-20 desde 17-02-2003 até ao dia actual. Como colunas, esta tabela tem a data e também a designação de cada um dos títulos (ver figura 3.3). Esta estruturação da tabela (e consequentemente da BD) concerteza não é a melhor, e provavelmente não obedece às regras de construção de tabelas em bases de dados. Será feita uma referência a tal na secção de melhoramentos.

  A actualização desta tabela é feita dinamicamente, sempre que é actualizada a tabela PHP_BOLSA_PSI20, sendo que a actualização diz respeito apenas à ultima linha da tabela (*data* = *hoje*). A tabela é ordenada crescentemente pela data.

Para visualizar as informações contidas na base de dados, implementei o ficheiro *cotacoes.php* que mostra o conteúdo actual da tabela PHP_BOLSA_PSI20. O ficheiro está alojado no endereço: `http://gnomo.fe.up.pt/~ee95086/PSTFC/cotacoes.php` .

Este ficheiro é actualizado (*refresh*) em cada 30 segundos (ver figura 3.4).

Database *ee95086* - table *PSI20_COTACOES* running on *localhost*

Showing records 0 - 30 (99 total)

SQL-query : [Edit]
SELECT * FROM `PSI20_COTACOES` WHERE 1 LIMIT 0, 30

Show: 30 rows starting from 30 > >>

		data	BCP	BES	BPI	BRISA PRIV	CIMPOR	COFINA	EDP	IBERSOL	IMPRESA	J.MARTINS	NOVABASE	PARAREDE	PORTUCEL
Edit	Delete	2003-02-17	1.97	12.23	2.13	5.01	3.222	2.13	1.56	3.38	1.92	6.59	5.41	0.19	1.19
Edit	Delete	2003-02-18	1.95	12.25	2.15	5.06	3.208	2.15	1.58	3.37	1.89	6.5	5.5	0.19	1.18
Edit	Delete	2003-02-19	1.94	12.18	2.16	5.01	3.234	2.15	1.54	3.37	1.89	6.29	5.45	0.19	1.19
Edit	Delete	2003-02-20	1.9	12.3	2.17	5.04	3.216	2.11	1.54	3.35	1.88	6.25	5.35	0.19	1.18
Edit	Delete	2003-02-21	1.86	12.32	2.16	5.03	3.214	2.13	1.54	3.35	1.86	6.35	5.37	0.19	1.2
Edit	Delete	2003-02-24	1.81	12.29	2.13	4.95	3.218	2.01	1.52	3.34	1.84	6.25	5.35	0.19	1.19
Edit	Delete	2003-02-25	1.64	12.2	2.09	4.9	3.202	2.04	1.46	3.3	1.77	6.12	5.28	0.18	1.19
Edit	Delete	2003-02-26	1.65	12.15	2.1	4.96	3.2	2.03	1.43	3.29	1.72	6.02	5.2	0.18	1.19

Figura 3.3 - Tabela PSI20_COTACOES da Base de Dados.

Cotações PSI20 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites History

Address http://gnomo.fe.up.pt/~ee95086/PSTFC/cotacoes.php Go Links

Cotações PSI20

Título	Cotação	Hora	Variação	Quantidade	Máximo	Mínimo	Compra	Venda
BCP	1.52	10:31	-0.65%	747009	1.53	1.51	1.51	1.52
BES	12.80	09:40	-0.78%	14756	12.90	12.80	12.80	12.85
BPI	2.43	10:31	-1.22%	105096	2.46	2.43	2.43	2.44
BRISA PRIV	4.86	10:34	-0.82%	411196	4.90	4.84	4.86	4.87
CIMPOR	3.31	10:08	0.00%	11349	3.33	3.31	3.31	3.33
COFINA	2.20	10:31	-2.22%	13174	2.21	2.15	2.18	2.20
EDP	1.85	10:34	-0.54%	671342	1.87	1.85	1.85	1.86
IBERSOL	3.51	09:38	-0.28%	168	3.51	3.51	3.51	3.54
IMPRESA	2.20	10:30	-0.90%	4025	2.20	2.19	2.19	2.21
J.MARTINS	6.89	10:32	0.00%	5513	6.90	6.82	6.89	6.90
NOVABASE	6.31	10:34	1.12%	2848	6.34	6.22	6.22	6.31
PARAREDE	0.19	09:09	0.00%	8300	0.20	0.19	0.19	0.20
PORTUCEL	1.29	10:13	-1.53%	31107	1.31	1.29	1.29	1.30
PT.MULTI	15.07	10:16	-1.18%	13746	15.35	15.05	15.07	15.14
PTELECOM	6.20	10:35	-0.64%	956367	6.25	6.17	6.20	6.21
SAG GEST	1.18	10:19	0.00%	1810	1.19	1.17	1.18	1.19
SEMAPA	2.82	10:10	-0.35%	1540	2.83	2.80	2.82	2.85
SONAE SGPS	0.49	10:35	0.00%	891248	0.50	0.48	0.48	0.49
SONAE.COM	2.05	10:13	0.49%	12077	2.06	2.04	2.04	2.05
T.DUARTE	0.91	10:27	1.11%	75023	0.92	0.88	0.90	0.91

2003-07-01 (Terça-feira) às 10:49

Done Internet

Figura 3.4 - Página *Web* para visualização da tabela PHP_BOLSA_PSI20.

Implementei ainda o ficheiro *historico.php* que apresenta o histórico de um certo título. Este ficheiro recebe como parâmetro a designação do título. A partir do ficheiro *cotacoes.php*, "clicando" num dos títulos, visualiza-se o histórico desse título (ver figura 3.5). Também este ficheiro é actualizado em cada 30 segundos.

Existe ainda um outro ficheiro auxiliar, *variaveis.php*, que contém algumas variáveis usadas nos outros dois ficheiros.

No que diz respeito à actualização da base de dados, a tabela PHP_BOLSA_PSI20 é actualizada pelo Navegador (ver secção seguinte), pelo que pode dizer-se que esta tabela apresenta os valores em tempo real, visto que o Navegador faz a extracção dos valores de um site de cotações on-line e insere os valores extraídos imediatamente na base de dados. Como a actualização é feita em cada 30 segundos, as cotações terão no máximo um diferimento de 30 segundos em relação ao valor real que figura no site de onde é extraída a informação. Quanto à actualização da tabela PSI20_COTACOES, esta é feita dinamicamente a partir do ficheiro *cotacoes.php*. Os valores que estão a ser visualizados nesse momento na coluna Cotação (referente ao valor mais actual da cotação) são colocados na última linha da tabela PSI20_COTACOES, tendo como referência a variável `$today = date("Y-m-d")`. Esta variável contém a data actual, logo os dados actuais vão ser inseridos na linha que tem a data de *hoje*.

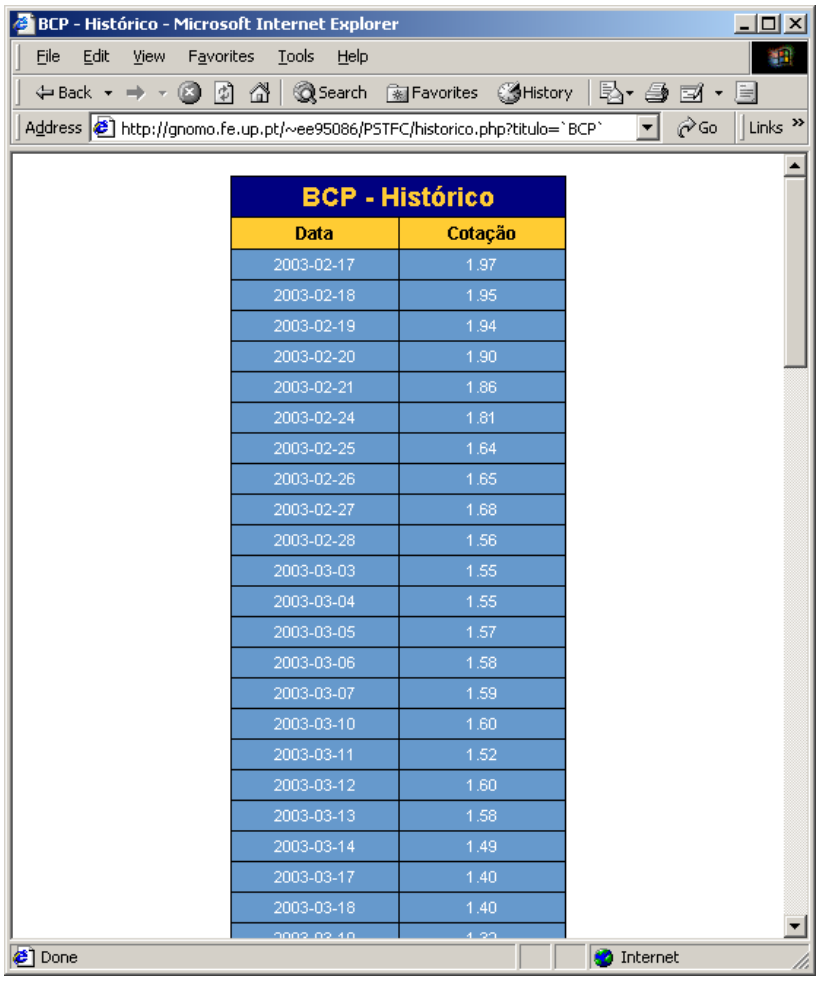


Figura 3.5 - Página *Web* para visualização do histórico.

3.4.2. Extracção das cotações

Antes de começar a implementação da extracção das cotações, consultei alguns trabalhos já realizados na mesma área [CA02, RCM02, RM02]
O ponto de partida para a implementação da extracção das cotações foi o *Web Croller* [W3WC]. Esta aplicação em Java pesquisa informação contida em páginas HTML e guarda-as num ficheiro de texto. O *Web Croller* foi adaptado de forma a abrir apenas a página pretendida e guardá-la num ficheiro. Este ficheiro é depois aberto por uma classe deste programa que retira a informação desejada e a guarda na base de dados.

A aplicação é constituída por 4 classes: `Navegador`, `GrabPage`, `Texto` e `filterAndUpdate` (optei por criar um *package* contendo as três últimas classes). O ponto de entrada para o programa é a classe `Navegador`. A primeira coisa que faz é usar o *stream* de *output* padrão para apresentar um ecrã inicial ao utilizador:

```
----------------------------------------
  C O T A C O E S    P S I 2 0    0.0

                FEUP 2003
----------------------------------------
```

De seguida cria um objecto da classe `GrabPage`. Esta última implementa todos os métodos necessários para trazer a página do servidor *Web*. É passado ao construtor da classe o URL da página da bolsa, construtor este que chama o método `dissect()`, por sua vez responsável pela obtenção do IP da máquina servidora, da porta do servidor *Web* e do ficheiro associado à página. Criado que está o objecto do tipo `GrabPage`, é agora chamado o método `grab()` existente neste mesmo objecto. Este método chama um outro método (`connect()`), responsável pela ligação ao servidor *Web*. Este lança o *socket* para o servidor e cria dois *streams*: de entrada e de saída, que permitirão usar um novo objecto `Writer` e outro `Reader` para comunicar com o servidor. Conseguida a ligação, o método `grab()` chama um outro método (`fetch()`) que irá usar o leitor e escritor mencionados para ir buscar a página à máquina servidora, guardando o conteúdo da página num *array* de *strings*. Por sua vez, este *array* de *strings* está implementado no interior de um objecto do tipo `Texto` que possui um método (`acrescentaLinha()`) que permite adicionar linhas de texto ao *array* mencionado. Sendo assim, o método `fetch()` chama o método `acrescentaLinha()` sobre o objecto do tipo `Texto`, até que não existam mais linhas na página que está a ser transferida. Finalmente, o método `fetch()` retorna a `grab()` o objecto do tipo `Texto`, sendo chamado um método para terminar a ligação com o servidor *Web*. O que o código principal recebe (*Navegador*) é um objecto do tipo `Texto`, mais uma vez, com um *array* de *strings* que representa o texto da página.

Concluída a parte inicial da aplicação, o "Navegador" possui já uma cópia local da página. A partir de agora poderá ser criado código para processar o conteúdo da mesma, que é a parte essencial deste programa. Para este efeito foi implementada uma nova classe: `filterAndUpdate`; à qual é passado através do construtor, o texto da página. Um dos factos importantes da página recebida é que a tabela de cotações está integralmente numa linha de texto e, deste modo, a primeira coisa que o construtor faz é procurar esta linha e guardá-la num objecto do tipo `String` (mais especificamente `tabLinha`). O navegador instancia um objecto da classe `filterAndUpdate` e

chama sobre o mesmo o método `constroiTabCotacoes()`, também nele implementado. Se não encontra esta linha simplesmente reporta um erro na página.

O método `constroiTabCotacoes()` actua imediatamente sobre a linha com a tabela de cotações, retirando-lhe (usando o método `replaceAll()`) todos os caracteres que não interessam entre os valores das cotações, ficando estes valores apenas separados pelo caracter ">". Depois, vai tentar criar uma tabela devidamente estruturada com estes valores - `tabCotacoes`. Para tal, como esta tabela possuirá 20 linhas, usa um ciclo `for` com 20 passagens para apanhar os valores das cotações presentes na tabLinha filtrada. Em cada passo, usa o método `split()` para apanhar cada um dos valores referentes a uma empresa e separados por ">", e depois procura o início da próxima linha (empresa) com o método `indexOf()`.

Uma vez filtrada a informação das cotações, é possível actualizar a base de dados com estes valores, e é o que se faz chamando um novo método - `connectDB()`. Essencialmente o que este método faz é instanciar um objecto com o *driver JDBC* de acesso à base de dados *MySQL*. O *driver* utilizado foi o *MM.MySQL*. Instanciado o *driver*, pode agora ser utilizado o `DriverManager` para criar uma ligação com a base de dados usando este mesmo *driver*, e é exactamente o que se faz chamando o método `getConnection()`. Finalmente, o método `connectDB()` cria ainda um objecto do tipo `Statement` que será usado para fazer perguntas à base de dados.

Regressando ao método inicial `constroiTabCotacoes()`, após estabelecida a ligação com a base de dados, é chamado o método `updateDB()` que actualizará a base de dados. Este último método simplesmente usa um ciclo for com 20 passos para actualizar as 20 linhas da tabela *PHP_BOLSA_PSI20*, usando como base a tabela `tabCotacoes` criada localmente. De referir que todos os passos que têm vindo a ser descritos são devidamente tratados com excepções para o caso de algum erro ocorrer. Por último, o método `constroiTabCotacoes()` termina a ligação com a base de dados (chamando `disconnectDB()`) e retorna o ciclo de execução para o `Navegador`.

Tendo corrido bem todo este processo, não lançadas nem apanhadas excepções, é restabelecido o controlo para o `Navegador` que invocará agora o método `Thread.sleep()`, "adormecendo" um certo tempo antes de repetir tudo o que aqui foi descrito.

3.4.3. Agente Bolsa

O agente Bolsa foi implementado sobre a plataforma JATLite, usando as camadas *Abstract Layer*, *Base Layer*, *KQML Layer* e *Router Layer*, fornecidas por essa plataforma.

O agente Bolsa é constituído por duas classes: a classe `Bolsa`, que implementa as funcionalidades deste agente e a classe `Wbolsa`, que implementa a interface gráfica.

Inicialmente é feita a ligação ao *Router* do JATLite. No construtor `Bolsa` é chamada a classe `TRouterID`, que através do método `seekConfFile()` verifica se existe algum ficheiro de configuração. O ficheiro de configuração contém as informações

relativas à ligação ao *Router* (*Nome do router*, *Host do router*, *Porta de ligação*, *Nome do router registrar*, *Porta do router registrar*) e é opcional. Caso não seja encontrado nenhum ficheiro de configuração, são usadas as configurações por defeito, as quais podem ser definidas na classe `TRouterID` através do método `setDefault()`. O agente Bolsa faz então a ligação ao router enviando o seu nome e *password*.

Após estabelecida a ligação ao *Router*, o agente Bolsa inicia a sua acção. É usado o *stream* de *output* padrão para apresentar uma mensagem inicial ao utilizador:

```
----------------[ ABERTURA DA BOLSA !]---------------
```

Na inicialização da Bolsa é feita a configuração da interface gráfica através das classes `PanelCotacoes` e `PanelMSG` do *package* `GUITools`. È definido o tamanho da janela relativa ao agente Bolsa (que contém como já foi referido anteriormente, as mensagens enviadas, as mensagens recebidas e a evolução das cotações), sendo ainda definido o tamanho das áreas referentes às mensagens enviadas e recebidas (`PanelMSG`) e também o tamanho da caixa de texto referente à evolução das cotações (`PanelCotacoes`). De seguida são colocadas no cabeçalho do painel referente às cotações as siglas dos títulos.

```
BCP . BES . BPI . BRISA. CIMPOR COFINA EDP . IBERSO IMPRES J.MART NOVABA
PARARD PORTUC PT.MUL PTELEC SAG GE SEMAPA SONAES SO.COM T.DUAR
```

Isto é feito com um ciclo for que vai inserindo as siglas no cabeçalho através do método `setCabecalho()`, da classe PanelCotacoes. De notar que todas as siglas foram colocadas com 6 caracteres para que todas as colunas tenham a mesma largura.

Depois é enviada uma mensagem de início ao agente Bolsa, que serve para fazer um teste à comunicação (se esta mensagem não for visualizada na área das mensagens recebidas é porque há algum erro de comunicação). Relembro que a mensagem é enviada ao *Router*, que por sua vez a reenvia ao agente Bolsa. A mensagem enviada ao Router é a seguinte (KQML):

```
(tell :sender GHOST :receiver " + bolsa.getName() + " :content (TEMPO
INICIAL))
```

Deve ser visualizada na área de mensagens recebidas:

```
GHOST/tell - <(TEMPO INICIAL)>
```

Para obter o nome do agente é usado o método `getName()`, que neste caso vai retornar o nome do agente Bolsa (BOLSA).

Em seguida começa a ser feita a actualização dos valores no painel das cotações. È invocado o método `actualiza()`, que vai fazer a ligação à base de dados para obter as informações desejadas, tanto da tabela das cotações actuais como da tabela do histórico. A primeira coisa que este método faz é a instanciação do *driver* de acesso à base de dados *MySQL*. Após ser efectuada a ligação à base de dados, é executado o seguinte *query*[3], para obter as informações acerca dos títulos, valores e quantidades actuais:

[3] Interrogação à base de dados.

```
SELECT sigla, quantidade, ult_cotacao, titulo FROM PHP_BOLSA_PSI20;
```

Os resultados do *query* são armazenados temporariamente num *array* bidimensional de *strings* (`tabela[][]`), depois são convertidos para objectos da classe `SeqAccao`[4] e inseridos num vector (`Vaccoes`). É ainda feito o armazenamento da designação dos títulos num *array* de *strings* (`titulos[]`), que será útil para fazer o *query* relativo aos históricos dos títulos.

O próximo passo é precisamente a obtenção dos históricos de todos os títulos, a partir da tabela PSI20_COTACOES da base de dados. Primeiro é executado um *query* para saber quantas linhas tem a tabela, ou seja, há quantos dias está a ser armazenado o histórico dos títulos:

```
SELECT count(*) FROM PSI20_COTACOES;
```

O resultado do *query* irá servir para inicializar a variável `histórico`:

```
historico = new double[20][count];
```

e para armazenar na variável `n_dias` o número de dias a que corresponde o histórico, ou seja, há quantos dias existe informação acerca do histórico.

De seguida é executado dentro de um ciclo `for` um outro *query* para a obtenção do histórico de todos os títulos:

```
SELECT data, `"+titulos[k]+"` FROM PSI20_COTACOES ORDER BY data ASC;
```

Dentro do ciclo `for` é obtida a designação de cada um dos títulos a partir do *array* `titulos[]`. No *query* é pedido à base de dados que o resultado seja ordenado ascendentemente pela data. O resultado do *query* é armazenado na variável `historico[][]`.

Após serem obtidos os valores da base de dados, é feita a actualização do painel das cotações, inserindo-lhe uma nova linha no topo, com as cotações actuais. Tal é feito com o método `meteStr()` da classe `PanelCotacoes`. Este método recebe uma *string* que já vem formatada pela classe `Wbolsa` e faz a inserção da linha na caixa de texto utilizando o método `insert()`. A formatação da *string* é feita através de um ciclo for, que vai fazendo a concatenação dos valores das cotações. Os valores são formatados para duas casas decimais:

```
DecimalFormat myForm = new DecimalFormat("0.00");
```

É ainda acrescentado o símbolo do Euro (`€`) antes da cada um dos valores.

Após cada actualização, será invocado o método `Thread.sleep()`, "adormecendo" durante 30 segundos e depois irá proceder a uma nova actualização, sendo repetidos os passos descritos.

[4] Ver descrição da classe no Apêndice 1.

Ainda no método `actualiza()`, após a obtenção das informações da base de dados, são também actualizadas algumas variáveis que dizem respeito à implementação dos indicadores. Assim, após terem sido guardados os valores dos históricos dos títulos, são calculadas as médias móveis exponenciais de 26 e 12 dias (referentes ao indicador MACD). Depois é calculado o MACD e também a média móvel exponencial de 9 dias (*signal*). Todos estes dados são guardados em *arrays* bidimensionais (`mm26[][]`, `mm12[][]`, `macd[][]` e `signal[][]`). É ainda feita em paralelo a extracção dos valores do MACD e do *signal* para ficheiros de texto, através do *stream* de *output*, com o objectivo de poderem ser importados no Excel, para visualização dos respectivos gráficos.

As fórmulas de cálculo do indicador MACD são as seguintes:

$$\mathrm{MACD}_i = \mathrm{EMA}_i^{(1)} - \mathrm{EMA}_i^{(2)}$$

$$\mathrm{signal}_i = (1 - k_3) * \mathrm{signal}_{i-1} + k_3 * \mathrm{MACD}_i$$

onde:

$$\mathrm{EMA}_i^{(1)} = (1 - k_1) * \mathrm{EMA}_{i-1}^{(1)} + k_1 * \mathrm{close}_i$$

$$\mathrm{EMA}_i^{(2)} = (1 - k_2) * \mathrm{EMA}_{i-1}^{(2)} + k_2 * \mathrm{close}_i$$

$$k_1 = \frac{2}{1 + N_1}, \quad k_2 = \frac{2}{1 + N_2}, \quad k_3 = \frac{2}{1 + N_3}$$

$$\mathrm{EMA}_{-1}^{(1)} = \mathrm{EMA}_{-1}^{(2)} = \mathrm{close}_0$$

$$\mathrm{signal}_{-1} = 0$$

$N_1 = 26, \quad N_2 = 12, \quad N_3 = 9$

EMA – *Exponential Moving Average*, Média Móvel Exponencial.
close – cotação de fecho.
N – nº de dias de cálculo da média.

Os ciclos que implementam o cálculo do indicador MACD são apresentados a seguir:

- **Média móvel exponencial de 26 dias**

```
for (int i=0; i<20; i++)
{
      mm26[i][0]=historico[i][0];
      for (int j=1; j<n_dias; j++)
      {
            mm26[i][j]=(1-k1)*mm26[i][j-1]+k1*historico[i][j];
      }
}
```

- **Média móvel exponencial de 12 dias**

```
for (int i=0; i<20; i++)
{
      mm12[i][0]=historico[i][0];
      for (int j=1; j<n_dias; j++)
      {
            mm12[i][j]=(1-k2)*mm12[i][j-1]+k2*historico[i][j];
      }
}
```

- **MACD**

```
for (int i=0; i<20; i++)
{
      for (int j=n_dias-60; j<n_dias; j++)
      {
            macd[i][j]=mm26[i][j]-mm12[i][j];
      }
}
```

- **Média móvel exponencial de 9 dias (*signal*)**

```
for (int i=0; i<20; i++)
{
      signal[i][0]=0;
      for (int j=n_dias-60; j<n_dias; j++)
      {
            signal[i][j]=(1-k3)*signal[i][j-1]+k3*macd[i][j];
      }
}
```

Para simular a concorrência entre os investidores, o valor da quantidade disponível para venda de um determinado título é diminuída sempre que há uma compra. Assim, se houver pedidos de compra de um determinado título por parte de dois investidores, aquele que lançar o pedido em segundo lugar poderá não ter disponível a quantidade desejada.

3.4.4. Agentes investidores

Os agentes investidores também foram implementados sobre a plataforma JATLite, usando as camadas *Abstract Layer*, *Base Layer*, *KQML Layer* e *Router Layer*.

Em termos de interface, não foi implementada nenhuma interface gráfica; toda a informação é visualizada numa janela de DOS (*prompt*).

Inicialmente os agentes fazem também a ligação ao *router*, de forma semelhante ao agente Bolsa, sendo que no caso dos investidores, o nome do agente, a *password* e a porta de ligação ao *router* são gerados quando é feita a parametrização.

Na inicialização de cada um dos agentes investidores, após ter recebido o(s) seu(s) parâmetros (específicos que cada um dos tipos de investidores), são carregadas as informações referentes ao seu capital e à sua carteira de títulos. As informações estão armazenadas em ficheiros com extensões *.cap* e *.inv*, respectivamente. Estas informações são carregadas invocando os métodos `getCapital()` e `getInvestimento()`. O valor do capital é de imediato armazenado na variável `capital` e a informação referente à carteira é armazenada no *array* `investimento[]`. Existe ainda um *array* auxiliar do tipo `boolean`,

`em_carteira[]`, que indica se o investidor tem esse título ou não. O *array* é inicializado com todos os seus elementos em `false` e quando é carregada a informação, se houver um título cuja quantidade seja maior que zero, o elemento referente a esse título será colocado em `true`. A informação acerca do capital e da carteira de títulos é mostrada ao utilizador.

3.4.4.1. Investidor baseado no indicador MACD

Este investidor recebe um parâmetro, que é o capital inicial, em milhares de euros (por exemplo, se o utilizador desejar que o capital inicial seja €2000, irá passar como parâmetro '2'). As variáveis que são dependentes do parâmetro (nome do agente, *password* e porta de ligação ao *router*) são de imediato inicializadas:

```
this.capital_ini = capIni*1000;
this.port=3000+capIni;

String identif= "INV"+capIni;
String passwd= "INV"+capIni;
```

Por exemplo, se o parâmetro passado for '2', as variáveis irão ficar com os seguintes valores:

`capital_ini = 2000`	(capital inicial)
`port = 3002`	(porta de ligação ao *router*)
`identif = INV2`	(nome do agente)
`passwd = INV2`	(*password*)

Facilmente se conclui que para quaisquer dois parâmetros diferentes se irá obter dois agentes distintos.

Após a inicialização do agente, e a obtenção das informações acerca do capital e da carteira de títulos, como já foi referido anteriormente, é enviada a primeira mensagem ao agente Bolsa, perguntando os títulos disponíveis. O envio das mensagens é implementado pelo método `sendMSG()`:

```
protected void sendMSG(String receiver, String perform, String MSG)
throws Exception
  {
    KQMLmessage sendkqml= new KQMLmessage();

    sendkqml.addFieldValuePair("performative", perform);
    sendkqml.addFieldValuePair("sender", this.getName());
    sendkqml.addFieldValuePair("receiver", receiver);
    sendkqml.addFieldValuePair("content", "("+ MSG + ")");

    sendMessage(sendkqml);
  }
```

Neste caso, para perguntar ao agente Bolsa quais os títulos disponíveis, o agente Investidor invoca o método da seguinte forma:

```
sendMSG("BOLSA","ask","TITULOS DISPONIVEIS");
```

Após ter enviado esta mensagem, o agente Investidor "adormece" durante 30 segundos (`Thread.sleep()`).

O agente Bolsa responde, enviando as informações acerca dos títulos, as suas cotações e quantidades existentes através da performativa "TITULOS". Também da parte do agente Bolsa é invocado o método `sendMSG()`, sendo que ele obtém o nome do agente para o qual deseja enviar a mensagem, através do método `kqml.getValue()`, obtido a partir da mensagem recebida:

```
sendMSG(kqml.getValue("sender"), "TITULOS", Smsg);
```

A variável `Smsg` é uma *string* onde estão concatenadas as *strings* contendo as siglas dos títulos, as cotações actuais e as quantidades disponíveis, separadas por vírgulas.

O agente Investidor recebe a *string* e invoca o método `define()` da classe `TAccoes`, que faz a separação da *string* em *tokens*[5]. São detectadas as vírgulas (ou seja o caracter '`,`') que fazem a separação dos valores e então após estar feita a separação em *tokens*, estes são armazenados nas variáveis correspondentes (`desig`, `quant`, `valor`), sendo depois colocados num objecto do tipo `TAccao`[6]. Os dados são depois armazenados no *array* `titulos[]` por parte da classe `Investidor`.

Em seguida, o Investidor envia outra mensagem ao agente Bolsa ("MACD"), perguntando os valores do indicador MACD:

```
sendMSG("BOLSA","ask","MACD");
```

Como resposta, o agente Bolsa irá enviar duas mensagens ao Investidor ("MACD" e "SIGNAL"):

```
sendMSG(kqml.getValue("sender"), "MACD", Smsg);
sendMSG(kqml.getValue("sender"), "SIGNAL", Ssgn);
```

A *string* `Smsg` contém a concatenação dos dois últimos valores (referentes aos dois últimos dias) de MACD de todos os títulos e a *string* `Ssgn` contém a concatenação dos dois últimos valores de *signal* de todos os títulos. Apenas são enviados os dois últimos valores, visto que apenas estes são necessários para a análise que o Investidor tem que fazer (comparando estes valores, o Investidor facilmente deduz se houve intersecção dos gráficos).

O Investidor recebe as *strings* e invoca o método `split()` da classe `TAccoes` para fazer a separação em *tokens* e armazenamento em variáveis, de forma semelhante ao que foi descrito anteriormente. Os dados são armazenados nos *arrays* bidimensionais `macd[][]` e `signal[][]` por parte da classe `Investidor`.

A partir daqui o Investidor começa a fazer a análise dos dados recebidos, implementando o *array* `indicador[]`, onde são armazenados os sinais de compra ou venda, caso existam:

```
for (int i=0; i<20; i++)
{
      if (macd[i][1]>signal[g][1] && macd[i][0]<signal[i][0])
      {
            indicador[i]="compra";
      }
```

[5] Sub-*strings*.
[6] Ver descrição da classe no Apêndice 1.

```
      else if (macd[i][1]<signal[i][1] && macd[i][0]>signal[i][0])
      {
            indicador[i]="venda";
      }
      else indicador[i]="0";
}
```

A análise feita é bastante simples; caso sejam detectadas intersecções entre os dois gráficos, será armazenada no *array* a informação de "compra" ou "venda", conforme o caso, ou então, se não for detectada qualquer intersecção, será armazenado "0".

Depois, o Investidor continua a sua análise, confrontando o *array* `indicador[]` com o *array* `em_carteira[]`, ou seja, em caso de haver sinal e compra e se não ele não tiver o título em carteira, irá efectuar a compra; e em caso de haver sinal de venda e se ele tiver o título em carteira, irá efectuar a venda.

Caso o agente Investidor decida fazer uma compra, irá ser inserido no *array* `investimento[]` as informações relativas ao título que pretende comprar (valor e quantidade). A quantidade é calculada mediante o capital que o agente irá investir. Foi implementado que o agente fará investimentos de 10% do seu capital actual.

```
investimento[x].valor=titulos[x].valor;
long n_accoes=(long) Math.round((0.1*capital)/investimento[x].valor);
investimento[x].quant=n_accoes;
```

É então enviada para o agente Bolsa a seguinte mensagem com a performativa "COMPRA":

```
sendMSG("BOLSA", "COMPRA", investimento[x].desig+","+investimento[x].quant);
```

Em seguida é invocado o método `updateInvestimento()`, que vai fazer a actualização do ficheiro que contém as informações da carteira de títulos. Depois, o agente "adormece" durante 10 segundos.

Após receber o pedido de compra do Investidor, o agente Bolsa irá responder com uma mensagem de realização da compra (performativa "COMPRA-PAGAMENTO"):

```
sendMSG(kqml.getValue("sender"), "COMPRA-PAGAMENTO", titulo + "," +
quant + "," + Valor);
```

A variável `Valor` contém o valor actual da cotação, obtido pelo agente Bolsa através do método `getValor()`. A variável `quant` é a quantidade que o agente Bolsa vai vender e é definida como sendo o valor mínimo entre a quantidade requisitada pelo agente Investidor e a quantidade existente disponível para venda:

```
quant = Math.min(SA.getQuant(), quantreq.longValue());
```

Quando o agente investidor receber a mensagem de realização da compra (performativa "COMPRA-PAGAMENTO") vai descontar do seu capital o montante referente ao investimento e vai actualizar o *array* `em_carteira[]`, colocando o elemento referente ao título no qual foi feito o investimento em `true`.

No caso de haver uma decisão de venda por parte do agente Investidor, este irá enviar ao agente Bolsa uma mensagem de venda (performativa "VENDA"), indicando qual o título que deseja vender e a respectiva quantidade.

```
sendMSG("BOLSA", "VENDA", investimento[x].desig+","+investimento[x].quant);
```

O elemento correspondente do *array* `em_carteira[]` é colocado em `false`, é actualizado o *array* `investimento[]` (`investimento[x].valor=0.0; investimento[x].quant=0;`) e é invocado o método `updateInvestimento()` para fazer a actualização do ficheiro que contém as informações relativas à carteira de títulos.

Como resposta à mensagem de venda, o agente Bolsa irá enviar ao agente Investidor a mensagem de aceitação de venda (performativa "VENDA-PAGAMENTO"):

```
sendMSG(kqml.getValue("sender"), "VENDA-PAGAMENTO", titulo + "," + quant + "," + Valor);
```

O agente Investidor ao receber esta mensagem vai actualizar o valor do seu capital, adicionando-lhe o montante resultante da venda.

Após ter efectuado as decisões de compra ou venda (ou nenhuma delas), o agente Investidor envia de novo ao agente Bolsa uma mensagem perguntando os títulos disponíveis (performativa "TITULOS DISPONIVEIS"), "adormece" durante 30 segundos e recomeça um novo ciclo, repetindo os passos descritos anteriormente.

Quanto à classe `InvestMACD`, que serve para parametrizar este agente Investidor, ela cria novos objectos da classe Investidor, passando-lhes o parâmetro desejado, que neste caso é o capital inicial. Esta classe pode ser chamada da linha de comandos, incluindo o parâmetro pretendido, da seguinte forma:

```
java InvestMACD capIni
```

Por exemplo, se pretendermos um Investidor deste tipo com um capital inicial de €2000, escreve-se na linha de comandos: `java InvestMACD 2`

Para a experimentação, implementei 3 agentes deste tipo (INV5, INV50, INV500), conforme descrito na tabela 3.1:

Capital inicial (€)	Nome do Agente
5000	INV5
50000	INV50
500000	INV500

Tabela 3.1 - Agentes investidores baseados no indicador MACD.

Para uma execução mais fácil foram criados 3 ficheiros *batch*, cada um contendo as instruções da linha de comandos para cada um dos agentes:

- **iniciar_inv5.bat**

```
TITLE INV5
java InvestMACD 5
```

- **iniciar_inv50.bat**

```
TITLE INV50
java InvestMACD 50
```

- **iniciar_inv500.bat**

```
TITLE INV500
java InvestMACD 500
```

A primeira linha (TITLE) serve apenas para se poder visualizar o nome do agente no título da janela.

3.4.4.2. Investidor baseado no indicador estocástico

A classe `InvStk` recebe dois parâmetros: o número de dias referentes à análise do indicador (`nDias`) e a margem de compra/venda. Como foi referido em secção anterior[7], a decisão de compra ou venda associada a este indicador é realizada de acordo com uma margem (por exemplo, margem 20 tem como tem como valores de referência os 20% e os 80%). Na implementação do agente investidor baseado no indicador estocástico vão ser usadas outras margens.

Também nesta classe algumas variáveis vão ser dependentes dos parâmetros: nome do agente, *password*, porta de ligação ao *router* e ainda a variável `Shist`, que é uma *string* que irá ser utilizada quando forem pedidos ao agente Bolsa os históricos desejados.

```
this.n_dias = nDias;
this.margem = margem;
this.port = nDias*100+margem;

margem_sup=100-margem;
margem_inf=margem;
Shist="HISTORICO_"+n_dias;

String identif= "INV"+n_dias+margem_inf;
String passwd= "INV"+n_dias+margem_inf;
```

Por exemplo se se passar os parâmetros ‘30’ e ‘20’, o agente irá ter os seguintes valores nas variáveis:

`n_dias = 30`	(nº de dias)
`margem = 20`	(margem)
`port = 3020`	(porta de ligação ao router)
`margem_sup = 80`	(margem superior)
`margem_inf = 20`	(margem inferior)
`Shist = "HISTORICO_30"`	(*string* utilizada na performativa de pedido do histórico)

[7] Ver secção 2.4.5.

```
identify = "INV3020"          (nome do agente)
passwd = "INV3020"            (password)
```

Após a inicialização é também carregada a informação acerca do capital e da carteira de títulos e depois é enviada a mensagem "TITULOS DISPONIVEIS", de forma igual ao agente anterior, sendo que o tratamento dos dados é feito da mesma maneira.

Depois de ser armazenada a informação nas respectivas variáveis, irá ser enviada para o agente Bolsa a mensagem de pedido do histórico referente ao nº de dias especificado para o agente:

```
sendMSG("BOLSA","ask",Shist);
```

No agente Bolsa foram implementados 3 históricos padrão: 15, 30 e 60 dias, pois foram esses os valores utilizados nos agentes na experimentação. Assim sendo, a mensagem enviada pelo agente Investidor ao agente Bolsa, será uma das seguintes:

```
sendMSG("BOLSA","ask","HISTORICO_15");
sendMSG("BOLSA","ask","HISTORICO_30");
sendMSG("BOLSA","ask","HISTORICO_60");
```

O agente Bolsa ao receber uma das três mensagens irá então fazer a concatenação dos valores dos últimos N dias (N=15,30,60) dos históricos e enviar uma das seguintes mensagens ao Investidor:

```
sendMSG(kqml.getValue("sender"), "HISTORICO_15", Smsg);
sendMSG(kqml.getValue("sender"), "HISTORICO_30", Smsg);
sendMSG(kqml.getValue("sender"), "HISTORICO_60", Smsg);
```

A variável `Smsg` é a *string* concatenada.

Depois de ser recebido o histórico, é invocado o método `splitHist()`, para fazer a separação em *tokens* e armazenamento dos valores do histórico, à semelhança do caso anterior. Os valores do histórico são então armazenados no *array* bidimensional `historico[][]`.

Depois de terem sido obtidos os históricos referentes aos últimos N dias, é efectuado um ciclo para detectar os máximos e os mínimos de cada título dentro desse conjunto de valores e é ainda calculado o valor do indicador estocástico (*stk*) para cada título:

```
for (int x=0; x<20; x++)
{
      minimo[x]=historico[x][0];
      maximo[x]=historico[x][0];
      for (int y=0; y<n_dias; y++)
      {
            if (historico[x][y] < minimo[x] && historico[x][y] > 0)
            {
                  minimo[x]=historico[x][y];
            }
                  if (historico[x][y] > maximo[x])
            {
                  maximo[x]=historico[x][y];
            }
            if (minimo[x] != maximo[x])
            {
```

```
            stk[x][y]=100*(historico[x][y]-minimo[x])/(maximo[x]-
minimo[x]);
            }
      }
}
```

Em paralelo com esta operação é também feita a extracção dos dados para ficheiros para que possam posteriormente ser importados no Excel. A extensão dos ficheiros é dependente dos parâmetros do agente: será .s15, .s30 ou .s60. Por exemplo, para o caso do título 'BCP', será:

```
FileOutputStream BCP_stk_out = new
                        FileOutputStream("..\\files\\BCP.s"+n_dias);
```

Em seguida começa a ser feita a análise dos dados, verificando-se se há sinais de compra ou venda:

```
for (int i=0; i<20; i++)
{
      if (stk[i][n_dias-1] >= margem_inf && stk[i][n_dias-2] < margem_inf)
      {
            indicador[i]="compra";
      }
      else
      if (stk[i][n_dias-1] <=  margem_sup && stk[i][n_dias-2] > margem_sup)
      {
            indicador[i]="venda";
      }
      else indicador[i]="0";
}
```

Se o estocástico (stk) estava abaixo/acima da margem inferior/superior para agora estar acima/abaixo dessa mesma margem, então há um sinal de compra/venda. Os sinais de compra ou venda (ou nenhum) são representados no *array* `indicador[]`, à semelhança do agente anterior.

A partir deste ponto, tudo se processa de forma idêntica ao agente anterior, tanto nas mensagens trocadas com o agente Bolsa, como nas actualizações das variáveis e ficheiros. Quanto aos investimentos, todos os investidores têm um capital inicial de €500000 e fazem investimentos de €25000.

No que diz respeito à classe `InvestStk`, ela funciona da mesma forma que a classe InvestMACD, criando objectos da classe `InvStk`; a única diferença é que neste caso são passados dois parâmetros:

```
java InvestStk nDias margem
```

Por exemplo, se for pretendido um agente que faça a análise baseada nos últimos 30 dias e que tenha uma margem de 20%, deverá ser escrito na linha de comandos:

```
java InvestStk 30 20
```

Para a experimentação foram criados 9 agentes deste tipo, conforme é especifcado na tabela 3.2:

		margem		
		90/10	80/20	70/30
nDias	15 dias	**INV1510**	**INV1520**	**INV1530**
	30 dias	**INV3010**	**INV3020**	**INV3030**
	60 dias	**INV6010**	**INV6020**	**INV6030**

Tabela 3.2 - Agentes investidores baseados no indicador estocástico.

Foram também criados ficheiros *batch*:

- **iniciar_inv1510.bat**

```
TITLE INV1510
java InvestStk 15 10
```

- **iniciar_inv1520.bat**

```
TITLE INV1520
java InvestStk 15 20
```

- **iniciar_inv1530.bat**

```
TITLE INV1530
java InvestStk 15 30
```

- **iniciar_inv3010.bat**

```
TITLE INV3010
java InvestStk 30 10
```

- **iniciar_inv3020.bat**

```
TITLE INV3020
java InvestStk 30 20
```

- **iniciar_inv3030.bat**

```
TITLE INV3030
java InvestStk 30 30
```

- **iniciar_inv6010.bat**

```
TITLE INV6010
java InvestStk 60 10
```

- **iniciar_inv6020.bat**

```
TITLE INV6020
java InvestStk 60 20
```

- **iniciar_inv1530.bat**

```
TITLE INV6030
java InvestStk 60 30
```

Foi ainda criado um outro ficheiro que permite iniciar todos os investidores em simultâneo (**iniciar.bat**):

```
start iniciar_inv5
start iniciar_inv50
start iniciar_inv500
start iniciar_inv1510
start iniciar_inv1520
start iniciar_inv1530
start iniciar_inv3010
start iniciar_inv3020
start iniciar_inv3030
start iniciar_inv6010
start iniciar_inv6020
start iniciar_inv6030
```

O comando `start` permite abrir uma nova janela e aí iniciar um novo agente.

4. Execução

A execução aqui descrita foi realizada no sistema operativo Windows 2000.
O primeiro passo para executar o SMA é iniciar o *router* do JATLite. Para tal é necessário fazer o *Setup*, "clicando" no ícone "Setup" existente dentro da pasta onde está instalado o JATLite (figura 4.1).

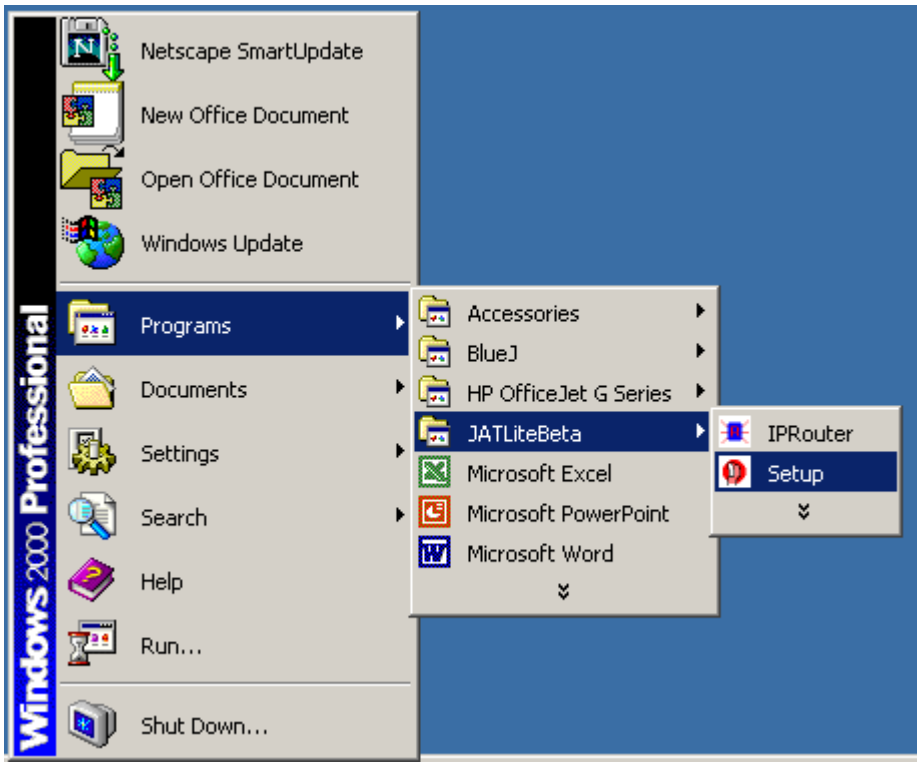


Figura 4.1 - Inicialização do *router* do JATLite.

Aparece então a seguinte janela onde podem ser colocados os parâmetros relativos à inicialização do *router* (figura 4.2):

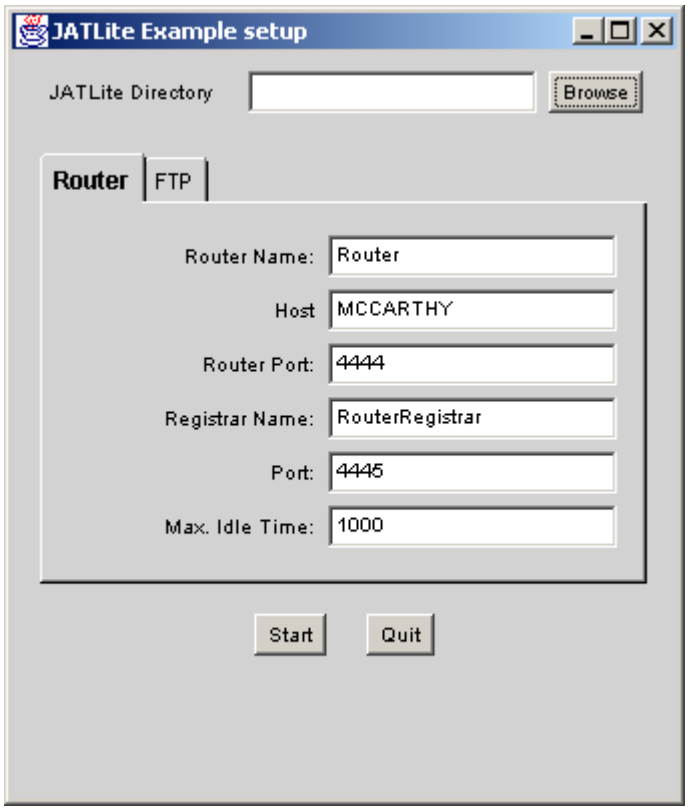


Figura 4.2 - Exemplo de inicialização do JATLite

De seguida inicia-se o router "clicando" no ícone "IPRouter". Aparece então a janela do *router*, apresentada na figura 4.3:

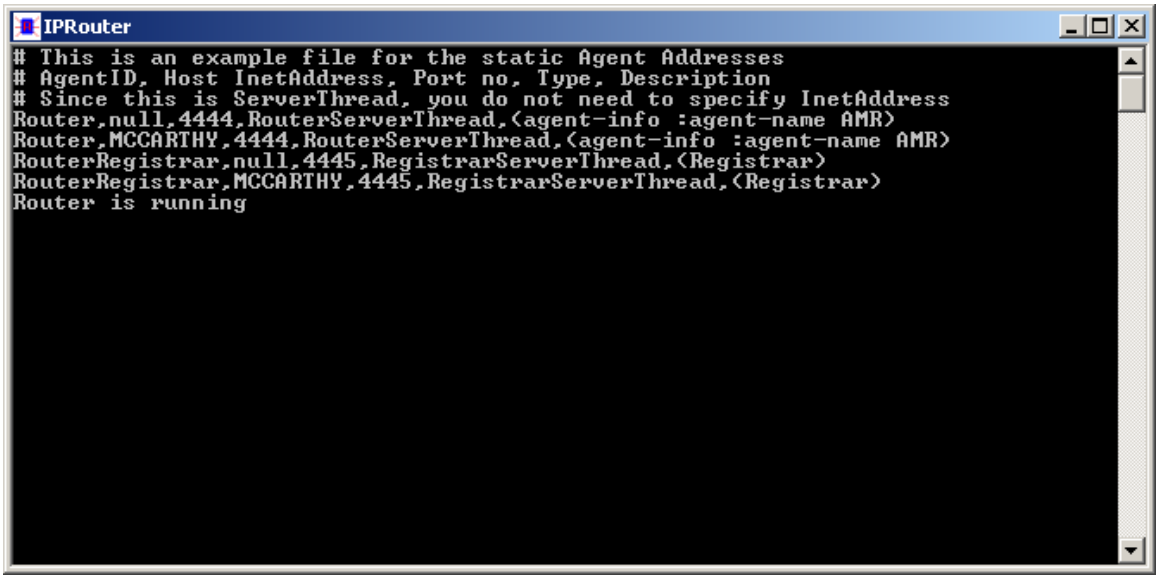


Figura 4.3 - *Router* do JATLite.

O próximo passo é iniciar o programa de extracção das cotações, para tal basta executar o ficheiro *batch* **iniciar_extr.bat**, conforme apresentado na figura 4.4.

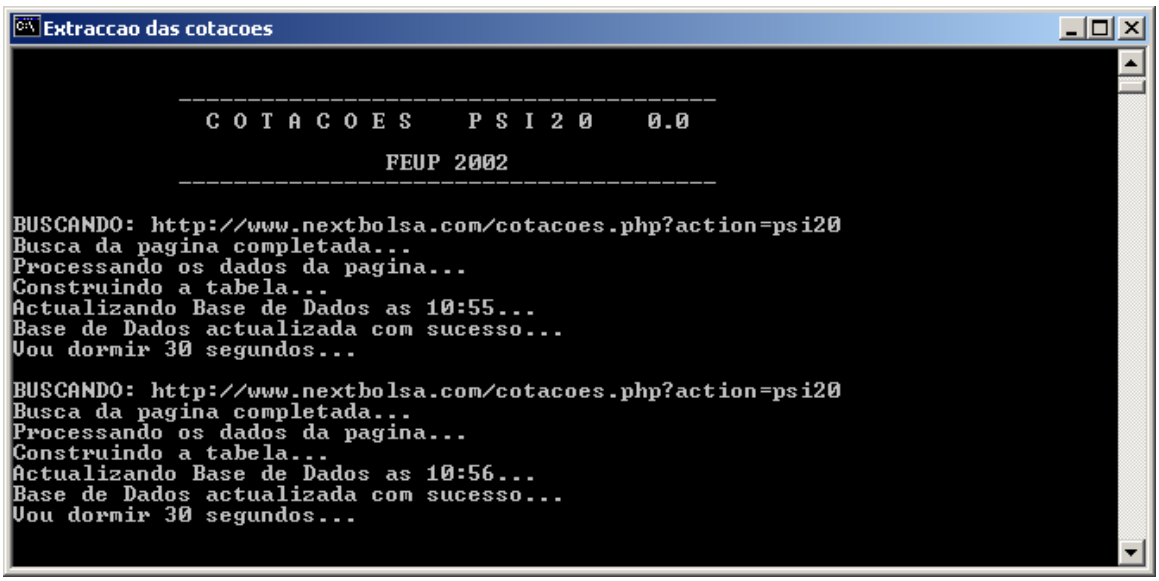


Figura 4.4 - Extracção das cotações.

Após ter sido feita a primeira extracção, já poderão ser visualizadas as cotações actualizadas na página *Web* implementada para o efeito e que já foi referida anteriormente (ver figura 3.4).

O passo seguinte será a iniciação do agente Bolsa, apresentada na figura 4.5. Para tal foi também criado um ficheiro *batch* (**iniciar_bolsa.bat**).

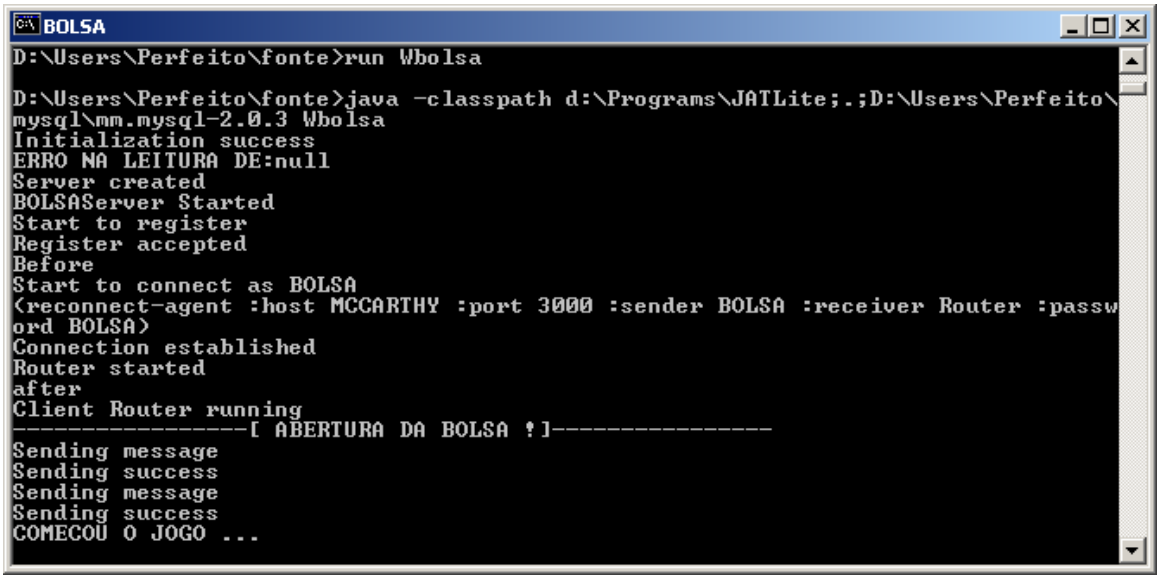


Figura 4.5 - Iniciação do agente Bolsa.

É então aberta a janela que implementa a interface gráfica do agente Bolsa (figura 4.6):

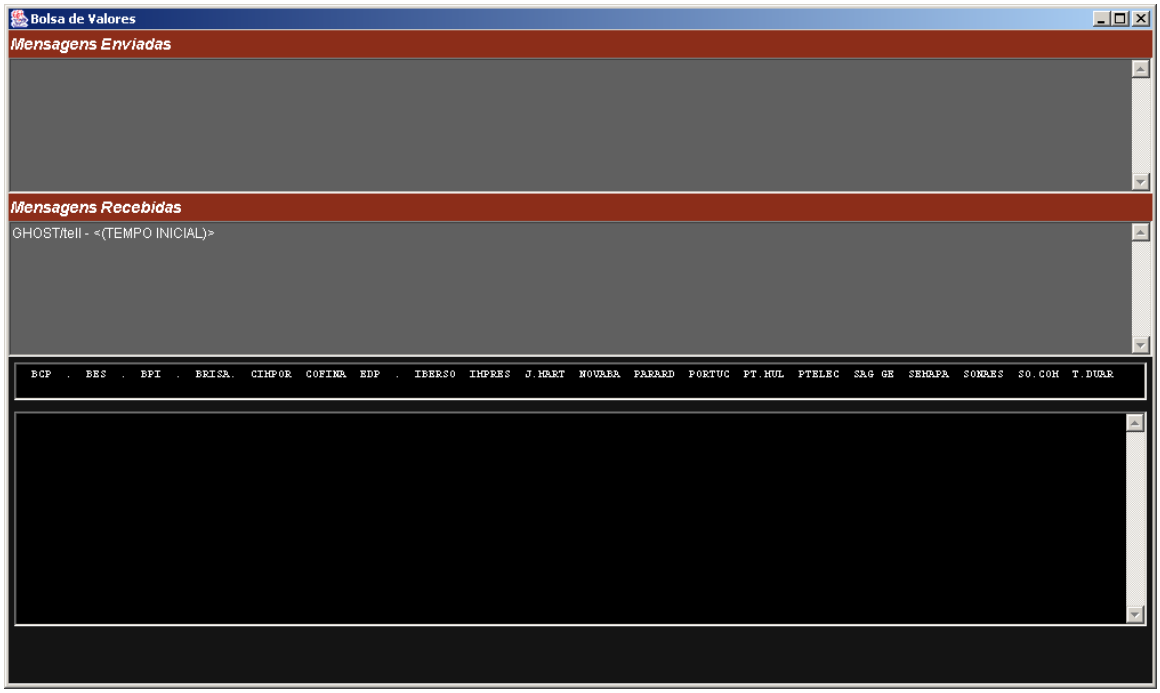


Figura 4.6 - Agente Bolsa.

Após estar o agente Bolsa a "correr" iniciam-se os agentes investidores com o ficheiro que já foi referido (**iniciar.bat**). A figura 4.7 apresenta a execução de um agente investidor baseado no indicador estocástico, com uma análise de 60 dias e margem de compra/venda de 30%.

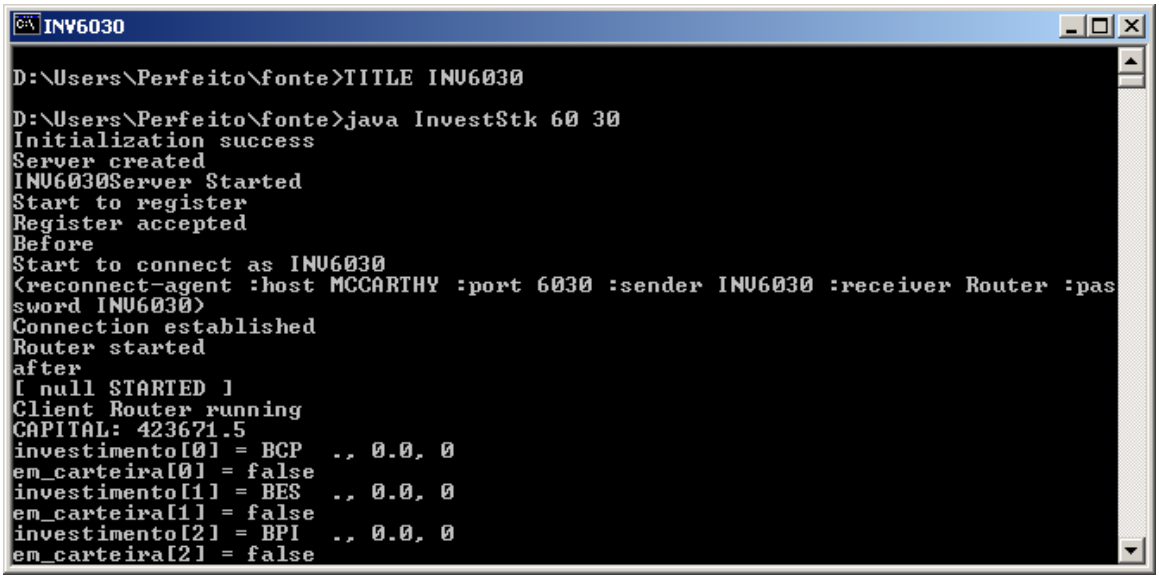


Figura 4.7 - Agente INV6030.

São apresentados de seguida alguns *screenshots* referentes a algumas das funcionalidades que já foram referidas.

Na figura 4.8 pode ver-se os indicadores de compra/venda no agente INV50, após este ter recebido as informações acerca dos valores de "MACD" e "signal" dos dois últimos dias e ter efectuado a análise dos mesmos.

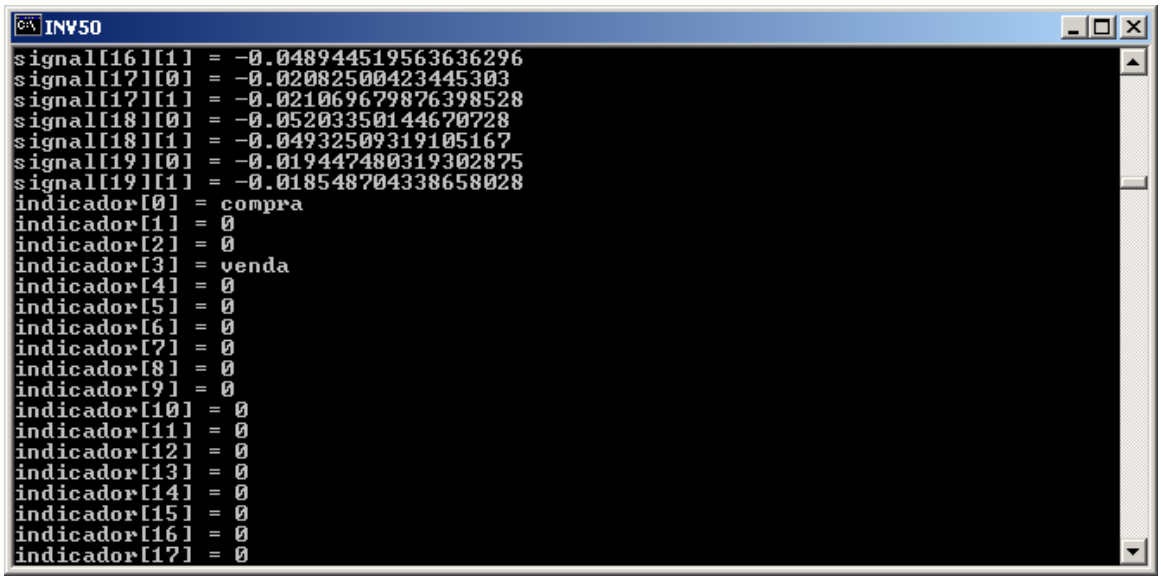


Figura 4.8 - Indicadores no agente INV50.

Repare-se no sinal de venda deduzido pelo agente INV50 (`indicador[3]=venda`). Na figura 4.9 vê-se a decisão de venda do título 3 por parte do agente INV50.

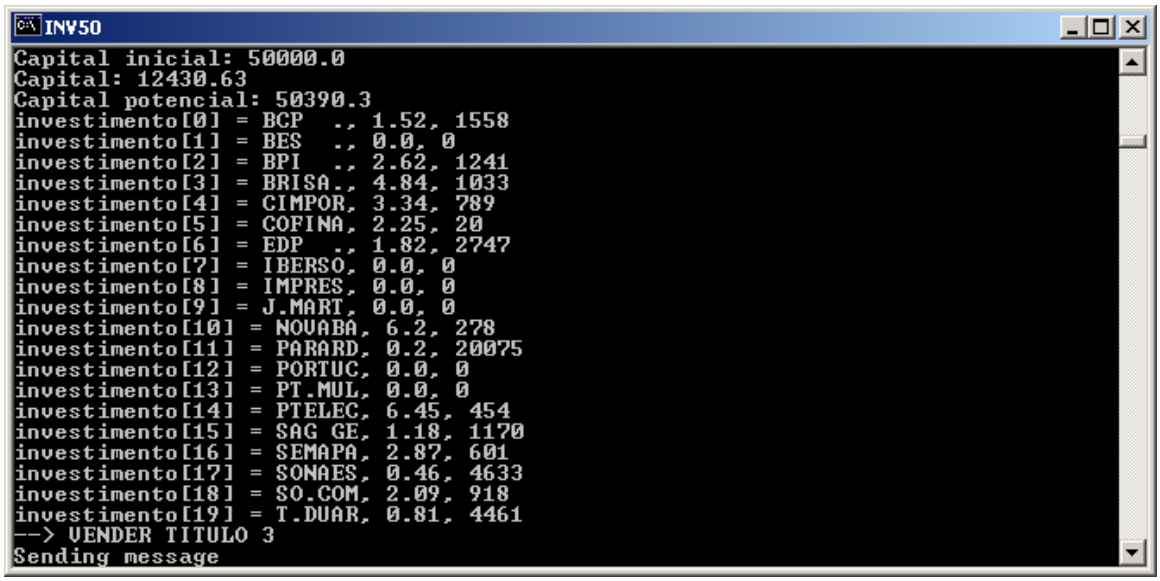


Figura 4.9 - Venda de um título.

Na figura 4.10 é apresentada a interface gráfica do agente Bolsa, onde se pode observar diversas mensagens enviadas e recebidas e ainda o painel com a actualização das cotações.

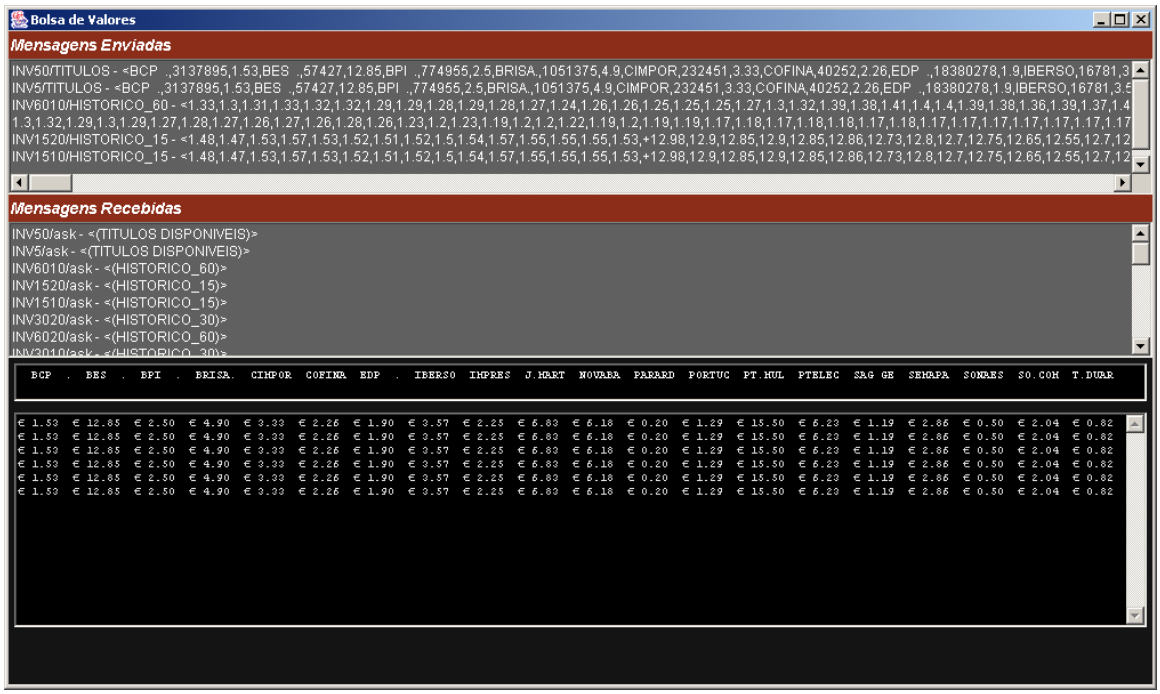


Figura 4.10 - Agente Bolsa em acção (I).

Na figura 4.11 podem observar-se os pedidos de pagamento por parte do agente Bolsa, referentes a compras efectuadas pelos investidores.

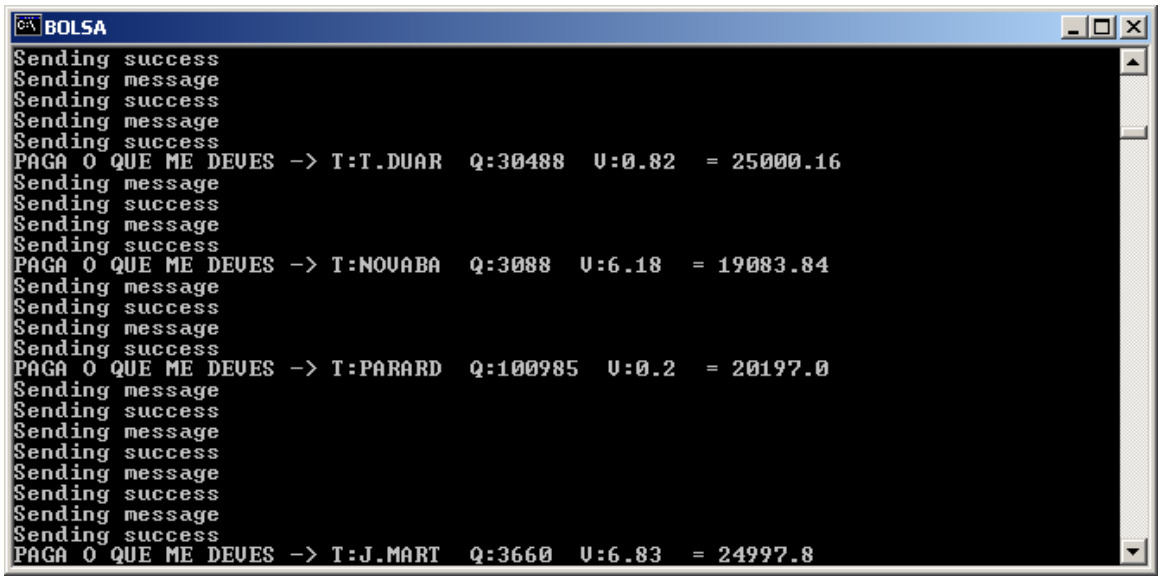


Figura 4.11 - Agente Bolsa em acção (II).

Na figura 4.12 estão representados os sinais de compra/venda deduzidos pelo agente INV1520 (agente investidor baseado no indicador estocástico, com uma análise de 15 dias e margem de compra/venda de 20%), após análise dos valores do STK. Repare-se no sinal de venda gerado para o título 19, após o STK ter passado dos 90% para os 70%, ou seja, estava acima dos 80% e passou a estar abaixo dos 80%.

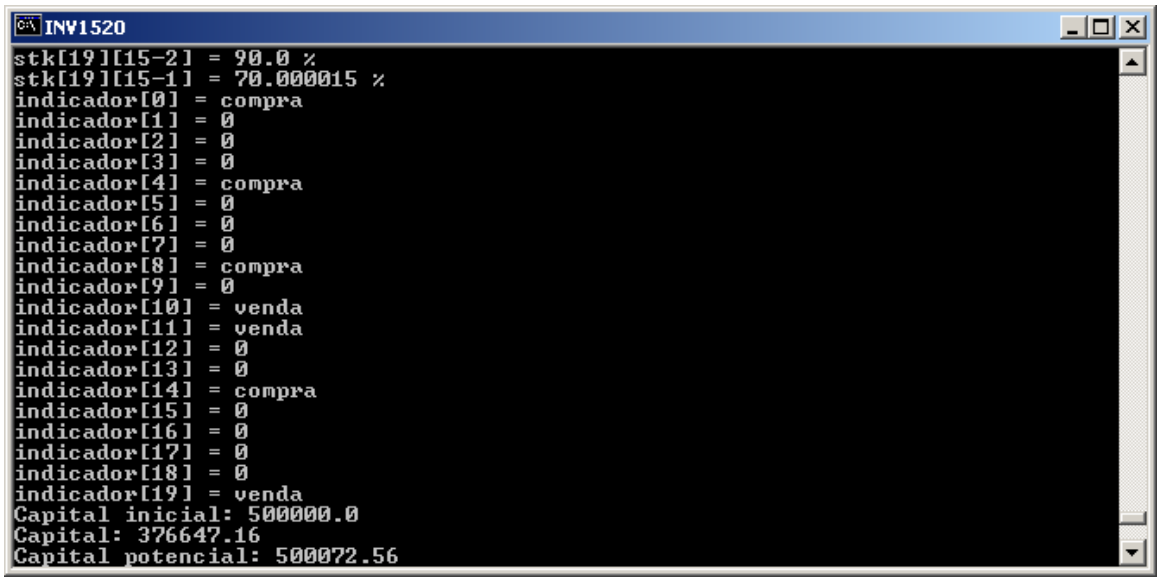


Figura 4.12 - Indicadores no agente INV1520.

Na figura 4.13 pode ver-se as decisões de compra e venda por parte do agente INV1510. Repare-se ainda na realização do pagamento relativo à compra e ao recebimento do pagamento relativo à venda.

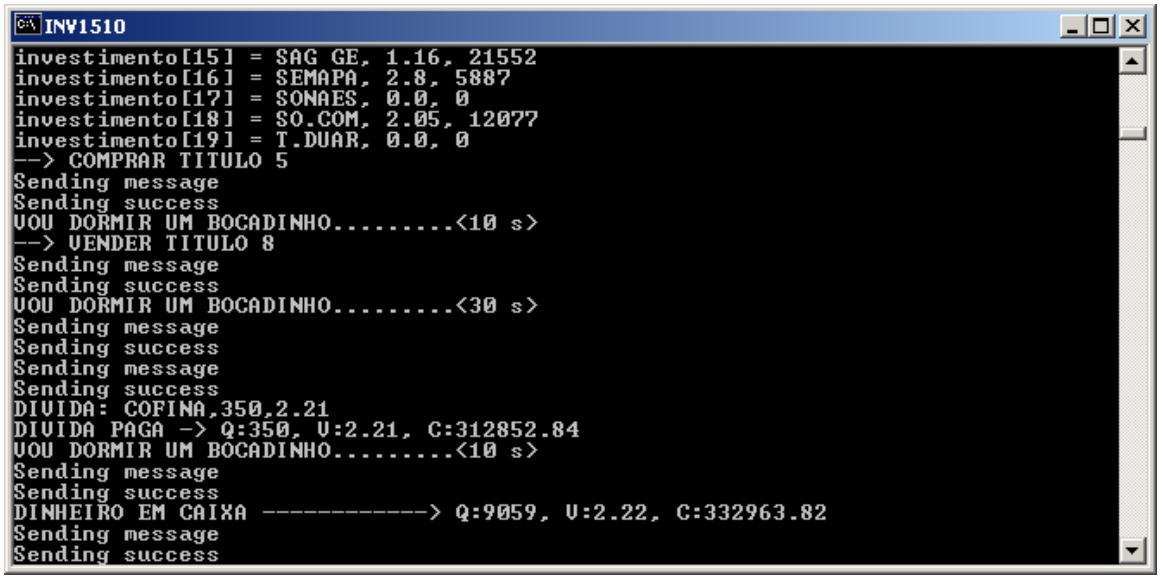


Figura 4.13 - Agente INV1510 em acção.

5. Resultados da experimentação

A experimentação foi realizada entre os dias 12/06/2003 e 09/07/2003, resultando um total de 20 dias úteis de funcionamento da Bolsa. Nesta experimentação foram incluidos três agentes investidores baseados no MACD (com capital inicial de €5000, €50000 e €500000), e nove agentes investidores baseados no indicador estocástico (com análise de 15, 30 e 60 dias e margem de compra/venda de 10%, 20% e 30%). O capital inicial dos agentes baseados no indicador estocástico é de €500000. No início da experimentação todos os agentes têm a sua carteira "vazia", isto é, nenhum dos agentes possui títulos.

5.1. Exposição dos resultados

De seguida são apresentados os resultados para cada um dos 12 agentes. São apresentados os valores da carteira (dinheiro e acções) e da rentabilidade (percentagem relativa ao valor inicial da carteira). Os valores apresentados são referentes ao final de cada dia.

INTENCIONALMENTE EM BRANCO

- **INV5**

Investidor baseado no indicador MACD, com capital inicial de €5000.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
0.00	5000.00	**5000.00**	0.000
1500.67	3499.33	**5000.00**	0.000
1531.91	3499.33	**5031.24**	0.621
1415.22	3614.23	**5029.45**	0.586
2496.68	2514.43	**5011.11**	0.222
2456.22	2514.43	**4970.65**	-0.590
3089.66	1945.14	**5034.80**	0.691
3262.88	1790.48	**5053.36**	1.056
3491.67	1516.32	**5007.99**	0.160
3624.57	1405.40	**5029.97**	0.596
3602.88	1405.40	**5008.28**	0.165
3132.61	1910.10	**5042.71**	0.847
2686.90	2356.50	**5043.40**	0.861
2680.02	2356.50	**5036.52**	0.725
2681.30	2356.50	**5037.80**	0.750
2168.88	2890.00	**5058.88**	1.164
2156.79	2890.00	**5046.79**	0.927
2099.60	2978.21	**5077.81**	1.532
2102.60	2978.21	**5080.81**	1.590
2375.86	2699.41	**5075.27**	1.483

Tabela 5.1 - Resultados experimentais do agente INV5.

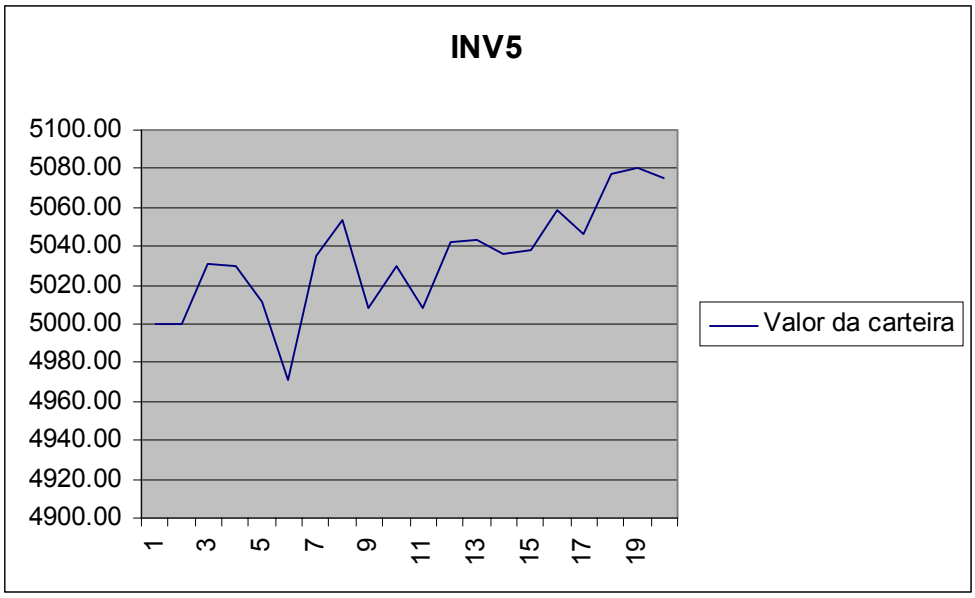


Gráfico 5.1 - Evolução do valor da carteira do agente INV5.

- **INV50**

Investidor baseado no indicador MACD, com capital inicial de €50000.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
0.00	50000.00	**50000.00**	0.000
14996.71	35003.29	**50000.00**	0.000
15308.45	35003.29	**50311.74**	0.620
17771.83	32521.86	**50293.69**	0.584
26381.05	23677.66	**50058.71**	0.117
25935.23	23677.66	**49612.89**	-0.780
31158.06	19178.32	**50336.38**	0.668
33247.81	17259.70	**50507.51**	1.005
36190.57	13811.23	**50001.80**	0.004
37825.77	12430.63	**50256.40**	0.510
37593.60	12430.63	**50024.23**	0.048
32909.09	17492.33	**50401.42**	0.796
28808.86	21507.23	**50316.09**	0.628
28776.92	21507.23	**50284.15**	0.565
28763.29	21507.23	**50270.52**	0.538
23669.70	26836.41	**50506.11**	1.002
23534.06	26836.41	**50370.47**	0.735
22395.13	28368.71	**50763.84**	1.505
28031.64	22693.32	**50724.96**	1.429
27702.67	22791.33	**50494.00**	0.978

Tabela 5.2 - Resultados experimentais do agente INV50.

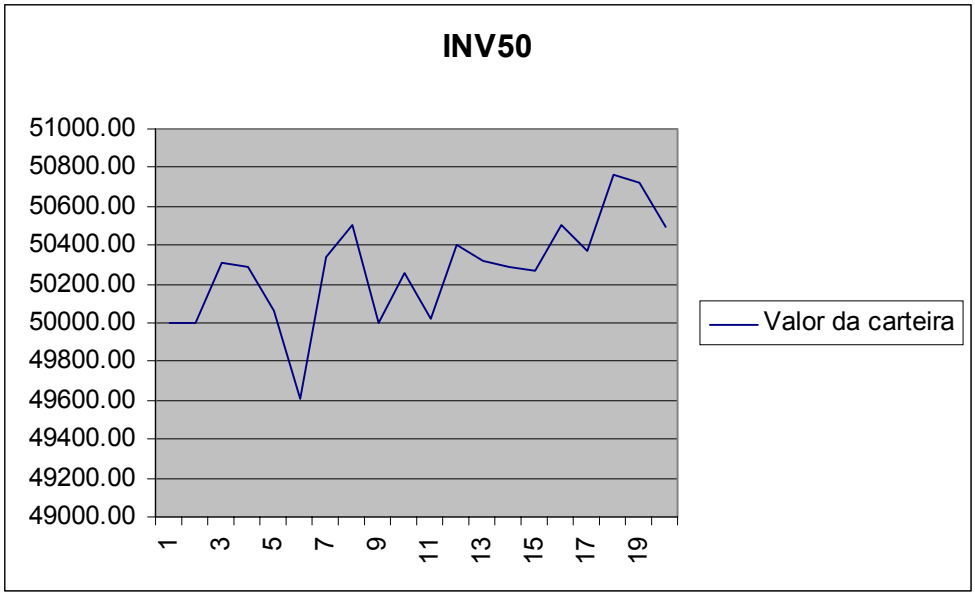


Gráfico 5.2 - Evolução do valor da carteira do agente INV50.

- **INV500**

Investidor baseado no indicador MACD, com capital inicial de €500000.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
0.00	500000.00	**500000.00**	0.000
150002.80	349997.20	**500000.00**	0.000
153120.95	349997.20	**503118.15**	0.620
101523.23	401414.59	**502937.82**	0.584
236834.75	263741.89	**500576.64**	0.115
233165.89	263741.89	**496907.78**	-0.622
289164.28	212868.11	**502032.39**	0.405
312214.67	191581.46	**503796.13**	0.754
328042.69	170526.39	**498569.08**	-0.287
347555.74	153474.21	**501029.95**	0.206
346628.53	153474.21	**500102.74**	0.021
299443.84	204096.11	**503539.95**	0.703
294972.16	204183.41	**499155.57**	-0.169
294903.71	204183.41	**499087.12**	-0.183
294696.15	204183.41	**498879.56**	-0.225
243497.53	257481.03	**500978.56**	0.195
242245.01	257481.03	**499726.04**	-0.055
222451.98	281113.95	**503565.93**	0.708
278864.56	224813.54	**503678.10**	0.730
274049.71	227144.86	**501194.57**	0.238

Tabela 5.3 - Resultados experimentais do agente INV500.

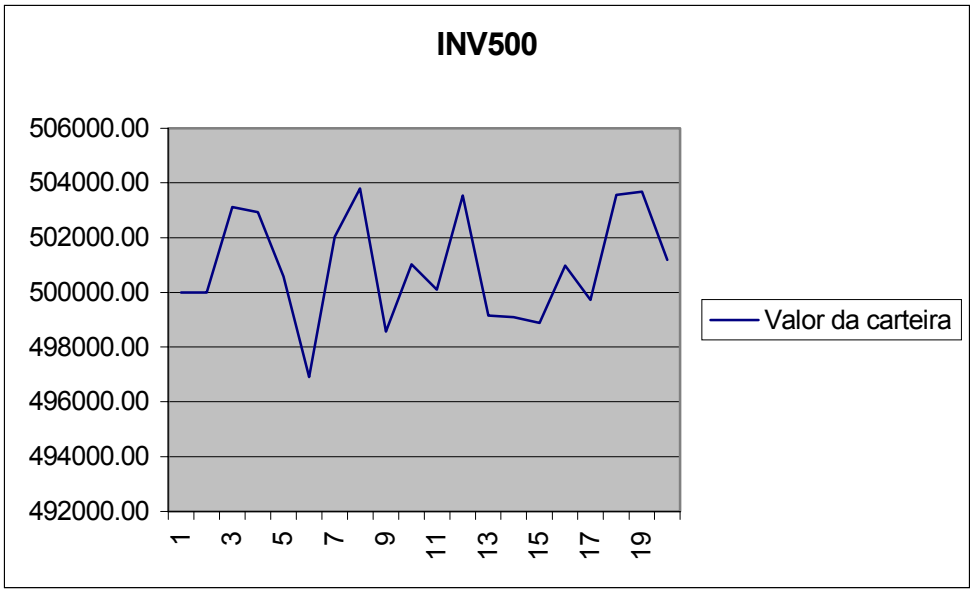


Gráfico 5.3 - Evolução do valor da carteira do agente INV500.

- **INV1510**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 15 dias e margem de 10%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
0.00	500000.00	**500000.00**	0.000
24999.95	475000.05	**500000.00**	0.000
25511.76	475000.05	**500511.81**	0.102
50438.28	450001.97	**500440.25**	0.088
74914.19	425004.29	**499918.48**	-0.016
74167.94	425004.29	**499172.23**	-0.166
124878.12	375003.43	**499881.55**	-0.024
124787.81	375003.43	**499791.24**	-0.042
123293.82	375003.43	**498297.25**	-0.342
175276.96	325000.43	**500277.39**	0.055
175350.59	325000.43	**500351.02**	0.070
196423.08	305406.53	**501829.61**	0.365
169653.34	330955.49	**500608.83**	0.122
118910.17	381251.25	**500161.42**	0.032
185388.62	313626.34	**499014.96**	-0.197
154607.51	343958.28	**498565.79**	-0.288
178275.39	318958.28	**497233.67**	-0.556
204557.22	293958.52	**498515.74**	-0.298
204719.64	293958.52	**498678.16**	-0.265
202789.88	293958.52	**496748.40**	-0.655

Tabela 5.4 - Resultados experimentais do agente INV1510.

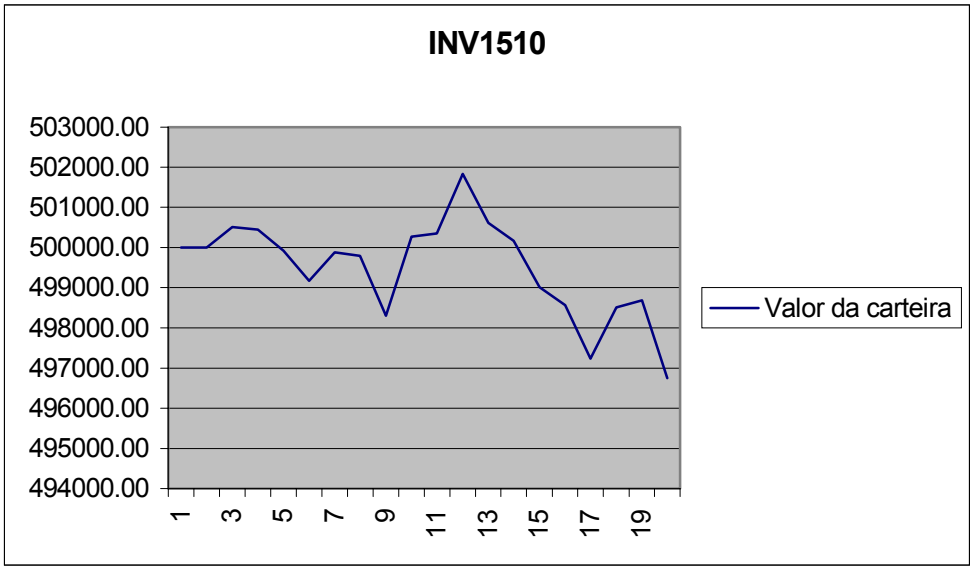


Gráfico 5.4 - Evolução do valor da carteira do agente INV1510.

- **INV1520**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 15 dias e margem de 20%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
25221.20	475001.34	**500222.54**	0.044
49998.61	450001.39	**500000.00**	0.000
50510.42	450001.39	**500511.81**	0.102
75585.30	425003.31	**500588.61**	0.118
74875.79	424335.95	**499211.74**	-0.158
74129.84	424335.95	**498465.79**	-0.308
124567.20	374334.53	**498901.73**	-0.220
124469.74	374334.53	**498804.27**	-0.240
122982.01	374334.53	**497316.54**	-0.540
132818.66	366911.53	**499730.19**	-0.054
133309.31	366416.87	**499726.18**	-0.055
184233.98	316416.71	**500650.69**	0.130
160458.27	341688.53	**502146.80**	0.428
157892.85	342888.00	**500780.85**	0.156
123310.83	376647.16	**499957.99**	-0.008
186126.65	312927.26	**499053.91**	-0.190
184663.82	312927.26	**497591.08**	-0.484
186282.12	312927.26	**499209.38**	-0.158
198886.70	300555.68	**499442.38**	-0.112
178555.01	318803.24	**497358.25**	-0.531

Tabela 5.5 - Resultados experimentais do agente INV1520.

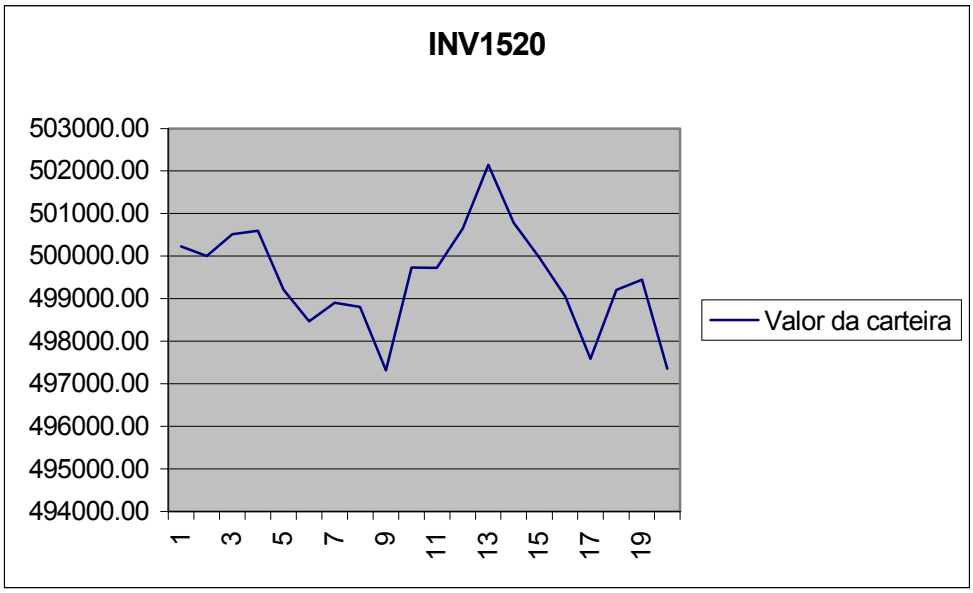


Gráfico 5.5 - Evolução do valor da carteira do agente INV1520.

- **INV1530**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 15 dias e margem de 30%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
50105.20	449703.96	**499809.16**	-0.038
24809.20	474776.76	**499585.96**	-0.083
49999.88	449776.92	**499776.80**	-0.045
99803.12	399781.00	**499584.12**	-0.083
99507.58	399781.00	**499288.58**	-0.142
98964.04	399781.00	**498745.04**	-0.252
99295.89	399850.38	**499146.27**	-0.171
130573.06	367988.54	**498561.60**	-0.289
129113.20	367988.54	**497101.74**	-0.583
155986.94	342982.24	**498969.18**	-0.207
181241.76	317890.97	**499132.73**	-0.174
250585.31	248806.85	**499392.16**	-0.122
228393.06	273541.13	**501934.19**	0.385
106595.91	393414.44	**500010.35**	0.002
189737.46	310027.82	**499765.28**	-0.047
195200.18	303545.16	**498745.34**	-0.252
213683.47	283911.36	**497594.83**	-0.483
214947.99	283911.36	**498859.35**	-0.229
216016.15	283911.36	**499927.51**	-0.015
189458.26	309016.11	**498474.37**	-0.306

Tabela 5.6 - Resultados experimentais do agente INV1530.

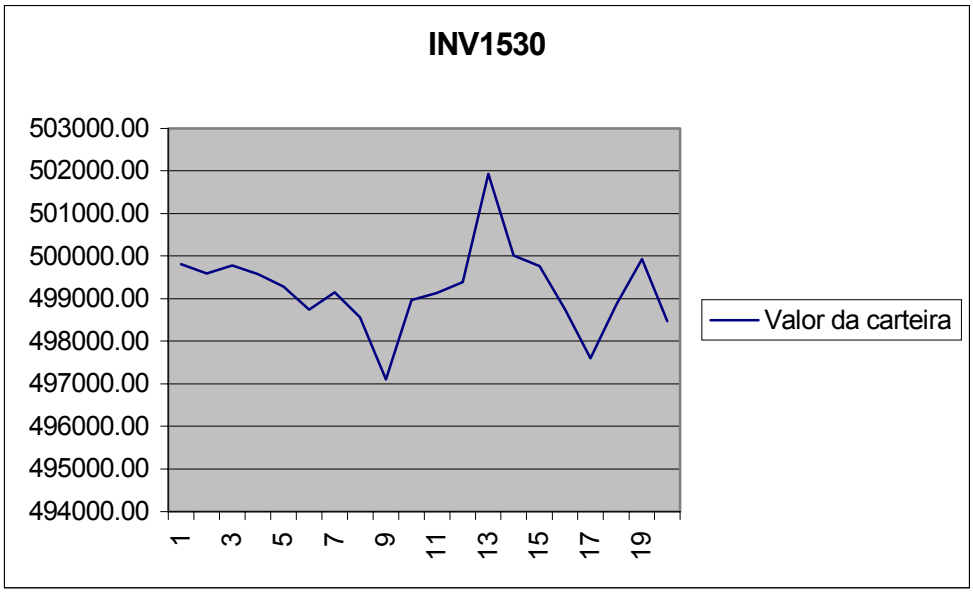


Gráfico 5.6 - Evolução do valor da carteira do agente INV1530.

- **INV3010**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 30 dias e margem de 10%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
25221.20	475001.34	**500222.54**	0.044
49998.61	450001.39	**500000.00**	0.000
50510.42	450001.39	**500511.81**	0.102
50816.26	450001.39	**500817.65**	0.163
50156.09	450001.39	**500157.48**	0.031
49801.76	450001.39	**499803.15**	-0.039
75762.68	425000.05	**500762.73**	0.152
76239.19	425000.05	**501239.24**	0.247
75663.70	425000.05	**500663.75**	0.133
150769.50	349942.03	**500711.53**	0.142
99857.85	401481.95	**501339.80**	0.267
100284.24	401481.95	**501766.19**	0.352
125642.06	376481.91	**502123.97**	0.423
125071.00	376481.91	**501552.91**	0.310
151511.29	351266.07	**502777.36**	0.552
124783.88	377304.81	**502088.69**	0.416
124068.83	377304.81	**501373.64**	0.274
125128.81	377304.81	**502433.62**	0.484
126090.19	377304.81	**503395.00**	0.674
99269.61	403150.11	**502419.72**	0.482

Tabela 5.7 - Resultados experimentais do agente INV3010.

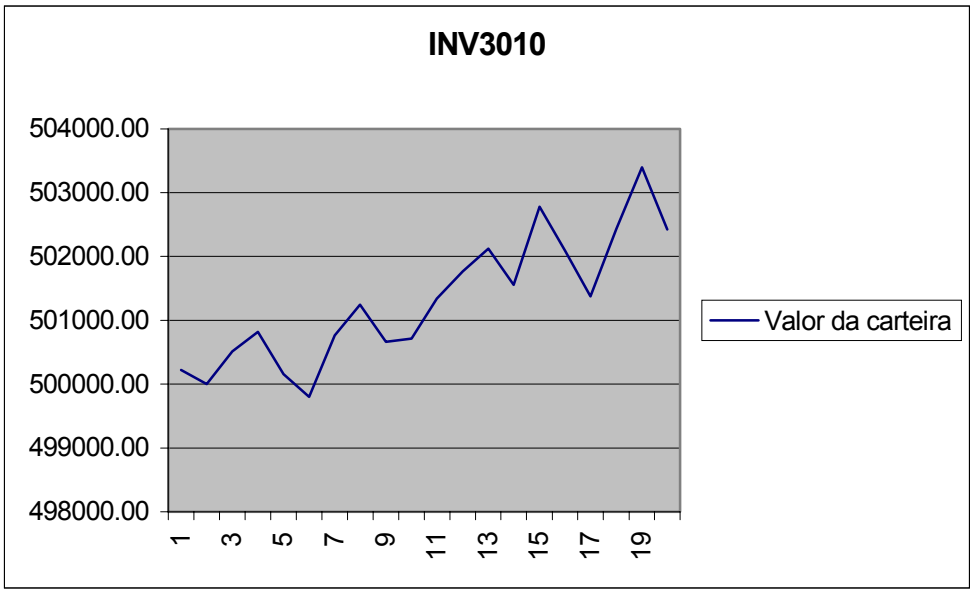


Gráfico 5.7 - Evolução do valor da carteira do agente INV3010.

- **INV3020**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 30 dias e margem de 20%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
25296.00	474704.00	**500000.00**	0.000
50072.75	449704.05	**499776.80**	-0.045
50584.56	449704.05	**500288.61**	0.058
50890.84	449704.05	**500594.89**	0.119
50230.23	449704.05	**499934.28**	-0.013
49875.90	449704.05	**499579.95**	-0.084
100840.67	399699.96	**500540.63**	0.108
101122.18	399699.96	**500822.14**	0.164
100351.25	399699.96	**500051.21**	0.010
126182.70	374650.36	**500833.06**	0.166
126050.41	375007.32	**501057.73**	0.211
127001.78	375007.32	**502009.10**	0.400
125800.68	376196.08	**501996.76**	0.398
125230.80	376196.08	**501426.88**	0.285
151467.21	351195.84	**502663.05**	0.530
149409.56	352079.49	**501489.05**	0.297
148600.75	352079.49	**500680.24**	0.136
149660.11	352079.49	**501739.60**	0.347
150416.85	352079.49	**502496.34**	0.497
149198.63	352079.49	**501278.12**	0.255

Tabela 5.8 - Resultados experimentais do agente INV3020.

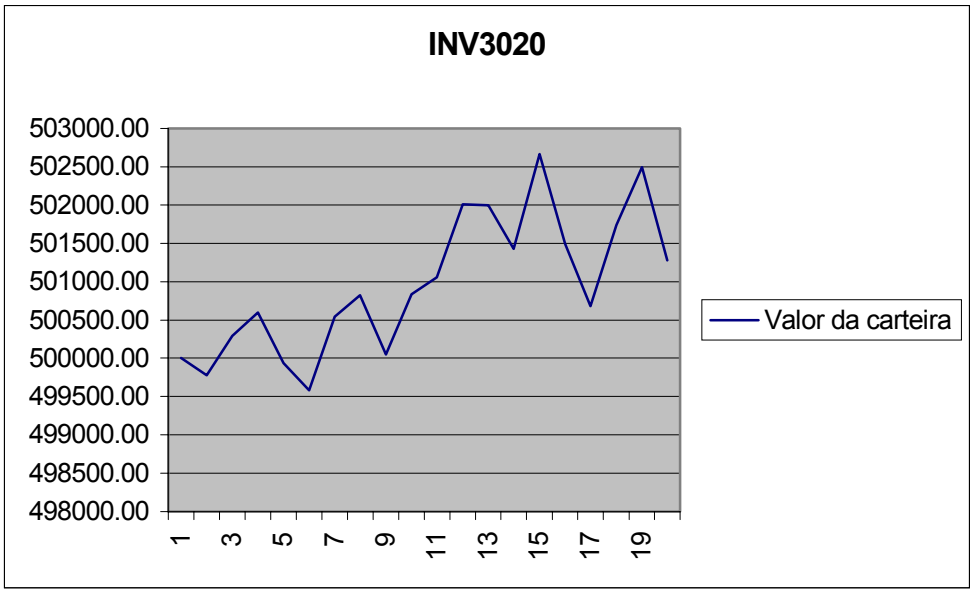


Gráfico 5.8 - Evolução do valor da carteira do agente INV3020.

- **INV3030**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 30 dias e margem de 30%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
0.00	500000.00	**500000.00**	0.000
0.00	500000.00	**500000.00**	0.000
24999.84	475000.16	**500000.00**	0.000
50224.88	450002.32	**500227.20**	0.045
99824.42	400004.60	**499829.02**	-0.034
99281.60	400004.60	**499286.20**	-0.143
130002.55	369822.55	**499825.10**	-0.035
130020.49	369822.55	**499843.04**	-0.031
128491.82	369822.55	**498314.37**	-0.338
178946.56	319774.08	**498720.64**	-0.257
153877.20	345202.72	**499079.92**	-0.184
154725.63	345202.72	**499928.35**	-0.014
154982.84	345202.72	**500185.56**	0.037
98456.91	400673.84	**499130.75**	-0.174
123859.84	375672.20	**499532.04**	-0.094
147429.55	351249.74	**498679.29**	-0.265
146490.89	351249.74	**497740.63**	-0.454
172655.50	326249.37	**498904.87**	-0.220
173802.28	326249.37	**500051.65**	0.010
172365.49	326249.37	**498614.86**	-0.278

Tabela 5.9 - Resultados experimentais do agente INV3030.

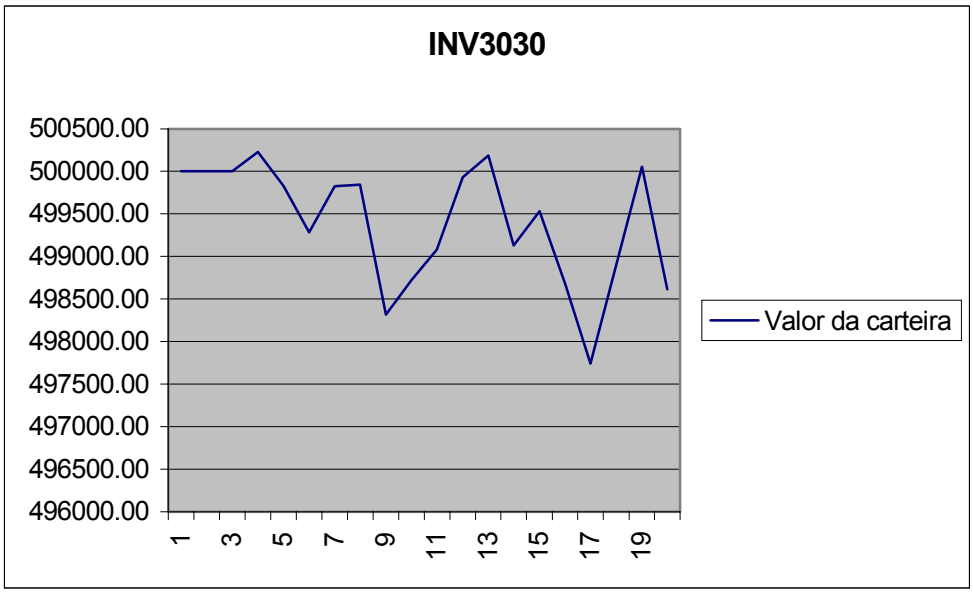


Gráfico 5.9 - Evolução do valor da carteira do agente INV3030.

- **INV6010**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 60 dias e margem de 10%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
0.00	500000.00	**500000.00**	0.000
24999.95	475000.05	**500000.00**	0.000
25511.76	475000.05	**500511.81**	0.102
25669.24	475000.05	**500669.29**	0.134
25157.43	475000.05	**500157.48**	0.031
24803.10	475000.05	**499803.15**	-0.039
25236.17	475000.05	**500236.22**	0.047
25393.65	475000.05	**500393.70**	0.079
24803.10	475000.05	**499803.15**	-0.039
24488.14	475000.05	**499488.19**	-0.102
24763.73	475000.05	**499763.78**	-0.047
49525.31	450002.25	**499527.56**	-0.095
49784.28	450002.25	**499786.53**	-0.043
49860.25	450002.25	**499862.50**	-0.028
50164.13	450002.25	**500166.38**	0.033
74004.35	425579.79	**499584.14**	-0.083
73124.47	425579.79	**498704.26**	-0.260
74175.42	425579.79	**499755.21**	-0.049
74927.66	425579.79	**500507.45**	0.101
48546.15	451382.79	**499928.94**	-0.014

Tabela 5.10 - Resultados experimentais do agente INV6010.

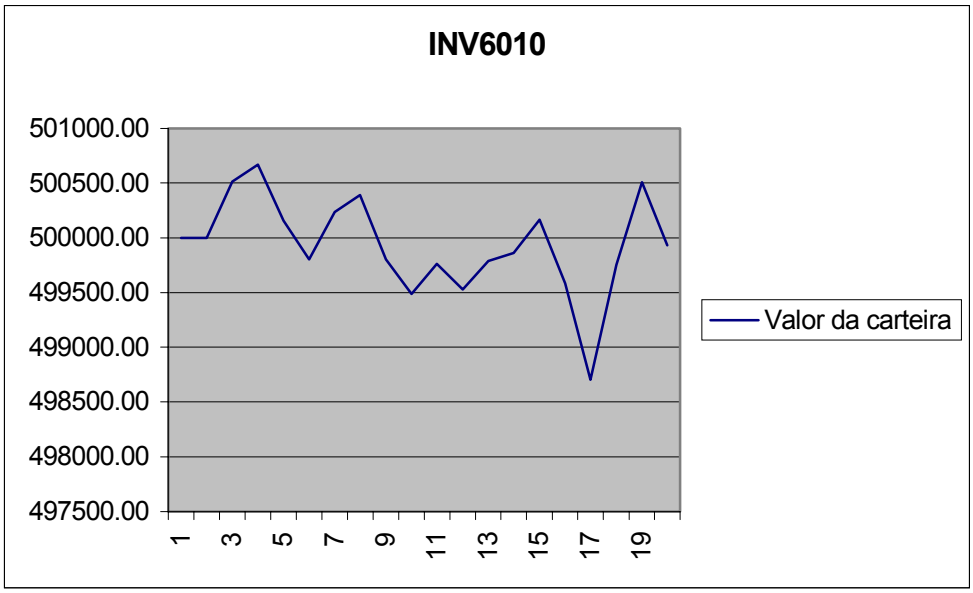


Gráfico 5.10 - Evolução do valor da carteira do agente INV6010.

- **INV6020**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 60 dias e margem de 20%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
0.00	500000.00	**500000.00**	0.000
0.00	500000.00	**500000.00**	0.000
24999.84	475000.16	**500000.00**	0.000
25154.16	475000.16	**500154.32**	0.031
24652.62	475000.16	**499652.78**	-0.069
24305.40	475000.16	**499305.56**	-0.139
24729.78	475000.16	**499729.94**	-0.054
24884.10	475000.16	**499884.26**	-0.023
24305.40	475000.16	**499305.56**	-0.139
23996.76	475000.16	**498996.92**	-0.201
48828.42	449999.96	**498828.38**	-0.235
48998.99	449999.96	**498998.95**	-0.201
49256.87	449999.96	**499256.83**	-0.149
49332.00	449999.96	**499331.96**	-0.134
49632.52	449999.96	**499632.48**	-0.074
49059.91	449999.96	**499059.87**	-0.188
48802.03	449999.96	**498801.99**	-0.240
49634.55	449999.96	**499634.51**	-0.073
74979.23	425000.48	**499979.71**	-0.004
74763.31	425000.48	**499763.79**	-0.047

Tabela 5.11 - Resultados experimentais do agente INV6020.

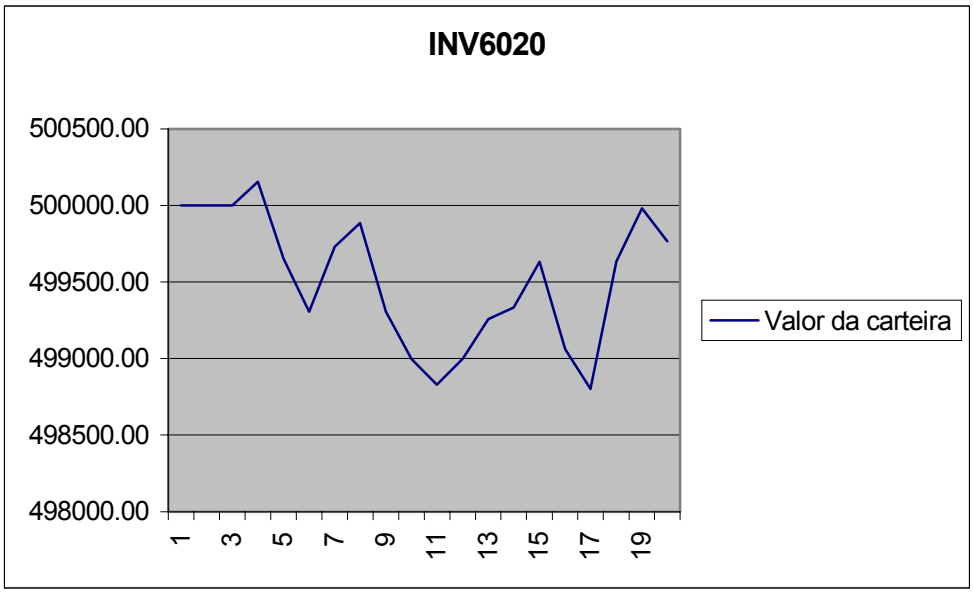


Gráfico 5.11 - Evolução do valor da carteira do agente INV6020.

- **INV6030**

Investidor baseado no indicador estocástico, com análise relativa aos últimos 60 dias e margem de 30%.

Valor das acções (€)	Capital (€)	Valor da carteira (€)	Rentabilidade (%)
1316.48	498672.64	**499989.12**	-0.002
1305.60	498672.64	**499978.24**	-0.004
1305.60	498672.64	**499978.24**	-0.004
26303.28	473674.96	**499978.24**	-0.004
25804.86	473674.96	**499479.82**	-0.104
25459.80	473674.96	**499134.76**	-0.173
51027.61	448673.62	**499701.23**	-0.060
51139.98	448673.62	**499813.60**	-0.037
50357.40	448673.62	**499031.02**	-0.194
74498.26	423671.50	**498169.76**	-0.367
75203.05	423671.50	**498874.55**	-0.226
75635.62	423671.50	**499307.12**	-0.139
75883.04	423671.50	**499554.54**	-0.089
75676.93	423671.50	**499348.43**	-0.130
77014.33	423671.50	**500685.83**	0.137
76206.68	423671.50	**499878.18**	-0.024
75812.00	423671.50	**499483.50**	-0.103
50821.66	449243.41	**500065.07**	0.013
51193.22	449243.41	**500436.63**	0.087
51178.98	449243.41	**500422.39**	0.084

Tabela 5.12 - Resultados experimentais do agente INV6030.

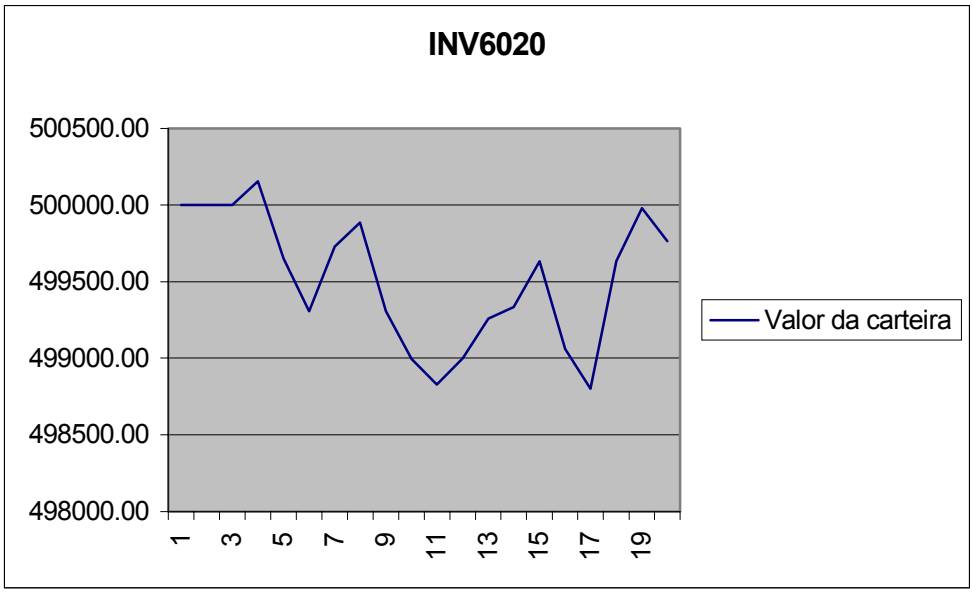


Gráfico 5.12 - Evolução do valor da carteira do agente INV6030.

5.2. Resumo das transacções

Os pontos seguintes enumeram todas as transacções realizadas pelos doze agentes investidores, em cada um dos 20 dias do período de experimentação.

- **12/06/2003**

INV1520 comprou 7418 acções do título IBERSOL.
INV1530 comprou 7440 acções do título IBERSOL.
INV1530 comprou 19084 acções do título PORTUCEL.
INV3010 comprou 7418 acções do título IBERSOL.
INV3020 comprou 7440 acções do título IBERSOL.
INV6030 comprou 1088 acções do título SAG GEST.

- **13/06/2003**

INV5 comprou 103 acções do título BRISA.
INV5 comprou 275 acções do título EDP.
INV5 comprou 79 acções do título PORTUGAL TELECOM.
INV50 comprou 1033 acções do título BRISA.
INV50 comprou 2747 acções do título EDP.
INV50 comprou 787 acções do título PORTUGAL TELECOM.
INV500 comprou 10331 acções do título BRISA.
INV500 comprou 27473 acções do título EDP.
INV500 comprou 7874 acções do título PORTUGAL TELECOM.
INV1510 comprou 3937 acções do título PORTUGAL TELECOM.
INV1520 comprou 3937 acções do título PORTUGAL TELECOM.
INV1530 vendeu 7440 acções do título IBERSOL.
INV3010 comprou 3937 acções do título PORTUGAL TELECOM.
INV3020 comprou 3937 acções do título PORTUGAL TELECOM.
INV6010 comprou 3937 acções do título PORTUGAL TELECOM.

- **16/06/2003**

INV1530 comprou 3858 acções do título PORTUGAL TELECOM.
INV3030 comprou 3858 acções do título PORTUGAL TELECOM.
INV6020 comprou 3858 acções do título PORTUGAL TELECOM.

- **17/06/2003**

INV5 vendeu 79 acções do título PORTUGAL TELECOM.
INV5 comprou 496 acções do título T. DUARTE.
INV50 comprou 20075 acções do título PARAREDE.
INV50 vendeu 787 acções do título PORTUGAL TELECOM.
INV50 comprou 4461 acções do título T. DUARTE.
INV500 vendeu 7874 acções do título PORTUGAL TELECOM.
INV500 comprou 97 acções do título T. DUARTE.
INV1510 comprou 1636 acções do título PT MULTIMEDIA.
INV1520 comprou 1636 acções do título PT MULTIMEDIA.
INV1530 comprou 3644 acções do título J. MARTINS.

INV1530 comprou 1636 acções do título PT MULTIMEDIA.
INV3030 comprou 3644 acções do título J. MARTINS.
INV6030 comprou 3834 acções do título PORTUGAL TELECOM.

- **18/06/2003**

INV5 comprou 110 acções do título BPI.
INV5 comprou 70 acções do título CIMPOR.
INV5 comprou 1606 acções do título PARAREDE.
INV5 comprou 40 acções do título PORTUGAL TELECOM.
INV50 comprou 1241 acções do título BPI.
INV50 comprou 789 acções do título CIMPOR.
INV50 comprou 454 acções do título PORTUGAL TELECOM.
INV500 comprou 13737 acções do título BPI.
INV500 comprou 8768 acções do título CIMPOR.
INV500 comprou 200707 acções do título PARAREDE.
INV500 comprou 5028 acções do título PORTUGAL TELECOM.
INV1510 comprou 1956 acções do título BES.
INV1520 comprou 1953 acções do título BES.
INV3030 comprou 1956 acções do título BES.
INV3030 comprou 19084 acções do título PORTUCEL.

- **19/06/2003**

Não houve transacções.

- **20/06/2003**

INV5 comprou 125 acções do título BCP.
INV5 comprou 95 acções do título COFINA.
INV5 comprou 364 acções do título SONAE.
INV50 comprou 1558 acções do título BCP.
INV50 comprou 20 acções do título COFINA.
INV50 comprou 4633 acções do título SONAE.
INV500 comprou 17249 acções do título BCP.
INV500 comprou 700 acções do título COFINA.
INV500 comprou 50205 acções do título SONAE.
INV1510 comprou 5187 acções do título BRISA.
INV1510 comprou 13736 acções do título BRISA.
INV1520 comprou 5187 acções do título BRISA.
INV1520 comprou 13587 acções do título BRISA.
INV1530 comprou 5187 acções do título BRISA.
INV1530 vendeu 3644 acções do título J. MARTINS.
INV3010 comprou 5187 acções do título BRISA.
INV3020 comprou 1961 acções do título BES.
INV3020 comprou 5187 acções do título BRISA.
INV3030 comprou 5187 acções do título BRISA.
INV3030 comprou 1493 acções do título IBERSOL.
INV6030 comprou 5187 acções do título BRISA.

- **23/06/2003**

INV5 comprou 74 acções do título SONAE.COM.
INV50 comprou 918 acções do título SONAE.COM.
INV500 comprou 10185 acções do título SONAE.COM.
INV1530 comprou 13298 acções do título EDP.
INV1530 comprou 5718 acções do título SAG GEST.

- **24/06/2003**

INV5 comprou 22 acções do título NOVABASE.
INV5 comprou 48 acções do título SEMAPA.
INV50 comprou 278 acções do título NOVABASE.
INV50 comprou 601 acções do título SEMAPA.
INV500 comprou 3090 acções do título NOVABASE.
INV500 comprou 661 acções do título SEMAPA.

- **25/06/2003**

INV5 comprou 94 acções do título SAG GEST.
INV50 comprou 1170 acções do título SAG GEST.
INV500 comprou 14451 acções do título SAG GEST.
INV1510 comprou 10081 acções do título BPI.
INV1510 comprou 3666 acções do título J. MARTINS.
INV1520 comprou 37115 acções do título PARAREDE.
INV1530 comprou 1969 acções do título BES.
INV3010 comprou 1969 acções do título BES.
INV3010 comprou 3666 acções do título J. MARTINS.
INV3010 comprou 1648 acções do título PT MULTIMEDIA.
INV3020 comprou 1648 acções do título PT MULTIMEDIA.
INV3030 comprou 10121 acções do título BPI.
INV3030 comprou 1648 acções do título PT MULTIMEDIA.
INV6030 comprou 3666 acções do título J. MARTINS.

- **26/06/2003**

INV1510 comprou 19231 acções do título PORTUCEL.
INV1510 vendeu 1636 acções do título PT MULTIMEDIA.
INV1520 comprou 3655 acções do título J. MARTINS.
INV1520 comprou 769 acções do título PORTUCEL.
INV1520 vendeu 1636 acções do título PT MULTIMEDIA.
INV1530 comprou 3704 acções do título J. MARTINS.
INV1530 vendeu 1636 acções do título PT MULTIMEDIA.
INV1530 comprou 12077 acções do título SONAE.COM.
INV3010 vendeu 1648 acções do título PT MULTIMEDIA.
INV3020 comprou 3704 acções do título J. MARTINS.
INV3020 vendeu 1648 acções do título PT MULTIMEDIA.
INV3030 vendeu 1648 acções do título PT MULTIMEDIA.
INV6020 comprou 3655 acções do título J. MARTINS.

- **27/06/2003**

INV5 vendeu 103 acções do título BRISA.
INV50 vendeu 1033 acções do título BRISA.
INV500 vendeu 10331 acções do título BRISA.
INV1510 comprou 100985 acções do título PARAREDE.
INV1520 comprou 10000 acções do título BPI.
INV1520 comprou 30488 acções do título T. DUARTE.
INV1530 comprou 11062 acções do título COFINA.
INV1530 comprou 3088 acções do título NOVABASE.
INV1530 comprou 30488 acções do título T. DUARTE.
INV6010 comprou 3660 acções do título J. MARTINS.

- **30/06/2003**

INV5 vendeu 496 acções do título T. DUARTE.
INV50 vendeu 4461 acções do título T. DUARTE.
INV500 vendeu 97 acções do título T. DUARTE.
INV1510 vendeu 13736 acções do título EDP.
INV1520 vendeu 13587 acções do título EDP.
INV1530 vendeu 13298 acções do título EDP.
INV3010 comprou 19084 acções do título PORTUCEL.
INV3020 vendeu 7440 acções do título IBERSOL.
INV3020 comprou 19084 acções do título PORTUCEL.

- **01/07/2003**

INV1510 vendeu 1956 acções do título BES.
INV1510 vendeu 5187 acções do título BRISA.
INV1510 vendeu 19231 acções do título PORTUCEL.
INV1510 comprou 12077 acções do título SONAE.COM.
INV1520 vendeu 1953 acções do título BES.
INV1520 vendeu 5187 acções do título BRISA.
INV1520 comprou 7530 acções do título CIMPOR.
INV1520 vendeu 769 acções do título PORTUCEL.
INV1530 vendeu 5187 acções do título BRISA.
INV1530 vendeu 3704 acções do título J. MARTINS.
INV1530 vendeu 3088 acções do título J. NOVABASE.
INV1530 comprou 8420 acções do título PARAREDE.
INV1530 vendeu 19084 acções do título PORTUCEL.
INV1530 vendeu 30488 acções do título T. DUARTE.
INV3030 vendeu 1956 acções do título BES.
INV3030 vendeu 5187 acções do título BRISA.
INV3030 vendeu 1493 acções do título IBERSOL.

- **02/07/2003**

INV1510 comprou 7508 acções do título CIMPOR.
INV1510 comprou 9059 acções do título IMPRESA.
INV1510 vendeu 100985 acções do título PARAREDE.

INV1510 comprou 21552 acções do título SAG GEST.
INV1510 comprou 5887 acções do título SEMAPA.
INV1520 comprou 16340 acções do título BCP.
INV1520 vendeu 3655 acções do título J. MARTINS.
INV1520 vendeu 37115 acções do título PARAREDE.
INV1520 vendeu 30488 acções do título T. DUARTE.
INV1530 comprou 16340 acções do título BCP.
INV1530 comprou 7508 acções do título CIMPOR.
INV1530 comprou 9059 acções do título IMPRESA.
INV1530 vendeu 8420 acções do título PARAREDE.
INV1530 comprou 5312 acções do título SEMAPA.
INV3010 comprou 21552 acções do título SAG GEST.
INV3020 comprou 10246 acções do título BPI.
INV3030 comprou 7508 acções do título CIMPOR.

- **03/07/2003**

INV5 vendeu 275 acções do título EDP.
INV50 vendeu 2747 acções do título EDP.
INV500 vendeu 27473 acções do título EDP.
INV1510 comprou 350 acções do título COFINA.
INV1510 vendeu 9059 acções do título IMPRESA.
INV1510 vendeu 3666 acções do título J. MARTINS.
INV1510 comprou 2320 acções do título NOVABASE.
INV1520 comprou 3356 acções do título NOVABASE.
INV1520 comprou 29100 acções do título PARAREDE.
INV1520 comprou 10287 acções do título SAG GEST.
INV1520 comprou 8993 acções do título SEMAPA.
INV1530 vendeu 9059 acções do título IMPRESA.
INV1530 comprou 3356 acções do título NOVABASE.
INV1530 comprou 29100 acções do título PARAREDE.
INV3010 vendeu 5187 acções do título BRISA.
INV3020 vendeu 5187 acções do título BRISA.
INV3020 comprou 21186 acções do título SAG GEST.
INV3030 comprou 20697 acções do título SAG GEST.
INV6010 comprou 20697 acções do título SAG GEST.

- **04/07/2003**

INV1510 comprou 19380 acções do título PORTUCEL.
INV1530 comprou 15220 acções do título PORTUCEL.

- **07/07/2003**

INV5 vendeu 1606 acções do título PARAREDE.
INV5 comprou 293 acções do título T. DUARTE.
INV50 vendeu 20075 acções do título PARAREDE.
INV50 comprou 3157 acções do título T. DUARTE.
INV500 vendeu 200707 acções do título PARAREDE.
INV500 comprou 21783 acções do título T. DUARTE.

INV1510 comprou 16556 acções do título BCP.
INV3030 comprou 12563 acções do título SONAE.COM.
INV6030 vendeu 5187 acções do título BRISA.

- **08/07/2003**

INV50 comprou 811 acções do título IBERSOL.
INV50 comprou 1197 acções do título IMPRESA.
INV500 comprou 11912 acções do título IMPRESA.
INV500 comprou 3987 acções do título J. MARTINS.
INV1520 comprou 5598 acções do título COFINA.
INV6020 comprou 21186 acções do título SAG GEST.

- **09/07/2003**

INV5 comprou 20 acções do título BES.
INV5 vendeu 72 acções do título CIMPOR.
INV5 comprou 1288 acções do título PARAREDE.
INV50 comprou 199 acções do título BES.
INV50 vendeu 789 acções do título CIMPOR.
INV500 comprou 1979 acções do título BES.
INV500 vendeu 8768 acções do título CIMPOR.
INV500 comprou 499 acções do título IBERSOL.
INV1520 vendeu 5598 acções do título COFINA.
INV1520 vendeu 29100 acções do título PARAREDE.
INV1530 vendeu 1969 acções do título BES.
INV3010 vendeu 3666 acções do título J. MARTINS.
INV6010 vendeu 3660 acções do título J. MARTINS.

5.3. Observações e comentários

Podem ser observados os resultados da concorrência entre os investidores através dos investimentos realizados no dia 17/06/2003. Os investidores INV5, INV50 e INV500 decidiram comprar acções do título T. DUARTE. Relembro que estes investidores têm a mesma estratégia de investimento, tendo como diferença apenas o capital investido, ou seja, espera-se sempre que estes três investidores invistam no mesmo título com as seguintes relações no que diz respeito ao capital investido (*invs*) nesse título:

invs(INV500) ≈ 10* invs(INV50) ≈ 100* invs(INV5)

ou seja, a quantidade (*quant*) terá também as seguintes relações:

quant(INV500) ≈ 10* quant(INV50) ≈ 100* quant(INV5)

No entanto no investimento (concorrente) acima referido (dia 17/06/2003), verificou-se que os três agentes lançaram pedidos de compra, sendo que os pedidos de INV5 (496 acções) e INV50 (4461 acções) foram satisfeitos; no entanto, o pedido de INV500 não pôde ser satisfeito pois não havia quantidade suficiente e apenas lhe foram atribuídas 97 acções. Os pedidos são satisfeitos pelo agente Bolsa por ordem de chegada.

O facto de haver apenas pequenas variações tanto nos lucros como nas perdas (rentabilidade) - menos de 0,5% na maioria dos casos - deve-se ao facto de os investidores terem uma carteira com dinheiro a mais, ou seja, com uma carteira de €500000 os investimentos deveriam ser muito superiores ao que foi implementado (€25000). O máximo valor que esteve investido por parte de um agente foi cerca de €150000, isto é, a maior parte do dinheiro esteve "parado", e neste caso, dinheiro "parado" é concerteza um mau investimento. Com um aumento do valor dos investimentos o valor da rentabilidade irá aumentar, visto que a rentabilidade é calculada tendo como base o valor total da carteira (valor em dinheiro mais o valor das acções). É fácil concluir que se por um lado o dinheiro "parado" não rende, e por outro lado um aumento do valor das acções leva a uma maior rentabilidade, será necessário diminuir a quantidade de dinheiro "parado" e aumentar o valor dos investimentos.

Pela análise dos gráficos observa-se que os investidores que têm como estratégia o indicador MACD tiveram maior sucesso. Quanto aos investidores que têm como estratégia o indicador estocástico, estão claramente divididos em 3 categorias, tendo como referência o número de dias da análise. Pode dizer-se que a análise mais coerente é sem dúvida a de 60 dias, visto que é muito mais ponderada e tal pode ser de facto observado, visto que os investidores que menos investimentos fizeram foram os que tinham uma análise de 60 dias. Os investidores que têm a análise de 15 dias, poderão sofrer as consequências do facto de possuir "pouca memória", sendo a sua análise muito menos ponderada. Quanto às margens de investimento, é fácil concluir que um agente que tenha uma margem de investimento de 10% será um agente mais cauteloso e calmo, e um agente que tenha uma margem de investimento de 30%, sendo portanto mais sensível às oscilações do mercado, será mais reflexivo e nervoso.

Em sublinhado encontram-se algumas palavras que poderão caracterizar os diferentes agentes.

6. Conclusões

Após ter concluído o trabalho faço um balanço positivo do mesmo e sinto ter cumprido os objectivos, no entanto é minha opinião que poderia ter sido mais desenvolvido, não fosse a falta de experiência da minha parte no que diz respeito à programação em Java. Ao longo do tempo fui adquirindo mais conhecimentos, que me permitiram progredir no trabalho, sendo que nos últimos tempos já me sentia à vontade com esta linguagem.

Apercebi-me do enorme potencial que são os sistemas multi-agente, não só pela realização deste trabalho em concreto, mas também pela consulta de vários documentos sobre este assunto, que fui encontrando ao longo das minhas pesquisas. Foi realmente bastante positivo para mim todo o estudo que fiz sobre os agentes e sistemas multi--agente e ainda sobre a plataforma JATLite.

Uma aplicação deste género é sem dúvida uma ferramenta extremamente útil para um investidor real na Bolsa, precisando no entanto de um maior desenvolvimento que vai para além de um trabalho final de curso, sendo necessário um maior estudo sobre as estratégias de investimento, factores de oscilação do mercado, etc.

7. Melhoramentos

São enormes as potencialidades de desenvolvimento e expansão deste trabalho. Refiro agora algumas das funcionalidades que penso serem bastante úteis e interessantes para o desenvolvimento deste trabalho.

Os melhoramentos mais óbvios são sem dúvida relativos à interface gráfica e às estratégias de investimento. É necessário que uma interface gráfica seja implementada nos agentes investidores, onde seja possível fazer a parametrização e onde se possa visualizar a evolução destes através de valores e gráficos. A interface do agente Bolsa também poderia ser desenvolvida, nomeadamente poderia mostrar mais informações sobre as cotações, como por exemplo as quantidades, e ainda poderiam ser visualizados gráficos sobre a evolução das cotações, sobre as médias móveis, etc. Quanto às estratégias de investimento, deveriam ser alvo de maior estudo e poderiam ser um pouco desenvolvidas, por exemplo, no indicador estocástico poderia ser utilizada uma média móvel e a partir daí obter os mínimos e os máximos. Poderiam ainda ser acrescentados outros indicadores.

Havendo uma maior diversidade de indicadores, poderá ser implementada uma interface gráfica geral para todos os investidores, e onde se poderá escolher qual ou quais os indicadores que esse investidor irá seguir e os respectivos parâmetros.

Para além de se ter estratégias de investimento feitas mediante a análise técnica, poderia ser feita uma incursão pela análise fundamental. Poderiam ser levados em conta os factores envolventes da análise fundamental: (1) estudo da economia; (2) estudo do sector de actividade em que a empresa está envolvida; (3) estudo dos rácios da empresa. Seria de facto interessante ter em conta factores económicos como por exemplo a inflação, taxas de juro, preço da gasolina, salário mínimo, etc. Os investimentos feitos pelos agente investidores teriam todos este factores em conta (*investir muito ou pouco face a esta situação económica?*; *este investimento tem futuro?*). Seria também importante fazer estudos dos vários sectores de actividades; os investidores iriam investir nos sectores mais promissores (*vou investir só neste sector porque é o que me dá mais garantias...*; *vou investir mais neste sector do que nos outros...*). Por último, a análise dos rácios das empresas seria um aspecto importante para decidir se um investimento nessa empresa será sólido (*esta empresa está a crescer, vou comprar mais acções dela...*; *esta empresa está a caminhar para a falência, vou vender todas as acções dela enquanto o prejuízo ainda é pequeno...*).

Seria interessante ter em conta também factores sociais; ter uma espécie de indicador da "moral" da sociedade. Tal indicador seria influenciado por factores tais como oscilações dos preços (não como factor económico, mas como a oscilação pode influenciar o modo de vida da sociedade), guerras, resultados desportivos, etc.

A base de dados deve ser alterada. A estrutura actual não é a mais correcta e é necessária uma reformulação das tabelas. Deve ser criada uma tabela EMPRESA (contém os dados da empresa) que está ligada a outra tabela (COTAÇÃO), que tem as informações sobre todas as cotações, tendo como campos a data, o nome da empresa e a cotação. Deverá existir ainda outra tabela que será sempre actualizada (ULTIMA_COTAÇÃO), que contém os valores actualizados das cotações, quantidades, abertura, fecho, máximos e mínimos.

Um outro aspecto importante poderia ser a implementação de uma certa "auto--aprendizagem", ou seja, os agentes investedores poderiam ter a capacidade de "aprender" com os seus erros. Por exemplo, em caso de terem efectuado um mau investimento que levou a terem prejuízo num certo título, eles seriam mais cuidadosos no futuro caso recebam um sinal de compra desse título.

Os investidores poderiam ter um comportamento diferente face aos seus investimentos, nomeadamente no valor do capital investido. Tal valor poderia ser mais variante de investidor para investidor. Poderá ser definida uma margem de investimento e o investidor iria decidir-se por um valor dentro dessa margem. Nessa decisão poderia levar em conta, por exemplo, quantos títulos tem já em carteira (se tiver já muitos títulos poderá ser mais cauteloso e investir o capital mínimo, ou então se for um agente de características mais arrojadas, irá investir mais...). Também na venda os investidores poderiam ter outro comportamento, ou seja, se receberem um sinal de venda de um título, poderão decidir não vender toda a quantidade que têm desse título se acharem que poderão lucrar com isso (se acharem que o valor desse título irá subir, poderão ter maior lucro se venderem mais tarde).

Apêndice 1 - Descrição das classes

Classe Investidor

```
java.lang.Object
  |
  +--java.lang.Thread
        |
        +--Abstract.AgentAction
              |
              +--BaseLayer.BAgentAction
                    |
                    +--KQMLLayer.KQMLAgentAction
                          |
                          +--
RouterLayer.AgentClient.RouterClientAction
                                |
                                +--Investidor
```

Todas as Interfaces Implementadas:
`java.lang.Runnable`

---

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

Classe que implementa o Agente Investidor baseado no Indicador MACD.

Campos

Campos herdados da classe RouterLayer.AgentClient.RouterClientAction
_deleteFPT, _deleteFPTSize, _mailQueue, _maxDeleteSize, _messageNumber, _myAddress, _registrarName, _routerName

Campos herdados da classe Abstract.AgentAction
_addresses, _connections, _durationTime, _endWith, _queue, _security

Campos herdados da classe java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Construtores

Investidor(int capIni)

Métodos

boolean	**Act**(java.lang.Object o)
protected void	**escutar**(KQMLLayer.KQMLmessage kqml)
void	**processMessage**(java.lang.String command, java.lang.Object obj)
void	**run**()
protected void	**sendErrorMessage**(KQMLLayer.KQMLmessage kqml)
protected void	**sendMSG**(java.lang.String receiver, java.lang.String perform, java.lang.String MSG)
void	**start**()
boolean	**terminus**()

Métodos herdados da classe RouterLayer.AgentClient.RouterClientAction

addToDeleteBuffer, addToDeleteBuffer, connect, connect, connect, ConvertStringToAddress, disconnect, endAction, getMailQueue, getMaxDeleteSize, getMyAddress, getRouterName, initDataMembers, listUsers, register, register, requestAddress, requestAddress, reserveMessage, reserveMessage, sendDeleteMessage, sendKQMLMessage, sendMessage, sendMessage, sendMessage, setMyAddress, setMyAddress, setRegistrarAddress, setRegistrarAddress, setRouterAddress, setRouterAddress, unregister, unregister

Métodos herdados da classe KQMLLayer.KQMLAgentAction

createReceiverThread, createReceiverThread, createServerThread, sendKQMLMessag

Métodos herdados da classe BaseLayer.BAgentAction

BroadCast

Métodos herdados da classe Abstract.AgentAction

getAddressTable, getConnectionTable, getDurationTime, getEndWith, getMessageQueue, getSecurity, setAddressTable, setConnectionTable, setDurationTime, setEndWith, setMessageQueue, setSecurity

Métodos herdados da classe java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, stop, stop, suspend, toString, yield

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Detalhes dos Construtores

Investidor

```
public Investidor(int capIni)
           throws java.lang.Exception
```

Detalhes dos Métodos

start

```
public void start()
```

Sobrepõe-se a:

`start` na classe `java.lang.Thread`

---

run

```
public void run()
```

Especificado por:

`run` na interface `java.lang.Runnable`

Sobrepõe-se a:

`run` na classe `BaseLayer.BAgentAction`

---

sendMSG

```
protected void sendMSG(java.lang.String receiver,
                       java.lang.String perform,
                       java.lang.String MSG)
                throws java.lang.Exception
```

`java.lang.Exception`

Act

```
public boolean Act(java.lang.Object o)
```

Sobrepõe-se a:

`Act` na classe `RouterLayer.AgentClient.RouterClientAction`

---

processMessage

```
public void processMessage(java.lang.String command,
                           java.lang.Object obj)
```

Especificado por:

`processMessage` na classe `RouterLayer.AgentClient.RouterClientAction`

---

sendErrorMessage

```
protected void sendErrorMessage(KQMLLayer.KQMLmessage kqml)
                         throws java.lang.Exception
```

`java.lang.Exception`

---

escutar

```
protected void escutar(KQMLLayer.KQMLmessage kqml)
```

---

terminus

```
public boolean terminus()
```

Classe InvestMACD

```
java.lang.Object
  |
  +--InvestMACD
```

---

public class **InvestMACD**
extends java.lang.Object

Classe para parametrização do Agente Investidor baseado no Indicador MACD.

Construtores

InvestMACD()

Métodos

static void	**main**(java.lang.String[] argv)

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Construtores

InvestMACD

```
public InvestMACD()
```

Detalhes dos Métodos

main

```
public static void main(java.lang.String[] argv)
                  throws java.lang.Exception

      java.lang.Exception
```

Classe InvestStk

```
java.lang.Object
  |
  +--InvestStk
```

---

public class **InvestStk**
extends java.lang.Object

Classe para parametrização do Agente Investidor baseado no Indicador Estocástico.

Construtores

InvestStk()

Métodos

static void	**main**(java.lang.String[] argv)

Métodos herdados da classe java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Construtores

InvestStk

```
public InvestStk()
```

Detalhes dos Métodos

main

```
public static void main(java.lang.String[] argv)
                  throws java.lang.Exception
```

java.lang.Exception

Classe InvStk

```
java.lang.Object
  |
  +--java.lang.Thread
        |
        +--Abstract.AgentAction
              |
              +--BaseLayer.BAgentAction
                    |
                    +--KQMLLayer.KQMLAgentAction
                          |
                          +--
RouterLayer.AgentClient.RouterClientAction
                                |
                                +--InvStk
```

Todas as Interfaces Implementadas:
java.lang.Runnable

---

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

Classe que implementa o Agente Investidor baseado no Indicador Estocástico.

Campos

Campos herdados da classe RouterLayer.AgentClient.RouterClientAction
_deleteFPT, _deleteFPTSize, _mailQueue, _maxDeleteSize, _messageNumber, _myAddress, _registrarName, _routerName

Campos herdados da classe Abstract.AgentAction
_addresses, _connections, _durationTime, _endWith, _queue, _security

Campos herdados da classe java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Construtores

InvStk(int nDias, int margem)

Métodos

boolean	**Act**(java.lang.Object o)
protected void	**escutar**(KQMLLayer.KQMLmessage kqml)
void	**processMessage**(java.lang.String command, java.lang.Object obj)
void	**run**()
protected void	**sendErrorMessage**(KQMLLayer.KQMLmessage kqml)
protected void	**sendMSG**(java.lang.String receiver, java.lang.String perform, java.lang.String MSG)
void	**start**()
boolean	**terminus**()

Métodos herdados da classe RouterLayer.AgentClient.RouterClientAction
addToDeleteBuffer, addToDeleteBuffer, connect, connect, connect,

ConvertStringToAddress, disconnect, endAction, getMailQueue, getMaxDeleteSize, getMyAddress, getRouterName, initDataMembers, listUsers, register, register, requestAddress, requestAddress, reserveMessage, reserveMessage, sendDeleteMessage, sendKQMLMessage, sendMessage, sendMessage, sendMessage, setMyAddress, setMyAddress, setRegistrarAddress, setRegistrarAddress, setRouterAddress, setRouterAddress, unregister, unregister

Métodos herdados da classe KQMLLayer.KQMLAgentAction
createReceiverThread, createReceiverThread, createServerThread, sendKQMLMessag

Métodos herdados da classe BaseLayer.BAgentAction
BroadCast

Métodos herdados da classe Abstract.AgentAction
getAddressTable, getConnectionTable, getDurationTime, getEndWith, getMessageQueue, getSecurity, setAddressTable, setConnectionTable, setDurationTime, setEndWith, setMessageQueue, setSecurity

Métodos herdados da classe java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, stop, stop, suspend, toString, yield

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Detalhes dos Construtores

InvStk

```
public InvStk(int nDias,
              int margem)
       throws java.lang.Exception
```

Detalhes dos Métodos

start

```
public void start()
```

Sobrepõe-se a:

`start` na classe `java.lang.Thread`

run

```
public void run()
```

Especificado por:

`run` na interface `java.lang.Runnable`

Sobrepõe-se a:

`run` na classe `BaseLayer.BAgentAction`

---

sendMSG

```
protected void sendMSG(java.lang.String receiver,
                       java.lang.String perform,
                       java.lang.String MSG)
                throws java.lang.Exception
```

`java.lang.Exception`

Act

```
public boolean Act(java.lang.Object o)
```

Sobrepõe-se a:

`Act` na classe `RouterLayer.AgentClient.RouterClientAction`

---

processMessage

```
public void processMessage(java.lang.String command,
                           java.lang.Object obj)
```

Especificado por:

`processMessage` na classe `RouterLayer.AgentClient.RouterClientAction`

---

sendErrorMessage

```
protected void sendErrorMessage(KQMLLayer.KQMLmessage kqml)
                         throws java.lang.Exception
```

`java.lang.Exception`

---

escutar

```
protected void escutar(KQMLLayer.KQMLmessage kqml)
```

---

terminus

```
public boolean terminus()
```

Classe Navegador

```
java.lang.Object
  |
  +--Navegador
```

---

public class **Navegador**
extends java.lang.Object

Sistema automático de recolha de páginas HTML a partir da Web. Módulo principal.

Construtores

Navegador()

Métodos

static void	**main**(java.lang.String[] args)

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Construtores

Navegador

```
public Navegador()
```

Detalhes dos Métodos

main

```
public static void main(java.lang.String[] args)
```

Classe Wbolsa

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Frame
                          |
                          +--Wbolsa
```

Todas as Interfaces Implementadas:

javax.accessibility.Accessible, java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.awt.event.WindowListener

---

public class **Wbolsa**
extends java.awt.Frame
implements java.awt.event.WindowListener

Classe que implementa o Agente Bolsa.

Sub-Classes

Sub-Classes herdadas da classe java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Sub-Classes herdadas da classe java.awt.Window
java.awt.Window.AccessibleAWTWindow

Sub-Classes herdadas da classe java.awt.Container
java.awt.Container.AccessibleAWTContainer

Sub-Classes herdadas da classe java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Campos

Campos herdados da classe java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Campos herdados da classe java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Campos herdados da interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Construtores

Wbolsa()

Métodos

void	**jogo**(java.lang.String conffile)
static void	**main**(java.lang.String[] argv)
void	**windowActivated**(java.awt.event.WindowEvent event)
void	**windowClosed**(java.awt.event.WindowEvent event)
void	**windowClosing**(java.awt.event.WindowEvent event)
void	**windowDeactivated**(java.awt.event.WindowEvent event)
void	**windowDeiconified**(java.awt.event.WindowEvent event)
void	**windowIconified**(java.awt.event.WindowEvent event)
void	**windowOpened**(java.awt.event.WindowEvent event)

Métodos herdados da classe java.awt.Frame
addNotify, finalize, getAccessibleContext, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, paramString, remove, removeNotify, setCursor, setExtendedState,

setIconImage, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Métodos herdados da classe java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getMostRecentFocusOwner, getOwnedWindows, getOwner, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindowStateListeners, hide, isActive, isFocusableWindow, isFocusCycleRoot, isFocused, isShowing, pack, postEvent, processEvent, processWindowEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, setCursor, setFocusableWindowState, setFocusCycleRoot, setLocationRelativeTo, show, toBack, toFront

Métodos herdados da classe java.awt.Container
add, add, add, add, add, addContainerListener, addImpl, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, remove, removeAll, removeContainerListener, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, setLayout, transferFocusBackward, transferFocusDownCycle, update, validate, validateTree

Métodos herdados da classe java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage,

prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus, transferFocusUpCycle

Métodos herdados da classe java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Métodos herdados da interface java.awt.MenuContainer
getFont, postEvent

Detalhes dos Construtores

Wbolsa

```
public Wbolsa()
```

Detalhes dos Métodos

jogo

```
public void jogo(java.lang.String conffile)
```

---

main

```
public static void main(java.lang.String[] argv)
```

---

windowClosed

```
public void windowClosed(java.awt.event.WindowEvent event)
```

Especificado por:
windowClosed na interface java.awt.event.WindowListener

---

windowOpened

```
public void windowOpened(java.awt.event.WindowEvent event)
```

Especificado por:
`windowOpened` na interface `java.awt.event.WindowListener`

---

windowDeiconified

```
public void windowDeiconified(java.awt.event.WindowEvent event)
```

Especificado por:
`windowDeiconified` na interface `java.awt.event.WindowListener`

---

windowIconified

```
public void windowIconified(java.awt.event.WindowEvent event)
```

Especificado por:
`windowIconified` na interface `java.awt.event.WindowListener`

---

windowActivated

```
public void windowActivated(java.awt.event.WindowEvent event)
```

Especificado por:
`windowActivated` na interface `java.awt.event.WindowListener`

---

windowDeactivated

```
public void windowDeactivated(java.awt.event.WindowEvent event)
```

Especificado por:
`windowDeactivated` na interface `java.awt.event.WindowListener`

---

windowClosing

```
public void windowClosing(java.awt.event.WindowEvent event)
```

Especificado por:
`windowClosing` na interface `java.awt.event.WindowListener`

Package accoes

Sumário das Classes

TAccao	Esta classe define uma "Acção da Bolsa", suas componentes e operações.
TAccoes	Esta Classe define um vector de accões e algumas operações que lhe dizem respeito.

Classe TAccao

```
java.lang.Object
  |
  +--accoes.Taccao
```

---

public class **TAccao**
extends java.lang.Object

Campos

java.lang.String	**desig**
long	**quant**
double	**valor**

Construtores

TAccao()
TAccao(java.lang.String S)
TAccao(TAccao ta)

Métodos

void	**set**(java.lang.String S)
void	**set**(TAccao ta)
void	**setValor**(double novoValor)

Métodos herdados da classe java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Campos

desig

public java.lang.String **desig**

---

quant

```
public long quant
```

---

valor

```
public double valor
```

Detalhes dos Construtores

TAccao

```
public TAccao()
```

TAccao

```
public TAccao(TAccao ta)
```

TAccao

```
public TAccao(java.lang.String S)
```

Detalhes dos Métodos

setValor

```
public void setValor(double novoValor)
```

---

set

```
public void set(TAccao ta)
```

---

set

```
public void set(java.lang.String S)
```

Classe TAccoes

```
java.lang.Object
  |
  +--java.util.AbstractCollection
        |
        +--java.util.AbstractList
              |
              +--java.util.Vector
                    |
                    +--accoes.TAccoes
```

Todas as Interfaces Implementadas:

java.lang.Cloneable, java.util.Collection, java.util.List, java.util.RandomAccess, java.io.Serializable

---

public class **TAccoes**
extends java.util.Vector

Campos

Campos herdados da classe java.util.Vector
capacityIncrement, elementCount, elementData

Campos herdados da classe java.util.AbstractList
modCount

Construtores

TAccoes()

Métodos

void	**define**(java.lang.String S)
int	**getDias**(java.lang.String S)
double	**getValor**(java.lang.String titulo)
long	**maxCotacao**()
void	**print**()
double[][]	**split**(java.lang.String S)

float[][]	**splitHist**(java.lang.String Str, int n_dias)
java.lang.String[]	**splitIndic**(java.lang.String S)

Métodos herdados da classe java.util.Vector
add, add, addAll, addAll, addElement, capacity, clear, clone, contains, containsAll, copyInto, elementAt, elements, ensureCapacity, equals, firstElement, get, hashCode, indexOf, indexOf, insertElementAt, isEmpty, lastElement, lastIndexOf, lastIndexOf, remove, remove, removeAll, removeAllElements, removeElement, removeElementAt, removeRange, retainAll, set, setElementAt, setSize, size, subList, toArray, toArray, toString, trimToSize

Métodos herdados da classe java.util.AbstractList
iterator, listIterator, listIterator

Métodos herdados da classe java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Métodos herdados da interface java.util.List
iterator, listIterator, listIterator

Detalhes dos Construtores

TAccoes

public **TAccoes**()

Detalhes dos Métodos

define

public void **define**(java.lang.String S)

---

getDias

public int **getDias**(java.lang.String S)

---

split

public double[][] **split**(java.lang.String S)

---

splitHist

```
public float[][] splitHist(java.lang.String Str,
                           int n_dias)
```

---

splitIndic

```
public java.lang.String[] splitIndic(java.lang.String S)
```

---

maxCotacao

```
public long maxCotacao()
```

---

getValor

```
public double getValor(java.lang.String titulo)
```

---

print

```
public void print()
```

Package extraccao

Sumário das Classes	
filterAndUpdate	Classe com os filtros implementados como métodos de classe.
GrabPage	Adaptação do ficheiro GrabPage do livro: Java Network Programming, Second Edition Merlin Hughes, Michael Shoffner, Derek Hamner Manning Publications Company; ISBN 188477749X
Texto	Guarda o texto como um array de linhas (String).

Classe filterAndUpdate

```
java.lang.Object
  |
  +--extraccao.filterAndUpdate
```

---

public class **filterAndUpdate**
extends java.lang.Object

Construtores

filterAndUpdate(Texto texto)

Métodos

void	**connectDB**()
void	**constroiResumo**()
void	**constroiTabCotacoes**()
void	**disconnectDB**()
void	**mostraInfo**()
void	**updateDB**()
void	**updateResumo**(java.lang.String queryResumo)

Métodos herdados da classe java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Construtores

filterAndUpdate

public **filterAndUpdate**(Texto texto)

Detalhes dos Métodos

constroiTabCotacoes

public void **constroiTabCotacoes**()

---

connectDB

```
public void connectDB()
```

---

disconnectDB

```
public void disconnectDB()
```

---

constroiResumo

```
public void constroiResumo()
```

---

updateDB

```
public void updateDB()
```

---

mostraInfo

```
public void mostraInfo()
```

---

updateResumo

```
public void updateResumo(java.lang.String queryResumo)
```

Classe GrabPage

```
java.lang.Object
  |
  +--extraccao.GrabPage
```

---

public class **GrabPage**
extends java.lang.Object

Campos	
protected java.lang.String	**file**

protected java.lang.String	**host**
protected int	**port**
protected java.io.BufferedReader	**reader**
protected java.io.Writer	**writer**

Construtores

GrabPage(java.lang.String textURL)

Métodos

protected void	**connect**()
protected void	**disconnect**()
protected void	**dissect**(java.lang.String textURL)
protected Texto	**fetch**()
Texto	**grab**()

Métodos herdados da classe java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Campos

writer

protected java.io.Writer **writer**

---

reader

protected java.io.BufferedReader **reader**

---

host

protected java.lang.String **host**

---

file

```
protected java.lang.String file
```

---

port

```
protected int port
```

Detalhes dos Construtores

GrabPage

```
public GrabPage(java.lang.String textURL)
         throws java.io.IOException
```

Detalhes dos Métodos

dissect

```
protected void dissect(java.lang.String textURL)
                throws java.net.MalformedURLException

      java.net.MalformedURLException
```

---

grab

```
public Texto grab()
           throws java.io.IOException

      java.io.IOException
```

---

connect

```
protected void connect()
                throws java.io.IOException

      java.io.IOException
```

---

fetch

```
protected Texto fetch()
               throws java.io.IOException

      java.io.IOException
```

---

disconnect

```
protected void disconnect()
                  throws java.io.IOException

      java.io.IOException
```

Classe Texto

```
java.lang.Object
  |
  +--extraccao.Texto
```

---

public class **Texto**
extends java.lang.Object

Construtores

Texto()

Métodos

void	**acrescentaLinha**(java.lang.String linha)

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Construtores

Texto

```
public Texto()
```

Detalhes dos Métodos

acrescentaLinha

```
public void acrescentaLinha(java.lang.String linha)
```

Package GUITools

Sumário das Classes

PanelCotacoes	Este "Panel" corresponde à segunda metade da janela da bolsa (Wbolsa.java).
PanelMSG	Painel contendo uma caixa de mensagens enviadas e outra de mensagens recebidas.

Classe PanelCotacoes

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--GUITools.PanelCotacoes
```

Todas as Interfaces Implementadas:
javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable

---

public class **PanelCotacoes**
extends java.awt.Panel

Sumário das Sub-Classes

Sub-Classes herdadas da classe java.awt.Panel
`java.awt.Panel.AccessibleAWTPanel`

Sub-Classes herdadas da classe java.awt.Container
`java.awt.Container.AccessibleAWTContainer`

Sub-Classes herdadas da classe java.awt.Component

`java.awt.Component.AccessibleAWTComponent,`
`java.awt.Component.BltBufferStrategy,`
`java.awt.Component.FlipBufferStrategy`

Campos

Campos herdados da classe java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Campos herdados da interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Construtores

PanelCotacoes(int dx, int dy, int max_seq)

Métodos

void	**meteStr**(java.lang.String S)
void	**setCabecalho**(java.lang.String S)

Métodos herdados da classe java.awt.Panel

addNotify, getAccessibleContext

Métodos herdados da classe java.awt.Container

add, add, add, add, add, addContainerListener, addImpl, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getInsets, getLayout, getListeners, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, paramString, preferredSize, print, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, removeNotify, setFocusCycleRoot, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, setLayout, transferFocusBackward, transferFocusDownCycle, update, validate, validateTree

Métodos herdados da classe java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage,

checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getGraphicsConfiguration, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getToolkit, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, show, size, toString, transferFocus, transferFocusUpCycle

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Detalhes dos Construtores

PanelCotacoes

```
public PanelCotacoes(int dx,
                     int dy,
                     int max_seq)
```

Detalhes dos Métodos

setCabecalho

```
public void setCabecalho(java.lang.String S)
```

meteStr

```
public void meteStr(java.lang.String S)
```

Classe PanelMSG

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--GUITools.PanelMSG
```

Todas as Interfaces Implementadas:
javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable

---

public class **PanelMSG**
extends java.awt.Panel

Sumário das Sub-Classes

Sub-Classes herdadas da classe java.awt.Panel
java.awt.Panel.AccessibleAWTPanel

Sub-Classes herdadas da classe java.awt.Container
java.awt.Container.AccessibleAWTContainer

Sub-Classes herdadas da classe java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Campos

Campos herdados da classe java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Campos herdados da interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Construtores

PanelMSG(int max_seq)

Métodos

void	**meteStrEnv**(java.lang.String S)
void	**meteStrRec**(java.lang.String S)

Métodos herdados da classe java.awt.Panel
addNotify, getAccessibleContext

Métodos herdados da classe java.awt.Container
add, add, add, add, add, addContainerListener, addImpl, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getInsets, getLayout, getListeners, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, paramString, preferredSize, print, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, removeNotify, setFocusCycleRoot, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, setLayout, transferFocusBackward, transferFocusDownCycle, update, validate, validateTree

Métodos herdados da classe java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getGraphicsConfiguration, getHeight, getHierarchyBoundsListeners, getHierarchyListeners,

getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getToolkit, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, show, size, toString, transferFocus, transferFocusUpCycle

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Detalhes dos Construtores

PanelMSG

```
public PanelMSG(int max_seq)
```

Detalhes dos Métodos

meteStrEnv

```
public void meteStrEnv(java.lang.String S)
```

---

meteStrRec

```
public void meteStrRec(java.lang.String S)
```

Package Sequencia

Sumário das Classes	
SeqAccao	Esta Classe dá origem às acções e contém operações sobre elas, tais como a obtenção da designação ou valor de uma certa acção.

Classe SeqAccao

```
java.lang.Object
  |
  +--java.lang.Thread
        |
        +--Sequencia.SeqAccao
```

Todas as Interfaces Implementadas:
`java.lang.Runnable`

---

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

Campos	
java.lang.String	**designacao**
java.lang.String	**qtd**
long	**quantidade**
long	**t0**
boolean	**terminus**
java.lang.String	**titulo**
java.lang.String	**val**
double	**valor**

Campos herdados da classe java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Construtores

SeqAccao(java.lang.String desig, long quant, double valor)

Métodos

java.lang.String	**getDesig**()
long	**getQuant**()
long	**getTime**()
java.lang.String	**getTitulo**()
double	**getValor**()
void	**run**()
void	**setQuant**(long quant)
void	**terminar**()

Métodos herdados da classe java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

Métodos herdados da classe java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Detalhes dos Campos

t0

public long **t0**

---

valor

public double **valor**

---

designacao

```
public java.lang.String designacao
```

---

quantidade

```
public long quantidade
```

---

terminus

```
public boolean terminus
```

---

val

```
public java.lang.String val
```

---

qtd

```
public java.lang.String qtd
```

---

titulo

```
public java.lang.String titulo
```

Detalhes dos Construtores

SeqAccao

```
public SeqAccao(java.lang.String desig,
                long quant,
                double valor)
```

Detalhes dos Métodos

run

```
public void run()
```

Especificado por:
`run` na interface `java.lang.Runnable`
Sobrepõe-se a:
`run` na classe `java.lang.Thread`

---

terminar

```
public void terminar()
```

---

getDesig

```
public java.lang.String getDesig()
```

---

getTitulo

```
public java.lang.String getTitulo()
```

---

getValor

```
public double getValor()
```

---

getTime

```
public long getTime()
```

---

getQuant

```
public long getQuant()
```

---

setQuant

```
public void setQuant(long quant)
```

Package tools

Sumário das Classes

Canivete	A incorporar algumas ferramentas genéricas.
TRouterID	Esta classe faz a gestão das principais referências de ligação a um "Router", por parte de um agente.

Classe Canivete

```
java.lang.Object
  |
  +--tools.Canivete
```

---

public class **Canivete**
extends java.lang.Object

Construtores

Canivete()

Métodos

java.lang.String	**strNcat**(java.lang.String s1, java.lang.String s2, int N)
java.lang.String	**strSpc**(int N)
static double	**trunc**(double x, int k)

Métodos herdados da classe java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Construtores

Canivete

```
public Canivete()
```

Detalhes dos Métodos

trunc

```
public static double trunc(double x,
                           int k)
```

---

strSpc

```
public java.lang.String strSpc(int N)
```

---

strNcat

```
public java.lang.String strNcat(java.lang.String s1,
                                 java.lang.String s2,
                                 int N)
```

Classe TRouterID

```
java.lang.Object
  |
  +--tools.TrouterID
```

---

public class **TRouterID**
extends java.lang.Object

Campos

java.lang.String	**host**
java.lang.String	**port**
java.lang.String	**router**
java.lang.String	**rport**
java.lang.String	**rrouter**

Construtores

TRouterID()
TRouterID(java.lang.String conffile)

Métodos

boolean	**seekConfFile**(java.lang.String file_name)

Métodos herdados da classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Detalhes dos Campos

router

```
public java.lang.String router
```

---

host

```
public java.lang.String host
```

---

port

```
public java.lang.String port
```

---

rrouter

```
public java.lang.String rrouter
```

---

rport

```
public java.lang.String rport
```

Detalhes dos Construtores

TRouterID

```
public TRouterID()
```

TRouterID

```
public TRouterID(java.lang.String conffile)
```

Detalhes dos Métodos

seekConfFile

```
public boolean seekConfFile(java.lang.String file_name)
```

Hierarquia das Classes

- class java.lang.Object
  - class java.util.AbstractCollection (implements java.util.Collection)
    - class java.util.AbstractList (implements java.util.List)
      - class java.util.Vector (implements java.lang.Cloneable, java.util.List, java.util.RandomAccess, java.io.Serializable)
        - class accoes.**TAccoes**
  - class tools.**Canivete**
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Panel (implements javax.accessibility.Accessible)
        - class GUITools.**PanelCotacoes**
        - class GUITools.**PanelMSG**
      - class java.awt.Window (implements javax.accessibility.Accessible)
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class **Wbolsa** (implements java.awt.event.WindowListener)
  - class extraccao.**filterAndUpdate**
  - class extraccao.**GrabPage**
  - class **InvestMACD**
  - class **InvestStk**
  - class **Navegador**
  - class accoes.**TAccao**
  - class extraccao.**Texto**
  - class java.lang.Thread (implements java.lang.Runnable)
    - class Abstract.AgentAction
      - class BaseLayer.BAgentAction
        - class KQMLLayer.KQMLAgentAction
          - class RouterLayer.AgentClient.RouterClientAction
            - class **Investidor**
            - class **InvStk**
    - class Sequencia.**SeqAccao**
  - class tools.**TrouterID**

Apêndice 2 - Protocolos e Performativas do AMR

Protocolo de Ligação TCP:

Certificação da ligação: O nível mais baixo da ligação usa uma certificação simples da ligação do router. Aceitando um pedido de ligação por *socket* do agente, o router emitirá esta mensagem ao agente.
Exemplo:
```
Router : 201 AMR Router
```
ou
```
Router : 500 Internal error
```
em caso de erro.

Performativas de Ligação:

RECONNECT-AGENT *:* O agente registado pode usar a performativa *'reconnect--agent'* para efectuar uma nova ligação ao AMR.

sender : Agente
receiver : Router
password : xxxxxxxx
host : host.dominio.com
port : 1234

Os campos *host* e *port* são opcionais. Se o agente emitir o host/port, o router verifica se aqueles são diferentes dos mais recentes. Se forem diferentes, o router atualizará o host/port. Isto supõe que o endereço do agente pode ser mudado em qualquer altura. Se o sistema não quiser verificar a senha, o campo da password pode ser omitido.
Exemplo:
```
(reconnect-agent :sender agente :receiver Router :password
xxxxxxxx :host xpto.abc.com :port 4321)
```

DISCONNECT *:* Termina a ligação.

sender : Nome do agente
receiver : Router
Exemplo:
```
(disconnect :sender Agente1 :receiver Router)
```

REGISTER *:* Regista-se no Router.
Este é um uso especial da especificação actual da linguagem KQML para esta performativa quando emitida a um AMR.

sender : Nome do agente
receiver : Router
password : xxxxxxxx

Exemplo:
```
(Register :sender Agente1 :receiver Router :password
xxxxxxxx)
```

IDENTIFY *:* O Router vai perguntar a identidade do agente que pretende registar-se.

sender : Router
receiver : Agente

Exemplo:
```
(identify :sender Router :receiver agente)
```

WHOIAM *:* O agente irá apresentar-se como resposta a um pedido de IDENTIFY.

sender : Nome do agente
receiver : Router
message-method : 'Método da mensagem', o qual é compreensível para o Router
KQML-extensions : Etiquetas para performativas que não são *standard* do KQML
Contact-information : A informação do contacto do agente. Inclui o nome do *host* e endereço de e-mail (opcional)
Description : Descrição do agente, que pode ser um anúncio (*advertise*)

Exemplo:
```
(whoiam :sender Agente1 :receiver Router :message-method
MessageRouter :content (contact-information :host
xpto.abc.com :port 1234 :email agente01@xpto.abc.com)
:KQML-extensions http://java.sun.com/newProtocol
:description (advertise :type DesignAgent))
```

UNREGISTER *:* Anula o seu registo no Router

sender : Nome do agente
receiver : Router
password : xxxxxxxx

Exemplo:
```
(unregister :sender Agente1 :receiver Router :password
xxxxxxxx)
```

LIST-AGENT *:* Pedido ao Router da lista dos utilizadores que estão actualmente *on-line*.

sender : Nome do agente
receiver : Router

Exemplo:
```
(list-users :sender Agente1 :receiver Router)
```

REGISTERED-AGENT *:* Resposta do Router ao pedido LIST-AGENT

sender : Router
receiver : Nome do agente
content : Lista dos agentes registados

Exemplo:
```
(users-agent :sender Router :receiver Agente1 :content
((Agente2 xxx.yyy.com connected) (Agente3 aaa.bbb.com
disconnected)))
```

REQUEST-ADDRESS *:* Pedido ao Router o endereço de um agente

sender : Nome do agente
receiver : Router
content : Nome do outro agente

Exemplo:
```
(request-address :sender Agente1 :receiver Router :content
Agente2)
```

AGENT-ADDRESS *:* Resposta do Router ao pedido REQUEST-ADDRESS

sender : Router
receiver : Nome do agente que fez o pedido
name : Nome do agente pretendido
host : Host do agente pretendido (*null*, se a informação não estiver disponível)
port : Nº da porta do agente pretendido (-1, se a informação não estiver disponível)
description : Descrição do agente (opcional)

Exemplo:
```
(address :sender Router :receiver Agente1 :name Agente2
:host xxx.yyy.com :port 1234 :description (agent-info
:service xxxxxx))
```

RESERVE-MESSAGE *:* Pedido ao Router de reserva de mensagem.

sender : Nome do agente
receiver : Router
time : Tempo de reserva (número longo - *long* - , compreensível pelo java)

content : Lista dos números das mensagens
host : Nome do host se a localização da recepção da resposta for diferente da actual (opcional)
port : Nº da porta se a localização da recepção da resposta for diferente da actual (opcional)

Exemplo:

```
(reserve-message :sender Agente1 :receiver Router :content
(1 3 5) :time 819374635243 :host abcd.abc.com :port 1234)
```

DELETE-MESSAGE *:* Pedido ao Router para apagar uma mensagem.

sender : Nome do agente
receiver : Router
content : Nº da mensagem

Exemplo:

```
(delete-message :sender Agente1 :receiver Router :content
3)
```

Apêndice 3 - Registos do trabalho

17/02/2003

- Elaboração da página web do projecto
- Pesquisa: Cotações on-line
- Estudo dos trabalhos de IWeb sobre extracção das cotações
- Pesquisa: JATLite
- Pesquisa: Ajuda para Java
- Requisição dos livros

18/02/2003

- Estudo da tecnologia de agentes
- Documento tirado do site: http://orca.st.usm.edu/~hwalia/stock.html
- Pesquisa: encontrados alguns links sobre agentes

19/02/2003

- Estudo da tecnologia de agentes
- Livro: "Agentes de Software na Internet"
  AGENTE (Pág. 70)

  Noção Fraca:
  - Autonomia
  - Sociabilidade
  - Reactividade (Percepção, Acção, Comunicação)
  - Pró-actividade
  - Persistência

  Noção Forte:
  - Mobilidade
  - Intencionalidade
  - Aprendizagem
  - Veracidade
  - Racionalidade
  - Benevolência
  - Características mentais (Conhrcimentos, Crenças, Intenções, Desejos)

- Consulta dos acetatos de IWeb (5º ano da Licenciatura em Engenharia Electrotécnica e de Computadores da FEUP)

20/02/2003

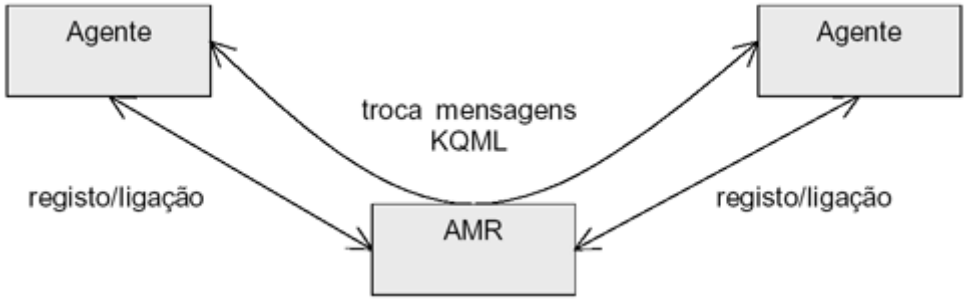
- Estudo da tecnologia de agentes
- Consulta dos acetatos de IWeb
- Consulta do livro "Agentes de Software na Internet"
- Consulta do livro "An Introduction to MultiAgent Systems"
- SISTEMAS MULTI-AGENTE
  Sistemas baseados na interacção de múltiplos agentes com um objectivo definido.
- Pesquisa: Sistemas Multi-Agente

24/02/2003

- Estudo da linguagem Java
- Livro "Java in a Nutshell"
- Execução de programas

26/02/2003

- Estudo do JATLite (Java Agent Template Lite)
- http://www-cdr.stanford.edu/ProcessLink/papers/JATL.html
- Pesquisa: JATLite
- http://www.str.isla.pt/cadeiras/pi/martinho/tese.pdf
- Agent Message Router (AMR) do JATLite na comunicação entre agentes:



- Camadas do JATLite (Arquitectura):
  - Abstract Layer
  - Base Layer
  - KQML Layer
  - Router Layer
  - Protocol Layer
- Linguagem KQML (Knowledge Query and Manipulation Language)
- ANS = Agent Name Server
- ANS vs AMR
  (http://www-cdr.stanford.edu/ProcessLink/papers/JATL.html, Cap. 3)
- Consulta:
  http://www.dsv.su.se/~lisa/Teaching/Int4/webb/api/API_users_guide.html
- Acetatos de IWeb sobre JATLite

27/02/2003

- Estudo da plataforma JATLite
- http://www.fe.up.pt/~eol/SMA/20002001/JCordeiro_Bolsa/relatorio.htm
- EXPERIMENTAÇÃO DO JATLITE (Ler README.txt)
    1. Setup
    2. IPRouter
    3. RCApplet → JATLite IPLayer Router Client / IPRCApplet (RIGHT)
        3.1. Register
        Agent Name: AGENTE01
        Password: agente01

        (vê-se no IPRouter)

        AGENTE01, null, -1, MessageRouter, (agent-info:password agente01)
        Register Request from AGENTE01 accepted

    4. RCApplet (LEFT)
        4.1. Register
        AGENTE02
        agente02

        (ver IPRouter)

    5. Connect
    6. Troca de mensagens…
    7. Log → ficheiro txt

- (No âmbito da linguagem KQML)
  ONTOLOGIA: Conjunto de conhecimentos específicos sobre determinado domínio de aplicação.
- Os logs estão na pasta:
  D:\Programas\JATLite\RouterLayer\Resource\incoming
- Definição da CLASSPATH
  D:\Programs\JATLite

03/03/2003

- Estudo da linguagem Java
- Execução de programas
- Exemplos

05/03/2003

- Estudo e experimentação da plataforma JATLite
- RouterClientAction
- RCActionExample.java
- Consulta dos códigos de fonte.zip

- Executar:
  1. Iniciar Router
  2. Iniciar Agente Bolsa (java Wbolsa)
  3. Iniciar Agente InvsCalmo (java WinvsCalmo)
  4. Iniciar Agente Nervoso (java InvsReflx)
- Experimentação e observação
- Agente Bolsa (Primeiros Passos)
- Extracção dos Valores da Bolsa
- http://gnomo.fe.up.pt/~ee95086/PSTFC/cotacoes.php

06/03/2003

- Implementação do Agente Bolsa
- Package accoes
  Accao.java
  - Definição da acção da bolsa (estrutura)

  Accoes.java
  - Define um vector de acções
- tabCotacoes[20][8]

11/03/2003

- Implementação do Agente Bolsa
- SeqAccao (String desig, long quant, double valor, double prob)
- Formato:
  | xx.xx |
- De 10 em 10 segundos
- De 30 em 30 linhas (300 seg = 5 min)
- Criar siglas (?) para alinhar valores

12/03/2003

- Implementação do Agente Bolsa
- Interface:
  - Mensagens enviadas para os agentes investidores
  - Mensagens recebidas dos agentes investidores
  - Evolução das cotações
- Cotações em tempo real (Extraídas do NextBolsa)
  (PSI20)
- Na área das cotações é apresentada a sigla da empresa cotada
- Usada a linguagem KQML, suportada na plataforma JATLite
- Performativas...
- Cotações são actualizadas cada 10 segundos
- A caixa de texto é limpa após 30 linhas (5 minutos)
- Às acções estão associados os atributos:
  - Nome
  - Quantidade
  - Valor

13/03/2003

- Teste do Agente Bolsa (e melhoramento)
- Apresentar com 2 casas decimais
- Implementação dos Agentes Investidores
  INV01, INV02
- Quais as estratégias?
- Agente INV02: obter máximos e mínimos logo à partida
  (Alterar nos Vectores)
- Vectores: (Evolução Mundo)
  - Vtitulos
  - Vvalores
  - Vmaximos
  - Vminimos
  - Vmedios
  - Vns
- regista (acções)
  → regista (designação, valor) *
- * regista (título, valor)
  Vai a Vtitulos ver qual o IndexOf(titulo) = i
  N: Época (long)
- Ver a média (cálculo e aplicações)
- Aplicação da estratégia...
  - Quando comprar?
  - Quando vender?
- Focar no Agente INV02 (Calmo)
- Melhorar INV02. (melhorar a estratégia, adaptar a um investimento coerente com as oscilações do mercado)
- Quais os parâmetros de INV02?
  (Quais os parâmetros (variáveis) que o guiam na sua estratégia de compra/venda?)
- Estudar a mudança das variáveis com as oscilações do mercado
- Colocar na janela do INV02 um campo com Capital potencial (instantâneo) (?)
- FACTOS (INV02)
  - Durante 20 épocas o agente faz 20 observações dos valores das acções, assim como Vmax e Vmin
  - Regista o valor mádio Vmed. [COMO?][PARA QUE SERVE?]
  - Espera que uma acção atinja um valor próximo de Vmin para proceder ao investimento [QUÃO PRÓXIMO?]
  - Mantém-se passivo até que o valor se aproxime de Vmax [QUANTO?], resolvendo então vender o seu investimento
  - Reinveste numa época futura quando alguma acção satisfizer as condições anteriores
- Características de INV02:
  - Calmo
  - Prudente
- Investe 50% do seu capital

18/03/2003

- Estudo do agente INV02

	V	Vmax	Vmin
BES	11.91	12.00	11.91

$$p_v = \left| \frac{v - v_{min}}{v_{max} - v_{min}} \right| = 0$$

DECISÃO: menor p_v

- Fixar $p_v = 0.5$
  (WinvsCalmo.java, 305) getFactorP(0.5)
- Vai buscar a maior variação percentual, menor que 0.5 (50%)
  O que está mais próximo de $p_v = 0 \rightarrow$ retorna título
- Fixar investimento = 50% do capital
- Para compra: vê todos os títulos e escolhe o melhor
- Para venda: vai ao título de invesimento, vê máximos e mínimos

  $$p_v = \left| \frac{v - v_{min}}{v_{max} - v_{min}} \right|$$

  vai testar o factor de 50% (getFactor(0.50)), e em caso de se verificarem as condições ele efectua a venda
- Média: vector Vmedios e função melhorMedia() não têm influência

? $$v_{medio} = \frac{1}{N+1} * (valor + N * v_{medio})$$

$$v_{medio} = \frac{1}{N+3} * (valor + v_{max} + v_{min} + N * v_{medio})$$

- getMedia()
  Dado um título, retorna o seu valor médio
- melhorMedia()
  Retorna o título que tem a média mais alta
- Justifica-se esperar 20 épocas para agir? Passar para 10 épocas (ou menos)
- Para teste: começa a agir após 3 épocas
- Fixar em 5 épocas?
- Campo do potencial implementado com sucesso
- Ideia:
  Implementar um contador que regista há quantas épocas já se está com o mesmo investimento
  Se estiver há mais de x épocas
  &&
  valor actual > valor de compra $\rightarrow$ VENDER

  Se valor actual < valor de compra
  $\rightarrow$ esperar mais y épocas
- Alterar limites?
  Limites actuais: compra a 50% de Vmin
  vende a 50% de Vmax

V	Vmin	Vmax
12	10	20

$$p_v = \left| \frac{v - v_{min}}{v_{max} - v_{min}} \right| = \frac{12-10}{20-10} = \frac{2}{10} = 0.2 \rightarrow 20\%$$

- O que pode ser alterado:
  - Nº de épocas de observação antes de agir
  - Limites do investimento (de compra e de venda)
  - Valor do investimento
  - Capital inicial
- Criar um clone de INV02 e alterar significativamente as características → INV03
- INV02: colocar $p_v >= 0.5$
- INV03: Limites: Compra a 75%
  Vende a 25%
- Não resulta: 2.32 2.37 2.30
  Alterar os limites para 60%, 40%
  Mas se comprar a 55% e vender a 45% perde dinheiro
- Não se justifica a mudança
  Investidor para investir a longo prazo
- Aumentar a margem (p.ex. 75%, 25%) e colocar como condição de venda: cappot > (cap + capinv)
- INV03 testado com sucesso
  Aplicar mudanças a INV02
- Focar em INV01
  Tentar colocar INV01 com interface gráfica

19/03/2003

- Aplicar mudanças efectuadas em INV03 ao agenrte INV02
- Fixar INV02 e aplicar melhoramentos e testes a INV03
- Agente INV02:
  - Aguarda 5 épocas, após as quais começa a agir
  - Limites de investimento: 50%, 50%
  - Valor do investimento: 50%
  - Capital inicial: 1000
- Análise do agente INV01
- INV01 investe o máximo capital possível
- Colocar cap. potencial
- Obrigar a investir apenas se $\varphi > 0$
- Interface gráfica para INV01
- Compras/Pagamentos e Vendas/Pagamentos:
  setInvest()
  setCapital()
  setCapInv()
- Interface OK
- Qual a estratégia de compra/venda?
- Observa durante N épocas, registando os valores
- Calcula dV, ddV e φ
  dV[i] = V[i+1] – V[i]

ddV[i] = dV[i+1]-dv[i]
φ = (1+ddV)*dV*V

- Compra as acções com maior φ (no momento)
- Num determinado momento poderá vender as acções caso haja um investimento mais promissor (mesmo que perca dinheiro)
  Também poderá vender logo que veja que vai ter lucros
- Quando vê que tem lucro vende logo as acções e compra de novo acções da mesma empresa, porque o φ é o mais elevado (IRREAL!)
- Retirar a opção de vender quando tem lucro!
- Ou então... se lucro for superior a x%, vender

$$\text{lucro} = \frac{\text{Cap.Pot.}}{\text{Cap.Invs.}}$$

$$\text{lucro de 5\%} \rightarrow \frac{\text{Cap.Pot.}}{\text{Cap.Invs.}} > 1.05$$

- Qual a percentagem mínima realista?
- Ou então... se cada acção subir mais de €0.0x, vender.
  €0.05 ?
  e se descer...
- Lucro

$$\frac{\text{cap.} + \text{cap.invs} + \text{dinvs}}{\text{cap.} + \text{cap.invs}} = 1 + \frac{\text{dinvs}}{\text{cap.} + \text{cap.invs}} > 1.05$$

  ou

$$\frac{\text{getValor(investimento.desig)}}{\text{investimento.valor}}$$

- Se a acção do investimento subir 0.5%, ele vende
- Ideia:
  Alterar os vectores dV[], ddV[] e φ[] apenas se houver variação de algum valor das cotações
  NÃO!
  Ele deve permanecer num estado de observação se não houver mudanças.
  Não faz sentido porque nesse caso poderá haver erros, visto que iria haver para uma certa acção que mantém o seu valor durante várias épocas um dV≠0 e ddV≠0

20/03/2003

- Aumentar o tempo de sono para ran(60000)
- Aumentar para INV02 e INV03 ?
- Passar para ran(30000) ?
- Colocar INV02 e INV03 a investir máximo
- Passar para ran(10000)
- Fixar em 10 segundos (se for menor que 10 s, ele faz duas leituras do mesmo valor)
- Fixar também em INV02 e INV03
- Estudo das estratégias de investimento
- www.portaldebolsa.com
- www.analisestecnicas.com

- Tentar obter valores das cotações dos últimos n dias para calcular médias, etc e também para fazer gráficos
- Tentar implementar estratégias baseadas na análise da variação dos valores das cotações nos últimos n dias
- É necessário fazer uma análise a médio-longo prazo
  6 meses a 1 ano (?)
  2 semanas a 1 mês (?)
- Necessário ter valores de abertura e fecho
- Indicador MACD
  Subtrai-se à média móvel exponencial de 26 dias a média móvel exponencial de 12 dias.
  Compara-se com o gráfico da média móvel exponencial de 9 dias (trigger)
- Cálculo da média móvel exponencial:

$$ME(a) = ME(a-1) + \left[\frac{2}{N+1} * P(a) - ME(a-1) \right]$$

  P(a): valor de fecho
  N: nº de dias para os quais se quer o cálculo
- Página com alertas
  http://pt.portaldebolsa.com/pt/analysis/view_triggers.asp
- Potencial aplicação para um agente.
  (Consultar esta página e investir de acordo com os alertas)
- Ou um agente para cada alerta
- Filtrar apenas para as acções da Bolsa simulada

25/03/2003

- Implementação do indicador MACD
- É necessário obter as cotações dos últimos n dias
- Armazenar na BD ? ✓
  1) Extrair de um site que tenha as cotações dos últimos n dias
  ~~ou~~ e
  2) Começar a inserir a partir de agora
- Cotações em .txt:
  http://www.bpionline.pt/docs/assync/bannertitulospub.txt
- Consulta do site: www.investidorglobal.pt
- Agente Bolsa: apresentar as cotações dos últimos n dias
- jsp ?
- Fazer um php com o historial ✓
- Tentar fazer gráficos (asp ?)
- Inserir as empresas que faltam.
  - É necessário ir buscar o historial
  - Introduzir no Excel
- Aumentar o Agente Bolsa ?
  (Para pôr as empresas todas)
- Ver o caso quando têm o preço de fecho

26/03/2003

- Indicador MACD
- Página php do histórico

- É necessário inserir na BD todos os históricos
  $today = date ("Y-m-d"); (PHP)
- Data OK, é necessário inserir todos os valores de fecho
- Hipótese:
  Ir sempre actualizando (UPDATE) sempre que é actualizado na BD a tabela das cotações
  UPDATE PSI20_COTACOES set 'titulo' = 'x.xx' where data = $today;
- Actualizar com o cotacoes.php ✓
  Criar um ficheiro .php com funções ✗
  DELETE FROM PSI20_COTACOES WHERE data = '$today';
  INSERT INTO PSI20_COTACOES (data) VALUES ('$today'); ✓
- Actualização de cotações, histórico e BD, OK!
- Agente Bolsa OK ?
  - Passar de 10 para 30 segundos ✓
  - Colocar todas as acções ✓
  - Implementar com java.swing (?)
- Agente Investidor
  - Implementar com java.swing (?)
  - Inicialmente fazer no DOS Prompt
  - Adquirir informações (como INV02) e mostrar
  - Aplicar indicador MACD
  - Apresentar MACD e Trigger

27/03/2003

- Fórmulas (utilizadas no Excel) para cálculo do MACD

$$\text{ME(hoje)} = \text{ME(ontem)} + \frac{2}{\text{N}+1} * \left(\text{V(ontem)} - \text{ME(ontem)}\right)$$

ME(x) → Média móvel exponencial no dia x
V(x) → Valor da cotação no dia x
N → nº de dias para os quais se pretende a média

$$\text{MACD(hoje)} = \text{ME(ontem)} + \left(\frac{2}{26+1} - \frac{2}{12+1}\right) * \left(\text{V(ontem)} - \text{ME(ontem)}\right)$$

$$\text{Trigger} = \text{ME(ontem)} + \frac{2}{9+1} * \left(\text{V(ontem)} - \text{ME(ontem)}\right)$$

$$\textit{MACD} - \textit{Trigger} = \left(\frac{2}{26+1} - \frac{2}{12+1} - \frac{2}{9+1}\right) * \left(\text{V(ontem)} - \text{ME(ontem)}\right)$$

- Implementar investidor que adquira os valores das cotações e mostre os valores de MACD e trigger para cada título
- Este investidor não irá investir apenas num título, mas em vários
- Necessário novas performativas ?
- Armazenar em BD as informações sobre a sua carteira
- Agente Bolsa tem histórico (últimos 30 dias)
- Investidor pergunta histórico à Bolsa
- O problema quando está a cotação em "fecho" é que não aparece a quantidade
- Colocar quantidade em 0

- Problema resolvido
- Necessário actualizar a quantidade na Bolsa
- O histórico pode ser implementado no SeqAccao através de um vector, para cada título:
  this.historico
  método getHistorico() getHistorico(Accao)
- Historico: (SeqAccao.java)
  historico[count][2]
  public String[][] getHistorico()
  {
  return this.historico;
  }
- Performativa:
  perform.equals("ask")
  content.equals("HISTORICO TITULO")
  ou " ("HISTORICOS")
- Mandar um vector com os históricos
  → Vhistoricos
  Vhistoricos = new Vector();
  inicializar(...)
- Criar uma classe do tipo SeqHist ?
  SeqHist (String designacao, String[][] historico)

01/04/2003

- Melhorar a actualização do histórico com base na análise da data (dia da semana)
- MACD – Fórmula:

$$MACD_i = EMA_i^{(1)} - EMA_i^{(2)} \qquad \text{(MACD 12/26)}$$

$$signal_i = (1 - k_3) * signal_{i-1} + k_3 * MACD_i \qquad \text{(MM9)}$$

onde:

$$EMA_i^{(1)} = (1 - k_1) * EMA_{i-1}^{(1)} + k_1 * close_i$$

$$EMA_i^{(2)} = (1 - k_2) * EMA_{i-1}^{(2)} + k_2 * close_i$$

$$k_1 = \frac{2}{1 + N_1}, \quad k_2 = \frac{2}{1 + N_2}, \quad k_3 = \frac{2}{1 + N_3}$$

$$EMA_{-1}^{(1)} = EMA_{-1}^{(2)} = close_0, \quad signal_{-1} = 0$$

(Fórmulas do site Investidor Global)

- Testar as fórmulas no Excel
- $N_1 = 26$, $N_2 = 12$, $N_3 = 9$
- Fórmulas OK: Implementar
- Usar vectores para armazenar todos os valores intermédios ou fazer o cálculo completo ?
- Armazenar todos os valores para comparar com os valores obtidos no Excel
- Criar "carteira de acções"

Vector do tipo Acção (Taccao)
Com as acções que o investidor possui
- Verifica o indicador MACD:
  Se MACD > signal && !em_carteira
  então compra();
  Se MACD < signal && em_carteira
  então vende();
- Tentar importar os valores para o Excel a partir da BD
- Colocar gráficos do MACD junto ao histórico
- Apenas reage no primeiro cruzamento das linhas
  em_estudo = true
  Vector ? – estudo para todos os títulos
  Após 1ª compra: em_estudo = false
- Não...
- MACD > signal && !em_carteira → compra();
  MACD > signal && em_carteira → em_estudo();
  MACD < signal && !em_carteira → em_estudo();
  MACD < signal && em_carteira → vende();

02/04/2003

- Actualizar a data no cotacoes.php sempre (javascript)
  (Não dá!)
- Fazer com data/hora do php
- $now = date ("d-m-Y, H:i");

03/04/2003

- Importante fazer importação dos valores para o Excel e apresentar gráfico do MACD junto ao histórico, mas fica para mais tarde
- Focar no Agente Investidor
- Fazer performativa para perguntar histórico
- Colocar getHistorico() dentro de SeqAccao ?
- ask ("HISTORICO") ou ask ("HISTORICO 'TITULO'") ?
- Investidor pergunta a Bolsa que títulos existem
  ask ("TITULOS EXISTENTES")
- Bolsa responde...
  tell ("TITULOS_EXISTENTES")
- Colocar os titulos num vector...
  Vtitulos
- No cotacoes.php apenas actualizar historico se o dia da semana for diferente de Sábado ou Domingo

07/04/2003

- É necessário estabelecer a comunicação entre o Agente Bolsa e o Agente Investidor.
  Investidor pergunta histórico, Bolsa fornece histórico, Investidor processa informação recebida
- Inicialmente não colocar a janela gráfica. Ver apenas no prompt.

09/04/2003

- Ver método Act() em Wbolsa e Winvs
- Comparar a maneira como trocada a informação e em que estruturas
- Fazer a analogia entre acções (normal) e histórico
- Como fazer o tratamento do histórico ?
- Implementar uma classe Historico.
  (À semelhança da classe Acção)
- Classe Historico:
  - Nome da Acção *
  - Vector com os valores dos últimos n dias
    * Nome da Acção ?
      ou um campo do tipo Acção ?

- No caso de ser Historico(Acção, Vhistorico):
  - O vector terá os valores até (hoje-1)
  - A análise do valor contido em "Acção" juntamente com os valores do histórico levará à decisão (ou não) de compra/venda
- Será correcto ?
  Deverá esperar pelo preço de fecho ?
- Tomar decisões apenas baseadas nos dias anterirores ?
  (O valor actual não interessa)...
- Após implementada a classe Histórico, é necessário fazer a comunicação entre Bolsa e Investidor
  Implementar as performativas
- É necessário actualizar mais dinamicamente a Base de Dados
- Reflectir o caso de haver títulos que entram e outros que saem da lista
- No caso de entrar um novo título, começar a armazenar a partir daí ou ir buscar histórico ?
- O histórico tem que ser prpriedade da Bolsa
- Caso da carteira de acções...
  Criar classe "Carteira" ?
  Ou apenas um vector/array ?

10/04/2003

- Classe Historico no package accoes ?
- À semelhança de TAccao e TAccoes fazer Historico e Historicos...
- Inicializar o histórico da mesma forma que é inicializada a acção
- No package Sequencia é necessário definir como é feita a sequência dos valores presentes no histórico
- Inicializar o histórico no Wbolsa

29/04/2003

- http://sourcefourge.net/projects/artstkmkt
- Organizar informação para escrita do relatório de progresso
- Ver links de projectos sobre Bolsas Virtuais com Agentes
- http://artstkmkt.sourcefourge.net
- Procura: Artificial Stock Market

- http://www.phy.duke.edu/~palmer/papers
- http://people.brandis.edu/~bleberon/wps.html

30/04/2003

- Organização da informação com vista à escrita do relatório de progresso
- http://www.swarm.org
- ftp://ftp.swarm.org/pub/swarm

02/05/2003 → 08/05/2003

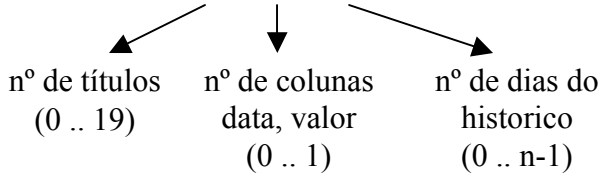
- Escrita do relatório de progresso

13/05/2003

- Testar Investidor de raíz
- Colocar argumentos para parametrizar os vários investidores
- Começar a partir da fonte original
- 1º) Comunicação com o Router
    - Comunicação OK
- 2º) Tratar e enviar histórico, performativas
    - No agente Bolsa obter histórico a partir da BD
    - Colocar em historico[][], quando recebe a info da BD

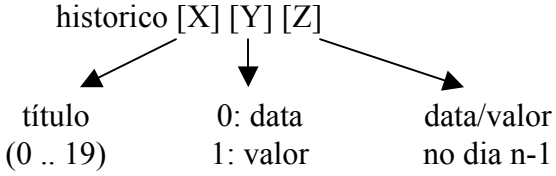
14/05/2003

- Tratar histórico (2)
  String [][][] historico;
  historico = new String [20][2][count]



historico [0][0][0] = 2003-02-17
historico [0][0][1] = 2003-02-18
historico [0][0][2] = 2003-02-19
historico [0][0][3] = 2003-02-20

historico [0][1][0] = 1.97
historico [0][1][1] = 1.95
historico [0][1][2] = 1.94
historico [0][1][3] = 1.90



- Envio do historico OK
  Vale a pena mandar a data ?
  Mandar apenas Cotações... ?
  Mandar título ?

15/05/2003

- Tratamento do histórico
- Separação em tokens
- É necessário colocar o arrray:
  float hist[][] = new float [20][X]
- Devia dar new float [20][] ?
- Necessário saber o nº de dias a que corresponde o histórico para determinar o tamanho do array, etc
- Enviar perforrmativa "N_DIAS" antes do histórico e armazenar numa variável (n_dias)
- Tratamento do nº de dias feito com sucesso
  O array fica com o tamanho correcto
- Envio do histórico OK
- Armazenamento do histórico por parte do Investidor OK
- Implementar arrays para guardar as médias móveis, etc
- ATENÇÃO: Os valores enviados pela bolsa para o investidor não estão actualizados (são os que são recolhidos quando a Bolsa é inicializada)
- Fazer uma função para actualizar permanentemente os valores ?
- As médias móveis, MACD, etc. são calculos feitos pelo Investidor e é ele que armazena os arrays

20/05/2003

- Fazer a actualização dos valores da Bolsa. Os valores que são enviados pela Bolsa para o Investidor são os valores que a Bolsa foi buscar à BD quando foi inicializada
- Método actualizar() ? ✓
- Será que o problema é apenas quando a Bolsa envia a informação para o Investidor ?
  Porque no painel os valores são actualizados...
- O array historico[][] é enviado para o Investidor
  Este array só é actualizado uma vez
  É necessário actualizá-lo sempre
- actualiza() : Actualiza tabela[][] e historico[][]
- Actualiza mas com cerca de 6 ciclos de atraso
- Diminuindo o tempo de espera do Investidor (sleep) de 30 para 5 segundos, já fica actualizado mais rápido
  (Atraso agora é 3 ciclos) – Nem sempre... às vezes é imediato
- Passo seguinte aplicar as fórmulas das médias móveis e construir arrays
- Ver ficheiros MACD_teste.xls
- Fórmulas:

$$MACD_i = EMA_i^{(1)} - EMA_i^{(2)} \qquad \text{(MACD 12/26) (MM26-MM12)}$$

$$signal_i = (1 - k_3) * signal_{i-1} + k_3 * MACD_i \qquad \text{(MM9)}$$

onde:

$$EMA_i^{(1)} = (1 - k_1) * EMA_{i-1}^{(1)} + k_1 * close_i$$
$$EMA_i^{(2)} = (1 - k_2) * EMA_{i-1}^{(2)} + k_2 * close_i$$
$$k_1 = \frac{2}{1 + N_1}, \quad k_2 = \frac{2}{1 + N_2}, \quad k_3 = \frac{2}{1 + N_3}$$
$$EMA_{-1}^{(1)} = EMA_{-1}^{(2)} = close_0$$
$$signal_{-1} = 0$$

$N_1 = 26, \quad N_2 = 12, \quad N_3 = 9$

EMA1[x][0] = historico [x][0]
EMA1[x][1] = (1-k_1)*EMA1[x][0] + k_1*historico[x][1]
EMA1[x][i] = (1-k_1)*EMA1[x][i-1] + k_1*historico[x][i] $i \in N$

EMA2[x][0] = historico [x][0]
EMA2[x][1] = (1-k_2)*EMA2[x][0] + k_2*historico[x][1]
EMA2[x][i] = (1-k_2)*EMA2[x][i-1] + k_2*historico[x][i] $i \in N$

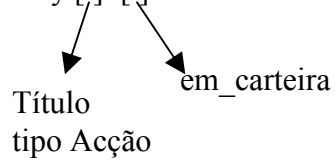
MACD[x][0] = 0
MACD[x][1] = EMA1[x][1] – EMA2[x][1]
MACD[x][i] = EMA1[x][i] – EMA2[x][i] $i \in N_0$

signal[x][0] = 0
signal[x][1] = (1-k_3)*signal[x][0] + k_3*MACD[x][1]
signal[x][i] = (1-k_3)*signal[x][i-1] + k_3*MACD[x][i] $i \in N$

- MM26 – OK
- MM12 – OK
- MACD – OK
- signal – OK
- Fazer exportação dos valores do MACD e de signal para ficheiro de texto e depois importar no Excel para fazer gráficos

21/05/2003

- Fazer a exportação para todos os títulos
- Atenção: é necessário dividir o valor das cotações da CIMPOR por 5 devido ao Stock Split
- Implementar carteira de títulos
- Array? Dimensões? 20? ✓ carteira[]
  Qual a info? Bool em_carteira? 1/0
  Preço de compra? Quantidade?
  É necessário definir capital inicial
  Array [ ] [ ]



- Ver mais ou menos como estão os Investidores base...
  INV01 e INV02, a partir do código original
- Ou na classe TAccao criar um campo em_carteira ?
- Capital inicial ? € 1000,00 ?
  € 5000,00 ?
  → € 50.000,00
  Investimentos de € 25.000,00
- Implementar vários investidores, cada um tem um capital diferente.
  Cada um investe mediante o seu capital
  Investidores com diferentes valores de carteira
  € 500000
  € 50000
  € 5000
- Fazer uma classe Carteira dentro do package accoes ?
- Características:
  Capital inicial
  Capital investido (?)
  Capital potencial (?)
  Títulos
  ...
- Colocar ficheiros do MACD e signal com outras extensões ✓
  - .macd
  - .sgn

- Classe Carteira
  public float capital_inicial;
  public TAccao Carteira;
- Alteração da classe TAccao (public boolean em_carteira)
  Acrescentado no construtor TAccao():
  em_carteira = false;
- Centralizar informação
  Bolsa faz todos os cálculos e insere na BD
  Arranjar diversas performativas para o Investidor obter o que pretende (MACD, signal, etc)
- Toda a info é apresentada na interface do Agente Bolsa (indicadores, gráficos, etc)
- Tratar info na Bolsa e tentar implementar de imediato com java.swing (?)

22/05/2003

- Agente Bolsa faz cálculos (e insere na BD)
- Fazer métodos para cálculos ✓
- A partir do historico[][][] pode-se fazer os outros arrays
- Pode ficar dentro do método actualiza()
  historico [k][0][j] = data
  historico [k][1][j] = valor
- Passar para historico[][]
  double [][] historico;
  historico = new double [20][count];
  (A data é eliminada)

- Bolsa faz os cálculos – OK
  Guarda os valores em arrays
- Performativas para enviar macd, signal, etc.
- Performativa HISTORICO – OK
- Performativa MACD
- Performativa SIGNAL
- Performativas:
  - 1º) No agente Bolsa o envio
  - 2º) No agente Investidor a recepção
- É necessário ver qual a sequência correcta das performativas trocadas entre Bolsa e Investidor
  Investidor/ask(TITULOS), Bolsa/tell(TITULOS)
  Investidor/ask(MACD), Bolsa/tell(MACD)
  Investidor/ask(SIGNAL), Bolsa/tell(SIGNAL)
- É necessário antes de MACD e SIGNAL, o Investidor receber N_DIAS
- String da performativa é muito grande (MACD)
- Mandar um título de cada vez ?
- Mandar os últimos 30 MACD/signal ?
- Mandar os últimos n MACD/signal ?
- Mandar o último MACD/signal ?
- Mandar apenas um alerta ?
- Array (boolean) com os títulos favoráveis para compra/venda ?

26/05/2003

- Implementar arrays que contenham alertas de compra/venda
- Possibilidade para os valores do array (String)
  0/compra/venda indicador[20]
- Como alocar os valores no array ?
- Inicialização/Alocação (?)
  Se macd [hoje] > signal [hoje]
  && macd [ontem] < signal [ontem]
  então indicador [i] = "compra"
  Se macd [hoje] < signal [hoje]
  && macd [ontem] > signal [ontem]
  então indicador [i] = "venda"
  else
  indicador [i] = "0"
- Performativa "INDICADOR"
- Ou então indicador [20][n_dias]
  Para armazenar todos os alertas
  Enviar para Investidor: indicador[i][hoje]
- O array indicador[][] foi implementado com sucesso
- Implementação da performativa "INDICADOR"
  → Investidor/ASK ("INDICADOR")
  Bolsa/TELL ("INDICADOR")
  P. ex. hoje: n_dias = 68
  Bolsa envia:
  indicador[i][67]
- Envio do INDICADOR – OK

- É necessário armazenar por parte do Investidor
- Implementar sequência de performativas
  - Investidor/ask ("TITULOS")
  - Bolsa/tell ("TITULOS")
    (Investidor armazena informação sobre os títulos)
  - Investidor/ask ("INDICADORES")
  - Bolsa/tell ("INDICADORES")
    (Investidor armazena indicadores e faz investimentos)
  - (sleep)
- Indicadores bem armazenados em array por parte do Investidor: indicadores[]
- É necessário implementar uma sequência correcta de performativas com vista à análise dos dados por parte do Investidor para que este possa fazer o investimento

27/05/2003

- Sequência das performativas
  - 1º) TITULOS DISPONIVEIS
  - 2º) INDICADORES
- Criar carteira de títulos e decisões de investimento
- Alternativa ao envio de indicadores: envio de macd e signal dos últimos 2 dias
- Classe Carteira
  Array do tipo TAccao
  Se ta.quantidade = 0
  então não tem este título
- Preencher o array
- Criar em_carteira (boolean)
  Preencher tudo com false ✓
- Com a análise de indicadores[] e em_carteira[], o Investidor toma a decisão de compra ou venda
- Array investimento[] ?
- Se indicadores [i] = = compra
  && em_carteira [i] = = false
  então comprar ;
  Se indicadores [i] = = venda
  && em_carteira [i] = = true
  então vender ;
  // else continue;
- Implementar o investimento
  (compra/venda)
- Passar investimento para investimento[]
- Fazer Carteira[] (TAccao)
  Inicializar com:
  - Título
  - Valor actual
  - Quantidade adquirida
- Fazer Investimento[] (TAccao)
  - Título
  - Valor actual
  - Quantidade adquirida

- Se não tem o título na carteira: valor do investimento = 0, quantidade = 0
- Inicializar investimento[]:
  - investimento[i].desig = "..."; (?)
  - investimento[i].quant = 0L;
  - investimento[i].valor = 0.0;

~~- Investimento[]: variável global e inicializada quando recebe performativa titulos~~

- Títulos[], variável global , inicializada quando é recebida performativa TITULOS (OK)
- Investimento[] inicializada ?
- Inserir valores aquando da análise indicadores/em_carteira

  Caso de <u>compra</u>:

  investimento[i].desig = titulos[i].desig
  investimento[i].valor = titulos[i].valor
  investimento[i].quant = x → calculado de acordo com o investimento e capital disponivel
  → Envia performativa para Bolsa ("COMPRA")

  Caso de <u>venda</u>:

  → Envia performativa para Bolsa ("VENDA")
  investimento[i].desig = titulos[i].desig
  investimento[i].valor = 0.0
  investimento[i].quant = 0L

  boolean inicializado ?
  inicializar investimento ?
- Inicialização de investimento – OK
- Necessário actualizar BD (NextBolsa em baixo o dia todo)
- Implementar ordens de compra e venda
- FACTO: Quando o Investidor vai fazer a análise dos dados (indicadores vs em_carteira), o array investimento[] está "limpo"

  investimento[i].desig = "TITULO"
  investimento[i].valor = 0.0
  investimento[i].quant = 0L
- Análise dos dados (indicadores vs em_carteira)

  Se indicadores[i].equals("compra")
  && em_carteira[i] = = false
  então | investimento[i].valor = titulo[i].valor
  | investimento[i].quant = quantidade → de acordo com capital investido
  | envia performativa Bolsa("COMPRA")
  | em_carteira = true

  Se indicadores[i].equals("venda")
  && em_carteira[i] = = true
  então | envia performativa Bolsa("VENDA")
  | em_carteira = false
  | investimento[i].valor = 0.0
  | investimento[i].quant = 0L

- Decisões a tomar: qual o capital investido em cada título ?
  Capital fixo ou percentagem (fixa/aleatória) do capital actual ?

28/05/2003

- Implementar ordem de compra/venda com as respectivas performativas
- NOTA: Forma como foi inicializado investimento[]:
  for (int b=0; b<20; b++)
  { investimento[b] = new TAccao();}
  → Devido ao construtor da classe TAccao:
  public TAccao(){
  desig = " ";
  quant = 0L;
  valor = 0.0;
  }
  As designações são colocadas quando é recebida a performativa "TITULOS":
  investimento[a].desig = ta.desig

- É necessário decidir: qual o valor do investimento ?
  Capital inicial = 500000
  Investimentos = 25000
- O capital investido só deve ser descontado depois de receber a ordem de pagamento
- É necessário verificar aquando da compra se existe a quantidade pretendida
- em_carteira passa para true apenas depois de efectuar pagamento da compra
  em_caeteira passa para false apenas depois de receber pagamento da venda
- Colocar a condição de quantidade > 0
  Quando quantidade = 0, não faz análise
- Performativas:
  Investidor: "COMPRA", Bolsa: "COMPRA-PAGAMENTO"
  Investidor: "VENDA", Bolsa: "VENDA-PAGAMENTO"
- Detalhes das performativas:
  Investidor → Bolsa ("COMPRA")
  investimento[x].desig, investimento[x].quant
  Bolsa → Investidor ("COMPRA-PAGAMENTO")
  titulo
- O Agente Bolsa faz automaticamente os cálculos para o caso de se pretender uma quantidade superior à existente
- Aparentemente, tudo OK, falta verificar VENDA
- ATENÇÃO: O sistema tem que estar "limpo"
  Dá erro se receber performativas que estão em fila de espera no Router
- Possibilidades de diversificação de estratégias
  1) Como está (MACD, parte do repouso)
  2) MACD, mas compra logo todos os titulos onde se verifique MACD > signal
- Mas atenção: o ideal era arranjar algum parâmetro que fosse alterado de agente para agente
- Uma hipótese é o capital inicial
- Para os investidores com menos capital é necessário fazer a verificação do investimento (ou então investe sempre percentagens)

- Para o caso do Agente "Nervoso":
  Como é a Bolsa que faz a análise macd/signal, terá que ser implementado um array estado[] com as variantes "comprar" e "vender"
- Performativa "ESTADO"
- REUNIÃO COM OS ORIENTADORES
  - Variante do Investidor baseado no MACD fica para depois
  - Implementar Indicador Estocástico
  - Tentar no Inv. MACD mandar os últimos 2 dias de macd/signal e ele faz a análise
  - Implementar performativas para tal

29/05/2003

- Inv. MACD: Bolsa envia os 2 últimos valores de macd[] e signal e o investidor analisa e obtém os sinais de compra/venda
  Implementar as performativas
- Obter fórmulas para os cálculos do indicador estocástico e começar a implementar este investidor
- Fazer a análise do indicador estocástico importando os dados no Excel, fazendo os gráficos correspondentes
- Envio de 2 performativas: "MACD" e "SIGNAL"
- Tentei colocar o caracter '#' como separador, mas não deu. Deve ser caracter ilegal
  Coloquei o caracter '+' como separador
- No investidor fazer separação dos tokens e armazenar em dois arrays, macd[][] e signal[][]
- Retirar do investidor performativa INDICADORES e para a separação utilizar o método split()
  public double[][] split (String S)
- Arrays macd[][] e signal[][], preenchidos
- Criar array indicador[] e preenchê-lo com os valores "compra", "venda" e "0" após análise de macd/signal
- Indicador[], OK
- Sequência das performativas
  Investidor (TITULOS), Bolsa (TITULOS)
  Investidor (MACD), Bolsa (MACD, SIGNAL)
  Investidor (TITULOS), ...
  Investidor (COMPRA), ...

03/06/2003

- Estudo do Indicador Estocástico

$$STK\% = 100 * \frac{(P - L)}{(H - L)}$$

onde:
P: último fecho
L: menor fecho dos últimos N períodos
H: maior fecho dos últimos N períodos

Análise:

- Sinal de compra:
  Se STK% desce abaixo dos 20% para depois subir acima dos 20%

- Sinal de venda:
  Se STK% sobe acima dos 80% para depois descer abaixo dos 80%

- Implementação do Indicador Estocástico
- Bolsa calcula mínimo e máximo
- O cálculo pode ser feito quando é feito o cálculo do macd
- Performativas
  Tentar:
  Investidor ("ESTOCASTICO N") (?)
- Para já, dos dias todos
- Investidor recebe mínimo e máximo e faz análise
- Mínimos e Máximos têm que ser:
  minimo[] e maximo[]
- Arrays minimo[] e maximo[] – OK
- Implementação da análise

$$stk[i][N] = 100 * \frac{historico[i][ontem] - minimo[i]}{maximo[i] - minimo[i]}(\%)$$

Tem que ser stk[][] para armazenar o valor do indicador em cada dia

```
double stk[][];
stk = new double [20][n_dias-1];
```

minimo[x]: valor mínimo registado até ao dia x
maximo[x]: valor máximo registado até ao dia x

- Dá erro a escrever para o ficheiro (.stk)
- Armazenar em minimo[][] e maximo[][] ?
  Guarda a informação acerca do mínimo/máximo em cada dia
- Retirando os ciclos relativos a minimo/maximo/stk não dá erro

04/06/2003

- Bolsa manda histórico dos últimos 30 dias e mínimos/máximos (?)
- Investidor faz cálculos (e gráficos)
- Começar a implementar investidor
- Bolsa manda apenas o histórico. Investidor verifca máximo/mínimo ✓
- Performativa ("HISTORICO 30")
- Implementar o investidor (InvStk) passo a passo e ir testando. Comunicação, etc.
- 1º InvStk pergunta TITULOS DISPONIVEIS – OK
- 2º Pergunta HISTORICO 30
- Comunicação: OK
- Em Bolsa implementar o envio dos últimos 30 valores
- Verificar o método split() em TAccoes()
- Necessário fazer splitHist()

- Performativa HISTORICO recebida – OK
  Armazenamento dos valores – OK
  historico[20][30]
- Implementar cálculos no Investidor
- Cálculos, stk[][] – OK
- Implementar decisões de compra/venda
- É necessário implementar duas variáveis
  boolean abaixo20[20]
  boolean acima80[20]
  1) Inicializa-se a false
  2) Se stk[i] < 20
  abaixo20[i] = true;
  Se stk[i] > 80
  acima80[i] = true;
  3) Se stk[i] > 20
  && abaixo20[i] = = true && em_carteira = = false
  | comprar[i];
  | abaixo20[i] = false;
  Se stk[i] < 80
  && acima80[i] = = true && em_carteira = = false
  | vender[i];
  | acima80[i] = false;

- Será melhor ter uma variável estado[] com 3 valores possíveis ?
  acima80 / neutro / abaixo20
  String estado[][]
- Decisão:
  (Se stk[i][hoje] > 20
  && estado[i][ontem].equals("abaixo20")
  && em_carteira[i] = = false)
  → COMPRAR
  (Se stk[i][hoje] < 80
  && estado[i][ontem].equals("acima80")
  && em_carteira[i] = = true)
  → VENDER
- Tudo OK

05/06/2003

- Experimentação do SMA
- Variantes para o InvStk:
  - nº de dias da análise (em vez de 30 podem ser 10, 20, 60, ...)
  - margens de compra/venda (em vez de 80/20, pode ser 70/30, ...)
- Exportação para o Excel, OK
  Visualização dos gráficos, OK
- Capital potencial: apresentar com 2 casas decimais
  (Arredondar)... Usei método trunc()
  Investidor e InvStk

09/06/2003

- Experimentação do SMA
- Implementação de InvStk10 ✓
- Criar InvStk10_7030 ➨
- Na análise dos dados (lançamento dos sinais) pode-se alterar:
  - Retirar estado[][]
  - Apenas comparar stk[i][29] com stk[i][28]
- Fazer alternativas para InvStk
  (Nº de dias / Margem)
- Como é necessário fazer a simulação ao longo de vários dias, é necessário armazenar os dados relativos à carteira dos investidores
  (txt ?)
- Alternativas:

	80/20	**70/30**	**90/10**
30 dias	INV3020	INV3030	INV3010
60 dias	INV6020	INV6030	INV6010
15 dias	INV1520	INV1530	INV1510

- Para fazer armazenamento dos dados relativos à carteira:
  - Quando a carteira (array investimento[]) é actualizada, o ficheiro (.crt ou .inv) é actualizado
  - Quando o agente é inicializado, é carregada a informação
- Alternativas para o MACD ?
  - Compra logo no início, se houver sinal que a seguir irá haver ordem de venda
  - Vários capitais iniciais

11/06/2003

- Fazer armazenamento dos investimentos num ficheiro (.inv)
- Fazer um método para escrever
  updateInv()
- Fazer um método para ler
  getInv()
- Fazer e testar no InvStk.java
  A partir deste fazer os outros
- Ver quando é feita actualização/inicialização de investimento[]
- 1º) Fazer a escrita para o ficheiro (deixar estar a inicialização)
  2º) Quando escrita estiver implementada, retirar a inicialização e implementar a leitura do ficheiro
- INV3020.inv
- Quando é carregada a informação, é necessário actualizar em_carteira
  (se quant > 0)
- Também é necessário armazenar info acerca do capital
- UpdateInvestimento() – OK
- Fazer getInvestimento() e getCapital()
- getCapital() – OK
- getInvestimento() – OK
- Colocar em_carteira = true se investimento[i].quant > 0 – OK

- Sempre que há investimentos (compra/venda) é necessário fazer updateInvestimento()

12/06/2003

- Fazer variantes do InvMACD
- Verificar quando Investidor não tem capital suficiente ×
- Implementar o armazenamento em ficheiro das informações do InvMACD
- Fazer gráficos de STK60 e STK15
- Variantes de InvMACD
  500000 Investe 5% do capital
  50000 10%
  5000 20%
- Retirar alguns printl, para a execução ficar menos lenta
- Colocar o sleep em 30 segundos
- Começar a preparar relatório
- Para fazer os vários investidores:

java InvestStk nDias margem

java InvestMACD capIni	(€/1000)

- *Inv MACD → investem 10% do capital*

17/06/2003 →

- Experimentação do SMA
- Escrita do relatório

Referências

[AFJM95] R. Armstrong, D. Freitag, T. Joachims, T. Mitchell. WebWatcher: A Learning Apprentice for the WWW. *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, 1995.

[Bat94] J. Bates. The Role of emotion in believable agents. Em [*Rie94*].

[BSY95] M. Balabanovic, Y. Shoham, Y. Yum. An Adaptative Agent for Automated Web Browsing. *Journal of Image Representation and Visual Communications*, 1995.

[CA02] Carlos Costa, Sérgio Almeida. *Visualização num Browser de Informação dinâmica de uma Base de Dados*. Informação na Web, Licenciatura em Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, 2002.

[Coe96] Hélder Coelho, *Inteligência Artificial Distribuída – Uma Introdução*, Departamento de Informática – Faculdade de Ciências da Universidade de Lisboa, 1996

[FG96] S. Franklin, A. Graesser. Is it an Agent or Just a Program?: A Taxonomy for Autonomous Agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag, 1996.

[FLM97] Tim Finin, Yannis Labrou e James Mayfield. KQML as an Agent Communication Language. *Software Agents, editado por Jeffrey M. Brandshaw*. AAAI/MIT Press, 1997.

[Gil+95] D. Gilbert, et al., IBM. *The Role of Intelligent Agents in the Information Infrastructure*, 1995.
`http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm`

[GK95] M. Genesereth, S. Ketchpel. *Software Agents*. Stanford University, 1995.
`http://logic.stanford.edu/sharing/papers/agents.ps`

[Hew77] C. Hewitt. Viewing Control Structures as Patterns of Passing Messages. *Artificial Intelligence*, 8(3), 1977.

[Mar99] Domingos Martinho, *Agentes nos Processos de Aquisição de Bens e Serviços na Administração Pública*, Departamento de Informática – Faculdade de Ciências da Universidade de Lisboa, 1999

[Nwa96] H. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11(3), Cambridge University Press, 1996.

[RCM02] Carlos Rocha, Helder Castro, Pedro Marques. *Acesso a uma Base de Dados Utilizando PHP*. Informação na Web, Licenciatura em

Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, 2002.

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

[Rie94] D. Riecken (editor). Tema especial: Intelligent Agents. *Communications of the ACM*, 37(7), Julho 1994.

[RM02] Gabriel Ribeiro, Filipe Mota. *Visualização de Informação dinâmica de uma BD (Servlets)*. Informação na Web, Licenciatura em Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, 2002.

[Roc01] Ana P. Rocha. *Metodologias de Negociação em Sistemas Multi-Agentes para Empresas Virtuais*, Departamento de Engenharia Electrotécnica e de Computadores - Faculdade de Engenharia da Universidade do Porto, Dezembro 2001.

[Sho93] Y. Shoham. Agent Oriented Programming. *Artificial Intelligence*, 60(1), 1993.

[Sil99] Alberto Silva. *Agentes de Software na Internet*, Centro Atlântico, Lda., 1999.

[Sun95] Sun Microsystems, Inc. *The Java Language: A White Paper*. 1995.

[VB90] S. Vere, T. Bickmore. A Basic Agent. *Computational Intelligence*, 6, 1990.

[W3AT] PortalDeBolsa.Com, *Arquivo ABC da Análise Técnica*.
`http://pt.portaldebolsa.com/pt/analysis/abc_at_archive.asp`

[W3CDR] `http://www.agent.org/society/meetings/workshop9702`

[W3Cor] João P. Cordeiro, *Bolsa Virtual Simulada*.
`http://www.fe.up.pt/~eol/SMA/20002001/JCordeiro_Bolsa/fonte.zip`

[W3JAT] JAT*Lite* Introductory FAQ, *CDR, Stanford University*, 1996, 1997, 1998, 1999.
`http://www-cdr.stanford.edu/ProcessLink/papers/JATL.html`

[W3JPC] Heecheol Jeon, Charles Petrie, Mark R. Cutkosky. *JATLite: A Java Agent Infrastructure with Message Routing*. CDR, Stanford University.
`http://www-cdr.stanford.edu/ProcessLink/papers/jat/jat.html`

[W3WC] Web Croller.
`http://www.fe.up.pt/~eol/IW/htmls/web0.5.tgz`

[WJ94] M. Wooldridge, N. Jennings. *Agent Theories, Architectures, and Languages: A Survey*. Workshop on Agent Theories, Architectures and

Languages (incluído em Eleventh European Conference on Artificial Intelligence), Amsterdam, The Netherlands, 1994.

[WJ95] M. Wooldridge, N. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2), Cambridge University Press, 1995.