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

Integração de Sistemas

2014 / 2015

Trabalho 3

Sistemas Multiagente (JADE) e XML
com **RELATÓRIO FINAL**

Duração: 4 aulas acompanhadas por docente

Entrega: 14 de Junho de 2015

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

Quinta de integração de Sistemas

Os agentes são muitas das vezes utilizados para replicar comportamentos verificados na natureza para sistemas informáticos e mecatrónicos, nomeadamente para sistemas de manufatura, veículos autónomos, organizações entre outros.

Neste trabalho é pedido que se modele e implemente um ambiente simulado que deverá representar o comportamento de dois tipos de animais diferentes, num ambiente também ele replicando um ambiente real. O ambiente é criado por três diferentes entidades com responsabilidades diferentes, uma que abstrai o ambiente onde a simulação acontece e outros dois responsáveis por abstrair agentes autónomos que interagem com o ambiente simulado.

Tabela 1- Agentes e as suas responsabilidades

Agente	**Responsabilidade**
EnvironmentAgent	Este agente é responsável por coordenar a simulação e disparar os pedidos aos outros agentes para atualizarem os seus estados. Este agente também tem a capacidade para remover ou adicionar as restantes entidades e atualizar a interface gráfica.
CowAgent	Este agente é responsável por abstrair uma vaca presente no ambiente simulado. Este agente deve garantir que a entidade se desloca de forma a procurar comida (*Grass*) e fugir dos lobos (*Wolfs*) de forma a garantir a sobrevivência.
WolfAgent	Este agente é responsável por abstrair um lobo presente no ambiente simulado. Este agente deve garantir que a entidade se desloca de forma a procurar comida que neste caso são as vacas (*Cows*).

O objetivo da infraestrutura é a dinâmica adaptação do sistema com base nas decisões de cada uma das entidades.

3. Troca de Mensagens

A estrutura que permite ao EnvironmentAgent enviar o estado do ambiente e receber as decisões das entidades do mesmo é fornecido pelo corpo docente, sobre a forma de *schema*, num ficheiro "*.xsd"*. A estrutura TMyPlace tem uma lista de TPlace's.

Cada lista TMyPlace envia uma lista com 9 elementos, estes novos elementos representam o estado do sítio onde se encontra o elemento e as posições envolventas, da seguinte forma:

Figure 1 - Representação do vector que é trocado entre o EnvironmentAgent e os restantes

- Grass, do tipo int:
  - Este valor pode ir de 0 (valor mínimo) até 3.
- Wolf, do tipo boolean:
  - Indica se nesta posição se encontra um lobo.
- Cow, do tipo boolean:
  - Indica se nesta posição se encontra uma vaca.
- Obstacle, do tipo boolean:
  - Indica se nesta posição se encontra um obstáculo.
- Entity, do tipo String:
  - Indica o nome da entidade (CowAgent ou WolfAgent) que se encontra nesta posição (*null* caso esteja vazia).
- Position, do tipo TPosition:
  - Contem as coordenadas desta posição (xx e yy do tipo int).

Este *schema* será também utilizado para serializar o objeto para uma String e assim poder ser trocado entre os agentes, nas diversas mensagens.

4. Implementação

Na implementação pedida será utilizado o seguinte material:

- Linguagem JAVA no IDE Netbeans;
- Plataforma JADE;
- JAXB: utilizado para criar as classes TMyPosition, TPosition e TPosition, através do *schema* e serializar objetos destes tipos;
- Windows 7 ou 8 recomendado (também é possível em Linux);
- Código fornecido pelos docentes da cadeira;

5. Planeamento das Aulas

Aula 1 – Manipulação do DF e Inicialização de Comportamentos

1. Estudo do código fornecido pelos docentes

   Leia atentamente todo o código fornecido familiarizando-se com as funcionalidades implementadas com os pontos a implementar.

2. Registo no Directory Facilitator (DF) do JADE

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

   No package Common e na classe DFInteraction preencha o método *static*, SearchInDFbyName que permite qualquer agente procurar no DF por agentes pelo nome.

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. Adicionar um Behaviour ao agente

   No package Cow e na classe com o nome CowAgent, adicione à função setup() o código necessário para que o agente CowAgent execute os behaviours codificados nas classes, as ontologias a utilizar estão já especificadas no package Common, em Constants:

      i. KillYourselfResponder;
      ii. UpdateStateResponder;

   No package Wolf e na classe com o nome WolfAgent, adicione à função setup() o código necessário para que o agente WolfAgent execute o behaviour codificado na classe:

      i. UpdateStateResponder.

   Relembrar que a ontologia já está definida.

Aula 2 – Comunicação

1. Iniciar uma conversa entre dois agentes em JADE

   No package Environment e na classe SendUpdateStateInitiator e KillYourselfInitiator implemente o método BuiltMessage. Este método estático deverá devolver uma ACLmessage que será usada para iniciar uma conversação entre agentes. O parâmetro receiver, contêm o AID do agente que vai receber a mensagem e o parâmetro content contêm o conteúdo a enviar já serializado, que é serializado através da função createPlaceStateContent pertencente à classe MessageManagement. Em ambos os casos, não esquecer de definir a Ontology da comunicação, as constantes que definem as ontologias necessárias estão disponíveis na classe Constants do package Common.

2. Receber uma **conversa**

   No package CowAgent, nas classes:

   i. KillYourselfResponder;
   ii. UpdateStateResponder;

   Que são responsáveis pelo lado do cliente da conversação iniciada no ponto anterior. Complete o método handleRequest de forma a que o CowAgent possa receber o estado do ambiente, perceber qual deve ser o próxima posição para onde se deve deslocar e responder com essa posição, no caso do UpdateStateResponder (Nesta fase inicial não se preocupe com a inteligência da entidade, preocupe-se apenas em implementar corretamente o protocolo de comunicação com uma posição válida). Importante referir que a resposta a este pedido deve ser enviada segundo uma lista TMyPlace, com apenas um elemento, em que nesse elemento está definido um TPlace e na posição desse TPlace a nova posição da entidade.

   No caso do KillYourselfResponder, é sinal que a entidade foi removida do ambiente e portanto o agente deve morrer.

   No package WolfAgent, na classe UpdateStateResponder implemente o protocolo de comunicação à semelhança do ponto anterior.

3. Serialização das mensagens, recorrendo ao schema e JAXB

   No package Common, na classe MessageManagement, preencher os métodos createPlaceStateContent e retrievePlaceStateContent, responsáveis por serializar um objeto TMyPlace numa String e vice-versa.

Aula 3 – Implementação da Inteligência das Entidades

1. Nos UpdateStateResponder, no WolfAgent e CowAgent, codificados na aula anterior implemente o código que permitirá às entidades inteligência perante o estado do ambiente envolvente.

2. Confira ao agente WolfAgent a decisão de abandonar a simulação caso muito tempo (turnos) sem comer.

Aula 4 – Finalização do trabalho

1. Finalizar implementação do trabalho prático.
2. Inicializar elaboração de relatório final.

6. Avaliação

A avaliação do trabalho tem a seguinte ponderação:

- Correta implementação e demonstração de funcionamento do trabalho previsto para as aulas 2, 3 e 4:
  - 14 valores
- Correta implementação e demonstração de funcionalidade extra não definidas:
  - 1 valor
- Relatório Final:
  - 5 valores

Docentes

André Rocha andre.rocha@uninova.pt

José Barata jab@uninova.pt