Implementación de Agentes

- Comunicación
- Control de flujo
- Clase Agente
- Creación de Comportamientos

Implementación

Modelo de Programación

Estructura de un agente

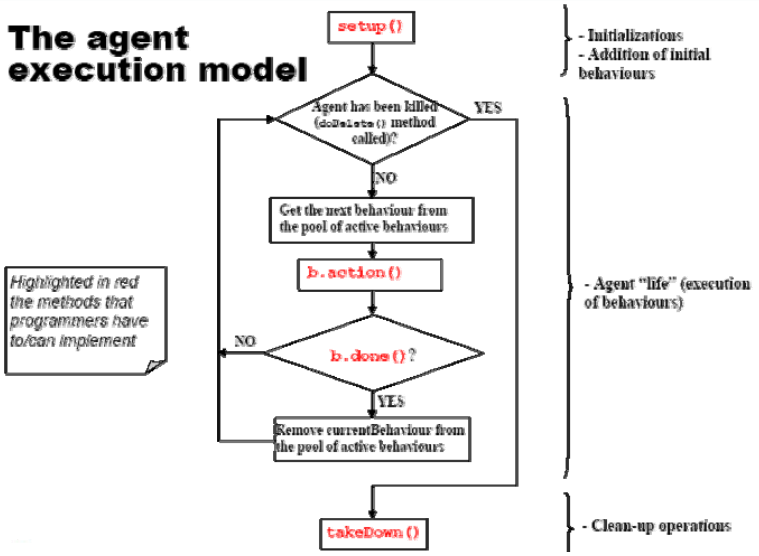
- Un agente JADE es una instancia de una clase de JAVA definida por el usuario que extiende la clase AGENTE básica (en el paquete `jade.core`).

```
public class MI_AGENTE extends Agent { ...
```

- El ciclo de vida de un agente JADE sigue el de FIPA
- El método *setup* es donde se inicializa el agente
- Las tareas que realiza un agente en JADE se estructuran en comportamientos
  - Subclases de la clase `Behaviour` en el paquete `jade.core.behaviours`
- super(): método para invocar al constructor del padre
- super. : para acceder a datos del padre
- getAMS() → the AID of the AMS

Implementación

Control de Flujo



Implementación

Modelo de Programación

Comportamientos

- Comportamiento: Básicamente es un ***Gestor de Eventos***: cómo un agente reacciona a un *evento* (recepción de un mensaje o interrupción de un *timer*)
- Código del ***Gestor de Eventos***: método **action**.
- Cada comportamiento se planifica según un algoritmo *round robin*.
- Ejemplos de Comportamientos incluidos en JADE:
  - SimpleBehaviour
  - OneShotBehaviour
  - WakerBehaviour
  - CyclicBehaviour
  - TickerBehaviour
  - FSMBehaviour
  - ReceiverBehaviour
  - ParallelBehaviour
  - SequentialBehaviour

Método **onWake**():
- Debe ser implementado.
- Es ejecutado después de un *timeout* dado.
- Después de esa ejecución el comportam. se completa.

Implementación

Modelo de Programación

Comportamientos

- Comportamiento: Básicamente es un ***Gestor de Eventos***: cómo un agente reacciona a un *evento* (recepción de un mensaje o interrupción de un *timer*)
- Código del ***Gestor de Eventos***: método **action**.
- Cada comportamiento se planifica según un algoritmo *round robin*.
- Ejemplos de Comportamientos incluidos en JADE:
  - SimpleBehaviour
  - OneShotBehaviour
  - WakerBehaviour
  - CyclicBehaviour
  - TickerBehaviour
  - FSMBehaviour
  - ReceiverBehaviour
  - ParallelBehaviour
  - SequentialBehaviour

Método **onTick**():
- Ejecutado periódicam. con un período dado.
- El comportam. se ejecuta para siempre a menos que su método stop() se ejecute

Implementación

Modelo de Programación

Comportamientos

- Extensiones de la clase **jade.core.behaviours.Behaviour**
- Métodos de la clase *Agent*:
  - **addBehaviour()**: Añade al agente la instancia de comportamiento que recibe como argumento
  - **removeBehaviour()**: Elimina el comportamiento de la lista de comportamientos
- Métodos de la clase *Behaviour*:
  - Fundamentales:
    - **action()**: Lo que hace realmente el agente. Se cambia de Comportamiento sólo cuando éste método del comportamiento planificado actualmente finaliza.
    - **done()**: Si ha acabado o no el agente
  - Adicionales:
    - **onStart()**: se invoca una vez sólo antes de la primera ejecución del método **action()**.
    - **onEnd()**: se invoca una vez sólo después de que el método **done()** devuelve **true**
- Variables de la clase *Behaviour* :
  - myAgent: ptr. al agente que está ejecutando el comportamiento

One Shot Behaviour

```
public class HelloOneShot extends Agent {
    protected void setup() {
        addBehaviour( new myBehaviour( this ));
    }

    class myBehaviour extends OneShotBehaviour {
        public myBehaviour( Agent a ) {
            super(a);
        }

        public void action() {
            System.out.println("Hello world. ");
            System.out.println("My name is " + getLocalName() );
        }
    }
}
```

Cyclic Behaviour

```
public class HelloCyclic extends Agent {
    protected void setup() {
        addBehaviour( new myBehaviour( this ));
    }

    class myBehaviour extends CyclicBehaviour {
        public myBehaviour( Agent a ) {
            super(a);
        }

        public void action() {
            System.out.println("Hello world. ");
            System.out.println("My name is " + getLocalName() );
        }
}
}
```

Waker Behaviour

```
public class Hello3 extends Agent {
    static long t0;
    protected void setup() {
        addBehaviour( new myBehaviour( this, 5000 ));
        t0 = System.currentTimeMillis();
    }

    class myBehaviour extends WakerBehaviour {
        public myBehaviour( Agent a, long dt ) {
            super(a, dt);
        }

        public void onWake() {
            System.out.println( "Start: "  + (System.currentTimeMillis()-t0) );
            System.out.println("Hello world. ");
            System.out.println("My name is " + getLocalName() );
        }
    }
}
```

Ticker behaviour

```
public class Hello2 extends Agent {
    static long t0;
    protected void setup() {
        addBehaviour( new myBehaviour( this, 5000 ));
        t0 = System.currentTimeMillis();
    }

    class myBehaviour extends TickerBehaviour {
        public myBehaviour( Agent a, long dt ) {
            super(a, dt);
        }

        public void onTick() {
            System.out.println( "Start: "  + (System.currentTimeMillis()-t0) );
            System.out.println("Hello world. ");
            System.out.println("My name is " + getLocalName() );
        }
    }
}
```