Gestão de Elevadores

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Agentes e Inteligência Artificial Distribuída
4º Ano 1º Semestre

Grupo T01_3:
Luís Barbosa - 201405729 - up201405729@fe.up.pt
Paulo Santos - 201403745 - up201403745@fe.up.pt
Sérgio Almeida - 201403074 - up201403074@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

10 de dezembro de 2017

Conteúdo

1 Objetivo

1.1 Descrição do cenário

O cenário do presente trabalho consiste num edifício de vários andares com diversos elevadores. Cada elevador possui uma carga máxima, que corresponde ao número máximo de pessoas que pode transportar.

Os elevadores comunicam entre si informação relevante. Quando há um novo pedido numa determinada zona do edifício, este deve ser alocado a um dos elevadores existentes naquela zona. Os elevadores possuem uma lista de pisos em que têm que parar, podendo a mesma ser alterada dinamicamente para se incluirem novos pedidos.

Periodicamente, os elevadores podem partilhar informação sobre os seus estados, no sentido de poderem atribuir tarefas a outros elevadores.

O programa deve permitir que o utilizador configure o sistema, como o número de pisos do edifício, número de elevadores e a carga máxima de cada elevador.

As necessidades dos utentes, como por exemplo a frequência de chamada de elevador em cada piso do edifício (devendo ser superior para o piso 0) e piso de destino, são geradas aleatoriamente.

Um elevador demora um tempo pré-definido a ir de um piso a outro, sendo que o intervalo de tempo correspondente à paragem do elevador para entrada/saída de utentes também é contabilizado.

1.2 Objetivos do trabalho

O trabalho tem como principal objetivo o desenvolvimento de um sistema multi-agente para a gestão eficiente dos elevadores de um determinado edifício, tendo em conta o cenário descrito acima. Para tal, será necessária a implementação de um algoritmo que permita gerir as chamadas aos elevadores de forma a que o transporte seja feito o mais rápido possível.

2 Especificação

2.1 Identificação e caracterização dos agentes

No nosso projeto, distinguem-se dois tipos de agentes: *Elevator* e *Building*.

2.1.1 *Building*

O *Building* é instanciado apenas uma vez, assim que o programa é iniciado. Regista-se no *DFService*, criando um *DFAgentDescription*, adicionando a este um novo serviço com tipo "Building". Este agente é responsável por gerar elevadores e simular os pedidos existentes num edifício por parte dos utilizadores ao alocar aos elevadores os pedidos. Tem a seguinte **arquitetura**:

- **properties** - informações sobre o número de andares e elevadores, se existe teclado para efetuar pedido e se a negociação entre elevadores é permitida
- **elevatorsProperties** - informações sobre o peso máximo, tempo de movimento e tempo de entrada/saída de pessoas
- **elevators** - lista com o AID de todos os elevadores
- **reqGenInterval** - valor do intervalo de geração de pedidos pelo edifício
- **randNumRequestsPerInterval** - valor aleatório do número de pedidos gerados por intervalo, caso o valor lido do ficheiro seja "rand"
- **numRequestsPerInterval** - valor lido do número de pedidos gerados por intervalo

Todas estas informações são lidas do ficheiro de configuração **default.properties**, de forma a tornar o processo de configuração do cenário mais generalizado.

Este agente tem o seguinte **comportamento**:

- Cria o número de pedidos definidos, definindo inicialmente o andar inicial, sendo que aqui se assumiu que o andar final era o mesmo que o inicial.
- Cada pedido é atribuído a um elevador aleatório, verificando se esse elevador possui teclado. Caso possua, o andar de destino é atualizado uma vez que já era conhecido. Caso contrário, mantém-se a assunção inicial.
- De seguida, cria-se uma *ACLMessage* do tipo *CFP* com o seguinte conteúdo: **Andar Inicial, Andar Destino, Tempo para o Andar Inicial**. Esta é enviada pelo *Building* para o **Elevator** escolhido anteriormente, utilizando o protocolo de interação *FIPA Contract Net*.

2.1.2 *Elevator*

O *Elevator* regista-se no serviço do *Directory Facilitator* com o tipo "Elevator", de forma a que o *Building* (e outros que subscrevam o *Directory Facilitator*) saiba que este existe. Tem a seguinte **arquitetura**:

- **properties** - informações sobre o peso máximo, tempo de movimento, tempo de entrada/saída de pessoas, ter/não ter teclado
- **state** - informações sobre o piso atual, peso atual, estado de movimento ("STOPPED", "GOING_UP"ou "GOING_DOWN") e nº de pessoas

- **internalRequests** - lista interna de pedidos a realizar, que é dinâmica, uma vez que pode ser alterada caso outros elevadores lhe atribuam um novo pedido, ou caso seja feito um novo pedido no *Building*
- **statistics** - estatísticas utilizadas na interface
- **startupTime** - tempo de inicialização
- **numElevators** - o elevador sabe qual elevadores existem no total

Este agente tem o seguinte **comportamento**:

Caso a negociação entre elevadores seja permitida, tenta negociar com outros elevadores os pedidos internos, através do envio de uma *ACLMessage* do tipo *CFP*, utilizando o protocolo de interação *FIPA Contract Net*. Os outros elevadores são adicionados como recetores da mensagem. Para cada pedido que ainda não tenha sido atendido, é procurado o pedido que vai ser mais demorado a executar, sendo esse o *initiator* do *Contract Net Behaviour*.

É feita a verificação da existência de algum pedido para o piso atual do elevador, e se ainda não foi atendido. Caso isto se verifique, o elevador para e é gerado um novo peso que não exceda a capacidade do mesmo. Este peso pretende simular a entrada de uma ou mais pessoas com ou sem pesos. Existem casos em que é gerado o valor 0 de forma a aproximar a uma situação real, pois por vezes não é possível satisfazer o pedido pois o elevador já se encontra lotado. Durante o tempo de entrada de pessoa, é assegurado que não se atendem outros pedidos.

A forma de o elevador decidir para que andar se deve dirigir tem por base o andar mais próximo do atual. É verificado todo o conjunto de pedidos do elevador, isto é, caso já tenha sido um pedido atendido, verifica-se qual é o andar de destino mais próximo. Caso seja um pedido externi, verifica-se qual é o andar de partida mais próximo.

O algoritmo de decisão tenta, em todos os andares, otimizar a distância ao próximo andar. Desta forma, garantimos que não ande um grande número de andares sem parar quando era mais eficiente ter parado a meio para atender mais pedidos.

Verifica, também, para todos os pedidos atendidos do elevador, se estes já se encontram no andar de destino. Quanto isto acontece, o elevador para e é simulada a saída da pessoa que tinha feito o pedido.

2.2 Estratégias e processos de raciocínio

De forma a comparar o desempenho dos elevadores com diferentes heurísticas, foram implementadas duas estratégias de alocação de pedidos. Estas duas estratégias suportam outros dois modelos: um em que existe apenas um botão de chamada, e o outro assume a existência de um teclado onde é possível indicar o piso de destino.

A **primeira** estratégia assume o sistema tradicional onde cada elevador possui uma estratégia fixa e individual: atende o pedido o elevador que se encontra mais próximo do piso onde a chamada foi efetuada. Para isto ser possível, os elevadores comunicam entre si através do protocolo de interação que será descrito mais à frente.

Para tornar o envio de requests para outros elevadores melhor, seria necessário pedir-lhes feedback para todos os requests e isso sobrecarregaria a rede de negociação, só se pedindo ajuda para o pior pedido de cada vez. Se não

podem ajudar para o pior, provavelmente não podem ajudar para outros. E iterativamente, ao alocar os piores pedidos a outros, optimiza-se/reduz-se os tempos de espera.

A **segunda** estratégia irá estimar o tempo que o elevador demoraria a responder a um pedido caso o mesmo fosse alocado a ele, tendo em conta a lista de pedidos que o elevador possui.

2.3 Protocolos de interação

FIPA CONTRACT NET (Sérgio)

3 Desenvolvimento

3.1 Plataforma/ferramenta utilizada e ambiente de desenvolvimento

A ferramenta escolhida para o desenvolvimento deste trabalho foi JADE. JADE é uma framework que permite o desenvolvimento de sistemas multiagente e é compatível com FIPA, isto é, segue os padrões de software estabelecidos para agentes heterogéneos e interativos e para sistemas baseados em agentes.

O projeto foi desenvolvido utilizando o IntelliJ IDEA Community Edition no sistema operativo Windows 10.

3.2 Estrutura da aplicação

(Sérgio já tem o diagrama de classes)

3.3 Detalhes relevantes da implementação

4 Experiências

4.1 Objetivo de cada experiência

4.2 Resultados

5 Conclusões

5.1 Da análise dos resultados das experiências levadas a cabo

5.2 Do desenvolvimento do trabalho e aplicabilidade de SMA ao cenário proposto

6 Melhoramentos

6.1 Sugestões para melhoramentos a introduzir no programa

De forma a conseguirmos ter uma melhor comparação era importante adicionar mais estratégias de respostas aos pedidos, conseguindo assim realizar mais experiências e extraindo novas conclusões.

Achamos que a interface podia ser ligeiramente melhorada, de forma a demonstrar melhor o funcionamento dos elevadores, como eles subirem/descerem e abrirem/fechar as portas à entrada/saída dos utentes. No entanto, consideramos que as estatísticas demonstradas conseguem transmitir uma boa ideia daquilo que se está a suceder.

7 Recursos

7.1 Bibliografia

Slides do Moodle sobre JADE
Documentação do JADE
Código fonte do JADE

7.2 Software

JADE (http://jade.tilab.com/)
Eclipse IDE for Java
IntelliJ IDEA Community Edition

7.3 Elementos do grupo

O grupo trabalhou de forma equitativa durante as aulas práticas e houve boa comunicação entre os membros, de forma a que cada elemento soubesse o que estava a ser feito.

8 Apêndice

8.1 Manual do utilizador

Para conseguir correr o projeto, deve:
1. Importar o projeto;
2. Adicionar a *library* do *JADE* ao *build path*;
3. Editar o ficheiro default.properties consoante o que deseja visualizar na interface;
4. Correr, utilizando a classe MyBoot como *main*.