Mercado Imobiliário - Relatório

Mestrado em Informática
Inês de Matos, 43538

1 INTRODUÇÃO

Neste relatório vai ser descrito o projeto: Mercado Imobiliário. Este projeto foi desenvolvido em Java sobre a plataforma JADE, de forma a criar um sistema multi-agente com diferentes tipos de agentes: os compradores, os vendedores e o mercado. Cada um deles irá ser descrito, tal como a comunicação feita entre eles, a estrutura do projeto e o mecanismo de distribuição de imóveis.

2 ESTRUTURA DO PROJETO

A estrutura do projeto divide-se em 5 classes e 1 ficheiro *.txt*. As classes correspondem à implementação dos agentes e de outras entidades auxiliares, para facilitar o processo de venda entre entidades.

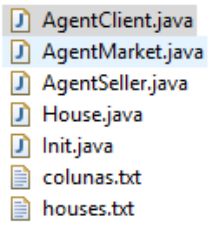

Figure 1: Estrutura do projeto em classes

2.1 CLIENTE

O agente comprador é representado pelo AgentClient. Quando este é executado é definido um perfil de compra do cliente, em que se tem em consideração vários fatores:

- o valor da casa
- a localização
- a tipologia
- o ano de construção da casa
- um lugar de estacionamento privado
- a presença de uma zona comercial
- os meios de transporte públicos,
- se a casa está equipada,
- a eficiência energética e o valor do IMI

Após definir os parâmetros mencionados acima, através de um comportamento cíclico, o cliente envia uma proposta aos vendedores existentes (neste caso 2) e aguarda a resposta de uma casa (um objeto do tipo House). Na receção o cliente vai avaliar a proposta recebida e enviar uma resposta, ao vendedor

correspondente, se aceita ou se rejeita. No caso de rejeição o cliente envia um código com a descrição da mesma, interpretável pelo vendedor.

2.2 VENDEDOR

O agente vendedor é representado pela classe AgentSeller e é constituído por dois comportamentos que correm em paralelo:

```
ParallelBehaviour parallel = new ParallelBehaviour(this, ParallelBehaviour.WHEN_ANY);

parallel.addSubBehaviour(new GuardaImoveis());
parallel.addSubBehaviour(new Negotiate());

addBehaviour(parallel);
```

Figure 2: Comportamentos do Agente Vendedor

O comportamento **GuardaImoveis()** é um comportamento cíclico, no qual o vendedor fica à espera que o agente mercado envie novas atualizações de casas. As casas são guardadas numa lista.

O comportamento **Negotiate()** é também um comportamento cíclico, embora este trata de negociar com os clientes qual a melhor casa, conforme os requisitos dos mesmos. Quando o vendedor recebe um requisito ou uma resposta, esta é avaliada de forma a satisfazer o cliente. Sempre que uma casa é vendida o vendedor envia uma mensagem ao Mercado para atualizar os dados.

2.3 MERCADO

O agente vendedor, representado pelo AgentMarket, também é constituído por dois comportamentos que correm em paralelo:

```
ParallelBehaviour parallel = new ParallelBehaviour(this, ParallelBehaviour.WHEN_ALL);

parallel.addSubBehaviour(new SacaImoveis(this,10000));
parallel.addSubBehaviour(new Vendida());

addBehaviour(parallel);
```

Figure 3: Comportamentos do Agente Mercado

O comportamento **SacaImoveis()** é um TickerBehaviour no qual de 10 em 10 segundos vai verificando atualizações no ficheiro, sempre que existe uma alteração, o AgentMarket distribui pelos vendedores. Esta distribuição é feita pelo valor do ID da entrada de uma casa, ou seja, os IDs pares vão para um vendedor e os ímpares para outro.

O comportamento **Vendida()** é um *CyclicBehaviour* no qual recebe mensagens do vendedor sempre que uma casa é vendida, de forma a alterar a disponibilidade daquela casa.

2.3.1 Mecanismo de distribuição

O mecanismo para distribuir regularmente os imóveis é feito pelo comportamento **SacaImoveis()**. Como já foi referido, de X em X tempo ele verifica no ficheiro *houses.txt* se existe alguma nova casa, caso exista, verifica se o ID desta é par ou impar de forma a distribuir pelos vendedores. Sempre que uma nova casa é vendida, o agente Mercado altera o ficheiro na linha respetiva do ficheiro *.txt*, na coluna de "Venda", passando de 0 para 1.

O ficheiro *houses.txt* segue a seguinte estrutura:

> *id;valor;localizacao;tipologia;area;ano;estacionamento;transportes;zcomercial;eficiencia ;equipada;imi;modocompra;venda*

Exemplo de uma entrada:

> *3;30000;Amadora;0;33;1988;0;1;0;C;0;75;1;0*

2.4 CLASSES AUXILIARES

2.4.1 Init

O agente de arranque do sistema é representado pela classe Init, no qual atualmente está preparada para executar os agentes em diferentes plataformas. A ideia seria simular a presença de vários agentes numa plataforma volátil, aquando os agentes *Market* e *Sellers* poderiam estar na mesma plataforma, mas em *containers* diferentes, mantendo uma certa permanência. Aqui também é onde se define os requisitos do cliente, de modo a fornecer independência para a geração de clientes, isto é, para não ser a própria classe do cliente a fazê-lo.

```
String valor = "0:150000:20";
List<String> locais = new ArrayList<String>();
locais.add("Telheiras");
locais.add("Parque das Nacoes");
locais.add("Lisboa");
locais.add("Odivelas");
String tipo = "1:0";
String ano = "-1";
int transportes = -1;
int estacionamento = -1;
int zcomercial = 0;
int equipada = 0;
String eficiencia = "C";
int imi = -1;
```

Figure 4: Requisitos de um cliente

Na Figure 4 estão indicadas as variáveis que um cliente considera.

1. **valor** – a primeira posição representa o valor inicial a que o cliente está disposto a comprar, a segunda representa o valor máximo e a terceira a tolerância sobre o valor máximo
2. **locais** – são guardadas as localidades desejadas pelo cliente, em que a primeira posição é a mais desejada e a última a menos

3. **tipo** – descreve quais são as tipologias requeridas pelo cliente, sendo a primeira posição a mais desejada e a último a menos.
4. **ano** – guarda o ano de construção da casa
5. **transportes**, **estacionamento**, **zcomercial**, **equipada** – cada variável guarda se o cliente quer ou não a característica, 0(zero) para ``não quer", 1 para ``quer" e -1 para indiferente
6. **eficiencia** – guarda a eficiência energética mínima requerida
7. **imi** – valor guardado do imposto anual, sendo o valor o máximo que o cliente quer pagar ou -1 para indiferente.

2.4.2 House

Esta classe representa uma casa, por cada objeto, de forma a ajudar a facilitar o processo de transações. Assim, em vez de enviar uma mensagem complexa na qual teria de ser depois digerida, o uso de um objeto representativo diminui o *workflow* de cada agente.

```
int id;
int valor; // valor
String local; //Local
String tipo; //"0-6"
String ano; // "1999"
int transportes; // 1 existe, 0 nÃ£o
int estacionamento; // 1 existe, 0 nÃ£o
int zcomercial;  // 1 existe, 0 nÃ£o
int equipada;  // 1 existe, 0 nÃ£o
String eficiencia; // "A", "B"
int imi;  //valor do imi
boolean vendida;
```

Figure 5: Variáveis de uma casa

3 COMUNICAÇÃO

Para a comunicação entre os agentes, foi usada a estrutura ACLMessage da FIPA.

3.1 CLIENTE

Recebe

- *ACLMessage.PROPOSE* do Vendedor sempre que há uma proposta

Envia

- ACLMessage.ACCEPT_PROPOSAL para o Vendedor se aceita a proposta
- ACLMessage.REJECT_PROPOSAL para o Vendedor se rejeita a proposta
- ACLMessage.CANCEL para o Vendedor se não quer negociar mais

3.2 VENDEDOR

Recebe

- ACLMessage.ACCEPT_PROPOSAL do Cliente se este aceita a proposta
- ACLMessage.REJECT_PROPOSAL do Cliente se este rejeita a proposta

- ACLMessage.CANCEL do Cliente se este não quer negociar mais
- ACLMessage.CFP do Mercado se este envia novas casas
-

Envia

- *ACLMessage.PROPOSE* para o Cliente sempre que tem há uma proposta
- ACLMessage.INFORM para o Mercado sempre que há uma casa vendida

3.3 MERCADO

Recebe

- ACLMessage.INFORM do Vendedor sempre que uma casa for vendida

Envia

- *ACLMessage.CFP* para o Vendedor sempre que existem novas casas para distribuir

4 NEGÓCIO

A realização do negócio entre os agentes foi a tarefa mais difícil devido à quantidade de variáveis existentes. Foi necessário reduzir a quantidade de combinações possíveis, de forma a ter um produto final com conteúdo funcional.

4.1 CLIENTE

Na parte dos clientes, existe um método – **avaliaProposta()** - que traça o perfil do cliente, isto é, conforme a casa recebida o cliente avalia consoante as variáveis que mais lhe convém.
Por exemplo, seguindo a Figure 4, o perfil que foi implementado considera as casas com um valor máximo de 20% do valor original, se a tipologia for pelo menos a segunda considerada e caso a localização da casa esteja entre as duas primeiras escolhidas pelo cliente.

4.2 VENDEDOR

No lado dos vendedores, o método **getAnHouse()** obtém uma casa, na qual é enviada para o cliente. Da primeira vez que o cliente comunica com o vendedor, o critério da primeira casa é simplesmente enviar a primeira em que o valor seja igual ou inferior ao valor pedido pelo cliente. A partir dai existe uma troca de mensagens entre o cliente e o vendedor, nas quais o vendedor tenta mitigar.

4.3 CÓDIGO DE COMUNICAÇÃO

No entanto, para que ambos os tipos de agentes se pudessem entender, a maneira mais fácil foi criar um código no qual representa uma série de respostas, este código também poderá ser combinado afim de criar mais precisão no entendimento da rejeição do cliente.

Table 1: Códigos

Código	*Representação*
0	Aceita a proposta
1	Rejeita o Valor
2	Rejeita a Localização
3	Rejeita a Tipologia
4	Rejeita o Ano
5	Rejeita a falta de Transportes
6	Rejeita a falte de Estacionamento
7	Rejeita a falta de Zona Comercial
8	Rejeita a falta de Equipamento
9	Rejeita a Eficiência
10	Rejeita o valor do IMI

Como foi dito mais acima, também existe a possibilidade de definir combinações para especificar as razões do cliente. Por exemplo se o cliente enviar um código ``02", significa que o cliente aceita se o vendedor alterar a localização para uma melhor; se o cliente enviar um ``23" significa que rejeita pela localização e pela tipologia. A inserção do``0" no código é importante porque significa que o cliente está mais próximo de aceitar a proposta. Caso o vendedor não tenha nenhuma casa que se adequo aos requisitos do cliente, este tenta reduzir o valor da última casa em X porcento (atualmente está em 10%)

5 CONCLUSÃO E TRABALHO FUTURO

Dada a complexidade do projeto, nem todas as funcionalidades foram possíveis serem feitas, isto é, havia uma expectativa de criar vários tipos de perfis tanto nos clientes como nos vendedores. Como também já tinha sido dito anteriormente no relatório, foi necessário reduzir o número de combinações entre variáveis, afim de se conseguir algum funcionamento base de como seria suposto o sistema funcionar. Para um sistema minucioso e mais completo, deveria de se ter em conta todas as variáveis e combinações, possibilitando assim vários tipos de interação distintos. Por exemplo, ter um cliente que olhe apenas para as caraterísticas ambientais da casa ou um cliente no qual não lhe interessa os valores, só a grandiosidade do imóvel.

Para trabalho futuro, seria então necessário remediar o assunto descrito acima, de forma a criar mais dinâmica de perfis e interações entre agentes. Outro ponto também a ser alterado, seria substituir o ficheiro *.txt* por uma base de dados convencional, que guardasse as casas, isto também tornaria o sistema mais eficiente e seguro.