Ano Lectivo 2012/2013

Relatório Final

Universidade do Porto

Faculdade de Engenharia

FEUP

Confiança em Redes Sociais

Agentes e Inteligência Artificial Distribuída

Gonçalo Pereira - 090509072 - ei09072@fe.up.pt
José Carlos Pereira - 090509012 - ei09012@fe.up.pt

Porto, 8 de Dezembro de 2012

Índice

Objectivo

Este trabalho foi realizado no âmbito da cadeira de Agentes e Inteligência Artificial Distribuída, do 1º semestre do 4º ano do MIEIC, da Faculdade de Engenharia da Universidade do Porto. E tem como propósito a implementação de um sistema constituído por vários agentes, daí a criação, definição e interacção destes.

A confiança desempenha um papel bastante importante nas redes sociais apesar de muitas vezes esse valor não ser compreendido pelos seus utilizadores. A confiança permite a formação de comunidades em redes sociais, avaliar a qualidade e credibilidade das informações, bem como determinar como a informação é passada de utilizador por utilizador na rede.

Com isto, de uma maneira geral, objectivo deste trabalho é desenvolver um sistema que simule a utilização de um mecanismo de confiança computacional numa rede social, no nosso caso, de um sistema de compra e venda de produtos online, como o *Ebay* ou o *Leiloes*. Pretendemos que os agentes possam interagir, de maneira a poderem realizar negócios de compra e venda entre si e assim o programa poder mostrar o comportamento dos vários agentes ao longo do tempo possibilitando-nos assim realizar vários tipos de testes e tirar conclusões. De maneira a facilitar a simulação do comportamento dos agentes na rede social utilizamos a ferramenta REPAST.

Este relatório tem como objectivo apresentar o resultado final do trabalho desenvolvido, explicando a maneira como implementamos o sistema, a sua estrutura, os seus módulos, classes e estruturas utilizadas, assim como as conclusões que tiramos.

Especificação

Identificação e caracterização dos agentes

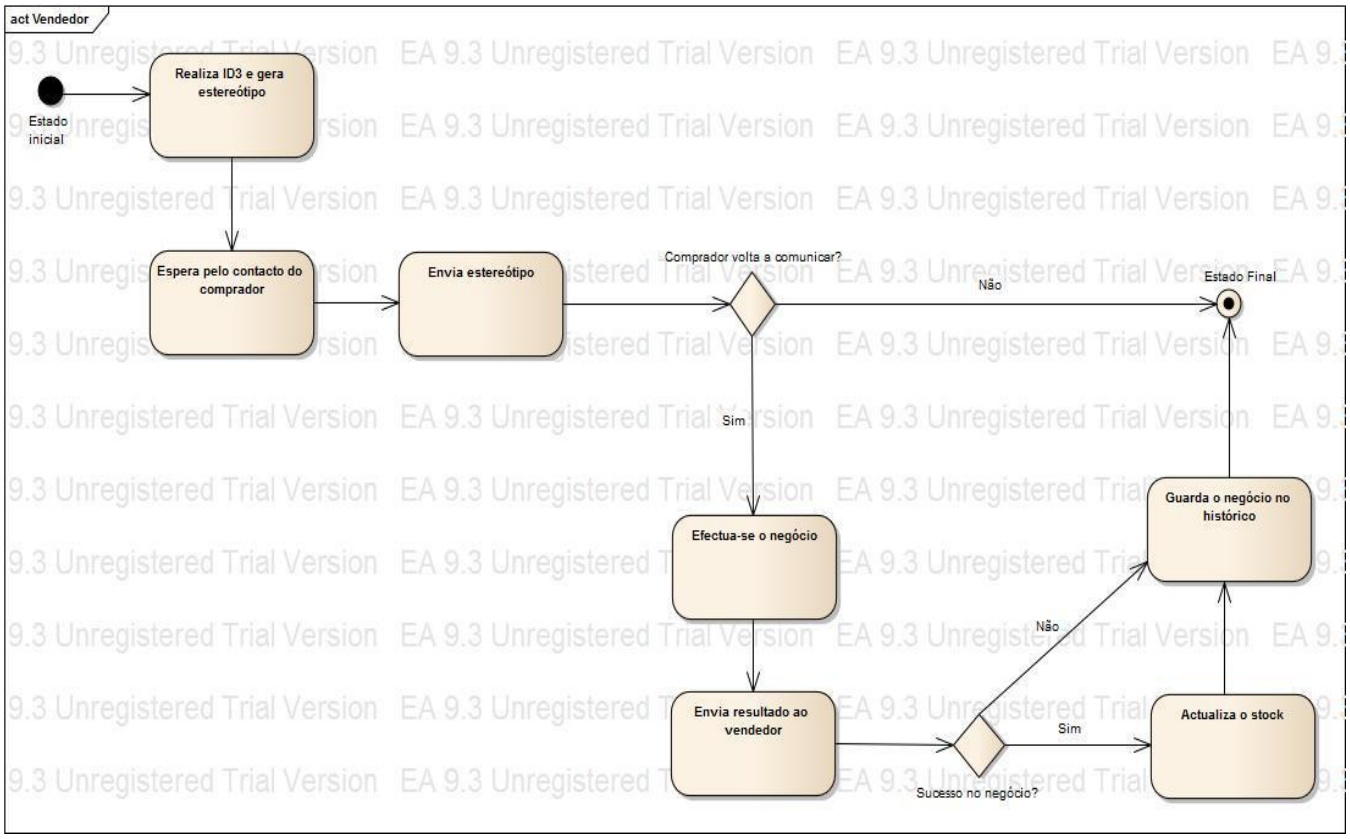
O sistema desenvolvido é constituído por dois tipos de agentes que irão interagir entre si: o **comprador** e o **vendedor** que representam os dois tipos de "papéis" que podemos ter num sistema de compra e venda online.

Agente Comprador

O comprador é um agente que mediante os produtos que tem na sua lista de compras escolhe de entre todos os vendedores, que possuem o produto que quer, o que conseguirá satisfazer as suas necessidades de compra de uma melhor forma (tendo em conta os preços e a confiança que deposita no vendedor). Conforme for negociando com os vendedores, este irá após cada iteração medir o nível de confiança do vendedor e formar uma opinião sobre este.

Cada comprador possui um *id* que é um inteiro que permite identificar e distinguir de uma melhor forma os compradores da rede social. Possuem também uma lista de compras que diz respeito aos produtos que o comprador pretende adquirir e que vai tentar "encontrá-los" nos stocks dos vendedores. Á medida que o comprador vai interagindo com os vendedores vai formalizando opiniões sobre cada um destes, essas opiniões criadas vão sendo adicionadas a uma lista que contêm todas as opiniões do comprador em relação aos vendedores. Uma opinião permite saber se um comprador já interagiu alguma vez com um determinado vendedor e contêm o valor de confiança que um comprador deposita num vendedor, confiança essa que é medida através do modelo *Sinalpha.*

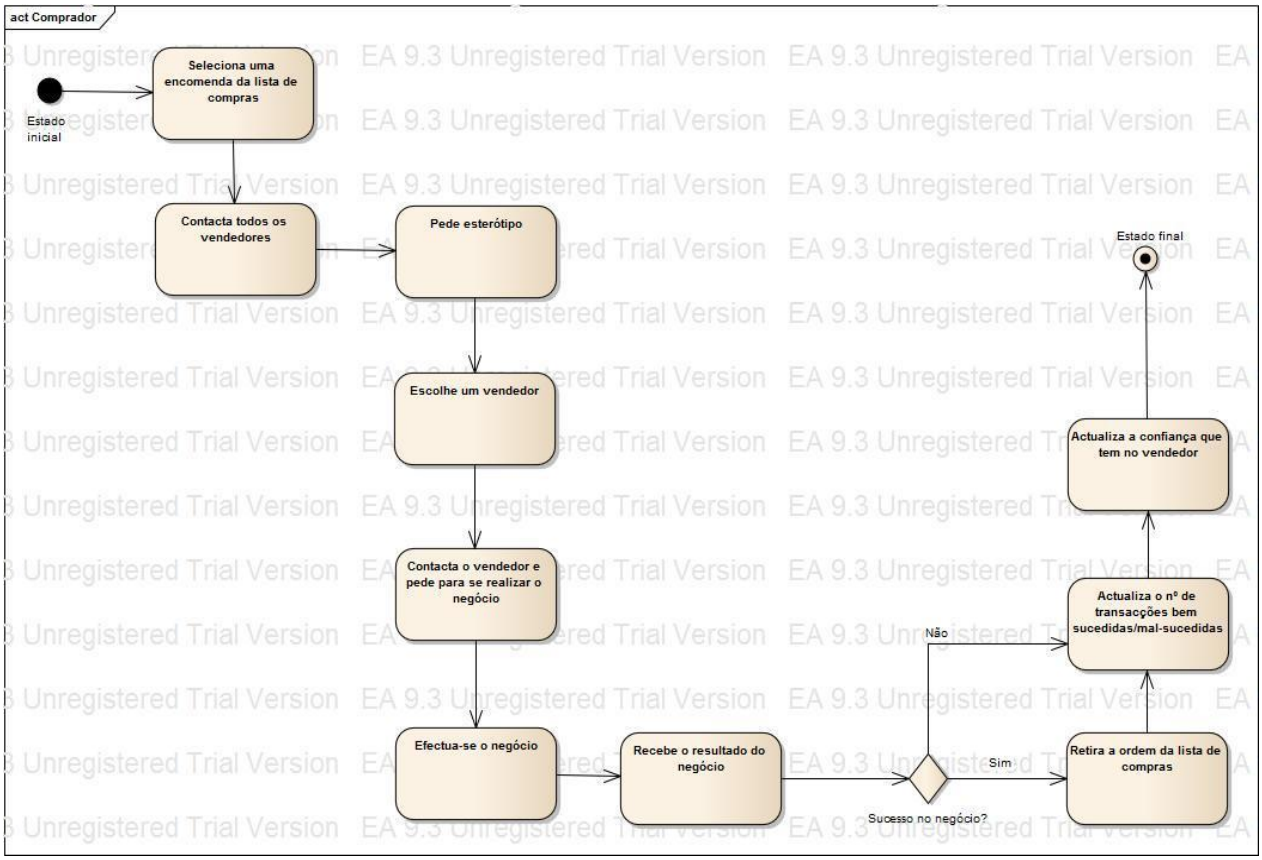
Também é importante referir que na interface gráfica do simulador os compradores são identificados pelos nós a verde.



Agente Vendedor

O vendedor é um agente que possui uma lista com vários produtos (encomendas)que deseja vender, que constituem o seu stock, cada uma dessas encomendas é definida pelo seu nome, preço, e quantidade disponível. Para além disso, dependendo de certas condições, o agente têm uma probabilidade, que é gerada aleatoriamente, de falhar uma entrega. Cada vendedor possui também um histórico de todas as encomendas que foram pedidas e os seus respectivos resultados (enviada ou não). Para guardar esse histórico utilizamos o algoritmo ID3.

Os vendedores são representados por nós azuis na interface gráfica.



Protocolos de interacção

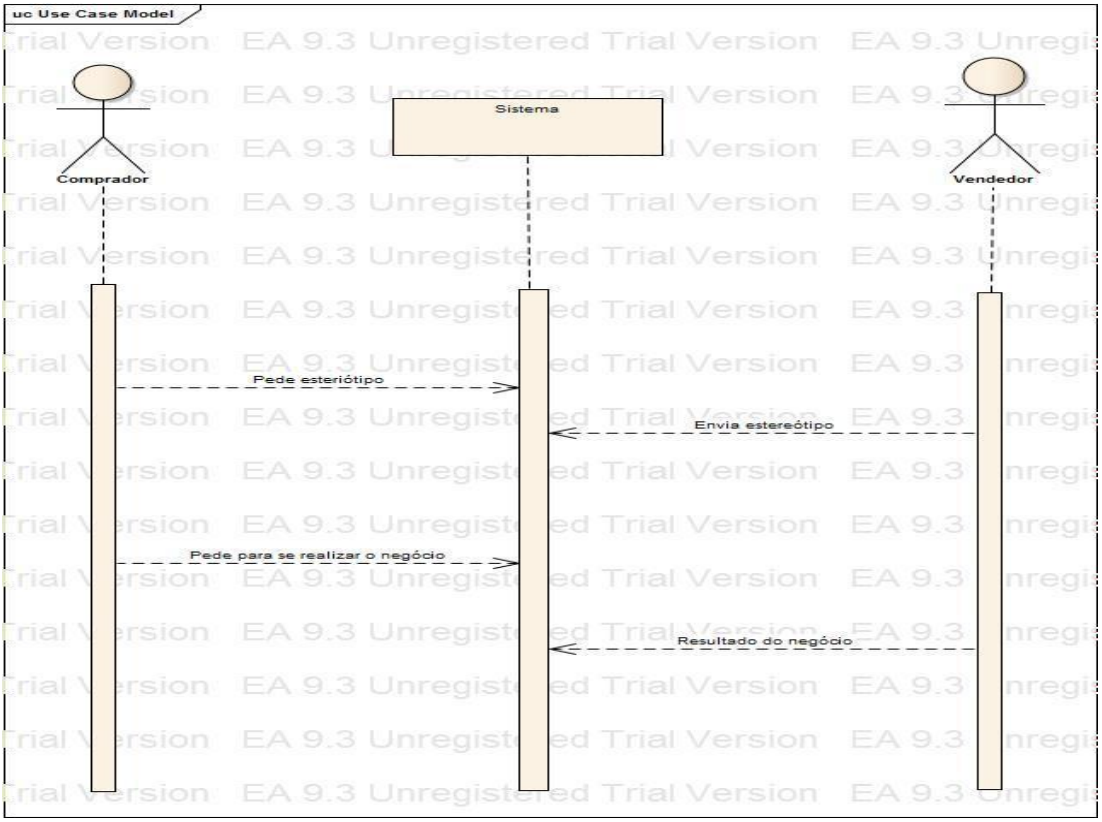
A interacção entre um comprador e um vendedor começa com um pedido do comprador que pretende obter um produto que irá ser fornecido pelo vendedor que satisfaça melhor as condições de compra. Para isso é executado o algoritmo *Contextual Fitness* que verifica os vendedores que possuem o produto especificado pelo comprador, se sim, e se o histórico de iterações passadas em casos similares se mostrarem ser de confiança o comprador irá considerar alguns vendedores para efectuar o negócio.

Após o comprador ter excluído os vendedores que não cumprem os critérios de venda que este deseja, de entre os que "sobraram" ele irá escolher um para efectuar o negócio. Para isso o comprador tem em consideração o preço do produto e a confiança que tem em relação ao tal vendedor. Nós usamos níveis de preferência para caracterizar cada caso e assim o

comprador consiga escolher o melhor vendedor. Caso, a confiança do vendedor seja superior a 0.9, ou seja muito alta, o comprador independentemente do preço escolhe esse vendedor, este é o nível 1 que é a primeira condição a ser verificada. O valor da confiança entre 0.6 e 0.9, corresponde ao nível 2. Se o comprador nunca tiver feito algum negócio com o vendedor o nível é o 3. Valor de confiança entre 0.4 e 0.6, corresponde ao nível 4 e finalmente, confianças inferiores a 0.4 correspondem ao nível 5 que é a ultima condição a ser verificada. Sempre que encontrar um vendedor com um nível de preferência num patamar mais elevado ele irá começar a ignorar todos os vendedores nos patamares inferiores.

Após o vendedor ter sido escolhido, é então efectuado o negócio entre os dois agentes, e a confiança do vendedor em relação ao comprador irá ser actualizada de acordo com o resultado desta, através do uso do algoritmo *Sinalpha*.

Além deste protocolo de iteração principal, existem outros 2 protocolos alternados para testar as várias diferenças entre as maneiras de escolher um vendedor, um protocolo idêntico ao principal mas sem contextual *fitness* e um protocolo em que irá escolher o vendedor apenas pelo preço.



Desenvolvimento

Plataforma/Ferramenta utilizada:

A plataforma que se utilizou para o desenvolvimento deste projecto é o *Repast* num ambiente Java. O *Repast* é uma plataforma open-source para o modelamento e simulação de agentes e suas interacções. Este disponibiliza uma série de serviços que permitem criar e modificar uma grande variedade de agentes e modelos de representação.

As suas principais características são:

- Enorme variedade de Agentes, Modelos e Bibliotecas;
- Orientada a objectos;
- Permite aceder e alterar dinamicamente os Agentes enquanto corre;
- Sistemas embutidos para a modelação dinâmica;
- Interface gráfica que permite analisar e estudar facilmente os resultados das simulações;
- Suporta uma grande variedade de linguagens;

Ambiente de desenvolvimento:

- **Linguagem de programação:** Java
- **Sistema Operativo:** Windows 7
- **IDE:** Eclipse
- **Plataforma utilizada:** Repast

Estrutura da aplicação:

O nosso trabalho encontra-se dividido em nove módulos: Comprador, Vendedor, Encomenda, ID3, Opiniao, Sinalpha, Model, MyDefaultDrawableNode e MyOpenSequenceGraph. De seguida, é feita uma descrição sobre cada um deles, assim como apresentamos o diagrama de classes do sistema.

Módulos

Módulo Comprador: implementa os objectos que representam os compradores na rede social. Para podermos representar graficamente um comprador tivemos de definir Comprador como subclasse da classe *OvalNetworkItem.*

Cada Comprador tem os seguintes membros-dado:

- **int id**: inteiro que permite a sua identificação;
- **Color color**: diz respeito à cor do seu nó na interface gráfica, neste caso, é o verde que é usado para simbolizar os compradores;
- **Vector<Opiniao> conf_vendedores**: contêm as opiniões que o comprador tem em relação aos vendedores com que interagiu;
- **Vector<Encomenda> lista_compras**: produtos que este pretende adquirir;

Convêm também referir os seguintes métodos:

- **void actualiza_opiniao(int pos, Opiniao opinion)** - actualiza uma opinião referente a um determinado vendedor através do conhecimento da sua posição no vector;
- **Encomenda SelectRandEncomenda() -** selecciona aleatoriamente uma encomenda da lista de compras do comprador e devolve-a;
- **Encomenda geraEncomenda() -** Gera uma encomenda dentro de certos parametros e adiciona-a à lista de compras;

Módulo Vendedor: implementa os objectos que representam os vendedores na rede social. Para podermos representar graficamente um vendedor tivemos de definir Vendedor como sub-classe da classe *OvalNetworkItem.*

Cada Vendedor tem os seguintes membros-dado:

- **int id:** inteiro que permite a sua identificação;
- **Vector<Encomenda> stock:** vector que contem os produtos que o vendedor possui em stock;
- **Vector<Integer> odds:** probabilidade de sucesso de uma entrega de cada encomenda em stock;
- **Vector<Encomenda> historico:** histórico de encomendas;
- **numero_sucesso:** inteiro com o número de negócios que tiveram sucesso;
- **numero_falhas**: inteiro com o número de negócios que falharam;

Convêm também referir os seguintes métodos:

- **void actualizaStock(int nova_quant,int pos):** actualiza a quantidade de uma determinada encomenda do stock do vendedor;
- **int generate_odds():** gera aleatoriamente a probabilidade de uma entrega ter sucesso que pode ter valores entre 1 e 100;

Módulo Encomenda: implementa os objectos que representam as encomendas que vão ser negociadas pelos compradores e vendedores na rede social.

Cada Encomenda tem os seguintes membros-dado:

- **double preco:** preço de uma encomenda;
- **int quantidade:** quantidade disponivel;
- **String nome:** nome do produto;
- **int resultado:** inteiro que indica se uma encomenda foi enviada com sucesso ou não. 0 se o estado da encomenda é neutro, 1 caso haja sucesso no envio, 2 caso falhe;

Módulo ID3: implementa o algoritmo ID3 que permite criar uma árvore de decisão que vai guardar o histórico dos negócios efectuados na rede social. De notar que esta classe foi definida com base noutro código que encontramos na Internet, o qual foi modificado e adaptado ao nosso programa.

Cada ID3 tem os seguintes membros-dado:

- **int numAttributes:** número de atributos a ter em conta para formar a árvore de decisão, no nosso caso, quatro atributos;
- **String []attributeNames**: array que contem os nomes dos atributos usados para formar o histórico: nome, quantidade, preço e resultado;
- **Vector []domains**: array de vectores constituído por quatro vectores um por cada atributo;

Convêm também referir os seguintes métodos:

- **double calculateEntropy(Vector data)** - calcula a entropia e retorna-a;
- **int readHistorico(Vector<Encomenda> teste)** - a partir de um vector de encomendas vai buscar a cada encomenda o valor dos seus atributos;
- **void printTree(TreeNode node, String tab)** - imprime a árvore de decisão;
- **Boolean ContextualFitness(TreeNode node, Encomenda enc)** - implementa o algoritmo do *Contextual Fitness;*

Módulo Opiniao: representa a opinião sobre um vendedor.

Cada Opiniao tem os seguintes membros-dado:

- **int id**: inteiro que identifica a Opiniao;
- **Sinalpha sinalpha**: objecto que possiblita o cálculo do valor da confiança;
- **double confianca**: valor de confiança de um comprador num vendedor;
- **boolean activado**: indica se já houve alguma interacção/negócio entre um determinado comprador e vendedor;

Convêm referir o seguinte método:

- **void actualiza_confianca(int sucesso) -** calcula o valor da confiança mediante o valor de *sucesso* e coloca a flag *activado* a TRUE, indicando que já houve pelo menos uma interação;

Módulo Sinalpha: implementa o modelo *Sinalpha* que permite o calcula da confiança.

Cada Sinalpha tem os seguintes membros-dado:

- **double alpha_ant**: variável que varia ao longo do tempo e permite calcular a confiança. α no início tem um valor de $3\pi/2$ e depois ao longo do tempo é dado por $\alpha = \alpha + \lambda.\omega$;

Convêm referir o seguinte método:

- **double calcula_confianca(int sucesso) -** este método calcula a confiança mediante o sucesso que houve num negócio, caso tenha havido sucesso na interacção (sucesso == 0) ou não (sucesso ==1);

Módulo MyDefaultDrawableNode: subclasse da classe *MyDefaultDrawableNode* que permite a representação de nós na interface gráfica. Foi necessário a criação desta subclasse para que pudéssemos distinguir os compradores e os vendedores da lista de agentes.

Cada MyDefaultDrawableNode tem os seguintes membros-dado:

- **int class_id**: inteiro que permite identificar se o agente é um Comprador (id == 1) ou um Vendedor (id == 2);
- **NetworkDrawable node**: objecto que vai permitir "desenhar" o nó;
- **Comprador comp:** caso queiramos representar graficamente um Comprador;
- **Vendedor vend**: caso queiramos representar graficamente um Vendedor;

Módulo MyOpenSequenceGraph: foi necessária a criação desta subclasse pois o nosso objectivo é criar um gráfico que mostrasse a relação confiança/tempo para cada comprador em relação aos vendedores, e não apenas um gráfico que avaliasse o sistema todo.

Cada MyOpenSequenceGraph tem os seguintes membros-dado:

- **int id**: inteiro que permite identificar o Comprador;
- **Vector<Opiniao> opinioes**: vector de opiniões, referentes aos vendedores, de um Comprador;
- **Vector<Integer> results**: vector de inteiros que vai guardar os valores de sucesso ou falha;

Módulo Model: este módulo contêm todos os métodos e funcionalidades que permitem executar o programa. Como herda os métodos da classe *SimModelImpl* é a partir desta classe que é executada a interface Repast.

Cada Model tem os seguintes membros-dado:

- **ArrayList<Node> agentList**: lista de agentes que vão constituir a rede social;
- **Schedule schedule**: objecto que permite o escalonamento das acções na interface gráfica;
- **DisplaySurface dsurf**: objecto que permite criar a janela onde está apresentada a rede social;
- **GraphLayout space**: objecto que permite adicionar á janela do programa o que pretendemos visualizar (agentes e as suas interacções);
- **int spaceSize**: tamanho da janela a apresentar aquando a execução do programa;
- **int TypeTest**: tipo de teste do sistema a ser executado;
- **int TypeGraph**: tipo de grafo a ser apresentado;
- **int Selected_test**: que teste foi seleccionado;
- **ArrayList<MyOpenSequenceGraph> graphsTrust**: array de gráficos que medem a confiança em função do tempo;

- **ArrayList<MyOpenSequenceGraph> graphsSucess**: array de gráficos que medem o sucesso em função do tempo;

Convêm também referir os seguintes métodos:

- **void buildModel()** - onde se criam os vendedores, compradores, encomendas, gráficos e os nós que vão ser adicionados á lista de agentes;
- **void buildDisplay() -** vai fazer o *display* dos gráficos e das janelas que devem ser apresentados na interface do sistema;
- **MyDefaultDrawableNode pesquisa_vendedor(int id) -** procura na lista de agentes um determinado vendedor através do seu *id* e retorna-o;
- **void buildSchedule() -** faz o escalonamento ao longo dos vários *tics* das acções a serem criadas e alteradas na interface gráfica do sistema;
- **Map<Boolean,Integer> HasMaterial(Encomenda order, Vendedor vend) -** verifica se um determinado vendedor possui no seu stock uma determinada encomenda requerida pelo comprador, se tem um preço menor ou igual e se tem o material em quantidade suficiente;
- **Vector<Integer> escolhe_vendedor(Comprador c1,Encomenda e1) -** mediante um comprador e o seu respectivo pedido (encomenda) determina qual o vendedor que deve ser escolhido, para melhor favorecer o comprador, através da confiança e do preço da encomenda. Retorna um vector de inteiros constituído pelo *id* do vendedor e pela posição da encomenda no stock do vendedor;

Classe MySeq: esta classe implementa a interface *Sequence* que é necessária à criação dos gráficos que são apresentados na interface *Repast.* Em vez de usarmos directamente *Sequence* foi necessário criar MySeq, que implementa os objectos que vão constituir os gráficos a ser apresentados que podem ser de confiança ou de sucesso.

Cada Model tem os seguintes membros-dado:

- **int j**: posição do vector necessária para ir buscar o valor de confiança ou sucesso consoante o gráfico que queremos construir;
- **MyOpenSequenceGraph graph1**: gráfico a ser criado. Pode ser um gráfico de confiança em função do tempo ou de sucesso em função do tempo;

Convêm referir o seguinte método:

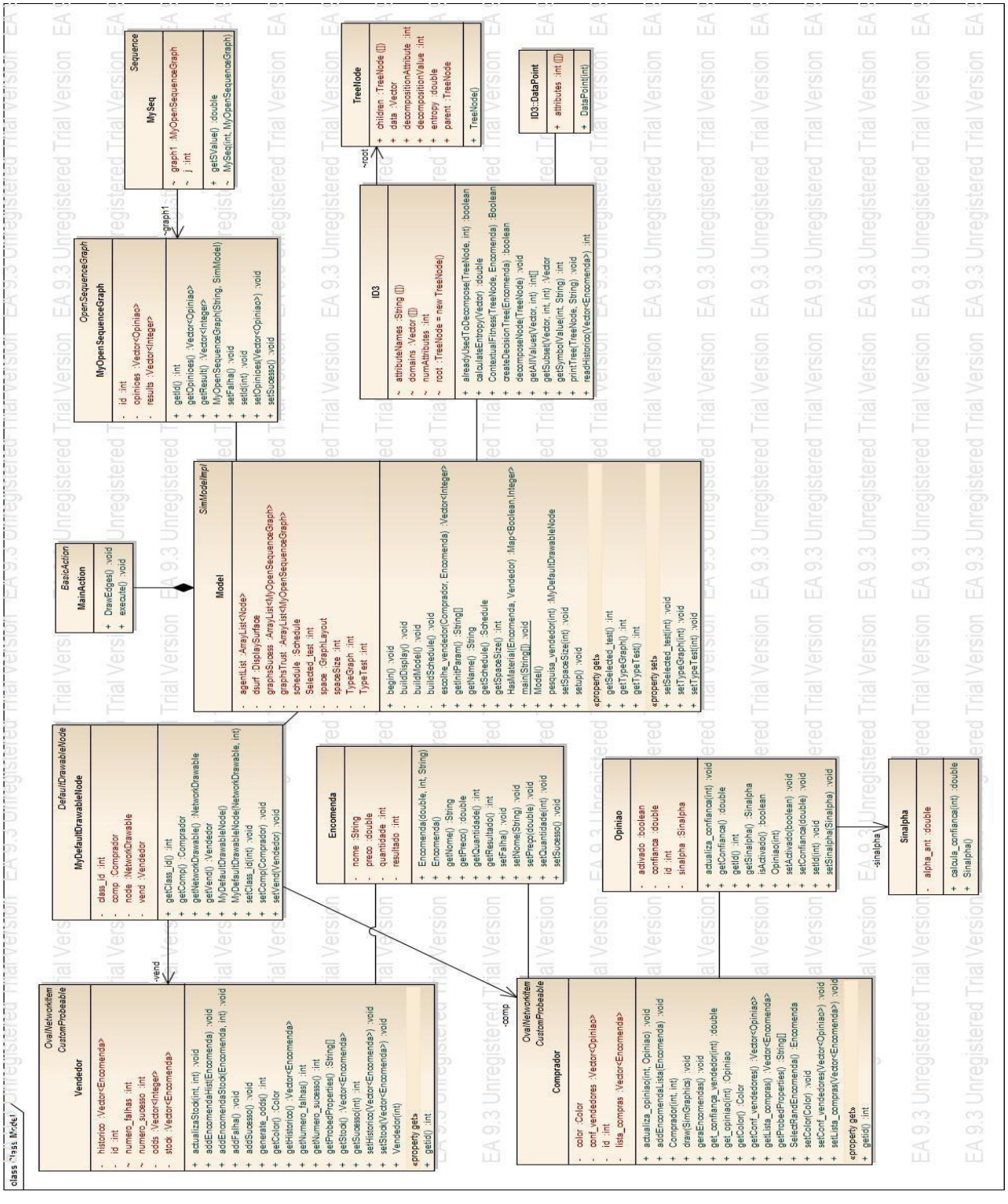
- **double getSValue() -** método que retorna o valor da variável que queremos que esteja presente no gráfico. Esse valor pode ser a confiança ou o valor de sucesso de envio de uma encomenda;

Classe MainAction: classe responsável pela execução do programa.

A classe MainAction tem os seguintes métodos:

- **void execute() -** método que executa o programa;
- **void DrawEdges() -** função que vai criar as *edges* que ligam os vários agentes da rede social;

Diagrama de classes



Detalhes relevantes da implementação

Todos os Vendedores começam com uma relação de confiança de 0.5 com os vendedores.

Na implementação do Sinalpha deste trabalho foi decidido que serão necessários 10 iterações para chegar a confiança máxima.

Após uma transacção ter sido realizada com sucesso, o material é removido do stock do vendedor é a encomenda removida da lista de compras do vendedor. Após todas as transacções terem sido realizadas ou não serem possíveis realizar, o programa irá correr indefinitivamente sem nenhuma modificação.

Sobre a rede social na interface gráfica os vendedores são representados com nós azuis, enquanto que os compradores tem a cor verde. As arestas que ligam os nós e representam as interacções entre os agentes e podem ter 3 cores consoante o nível de confiança que o comprador tem pelo vendedor. Caso, a confiança seja alta a aresta apresenta-se como verde, se a confiança entre esses dois agentes for intermédia a cor da aresta é amarela, para níveis de confiança baixos a aresta é vermelha.

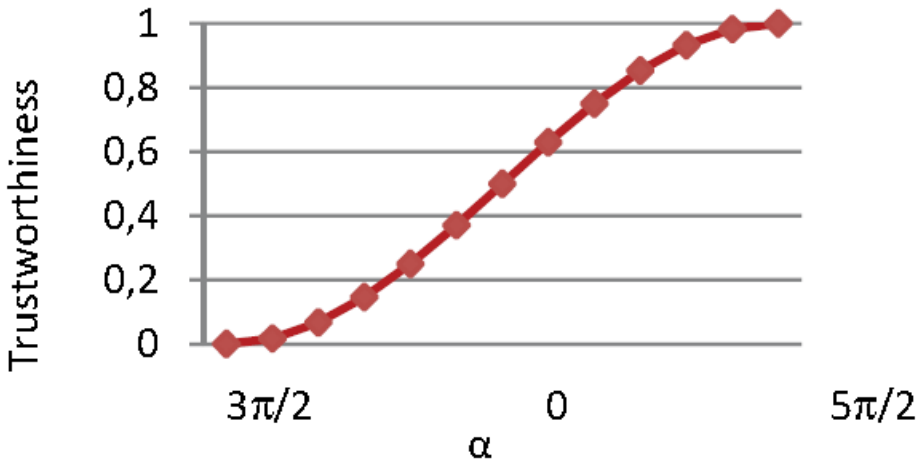
Adicionalmente à janela onde se encontra a rede social optamos também pelo programa criar gráficos por cada Comprador da rede social, de maneira a perceber a variação dos valores de confiança e do número de sucessos no envio de encomendas. Existem 2 tipos de gráficos, um que mostra a confiança do Comprador relativamente a todos os vendedores e outro que mostra a quantidade de iterações com sucesso e falhanço.

Modelo Sinalpha

O Sinalpha é um modelo que se baseia na agregação de vários factores que nos permitem medir a confiança, confiança essa que pode ser representada através de uma S-like curve. Este modelo baseia-se em três propriedades:

- **Assimetria** - determina que a confiança é uma coisa difícil de ganhar, no entanto, é fácil de perder. No nosso caso, um vendedor que envie sempre os produtos correctamente para o comprador vai ganhando a sua confiança pouco a pouco, no entanto, se houver uma vez que este não envia a compra, perde logo a maior parte da confiança que tinha com o comprador;

- **Maturidade** - a medição da fiabilidade de um agente pode ser diferente em várias fases do seu crescimento, em algumas o declive de crescimento é maior e noutras menor;
- **Distinguível** - distinção dos vários tipos de comportamentos do passado dos agentes;



Na figura anterior, podemos visualizar a S-like curve que pode ser dividida em 3 fases de ganho e perda de fiabilidade de um agente: criação de confiança (1º terço da curva), quando um utilizador novo começa a fazer parte da rede social, verificamos que o ganho de confiança no início é mais lento; obtenção de confiança (2º terço), o utilizador começa a obter, de uma forma mais rápida, a confiança dos outros utilizadores da rede social; e por fim tirar partido da confiança ganha até ao momento (3º terço), o utilizador já ganhou a confiança da rede social, nesta fase, a confiança do utilizador tem tendência para ir diminuindo de uma forma lenta.
A fórmula que nos dá o valor de confiança, em função de α, de um agente da rede social é a seguinte:

$$\mathbf{y(\alpha) = \delta.\sin \alpha + \delta}$$

Em que δ é uma constante de valor igual a 0.5, α é um valor que se pode situar entre 3π/2 e 5π/2, e que permite assim à fórmula principal retornar valores de confiança dos agentes entre 0 e 1. O α no início tem um valor de 3π/2 e depois ao longo do tempo é dado por $\alpha = \alpha + \lambda.\omega$. Em que ω é uma constante que representa o ritmo de ganho de confiança e λ é uma variável que pode ter o valor +1 se obtivermos uma evidência positiva de um agente, e -1.5, se pelo contrário, se verificar uma evidência negativa nas acções do agente. Verificamos que se ocorrer uma evidência negativa o λ é negativo, porque como vimos anteriormente, é mais fácil perder a confiança do que ganhá-la.

De uma maneira geral, o Sinalpha é usado após o contextual fitness para obter o valor actual de confiança dos agentes da rede social.

Contextual Fitness

O contextual fitness é um modelo CTR (Computational Trust and Reputation) que permite avaliar a *situational* awareness trust dos agentes, isto é, ele simula a situação em que um trustier agent conhece o historial do agente com que irá comunicar mas o trustier agent pretende realizar modificações subtis nesta comunicação o que introduz um factor de incerteza, a contextual ignorance.

Este modelo avalia a confiança através do uso de evidências passadas e a criação de um estereótipo a partir destas. Os estereótipos são comparados com as propostas existentes de forma a um valor de Contextual fitness ser derivado, esse valor é usado para se estimar uma confiança global.

Neste projecto é implementada uma versão alterada deste algoritmo em que os passos são idênticos excepto no ponto onde corre a derivação o valor de confiança do contextual fitness. Nesta etapa quando ocorre a comparação entre estereótipos e as propostas dependendo do resultado o trustier, pura e simplesmente, exclui, ou não, o agente trustee da lista de agentes com que pode comunicar em vez de gerar um valor.

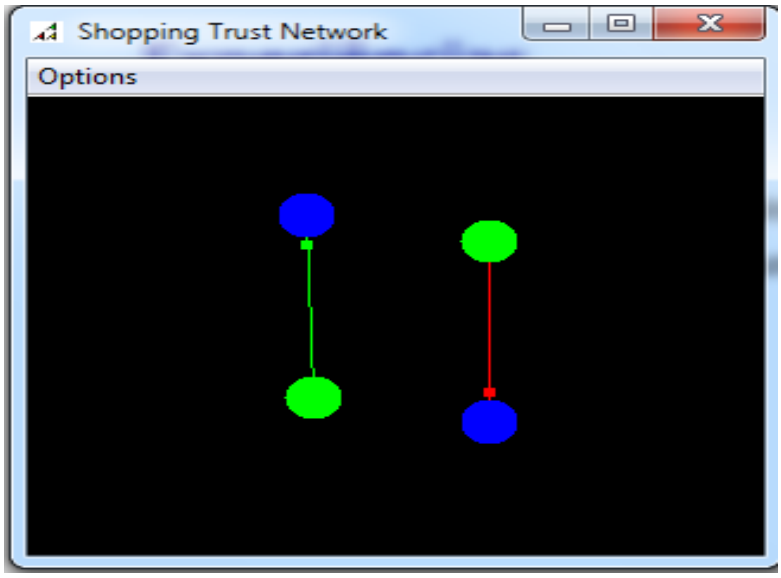
Experiências

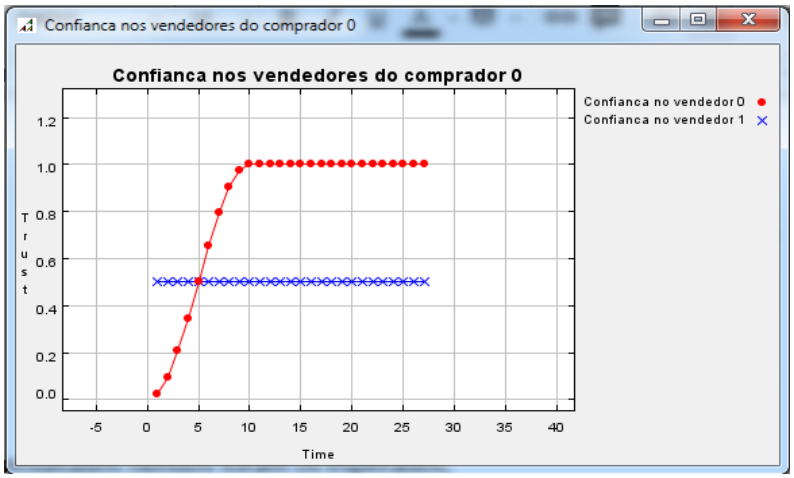
Todos os materiais desejados por os compradores estão divididos em 10 encomendas de forma a permitir que o Sinalpha chegue ao tecto de confiança.

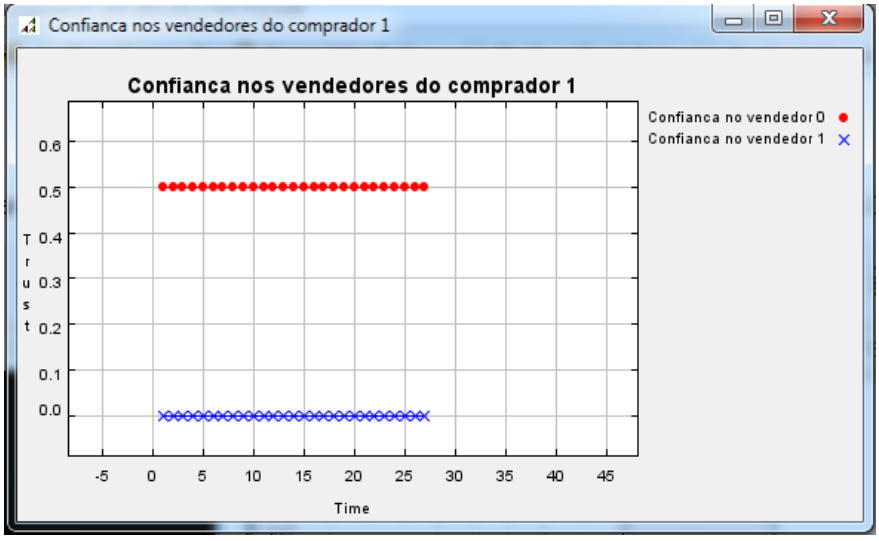
- **Experiencia 1:**

Teste com 2 Compradores e 2 Venderes. Cada vendedor deseja um material que existe no stock de um dos vendedores, um vendedor tem 100% de sucesso de realizar a transacção enquanto o outro 0%. Neste teste os compradores devem encontrar o vendedor com o material correspondente e actualizar a sua confiança de acordo com o resultado da transacção.

Resultados:


Shopping Trust Network
Options

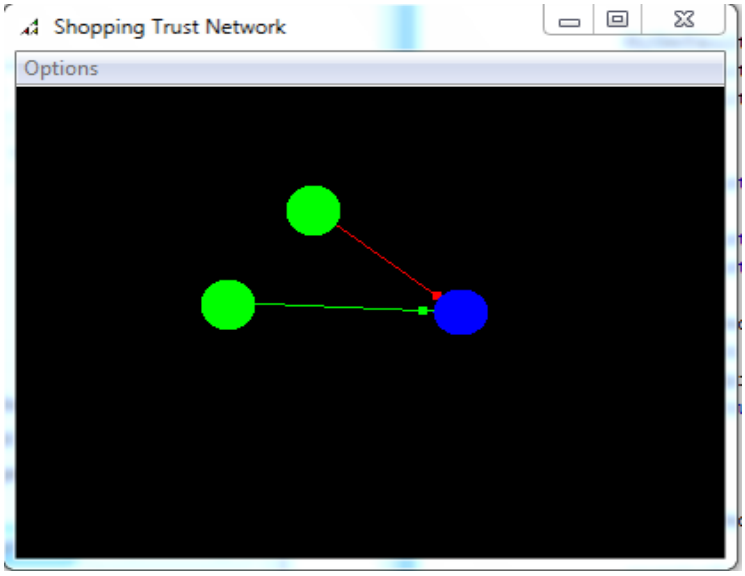

Confianca nos vendedores do comprador 0
Confianca nos vendedores do comprador 0
Confianca no vendedor 0
Confianca no vendedor 1
Trust
Time
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-5
0
5
10
15
20
25
30
35
40

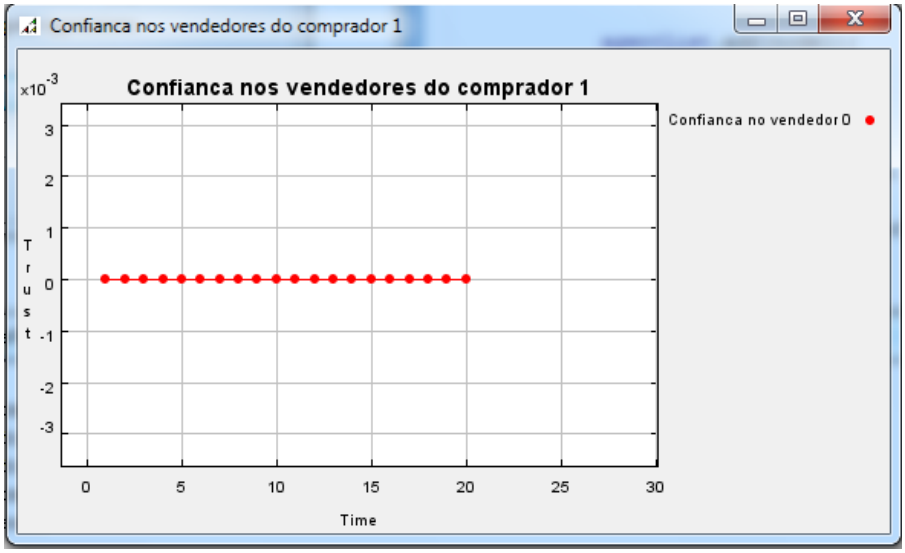

Confianca nos vendedores do comprador 1
Confianca nos vendedores do comprador 1
Confianca no vendedor 0
Confianca no vendedor 1
Trust
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0
-5
0
5
10
15
20
25
30
35
40
45

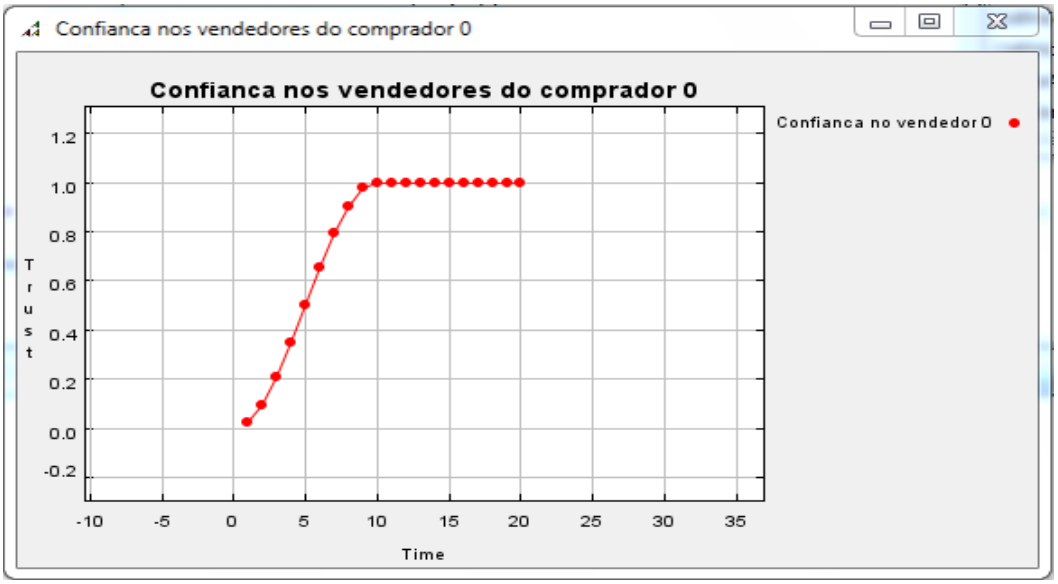
Os resultados obtidos foram os esperados, as transacções do Comprador 0 com Vendedor 0 tiveram sempre sucesso logo o seu valor de confiança aumentou até ao patar máximo, o oposto aconteceu entre o Comprador 1 e o Vendedor 1 em que as transacções falharam sempre, logo a confiança manteu-se a 0.

- **Experiencia 2:**

Existem 2 compradores, estes procuram materiais que existem no stock de um vendedor, para um desses materiais o vendedor tem 100% de realizar a transacção correctamente enquanto para o outro 0%. Esta experiência serve para mostrar 2 Compradores a interagirem com um único Vendedor mas numa situação em que os materiais têm probabilidades diferentes de sucesso.
Resultado:





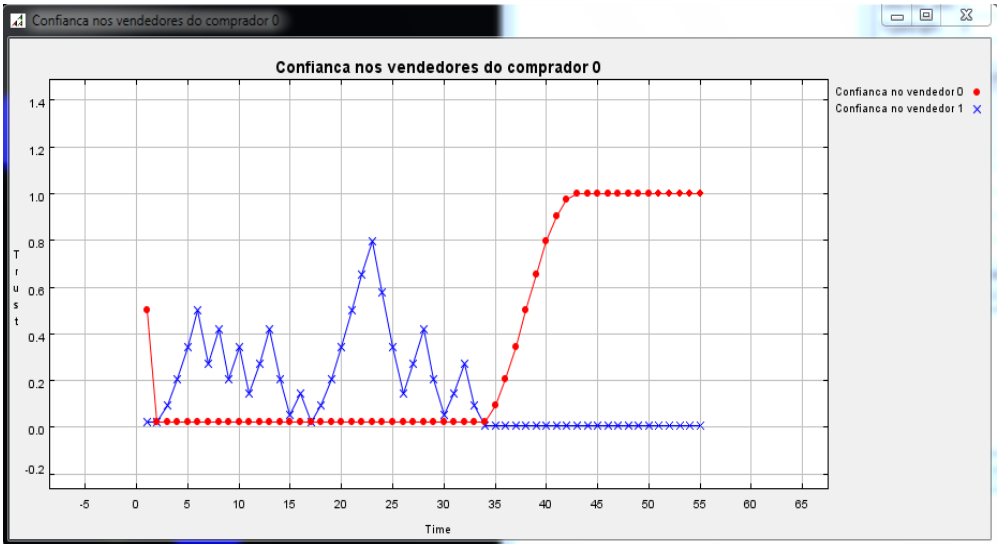


Como esperado, ambos os compradores formularam opiniões diferentes do mesmo vendedor devido as probabilidades de sucesso diferentes dos materiais.

- **Experiencia 3:**

Teste em que o comprador procura um material que existe em ambos os 2 vendedores. Num com 100% hipóteses de sucesso e noutro com 50%. Nesta experiencia ele deve tentar realizar transacções com ambos dando sempre preferência ao que têm mais confiança num dado momento, ao fim de várias iterações ele deve apenas contactar o vendedor com 100% de confiança. Foram corridos vários testes até ser encontrado um que melhor exemplifica o que nós estamos a tentar provar com esta experiencia.

Resultados:

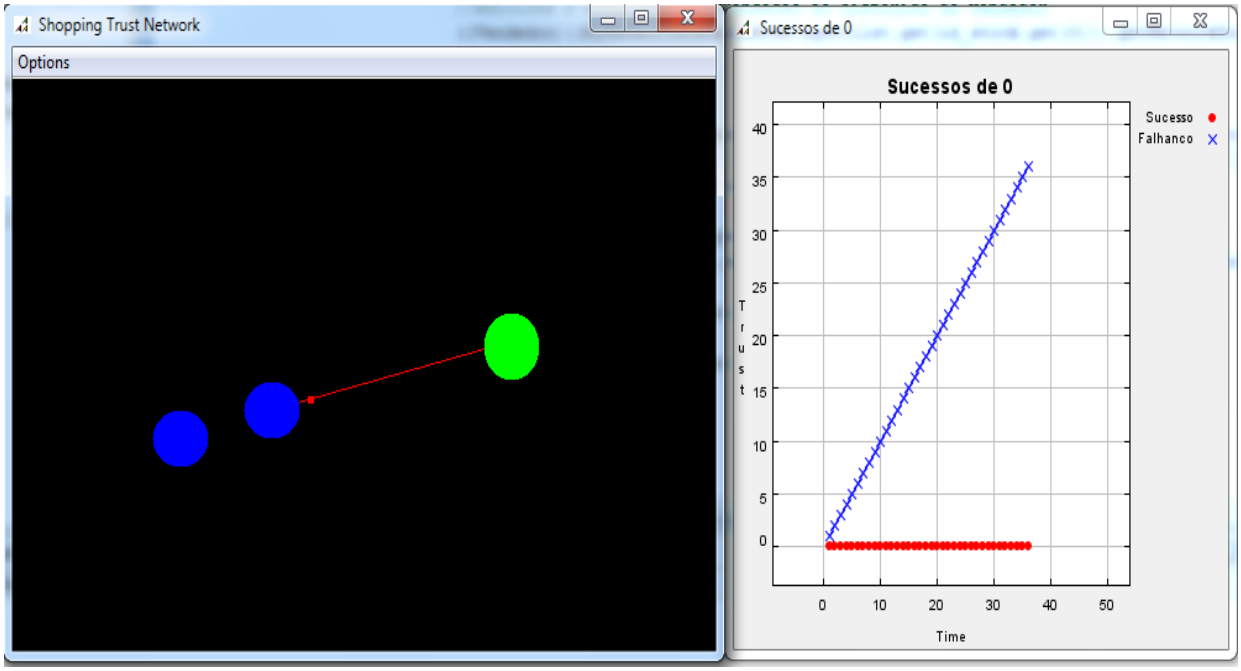


O resultado foi o esperado. Inicialmente ele seleccionou aleatoriamente o vendedor 1 com 50% de probabilidade, á medida que a confiança aumentava ele começou a escolher-lho exclusivamente. Mas ao longo que o tempo passa-se com o aumento nas transacções com falha a confiança foi variando até voltar a um valor muito baixo, fazendo com que escolhe-se o vendedor 0 com 100%, ai a sua confiança foi aumentando até ao valor tecto sem nenhuma interrupção.

Experiencia 4:

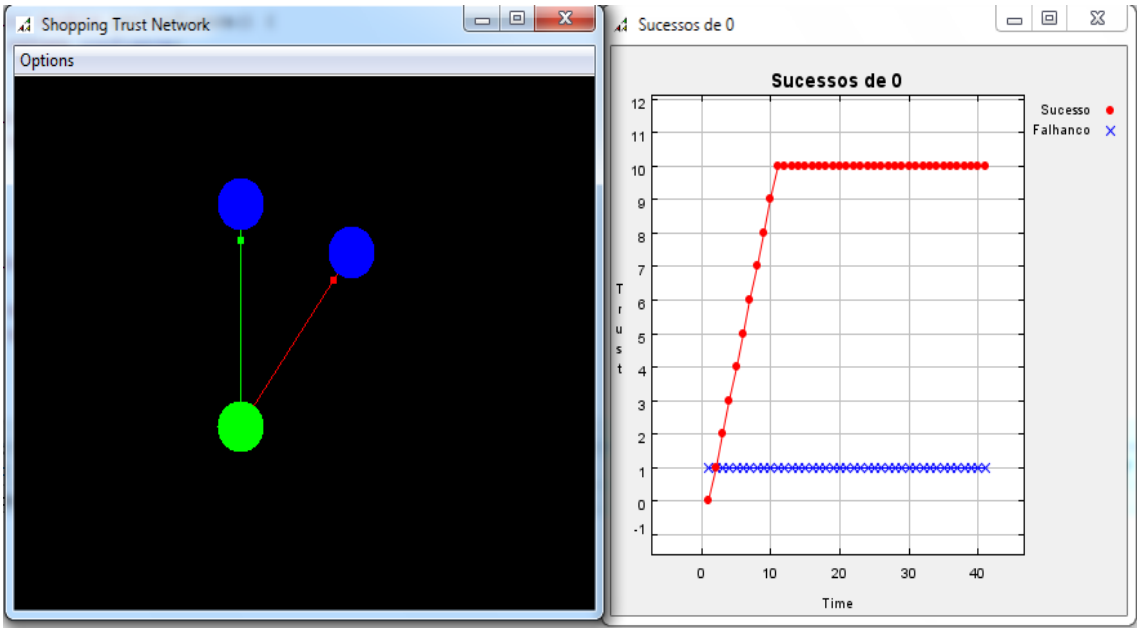
Teste com 1 comprador que procura um material em 2 vendedores. Um vendedor terá 100% hipóteses de sucesso e outro 0% mas com o material com um preço mais barato. Ambos os vendedores começam com um histórico considerável de transacções com esse material. O objectivo desta experiencia é mostrar como o *Sinalpha* + *contextual fitness* é superior á escolha com base apenas no *Sinalpha* e á escolha que têm apenas em consideração o preço menor.

Teste com base no preço menor:



Como se pode tendo em conta apenas o preço menor ele irá sempre comunicar com o vendedor que oferece o preço mais barato apesar de este falhar sempre na transacção

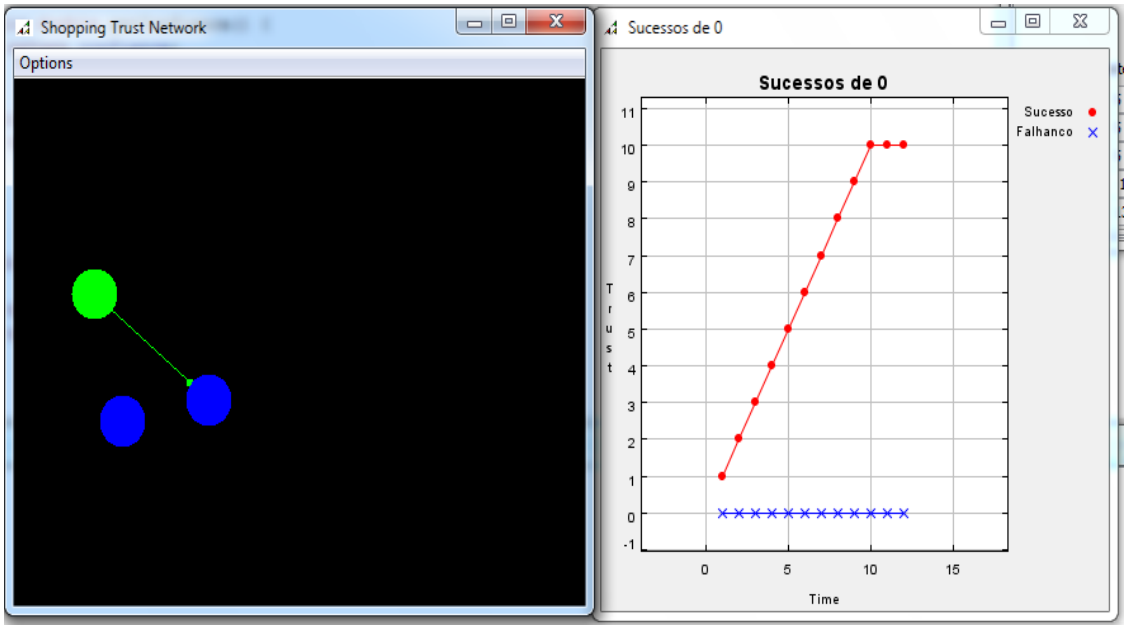
Teste com base apenas no *Sinalpha*:



Pode-se verificar que ele tentou entrar em contacto primeiro com o vendedor com o preço mais barato e falhou essa transacção. Ele de seguida calculou que este vendedor não era de confiança e tentou o outro vendedor mais caro. Neste vendedor todas as transacções para as

10 encomendas tiveram sucesso, provando assim que obviamente é importante o uso de confiança na comunicação numa rede social.

Teste com base no *Sinalpha + Contextual Fitness*:



Neste teste final como se pode ver o comprador não falhou uma única vez. Um estereótipo irá ser formado para cada vendedor partir do seu histórico. Através dos estereótipos o Comprador irá verificar que o Vendedor que vende mais barato falha em todas as transacções e por isso não irá sequer tentar realizar uma transacção com ele.

Logo podemos concluir que com o uso de *Sinalpha + Contextual Fitness* podemos obter transacções mais fiáveis do que nos outros casos anteriores.

Conclusões

Após uma análise do trabalho desenvolvido concluímos que conseguimos a maior parte os objectivos que nos propomos cumprir e com a excepção do bug referido na secção dos melhoramentos a rede de agentes está a funcionar como esperávamos.

Através dos resultados obtidos na fase de testes do sistema, tal como foi referido antes, podemos concluir que o uso do *Sinalpha* em conjunto com o *Contextual Fitness* permite ao comprador escolher sempre os vendedores mais fiáveis com o mínimo de transacções falhadas, o que faz disto um método ideal para ser usado numa rede social de agentes onde a confiança influencia decisões.

Durante o desenvolvimento do trabalho não sentimos muitas dificuldades em relação à parte lógica do programa. Por outro lado, o que nos dificultou mais o avanço do trabalho foi mesmo a falta de experiencia e falta se informação sobre o uso do *Repast,* o que nos fez perder muito tempo na implementação da interface do sistema.

Melhoramentos

- Melhorar a interface gráfica de maneira a que o utilizador do programa pudesse interagir de uma forma mais intuitiva com o sistema;

- Existe um bug que não foi corregido a tempo, devido a uma falha no nosso planeamento, em que caso aja desde o início varias falhas consecutivas num agente com uma encomenda com uma probabilidade de falha o *Contextual Fitness* poderá indicar que esse agente irá sempre falhar para essa situação quando na realidade pode suceder.

- Maneira do utilizador criar vários casos de testes sem ter de modificar o código directamente.

Recursos

Bibliografia

http://paginas.fe.up.pt/~niadr/PUBLICATIONS/2010/ams10-103-final.pdf
http://repositorio-aberto.up.pt/bitstream/10216/15129/2/18662.pdf
http://repast.sourceforge.net/repast_3/how-to/network.html
http://repast.sourceforge.net/repast_3/how-to/chart.html

Software

- Eclipse Indigo
- Repast

Percentagem de Trabalho Realizado por cada Elemento do Grupo

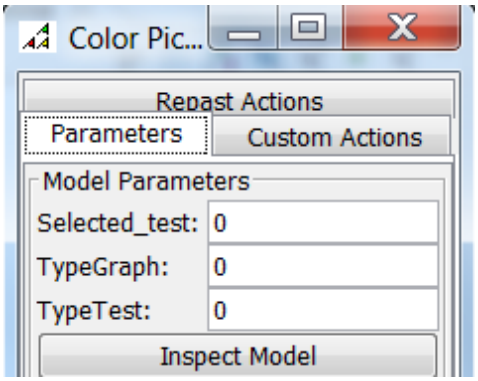
- **Gonçalo Pereira** - 50%
- **José Pereira** - 50%

Apêndice

Manual de Utilizador

O utilizador ao iniciar o programa depara-se com 3 variáveis que pode modificar, sendo estas:

- Selected_Test: Qual dos testes pre-feitos deseja correr.
- TypeGraph: O tipo de gráfico que deseja que seja apresentado
- TypeTest: Que tipo de protocolo de comunicação que deseja que corra



O Selected_Test tem 5 testes possíveis:

- 0 : Identico à Experiencia 1
- 1 :Identico à Experiencia 2
- 2: Identico à Experiencia 3
- 3: Identico à Experiencia 4
- 4: Teste com 2 compradores e 2 vendedores. Um dos compradores procura 2 materiais que existem num dos dos vendedores com 100%e e 50% de sucesso. E o outro comprador procura um material que existe no vendedor que resta com 0% de hipóteses. Neste caso em vez de uma única ordem cada comprador começará com centenas de ordens idênticas para mostrar a variação de confiança ao longo de um período elevado de tempo.
- 5: Existe um vendedor e 5 compradores. O vendedor irá sempre gerar encomendas novas de materiais existentes nos vendedores, tendo estes diferentes probabilidades de sucesso. O bug referido nos melhoramentos aparece aqui para o teste com *Contextual Fitness*

O TypeGraph tem 2 gráficos possíveis:

- 0: Gráfico da variação da confiança do comprador com os vários vendedores ao longo do tempo.
- 1:Grafico com os resultados das transacções ao longo do tempo de um Comprador.

O TypeTest tem 3 protocolos possíveis

- 0: Teste com confiança (Sinalpha) e *Contextual fitness*
- 1:Teste com confiança(Sinalpha)
- 2:Teste em que o comprador selecciona os vendedores apenas com o melhor preço em consideração.

Infelizmente, não foi possível criar a tempo uma maneira do utilizador inserir os seus próprios agentes e compradores. Portanto foi decido mostrar como se criar o código correcto no mybuildModel() de forma a criar uma rede para futuros testes:

Criar vendedor:

Vendedor v1 = new Vendedor(id);

O valor do id tem que ser sempre o valor do anterior +1 para o código correctamente.

Adicionar encomenda ao stock:

v1.addEncomendaStock(new Encomenda(Preço,Quantidade,Material),Probabilidade_de_sucesso);

Comprador:

Comprador c1 = new Comprador(indice,numero_de_vendedores_no_sistema);

Adicionar Encomenda a lista de comprar:

c1.addEncomendaLista(new Encomenda(Preço,Quantidade,Material));

Criar nó no grafico

MyDefaultDrawableNode node1 = new MyDefaultDrawableNode(Comprador/Vendedor,Tipo);
agentList.add(node1);

Tipo é 1 se for comprador e 2 se for vendedor.

Para criar os graficos de confiança para comprador é necessario:

MyOpenSequenceGraph g1 = new MyOpenSequenceGraph("Confianca nos vendedores do comprador "+comprador.getid(), this);
g1.setId(comprador.getid());
g1.setOpinioes(comprador.getConf_vendedores());
graphsTrust.add(comprador);

Para criar os gráficos dos resultados para cada comprador é necessário:

```
MyOpenSequenceGraph g1s =new MyOpenSequenceGraph("Sucessos de "+comprador.getid(), this);
graphsSucess.add(comprador);
```