UNIVERSIDADE NOVA DE LISBOA
FCT - Departamento de Engenharia Electrotécnica
Secção de Robótica e Manufatura Integrada

Integração de Sistemas

2017 / 2018

Trabalho 3

Sistemas Multiagente (JADE)

Duração: 4 aulas acompanhadas por docente

Entrega: 17 de Junho de 2018

1. INTRODUÇÃO

A definição de agente não é consensual. Desta forma, surgem múltiplas definições que variam de acordo com a cultura dos diversos proponentes. A noção de que um agente deve ter alguma autonomia é um dos poucos pontos de contacto entre as diferentes definições. No âmbito deste trabalho considerar-se-á que um agente é um programa que, sobre um determinado ambiente onde está situado, é capaz de tomar ações autónomas com base nos seus objetivos e na perceção de alterações ambientais, no sentido de cumprir o objetivo para que foi desenvolvido. Esta definição suporta a existência de um mundo não determinista pelo que a informação que o agente recolhe do meio é uma parcela da realidade. Assim, o agente deve estar preparado para lidar com informação incerta e incompleta. De acordo com esta definição um agente deve denotar algumas das seguintes características:

- Autonomia: um agente pode operar em autocontrolo sem interação de terceiros;
- Sociabilidade: os agentes podem interagir com outros agentes e entidades cumprindo as regras sociais que pautam a sua interação;
- Reatividade: os agentes são capazes de reagir a mudanças no ambiente.
- Pró-Atividade: os agentes não são entidades puramente reativas, isto é, têm um comportamento orientado a objetivos podendo tomar iniciativa durante uma operação;
- Adaptabilidade: os agentes têm a capacidade de se adaptarem a alterações no ambiente.

As interações com o meio ambiente (mundo e outros agentes) são fundamentais no contexto dos sistemas multiagente. Neste sentido, um sistema multiagente pode ser definido como uma rede de “solucionadores” de problemas que cooperam na resolução de um problema global do qual apenas conhecem partes. Está implícita a noção de que uma sociedade de agentes fornece um nível de funcionalidade em que o todo é maior do que a soma da contribuição das partes. Este conjunto de características torna os conceitos de agente e sistema multiagente poderosas ferramentas de modelação de sistemas de natureza distribuída.

2. Apresentação do Problema

Sistema de Elevadores

Os elevadores convencionais utilizam controladores com pouca capacidade de processamento e com regras pré-definidas, que permitem controlar um conjunto de elevadores previamente escolhidos e inalteráveis durante a execução.
Para além deste problema da falta de flexibilidade durante a execução do sistema, os elevadores requerem uma configuração local, específica para cada caso.
Com a constante evolução dos dispositivos computacionais, como por exemplo o aumento de capacidade de computação e de memória dos controladores, é agora possível apresentar e desenvolver soluções que sem estes novos recursos seriam impossíveis de se desenvolver.
Com vista à resolução destes problemas, de falta de flexibilidade e inteligência, destes sistemas é proposta uma abordagem baseada num ambiente multiagente responsável por gerir os movimentos de cada um dos elevadores e as decisões a tomar durante a execução, tais como:

- Que elevador deve se movimentar a cada momento;
- Qual o destino de um elevador quando este inicia um novo movimento;
- Se um elevador deve, ou não, corrigir o seu destino caso receba um novo pedido;
- Etc.

É proposta uma arquitetura constituída por dois tipos de agentes, com diferentes responsabilidades e abstrações:

- **ElevatorAgent;**
- **OrchestractorAgent.**

Na Tabela 1 são apresentados os dois tipos de agentes presentes na arquitetura e as suas responsabilidades.

Tabela 1- Agentes e as suas responsabilidades

Agente	**Responsabilidade**
ElevatorAgent	Este agente abstrai cada elevador presente no sistema. Tem a capacidade de verificar e atuar no *hardware*, como por exemplo: • Verificar a lista de destinos pretendidos pelos utilizadores que se encontram no interior do elevador; • Mandar o elevador para uma posição (andar) específica; Verificar se o elevador está numa posição específica (andar);

OrchestractorAgent	É a entidade de mais alto nível. Tem conhecimento de todos os elevadores presentes no sistema e sabe também para que andares deve encaminhar cada um dos elevadores, para respeitar os pedidos dos utilizadores que se encontram no interior e exterior dos elevadores.

O objetivo da infraestrutura é oferecer uma gestão inteligente e eficiente dos recursos (Elevadores). Como tal é preciso fazer-se um trabalho prévio de estudo de regras e comportamentos que permitirão essa mesma gestão, tendo em atenção alguns tópicos, tais como:

- Critério para decidir qual o próximo destino;
- Quando um utilizador introduz um novo destino, e o elevador se encontra em movimento, o elevador deverá verificar se será uma boa opção alterar o seu destino ou não, caso esse destino se encontre em uma posição intermédia ou próxima;
- Balancear o tempo despendido para cada pedido evitando que um pedido demore demasiado tempo, com o tempo total de resposta a todos os pedidos.

3. Implementação

Na implementação pedida será utilizado o seguinte material:

- Linguagem JAVA no IDE Netbeans;
- Plataforma JADE;
- Windows 7, 8 ou 10 recomendado (também possível em Linux e OSX);
- Código fornecido pelos docentes da disciplina.

O projeto fornecido pelos docentes da disciplina contem sete *packages* diferentes. Os alunos deverão trabalhar nos *packages* reservados à implementação dos agentes **ElevatorAgent** e **OrchestractorAgent**, em que os nomes dos *packages* são respetivamente ElevatorAgent e OrchestractorAgent.

No *package Common* é fornecida uma biblioteca que permite serializar qualquer conteúdo, de forma a este mesmo conteúdo ser enviado no *content* de uma mensagem entre agentes, e o processo inverso é também fornecido nessa mesma biblioteca, com o nome Serialization.

De forma a testar a implementação desenvolvida pelos alunos é fornecido um simulador já integrado no projeto fornecido pelos docentes. Este simulador corre automaticamente aquando do lançamento do projeto. É possível ver uma descrição deste simulador mais à frente.

Planeamento das Aulas

Aula 1 – Instalação, teste e familiarização com a plataforma

1. Siga as instruções fornecidas pelos docentes e instale o JADE. Teste a plataforma.
2. De forma a testar a plataforma e como auxilio ao desenvolvimento durante o projeto consulte os tutoriais fornecidos pelos docentes.

Aula 2 – Utilização do Directory Facilitator (DF) e Adicionar um novo *Behaviour*

1. Leia atentamente todo o código fornecido, familiarizando-se com as funcionalidades implementadas e com os pontos a implementar.
2. Registo no Directory Facilitator (DF) do JADE

   No package *Common*, na classe *DFInteraction* preencha o método *static*, *RegisterInDF* que permite a qualquer agente inscrever-se no DF e aos seus serviços.

3. Procura no Directory Facilitator (DF) do JADE

   No package *Common* e na classe DFInteraction preencha o método *static*, *SearchInDF* que permite a qualquer agente procurar no DF por agentes que disponibilizem um determinado serviço.

4. Remover agente do Directory Facilitator (DF) do JADE

   No package *Common* e na classe *DFInteraction* preencha o método *static*, *DeregisterFromDF* que limpa o registo do agente no DF.

5. Adicione ao ***ElevatorAgent*** um novo *behaviour* do tipo *TickerBehaviour*, que está implementado na classe *UpdateDestinies*. Este *behaviour* deve ser lançado no método *setup()*, na classe ***ElevatorAgent*** e irá correr periodicamente, com intervalos de execução de 100 ms. Este tempo está definido no ficheiro *Constants* do *package Common*.
   Periodicamente este *behaviour* irá verificar a lista de destinos pretendidos pelos utilizadores deste elevador, caso sejam encontradas mudanças, é adicionado um novo *behaviour* que implementa uma comunicação do tipo *FIPA Request*.
   Este *behaviour* pode também ser usado para adicionar um método que corre sempre que houver uma alteração nos destinos, tal como a introdução de novos destinos pretendidos.

Aula 3 – Implementação do *Orchestractor*

1. Implemente todos os *behaviours* e regras que permitem a coordenação dos elevadores sem que estes respondam à introdução de novos destinos durante um movimento.
   Nesta fase o ***OrchestractorAgent*** deve ser capaz de receber todos os estados dos elevadores, tais como destinos pretendidos e a posição no final de cada movimento, e deve também ordenar novos movimentos, depois de consultar a lista de destinos, tanto do ***OrchestractorAgent*** como do ***ElevatorAgent***, e as regras de forma a tomar a melhor decisão.

Aula 4 – Implementação de Funcionalidade Extra

1. Implementação de autonomia nos elevadores durante a execução

   Implemente todos os behaviours e regras que achar necessário, para que o *ElevatorAgent* redefina um novo destino, autonomamente, durante um movimento, caso assim se justificar. Por exemplo, se um elevador estiver a descer do andar 50 para o andar 10 e durante a descida for introduzido o destino 20, este deve parar no andar 20, comunicando ao *OrchestractorAgent* essa alteração e voltando a renegociar o destino que se segue.

4. Avaliação

A avaliação do trabalho tem a seguinte ponderação:

- Correta implementação e demonstração de funcionamento do trabalho previsto para a aula 2:
  - 10 valores
- Correta implementação e demonstração de funcionamento do trabalho previsto para as aulas 2 e 3:
  - 13 valores
- Correta implementação e demonstração de funcionamento do trabalho previsto para as aulas 2,3 e 4:
  - 18 valores
- Correta implementação e demonstração de funcionamento do trabalho previsto para as aulas 2,3 e 4 e com funcionalidades extra não definidas:
  - 20 valores

5. Descrição do Simulador

Console

Quando o projeto do *Netbeans* for lançado, irão aparecer duas caixas, uma delas para simular o *hardware*, no qual o **OrchestractorAgent** deve verificar os

pedidos realizados pelos utilizadores na parte exterior dos elevadores, e outra que é a consola de simulação. Nesta consola é possível configurar e lançar cada um dos **ElevatorAgents** que se desejam.

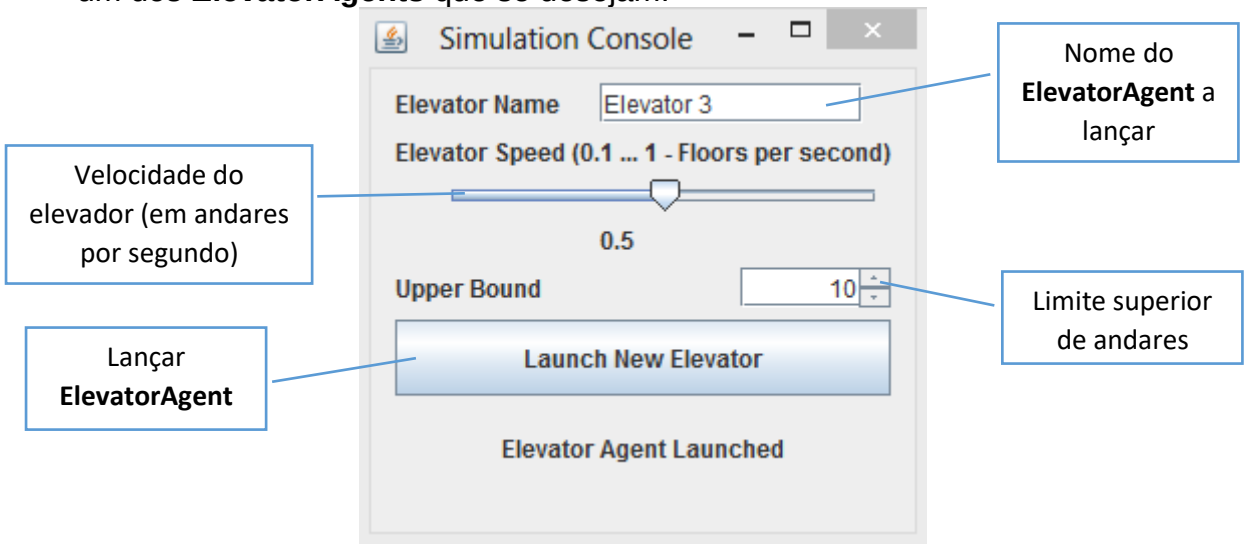


Figura 1 – Consola para lançar e configurar cada ElevatorAgent

Nesta consola é possível definir:

- Nome do agente;
- Velocidade do elevador, em andares por segundo (pode ser diferente entre elevadores);
- Limite superior de andares.

A consola irá informar o utilizador se o lançamento do agente se realizou com sucesso ou não.

OrchestractorAgent

Este agente necessita de saber em que andares é que os utilizadores "chamaram" um elevador, ou seja, quando um utilizador pretende "chamar" um elevador para o usar. Como tal foi criada uma interface gráfica em que se pode "chamar" um elevador indicando o andar em que nos encontramos.
Os andares em que estas chamadas foram efetuadas encontram-se listados nesta mesma interface gráfica e devem ser removidas por este agente quando um dos elevadores é destacado para este serviço.

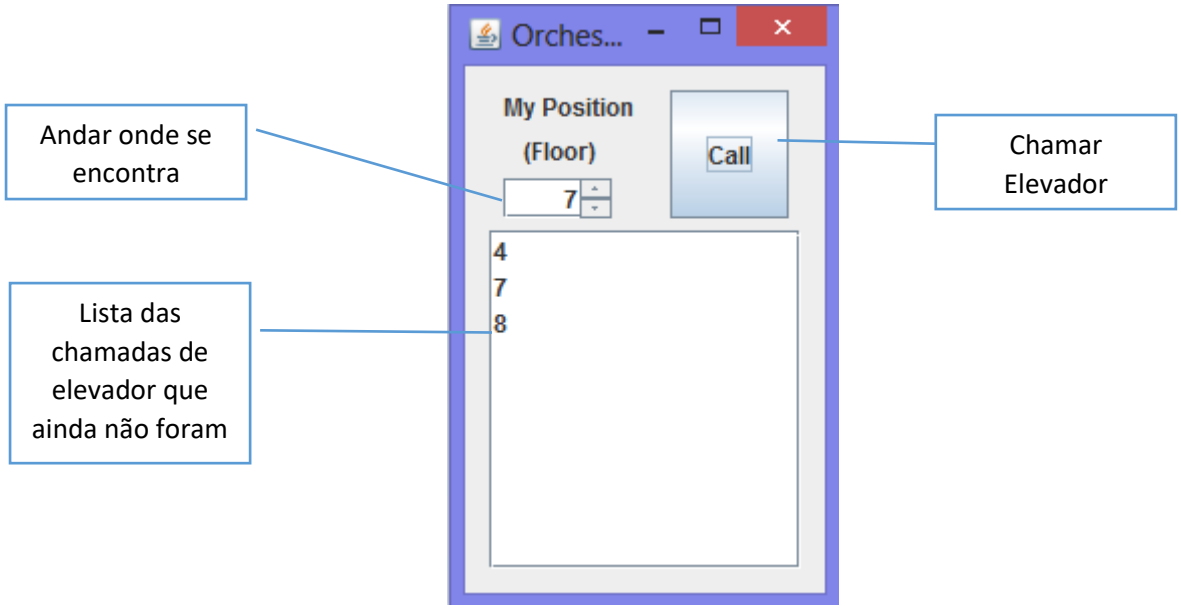


Figura 2 - Interface gráfica que permite simular as "chamadas" de um elevador

O **OrchestraAgent** tem uma interface definida para comunicar com o *hardware*, neste caso simulado pela interface gráfica, com os seguintes métodos:

- public ArrayList<Integer> calls() – Retorna a lista das chamadas por tratar;
- public void removeCall(int floor) – Remove da lista de chamadas um andar que foi servido;
- public boolean initHardware(OrchestractorAgent agent) – Este método é chamado para inicializar o *hardware*, neste caso lança a respetiva interface gráfica.

ElevatorAgent

Para cada **ElevatorAgent** lançado, será também lançada uma interface gráfica, onde é possível saber em que andar o elevador se encontra, os destinos pretendidos pelos utilizadores que se encontram no interior do mesmo e onde é possível também introduzir novos destinos desejados. Esta interface gráfica tem o seguinte aspeto:

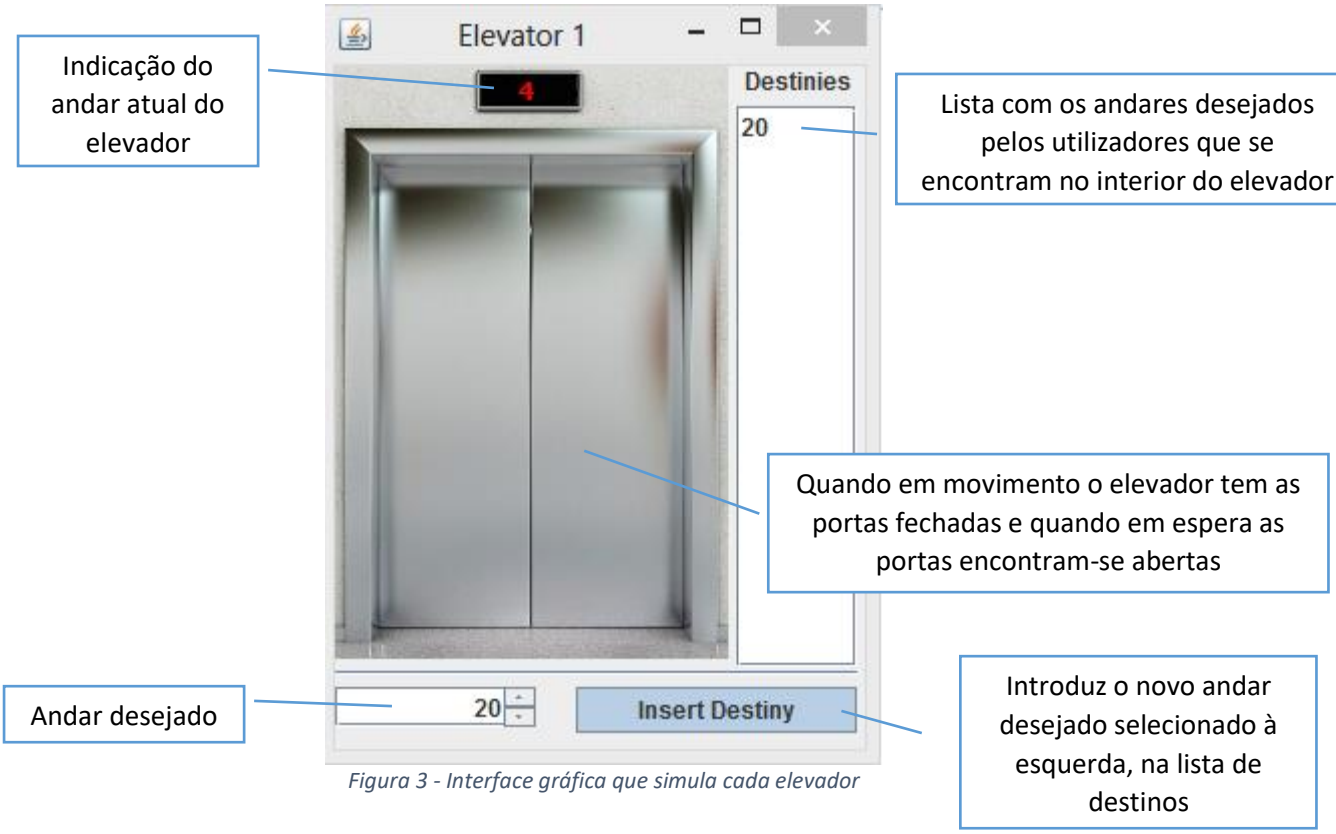


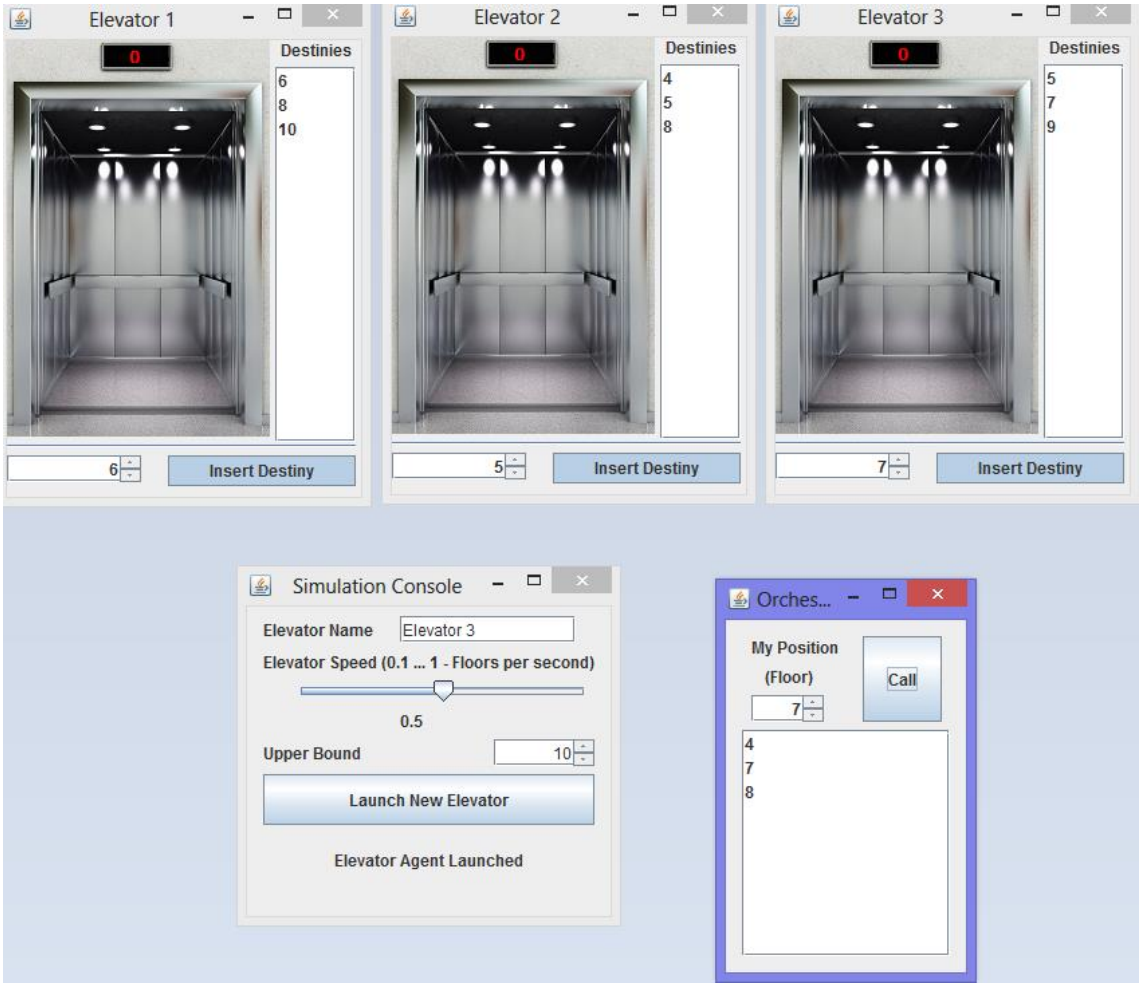
Figura 3 - Interface gráfica que simula cada elevador

O **ElevatorAgent** tem uma interface definida para comunicar com o *hardware*, neste caso simulado pela interface gráfica, com os seguintes métodos:

- public boolean arrivePosition(int position) – retorna verdadeiro se o elevador se encontrar nesta posição (andar) e em espera;
- public ArrayList<Integer> destinies() – retorna a lista dos destinos desejados pelos utilizadores que se encontram no interior do elevador;
- public boolean goToPosition(int position) – inicia a rotina que encaminha o elevador para o andar desejado;
- public boolean initHardware(OrchestractorAgent agent) – Este método é chamado para inicializar o hardware, neste caso lança a respetiva interface gráfica.

Ambiente de Simulação

Quando lançados três agentes do tipo **ElevatorAgent**, deverá ficar com um ambiente de simulação com o seguinte aspeto:



Neste ambiente de simulação teremos:

- Três interfaces gráficas para os três **ElevatorAgents**;
- Uma Consola para lançar **ElevatorAgents**;
- Uma interface gráfica para simular os espaços exteriores aos elevadores.

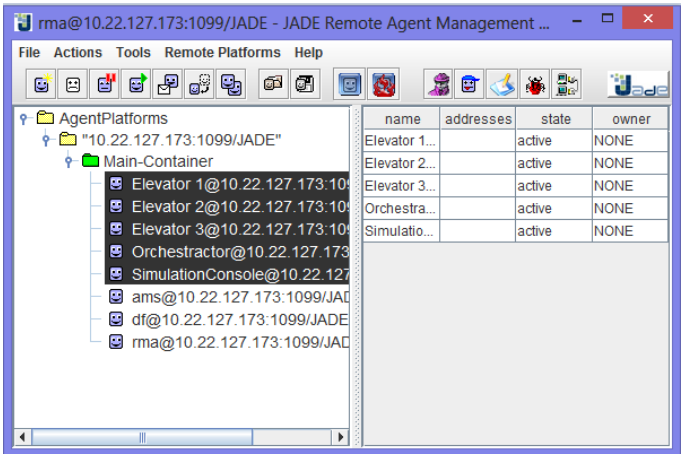


Figura 4 - Agentes a correr durante a simulação

6. Material Auxiliar

http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html

http://jade.tilab.com/documentation/tutorials-guides/

Docentes

André Rocha andre.rocha@uninova.pt

José Barata jab@uninova.pt