Laboratorio 2 Comportamientos

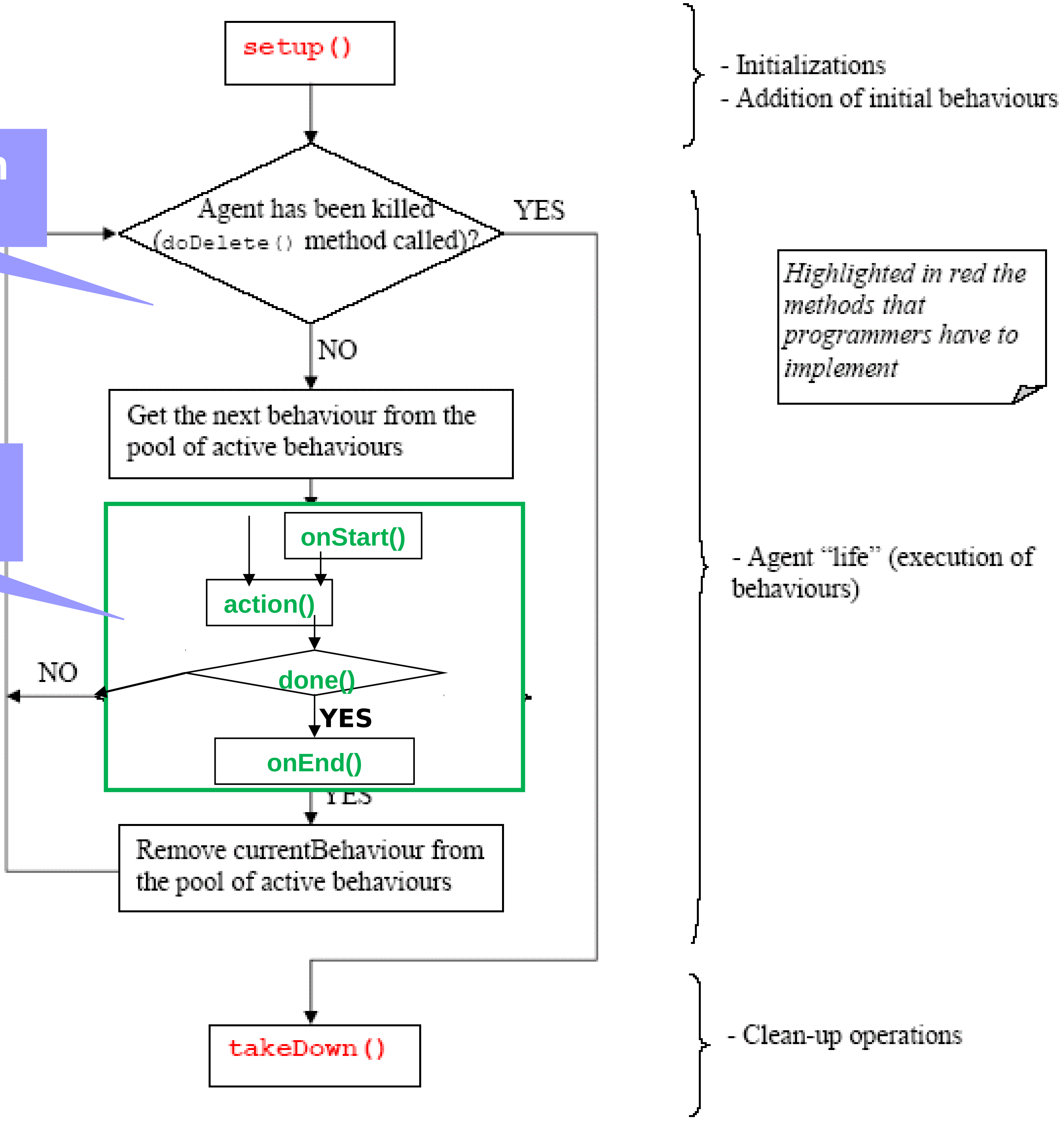
Técnicas Avanzadas de Inteligencia Artificial

Dpt. Lenguajes y Sistemas Informáticos.

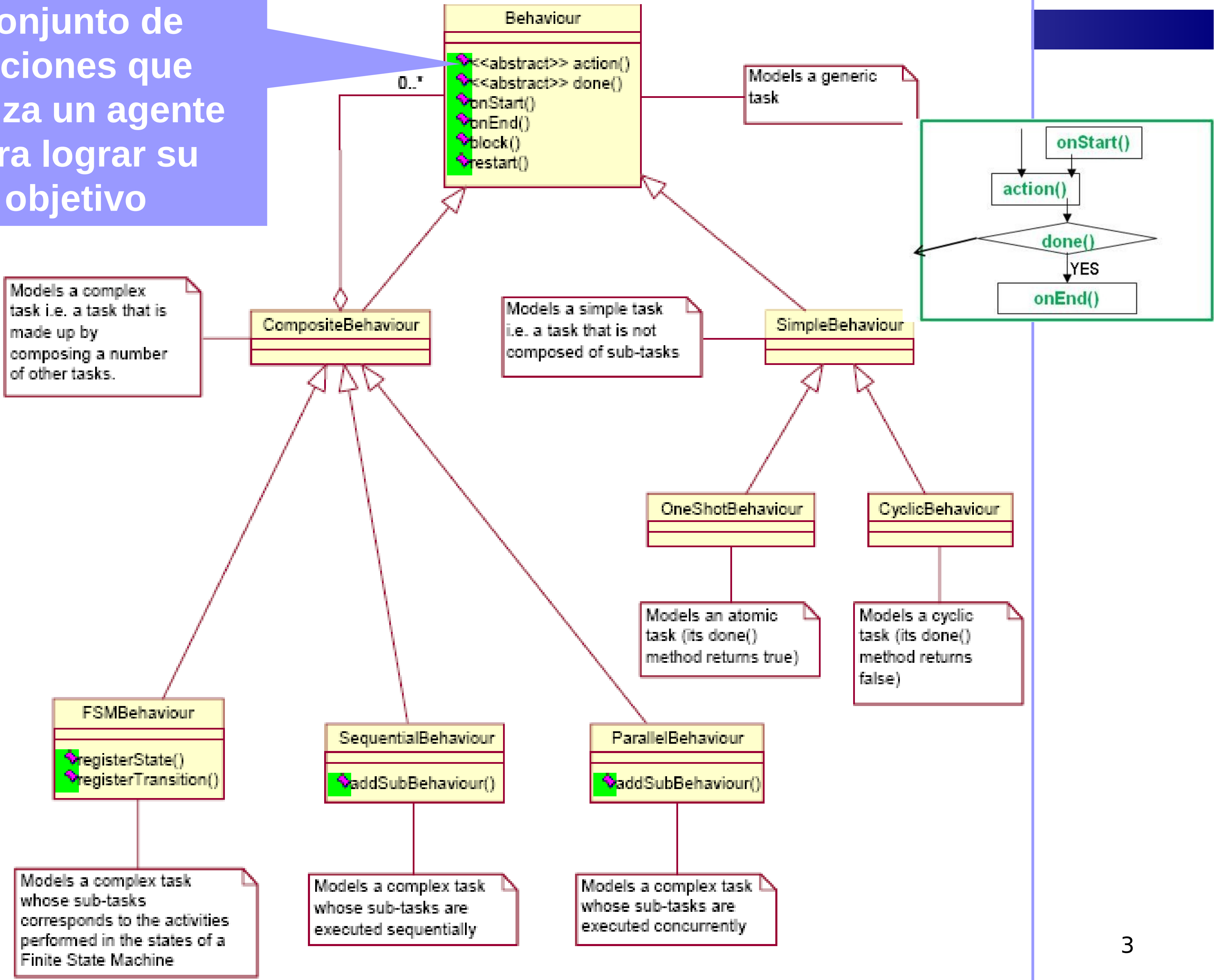
FISS. UPV-EHU

Hilo de ejecución de un agente

Ejecución del comportamiento


setup()
- Initializations
- Addition of initial behaviours
Agent has been killed (doDelete() method called)?
YES
NO
Get the next behaviour from the pool of active behaviours
onStart()
action()
done()
NO
YES
onEnd()
Remove currentBehaviour from the pool of active behaviours
Highlighted in red the methods that programmers have to implement
- Agent "life" (execution of behaviours)
takeDown()
- Clean-up operations

Conjunto de acciones que realiza un agente para lograr su objetivo



Behaviour
<<abstract>> action()
<<abstract>> done()
onStart()
onEnd()
block()
restart()
0..*
Models a generic task
onStart()
action()
done()
YES
onEnd()
Models a complex task i.e. a task that is made up by composing a number of other tasks.
CompositeBehaviour
Models a simple task i.e. a task that is not composed of sub-tasks
SimpleBehaviour
OneShotBehaviour
CyclicBehaviour
Models an atomic task (its done() method returns true)
Models a cyclic task (its done() method returns false)
FSMBehaviour
registerState()
registerTransition()
SequentialBehaviour
addSubBehaviour()
ParallelBehaviour
addSubBehaviour()
Models a complex task whose sub-tasks corresponds to the activities performed in the states of a Finite State Machine
Models a complex task whose sub-tasks are executed sequentially
Models a complex task whose sub-tasks are executed concurrently

ÍNDICE

3. Comportamientos- Behaviours

Actividades a realizar

- Crear paquete: **examples.behaviours**
- Importar clases: **SimpleAgent.java, TimeAgent.java, ComplexAgent.java y FSMAgent.java**
- Hay cuatro ejercicios:

  7. SimpleAgent.bat **8. TimeAgent.bat**

  9. ComplexAgent.bat **10. FSMAgent.bat**

- Crear los cuatro interfaces para ejecución o trabajaremos directamente con los .bat.

3.1. SimpleAgent

```
protected void setup() {
   System.out.println("Agente "+getLocalName()+" ha empezado.");

   // Add the CyclicBehaviour
   addBehaviour(new CyclicBehaviour(this) {
     public void action() {
       System.out.println("Ciclando");
     }
   });

   // Add the generic behaviour
   addBehaviour(new FourStepBehaviour());
}
```

3.1. SimpleAgent

```
private class FourStepBehaviour extends Behaviour {
   private int step = 1;

   public void action() {
     switch (step) {
     case 1:
       // Perform operation 1: print out a message
   …

    public boolean done() {
          return step == 5;
       }
```

3.1. SimpleAgent

```
public void action() {
    switch (step) {
    case 1: System.out.println("Operacion 1");      break;
    case 2: System.out.println("Operacion 2. Incluyo un comportamiento one-shot.");
        myAgent.addBehaviour(new OneShotBehaviour(myAgent) {
            public void action() {
              System.out.println("One-shot");
              }
            });
            break;
   case 3: System.out.println("Operacion 3");      break;
   case 4: System.out.println("Operacion 4");      break;
   }
   step++;
  }
```

Activar Dummy e Introspector

3.2. TimeAgent

```
public class TimeAgent extends Agent {

  protected void setup() {
    System.out.println("Agente

  // Add the TickerBehaviour (p      1 sec)
    addBehaviour(new TickerBehaviour(this, 1000) {
      protected void onTick() {
    …
 // Add the WakerBehaviour (wakeup-time 10 secs)
    addBehaviour(new WakerBehaviour(this, 10000) {
      protected void onWake() {
```

Comportamiento Cíclico- Cada X tiempo realiza una acción **-> onTick()**

X miliseg.

Comportamiento Único- Pasado X tiempo realiza una sólo vez la acción **-> onWake()**

X miliseg.

3.3. ComplexAgent

```
public class ComplexBehaviourAgent extends Agent {

class SingleStepBehaviour extends OneShotBehaviour {
    private String myStep;

    public SingleStepBehaviour(Agent a, String step) {
      super(a);
      myStep = step;
    }
    public void action() {
      System.out.println("Agente "+getName()+": Paso "+myStep);
    }
    public int onEnd() {
        reset();
                    System.out.println("Termina el comportamiento one-shot …
     “));
        return super.onEnd();
     }
}
```

3.3. ComplexAgent

```
public class ComplexBehaviourAgent extends Agent {
…
 protected void setup() {
   SequentialBehaviour myBehaviour1 = new SequentialBehaviour(this) {
     public int onEnd() {
       reset();
       System.out.println("Termina el comportamiento secuencial “…);
       return super.onEnd();
     }
   };
   myBehaviour1.addSubBehaviour(new SingleStepBehaviour(this, "1.1"));
   myBehaviour1.addSubBehaviour(new SingleStepBehaviour(this, "1.2"));
   myBehaviour1.addSubBehaviour(new SingleStepBehaviour(this, "1.3"));
   addBehaviour(myBehaviour1);

   SequentialBehaviour myBehaviour2 = new SequentialBehaviour(this);
   SequentialBehaviour myBehaviour2_1 = new SequentialBehaviour(this);
   SequentialBehaviour myBehaviour2_2 = new SequentialBehaviour(this);
    …
```

3.3. ComplexAgent

```
myBehaviour1.addSubBehaviour(new SingleStepBehaviour(this, "1.1"));
myBehaviour1.addSubBehaviour(new SingleStepBehaviour(this, "1.2"));
myBehaviour1.addSubBehaviour(new SingleStepBehaviour(this, "1.3"));
addBehaviour(myBehaviour1);

SequentialBehaviour myBehaviour2 = new SequentialBehaviour(this);
SequentialBehaviour myBehaviour2_1 = new SequentialBehaviour(this);
SequentialBehaviour myBehaviour2_2 = new SequentialBehaviour(this);

myBehaviour2_1.addSubBehaviour(new SingleStepBehaviour(this, "2.1.1"));
myBehaviour2_1.addSubBehaviour(new SingleStepBehaviour(this, "2.1.2"));
myBehaviour2_1.addSubBehaviour(new SingleStepBehaviour(this, "2.1.3"));

myBehaviour2_2.addSubBehaviour(new SingleStepBehaviour(this, "2.2.1"));
myBehaviour2_2.addSubBehaviour(new SingleStepBehaviour(this, "2.2.2"));
Behaviour b = new SingleStepBehaviour(this, "2.2.3");
myBehaviour2_2.addSubBehaviour(b);

myBehaviour2.addSubBehaviour(myBehaviour2_1);
myBehaviour2.addSubBehaviour(myBehaviour2_2);
myBehaviour2.addSubBehaviour(new SingleStepBehaviour(this, "2.3"));
myBehaviour2.addSubBehaviour(new SingleStepBehaviour(this, "2.4"));
myBehaviour2.addSubBehaviour(new SingleStepBehaviour(this, "2.5"));
addBehaviour(myBehaviour2);
```

Visualizarlo en la pizarra

3.4. FSMAgent

```
protected void setup() {
    FSMBehaviour fsm = new FSMBehaviour(this) {
     public int onEnd() {
        System.out.println("FSM comportamiento completado.");
        myAgent.doDelete();
        return super.onEnd();
     }
     };
fsm.registerFirstState(new NamePrinter(), STATE_A);
…
fsm.registerState(new RandomGenerator(4), STATE_E);
…
fsm.registerLastState(new NamePrinter(), STATE_F);
…
fsm.registerDefaultTransition(STATE_B, STATE_C);
fsm.registerTransition(STATE_C, STATE_C, 0);
…
```

3.4. FSMAgent

```
private  class NamePrinter extends OneShotBehaviour {
    public void action() {
      System.out.println("Ejecutando comportamiento
                                                  "+getBehaviourName());
    }
}
```

3.4. FSMAgent

```
private class RandomGenerator extends NamePrinter{

    private int maxExitValue;
    private int exitValue;

     private RandomGenerator(int max) {
          super();
          maxExitValue = max;
     }
     public void action() {
          super.action();
          exitValue = (int) (Math.random() * maxExitValue);
          System.out.println("Valor obtenido es "+exitValue);
     }
     public int onEnd() {
          return exitValue;
     }
}
```