Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

[DEIM]

Multi-Agent Systems

Jordi Pascual – jordi.pascual@urv.cat

JADE agents and behaviours

MESIIA – Master's Degree in Computer Security Engineering and Artificial Intelligence
MAI - Master's Degree in Artificial Intelligence

Outline

1. JADE Agents
2. Behaviours
3. More resources

1. JADE Agents

To create a new JADE agent:

1. Create a new class that extends from *jade.core.Agent*

```
import jade.core.Agent;

public class PingAgent extends Agent { }
```

We will use as an example the **PingAgent** class that can be found in the **JadeExample** project in the *urv.imas* package

1. JADE Agents

2. Define the agent lifecycle. To do so, two methods must be overridden
   - **setup()**: called only once when initializing the Agent. Tasks:
     1. DF registration
     2. Start at least one Behaviour. You can start as many as you need
   - **takeDown()**: called when the platform kills the agent. Tasks:
     1. DF deregistration
     2. Close any open resources, like files, database access, etc.

On initializing, the **PingAgent** does register to the DF and creates a *WaitPingAndReplyBehaviour*

On terminating, it does log a message

1. JADE Agents

```
@Override
protected void setup() {
        // Registration with the DF
        DFAgentDescription dfd = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType("PingAgent");
        sd.setName(getName());
        sd.setOwnership("TILAB");
        dfd.setName(getAID());
        dfd.addServices(sd);
        try {
                DFService.register(this,dfd);
                WaitPingAndReplyBehaviour PingBehaviour = new  WaitPingAndReplyBehaviour(this);
                addBehaviour(PingBehaviour);
        } catch (FIPAException e) {
                myLogger.log(Logger.SEVERE, "Agent "+getLocalName()+" - Cannot register with DF", e);
                doDelete();
        }
}


@Override
protected void takeDown() {
        myLogger.log(Logger.INFO, "Agent " + getLocalName() + " terminating");
        super.takeDown();
}
```

2. Behaviours

- Are actually the real workers who do the jobs
- Agents act as a “container” of behaviours
- Each behaviour is a subprocess or thread
- Behaviours extend the **Behaviour** class or one of the available prototypes
- JADE has a **non-pre-emptive scheduler** which decides which behaviour runs at a given time
- Each time a behaviour is scheduled, a new **step** will execute
- At each **step** of the behaviour, the scheduler calls the **action()** method
- To determine if the behaviour has finished, the scheduler calls the **done()** method

2. Behaviours

Basic behaviour example:

- Two states
- First step runs STATE1
- Second step runs STATE2
- After STATE2, *finished = true*; will disable scheduling this behaviour

```
public class MyBehaviour extends Behaviour {

        private enum State { STATE1, STATE2}
        private State currentState = State.STATE1;
        private boolean finished = false;

        @Override
        public void action() {
                switch (currentState) {
                        case STATE1 -> {
                                myLogger.log(Logger.INFO, "Entering
                                first state");
                                currentState = State.STATE2;
                        }
                        case STATE2 -> {
                                myLogger.log(Logger.INFO, "Entering
                                second state");
                                finished = true;
                        }
                }
        }

        @Override
        public boolean done() {
                return finished;
        }
}
```

2. Behaviours: Simple Behaviours

The simplest Behaviour is the **SimpleBehaviour** class. It addresses **atomic** tasks

Some subclasses:

- **OneShotBehaviour**: it executes just once
- **CyclicBehaviour**: runs continuously forever
- **TickerBehaviour**: runs periodically
- **WakerBehaviour**: one shot task which is executed after a given timeout is elapsed

2. Behaviours: Simple Behaviours

One shot task example

```
public class SimpleAgent extends Agent {


        private final Logger myLogger = Logger.getMyLogger(getClass().getName());

        @Override
        protected void setup() {
                addBehaviour(new SimpleBehaviour() {
                        boolean finished = false;

                        @Override
                        public void action() {
                                myLogger.log(Logger.INFO, "Running just once");
                                finished = true;
                        }

                        @Override
                                public boolean done() {
                                return finished;
                        }
                });
        }
}
```

2. Behaviours: Simple Behaviours

One shot task after a timeout example

```
public class WakerSample extends Agent {

    private final Logger myLogger = Logger.getMyLogger(getClass().getName());


    @Override
    protected void setup() {
        addBehaviour(new WakerBehaviour(this, 250) {

            @Override
            protected void onWake() {
                myLogger.log(Logger.INFO, "One shot task after a 250 ms
                delay");
            }
        });
    }
}
```

2. Behaviours: Composite Behaviours

Can be seen as **containers of Behaviours**. They have an internal scheduler to handle the sub-behaviours execution

The main class is the **CompositeBehavior**

Some subclasses:

- **SequentialBehaviour**: executes the sub-behaviours in sequential order in a FIFO fashion
- **ParallelBehaviour**: runs all sub-behaviours asynchronously
- **FSMBehaviour**: a Finite State Machine defined by the user is used to decide which sub-behaviours are scheduled

2. Behaviours: Composite Behaviours

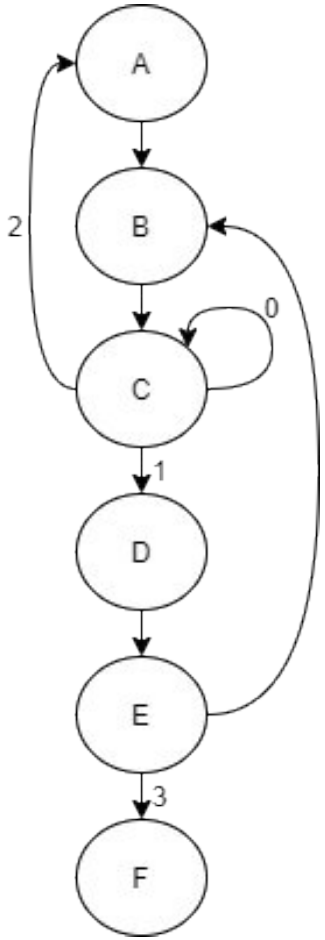
Sequential Behaviour example

```
public class SequentialAgent extends Agent {


        private final Logger myLogger = Logger.getMyLogger(getClass().getName());

        @Override
        protected void setup() {
                SequentialBehaviour sequentialBehaviour = new SequentialBehaviour();
                sequentialBehaviour.addSubBehaviour(new WakerBehaviour(this, 500) {

                        @Override
                        protected void onWake() {
                                myLogger.log(Logger.INFO, "First sub-behaviour");
                        }
                });
                sequentialBehaviour.addSubBehaviour(new WakerBehaviour(this, 1000) {

                        @Override
                        protected void onWake() {
                                myLogger.log(Logger.INFO, "Second sub-behaviour");
                        }
                });
                addBehaviour(sequentialBehaviour);
        }
}
```

2. Behaviours: Composite Behaviours

FSM Behaviour example

1. Register a single Behaviour as the **initial state** (*registerFirstState*)
2. Register one or more Behaviours as the **final states** of the FSM (*registerLastState*)
3. Register one or more Behaviours as the **intermediate states** of the FSM (*registerState*)
4. Register the **transitions** between states (*registerTransition* or *registerDefaultTransition*)



2. Behaviours: Composite Behaviours

- Additional examples are available at the *urv.imas.behaviours* package in the **JadeExample** project
- Example Behaviours are created as inner classes. They can also be created in separate classes

```
protected void setup() {
        FSMBehaviour fsm = new FSMBehaviour(this) {
                public int onEnd() {
                        System.out.println("FSM behaviour completed.");
                        myAgent.doDelete();
                        return super.onEnd();
                }};
        // Register state A (first state)
        fsm.registerFirstState(new NamePrinter(), STATE_A);
        // Register state B
        fsm.registerState(new NamePrinter(), STATE_B);
        // Register state C
        fsm.registerState(new RandomGenerator(3), STATE_C);
        // Register state D
        fsm.registerState(new NamePrinter(), STATE_D);
        // Register state E
        fsm.registerState(new RandomGenerator(4), STATE_E);
        // Register state F (final state)
        fsm.registerLastState(new NamePrinter(), STATE_F);
        // Register the transitions
        fsm.registerDefaultTransition(STATE_A, STATE_B);
        fsm.registerDefaultTransition(STATE_B, STATE_C);
        fsm.registerTransition(STATE_C, STATE_C, 0);
        fsm.registerTransition(STATE_C, STATE_D, 1);
        fsm.registerTransition(STATE_C, STATE_A, 2);
        fsm.registerDefaultTransition(STATE_D, STATE_E);
        fsm.registerTransition(STATE_E, STATE_F, 3);
        fsm.registerDefaultTransition(STATE_E, STATE_B);
        addBehaviour(fsm);
}
```

3. More resources

- FIPA Protocols
- JADE Javadoc
- JADE Guides
- JADE Maven Setup for Beginners