Práctica 4: JADE

Sistemas de Ayuda a la Toma de Decisiones

Pedro Allué Tamargo (758267) Juan José Tambo Tambo (755742)
Jesús Villacampa Sagaste (755739)

18 de noviembre de 2020

Índice

1. Ejercicio 1

Para este ejercicio se ha procedido a crear el agente `Ej1_Agente`. Este agente deberá saludar al ser creado, por ello se sobrescribirá el método `setup()`.
Para pasar un argumento al agente se hará en la creación del mismo en la opción *arguments* (Figura 1). En el código del agente se utilizarán la siguiente línea:

```
contador = Integer.parseInt(getArguments()[0].toString());
```

Esta línea utilizará el método `getArguments()` para obtener un array de objetos con los argumentos. Una vez obtenido el primer elemento se convertirá a `String` y se utilizará el método `parseInt()` de la clase *wrapper* `Integer`.
Se pide que el agente implemente un comportamiento tipo `CyclicBehaviour` para decrementar esta variable `contador` y cuando llegue a 0 termine la ejecución del agente. El comportamiento `CyclicBehaviour` tiene la peculiaridad de que su método `done()` siempre devuelve `false` y por lo tanto no terminará nunca a no ser que se especifique en el método `action()`. Por lo tanto el código de dicha clase será el siguiente:

```
package Ejercicio1;

import jade.core.behaviours.CyclicBehaviour;

class ContadorExternoBehaviour extends CyclicBehaviour {
    private final Ej1_Agente agente;

    public ContadorExternoBehaviour(Ej1_Agente agente) {
        this.agente = agente;
    }

    @Override
    public void action() {
        System.out.format("Contador: %d", agente.getContador());
        agente.setContador(agente.getContador() - 1);
        if (agente.getContador() == 0) {
            // El agente se suicida
            agente.doDelete();
        }
    }
}
```

El código del agente descrito se puede encontrar en el Listing 5.1.

2. Ejercicio 2

2.1. Parte 1

La comunicación entre los dos agentes (*Ej2_Envia* y *Ej2_Recibe*). En el código del comportamiento de *Ej2_Envia* se puede observar como el agente pide datos por entrada estándar para conocer al destinatario del mensaje y el texto a enviar. Tras esto se puede observar que el agente crea un objeto de la clase `ACLMessage` (*Agent Comunication Language Message*) de tipo `REQUEST` (petición). Tras esto, crea un objeto que identifica al agente destinatario (`AID`) y le pasa como parámetros el nombre del agente y además el *booleano* `AID.ISLOCALNAME` que indica que el nombre del agente solo es válido localmente. Despues se le asigna este identificador `AID` al destinatario y el texto como contenido del mensaje para después enviarlo utilizando el método `send()` del agente asociado al comportamiento.

En el código del comportamiento del *Ej2_Recibe* ocurre la operación de recepción del mensaje enviado por el agente del comportamiento anterior. En este método `action` se puede observar como se produce una recepción de un mensaje (`ACLMessage`) con carácter síncrono, es decir, bloqueante. Tras esto el comportamiento muestra por la salida estándar el nombre del agente del que recibe el mensaje y su contenido.

2.2. Parte 2

Ahora se pide modificar el código de los comportamientos para que los agentes se reenvíen entre sí el mensaje inicial un número determinado de veces, el cual se pasará como parámetro a uno de los agentes en su creación. Para la implementación de estos comportamientos se ha partido desde una aproximación anterior al apartado anterior. Uno de los agentes será el que envíe primero el mensaje del texto y el otro el que responda con el. Esto ocurrirá tantas veces como se indique en el parámetro.

Para la implementación de este comportamiento se han utilizado máquinas de estado finitas (*FSM*). Para el comportamiento que envía el texto por primera vez se ha utilizado el siguiente autómata (Figura 2)

Para el comportamiento del agente de recepción se ha utilizado la máquina de estados complementaria a la de la Figura 2. Se puede observar la máquina de estados de recepción en la Figura 3.

Para la implementación de la máquina de estados en los comportamientos se ha heredado del comportamiento `FSMBehaviour`. El código del comportamiento de envío se puede encontrar en el Listing 5.2 y el código del comportamiento de la recepción en el Listing 5.3.

3. Ejercicio 3

En este ejercicio, hay dos agentes, *Ej3_EmisorPasajeVuelo* y *Ej3_ReceptorPasajeVuelo*. El funcionamiento consiste en que el primero envía toda la información necesaria para la reserva de un vuelo al segundo.

Para la implementación de este comportamiento, en el agente *Ej3_EmisorPasajeVuelo* se puede observar como el agente pide datos por entrada estándar para conocer al destinatario del mensaje y la información necesaria para reservar el vuelo, que consta del origen, destino, tarifa y aerolínea (Figura 4) y que se encapsula en un objeto de tipo *Ej3_PasajeVuelo*. Tras esto se puede observar que el agente crea un objeto de la clase `ACLMessage` (*Agent Comunication Language Message*) de tipo `CONFIRM` (confirmación). A continuación, crea un objeto que identifica al agente destinatario (`AID`) y le pasa como parámetros el nombre del agente y además el *booleano* `AID.ISLOCALNAME` que indica que el nombre del agente solo es válido localmente. Despues se le asigna este identificador `AID` al destinatario y el objeto *Ej3_pasajeVuelo* como contenido del mensaje para después enviarlo utilizando el método `send()` del agente asociado al comportamiento.

En el código del comportamiento del *Ej3_ReceptorPasajeVuelo* ocurre la operación de recepción del mensaje enviado por el agente del comportamiento anterior. En este método `action` se puede observar como se produce una recepción de un mensaje (`ACLMessage`) con carácter síncrono, es decir, bloqueante. Tras esto, se comprueba que el mensaje es del tipo `CONFIRM` y su contenido es un objeto de tipo *Ej3_PasajeVuelo*. Si se cumplen estas dos condiciones se muestra por la salida estándar toda la información de la reserva del vuelo, que se encuentra encapsulada en el objeto recibido (Figura 5).

4. Anexo 1: Figuras

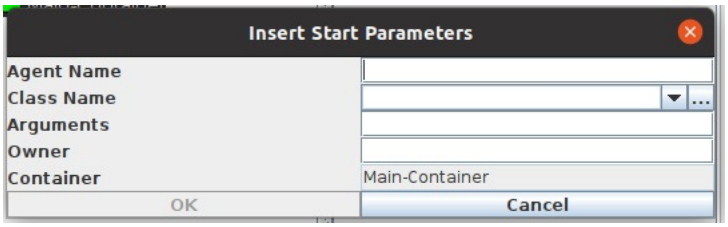

Figura 1: Captura de pantalla del menú de creación de agentes en *JADE*

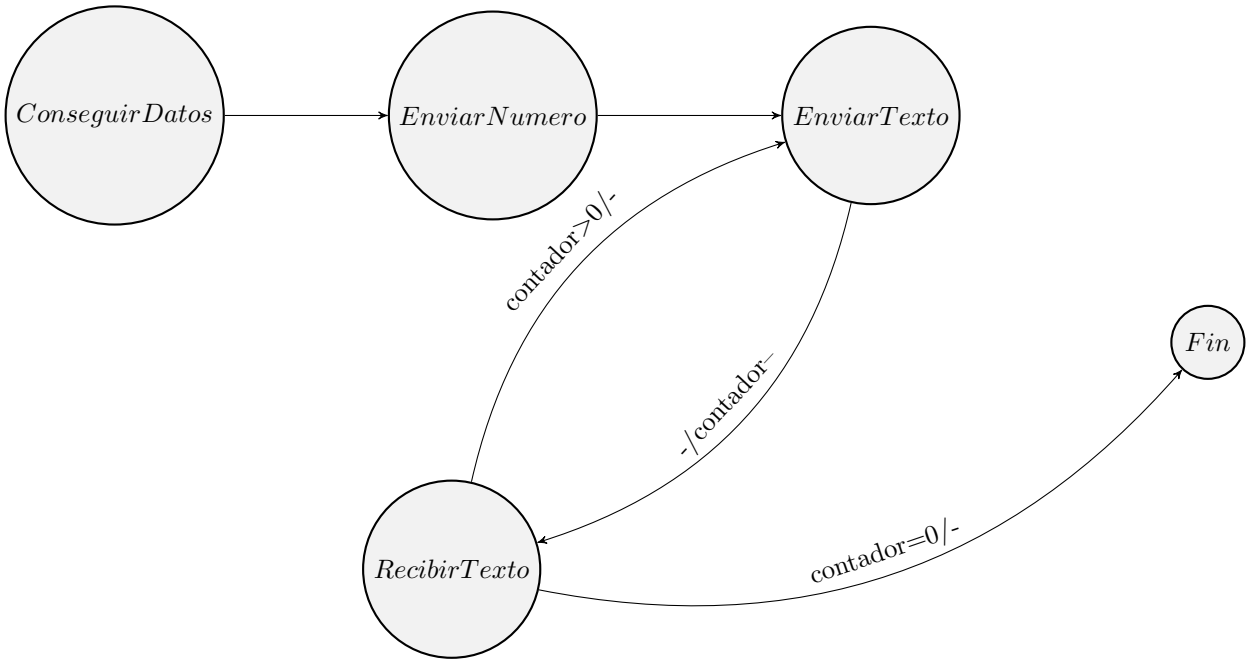

Figura 2: Máquina de estados finita del autómata de envío

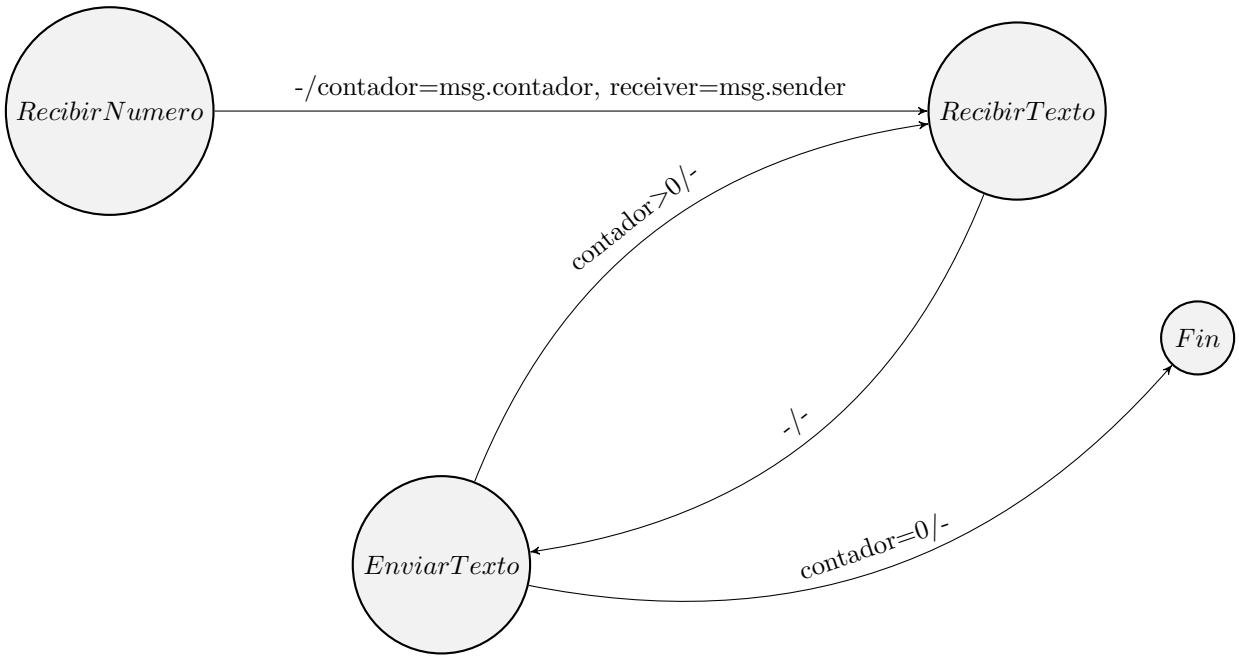

Figura 3: Máquina de estados finita del autómata de recepción

```
Escriba el nombre del Agente Receptor:
Receiver
Escriba el origen:
Zaragoza
Escriba el destino
Viena
Escriba la tarifa
DIRECT
Escriba la aerolinea
Ryanair
```

Figura 4: Captura de pantalla de la información a introducir por entrada estándar para el agente *Ej3_EmisorPasajeVuelo*

```
recibido
Origen: Zaragoza
Destino: Viena
Tarifa: DIRECT
Aerolinea: Ryanair
```

Figura 5: Captura de pantalla de la información que recibe*Ej3_ReceptorPasajeVuelo*

5. Anexo 2: Códigos

5.1. Código del agente del Ejercicio 1

```
package Ejercicio1;


import jade.core.Agent;

public class Ej1_Agente extends Agent {

    // Contador del agente
    private int contador;

    @Override
    protected void setup() {
        // El agente acepta un parametro
        contador = Integer.parseInt(getArguments()[0].toString());
        System.out.format("Hola, mi nombre es: %s. UUID: %s\n", getLocalName(),
            getAID().getName());
        addBehaviour(new ContadorExternoBehaviour(this));
        super.setup();
    }

    @Override
    protected void takeDown() {
        System.out.printf("El agente %s muere\n", getLocalName());
        super.takeDown();
    }

    public int getContador() {
        return contador;
    }

    public void setContador(int contador) {
        this.contador = contador;
    }
}
```

5.2. Código del comportamiento de la máquina de estados de envío

```
package Ejercicio2;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.FSMBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;

import java.util.Scanner;

// Documentation: https://jade.tilab.com/doc/api/jade/core/behaviours/
    FSMBehaviour.html
public class Ej2_Envia_FSM extends FSMBehaviour {

    // Text to send
    private String text;

    private static final int EV_VE_ENVIAR_NUMERO = 0;
    private static final int EV_VE_RECIBIR_TEXTO = 1;
    private static final int EV_VE_ENVIAR_TEXTO  = 2;
    private static final int EV_VE_FIN           = 3;

    private final String conseguir_datos = "Conseguir datos";
    private final String enviar_numero = "Enviar numero";
    private final String enviar_texto = "Enviar texto";
    private final String recibir_texto = "Recibir texto";
    private final String fin = "Fin";

    private int contador;
    private AID receiver;


    public Ej2_Envia_FSM(Agent a, int nVeces) {
        super(a);
        contador = nVeces;
        declareStates();
        declareTransitions();
    }

    private void declareTransitions() {
        // Ira al estado de fin desde recibir (porque empieza enviando)
        registerTransition(conseguir_datos, enviar_numero, EV_VE_ENVIAR_NUMERO);
        registerTransition(enviar_numero, enviar_texto, EV_VE_ENVIAR_TEXTO);
        registerTransition(enviar_texto, recibir_texto, EV_VE_RECIBIR_TEXTO);
        registerTransition(recibir_texto, enviar_texto, EV_VE_ENVIAR_TEXTO);
        registerTransition(recibir_texto, fin, EV_VE_FIN);
    }

    private void declareStates() {
        // 1st state is take receiver data and text to send
        registerFirstState(new OneShotBehaviour() {
            @Override
            public void action() {
                Scanner scanner = new Scanner(System.in);

                System.out.println("Nombre del destinatario? ");
                String receiverName = scanner.nextLine();
                receiver = new AID(receiverName, AID.ISLOCALNAME);
```

```
            System.out.println("Texto a enviar?");
            text = scanner.nextLine();

            System.out.format("%s\n", "--------------------------------------");
        }

        @Override
        public int onEnd() {
            return EV_VE_ENVIAR_NUMERO;
        }
    }, conseguir_datos);

    registerState(new OneShotBehaviour() {
        @Override
        public void action() {
            System.out.printf("Agente: %s . Estado inicial (Enviar numero)\n
                ", myAgent.getLocalName());
            ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);

            aclMessage.addReceiver(receiver);
            aclMessage.setContent(Integer.toString(contador));

            myAgent.send(aclMessage);
        }

        @Override
        public int onEnd() {
            return EV_VE_ENVIAR_TEXTO;
        }
    }, enviar_numero);

    registerState(new OneShotBehaviour() {
        @Override
        public void action() {
            System.out.printf("Agente: %s . Estado: Enviar texto\n", myAgent
                .getLocalName());
            ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
            aclMessage.addReceiver(receiver);
            aclMessage.setContent(text);

            myAgent.send(aclMessage);
            contador--;
            System.out.printf("Agente %s . Enviado. Quedan %d envios\n",
                myAgent.getLocalName(), contador);
        }

        @Override
        public int onEnd() {
            return EV_VE_RECIBIR_TEXTO;
        }
    }, enviar_texto);

    registerState(new OneShotBehaviour() {
        @Override
        public void action() {
            System.out.printf("Agente: %s . Estado: Recibir texto\n",
                myAgent.getLocalName());
            ACLMessage aclMessage = myAgent.blockingReceive();
            System.out.printf("Agente %s . Recibido: %s\n", myAgent.
                getLocalName(), aclMessage.getContent());
        }
```

```
            @Override
            public int onEnd() {
                // Tiene que hacer una transicion mas para recibir el texto.
                return contador == 0 ? EV_VE_FIN : EV_VE_ENVIAR_TEXTO;
            }
        }, recibir_texto);

        registerLastState(new OneShotBehaviour() {
            @Override
            public void action() {
                System.out.printf("El agente emisor %s ha terminado\n", myAgent.
                    getLocalName());
                myAgent.doDelete();
            }

            @Override
            public int onEnd() {
                return -1;
            }
        }, fin);
    }


}
```

5.3. Código del comportamiento de la máquina de estados de recepción

```
package Ejercicio2;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.FSMBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;

// Documentation: https://jade.tilab.com/doc/api/jade/core/behaviours/
    FSMBehaviour.html
public class Ej2_Recibe_FSM extends FSMBehaviour {

    private final String recibir_numero = "Recibir numero";
    private final String recibir_texto = "Recibir texto";
    private final String enviar_texto = "Enviar texto";
    private final String fin = "Fin";

    private final int EV_VE_RECIBIR_TEXTO = 0;
    private final int EV_VE_ENVIAR_TEXTO = 1;
    private final int EV_VE_FIN = 2;


    private AID receiver;
    private int contador;
    private String text;

    public Ej2_Recibe_FSM(Agent a) {
        super(a);
        declareStates();
        declareTransitions();
    }

    private void declareTransitions() {
        // Ira al estado de fin desde enviar (porque empieza recibiendo)
        registerTransition(recibir_numero, recibir_texto, EV_VE_RECIBIR_TEXTO);
        registerTransition(recibir_texto, enviar_texto, EV_VE_ENVIAR_TEXTO);
        registerTransition(enviar_texto, recibir_texto, EV_VE_RECIBIR_TEXTO);
        registerTransition(enviar_texto, fin, EV_VE_FIN);
    }

    private void declareStates() {
        // 1st state is send nVeces to receiver
        registerFirstState(new OneShotBehaviour() {
            @Override
            public void action() {
                System.out.printf("Agente: %s . Estado inicial (Recibir numero)\
                    n", myAgent.getLocalName());
                ACLMessage aclMessage = myAgent.blockingReceive();

                contador = Integer.parseInt(aclMessage.getContent());
                receiver = aclMessage.getSender();
                System.out.printf("Agente %s . Contador=%d . Emisor=%s\n",
                    myAgent.getLocalName(), contador, receiver.getName());
            }

            @Override
            public int onEnd() {
                return EV_VE_RECIBIR_TEXTO;
            }
```

```
        }, recibir_numero);

        registerState(new OneShotBehaviour() {
            @Override
            public void action() {
                System.out.printf("Agente: %s. Estado: Recibir texto\n",
                    myAgent.getLocalName());
                ACLMessage aclMessage = myAgent.blockingReceive();
                if (text == null) {
                    text = aclMessage.getContent();
                }
                System.out.printf("Agente %s. Recibido: %s\n", myAgent.
                    getLocalName(), aclMessage.getContent());
            }

            @Override
            public int onEnd() {
                return EV_VE_ENVIAR_TEXTO;
            }
        }, recibir_texto);

        registerState(new OneShotBehaviour() {
            @Override
            public void action() {
                System.out.printf("Agente: %s. Estado: Enviar texto\n", myAgent
                    .getLocalName());
                ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
                aclMessage.addReceiver(receiver);
                aclMessage.setContent(text);

                myAgent.send(aclMessage);
                contador--;
                System.out.printf("Agente %s. Enviado. Quedan %d envios\n",
                    myAgent.getLocalName(), contador);
            }

            @Override
            public int onEnd() {
                return contador == 0 ? EV_VE_FIN : EV_VE_RECIBIR_TEXTO;
            }
        }, enviar_texto);

        registerLastState(new OneShotBehaviour() {
            @Override
            public void action() {
                System.out.printf("El agente receptor %s ha terminado\n",
                    myAgent.getLocalName());
                myAgent.doDelete();
            }

            @Override
            public int onEnd() {
                return -1;
            }
        }, fin);
    }


}
```