BDI4JADE (I)

Sergio Alvarez

salvarez@cs.upc.edu

SID2022

https://kemlg.upc.edu

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA

JADE

Implementando BDI

- La complejidad de implementar razonadores basados en lógicas modales es alta
  - Para algunas lógicas como LTL es relativamente sencillo (e.g. autómatas de Büchi)
- Para poder programar agentes proactivos que puedan funcionar en tiempo real se suele simplificar la lógica y el ciclo de razonamiento
- Opciones:
  - JASON: http://jason.sourceforge.net/wp/description/
  - SPADE-BDI: https://spade-bdi.readthedocs.io/en/latest/readme.html
  - 2-APL: http://apapl.sourceforge.net/
  - JACK: http://www.agent-software.com.au/media/documentation/jack/jackdocs.html

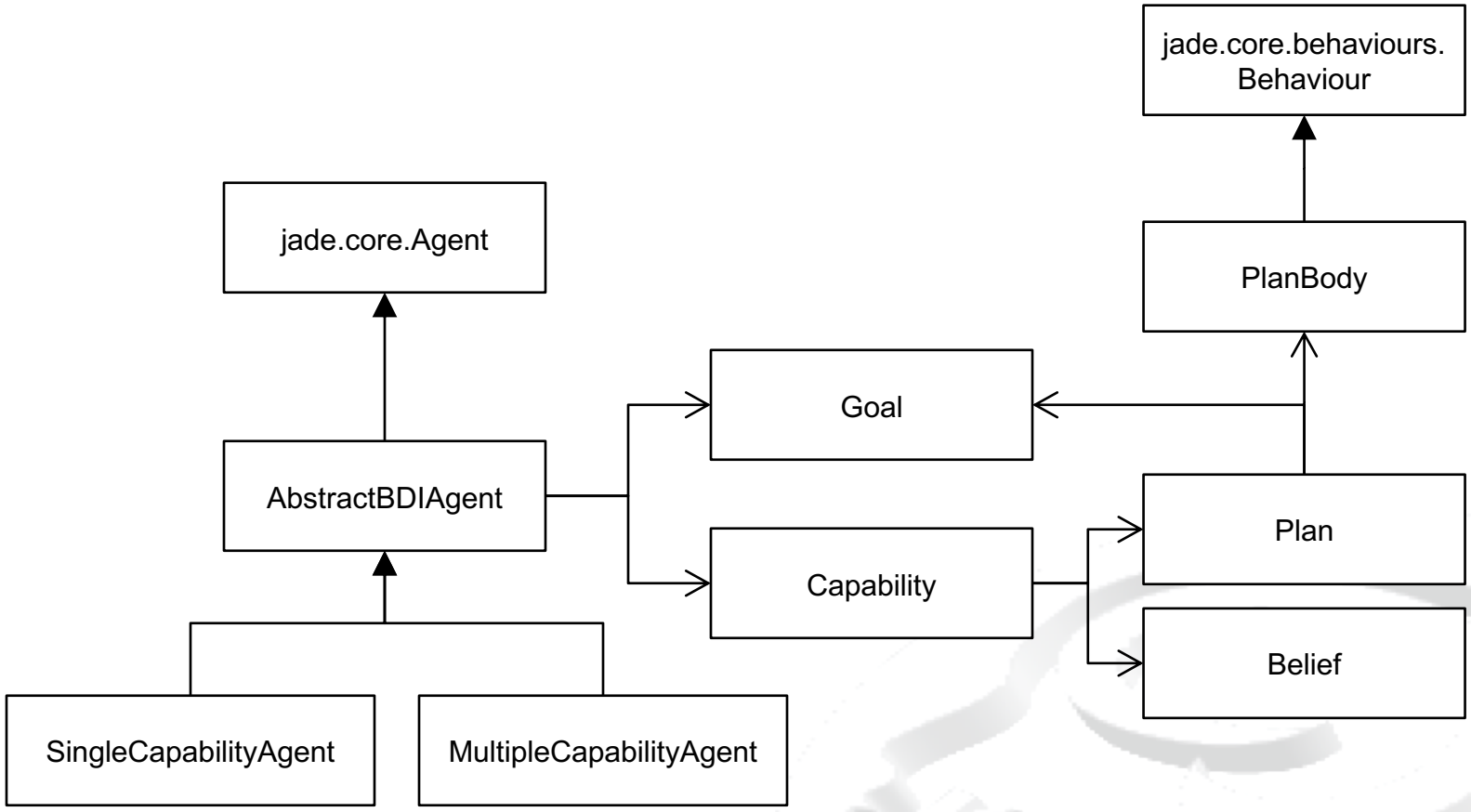
BDI4JADE

- "BDI4JADE: a BDI layer on top of JADE", *Ingrid Nunes, Carlos J.P. de Lucena, and Michael Luck*
  [https://kclpure.kcl.ac.uk/portal/files/30740370/Nunes_BDI4JADE_2011.pdf]
- Basado 100% en Java en lugar de lenguajes específicos
- Utiliza elementos básicos de JADE: Agent, Behaviour
- Jarfiles:
  - https://github.com/ingridnunes/bdi4jade/releases/download/v2.0/bdi4jade.jar
  - https://dlcdn.apache.org//commons/logging/binaries/commons-logging-1.2-bin.zip

Simplificaciones sobre BDI teórico

- Las *intentions* son siempre *goals (desires)* y son implícitas (no se exponen).
- El ciclo de razonamiento no planifica a partir de los *goals*, sino que los *plans* son parte del diseño e implementación del agente.
- Cada agente tiene asociado un conjunto de *capabilities*, que son conjuntos de *beliefs* y *plans*
- Cada plan tiene asociado el *goal* que puede cumplir y un *plan body*, que es un behaviour de JADE

BDI4JADE


jade.core.behaviours.
Behaviour
jade.core.Agent
PlanBody
Goal
AbstractBDIAgent
Plan
Capability
Belief
SingleCapabilityAgent
MultipleCapabilityAgent

Creando un goal sencillo

- Un goal se crea implementando la interfaz `bdi4jade.goal.Goal`
- Puede tener los atributos y métodos que queramos (si hace falta parametrizar)

```
public class HelloGoal implements Goal {
   private final String text;

   public HelloGoal(String text) {
      this.text = text;
   }

   public String getText() {
      return text;
   }
}
```

Creando un plan body

- Un plan body es un **behaviour** con acceso a la base de beliefs de su capability y a su goal actual y que cumple:

```
done() == (getEndState() != null)
```

- BDI4JADE incluye implementaciones para ejecutar planes secuenciales, paralelos o con FSM pero podemos simplemente extender `AbstractPlanBody` y sobreescribir `Behaviour.action()`

```
public class HelloPlanBody extends AbstractPlanBody {
   @Override
   public void action() {
      HelloGoal goal = (HelloGoal) getGoal();
      System.out.println("Hello " + goal.getText());
      setEndState(Plan.EndState.SUCCESSFUL);
   }
}
```

Creando un plan

- La clase de plan abstracta es `AbstractPlan` pero `DefaultPlan` será suficiente para la mayoría de casos
- La constructora puede recibir una subclase de `Goal` (o `GoalTemplate`) y una subclase de `PlanBody`

```
Plan plan = new DefaultPlan(HelloGoal.class,
                            HelloPlanBody.class);
```

Crear un agente BDI sencillo

- Si sólo queremos definir una *capability* para nuestro agente hay que extender la clase `bdi4jade.core.SingleCapabilityAgent`
- En la constructora hay que añadir:
  - Los goals iniciales
  - Los plans

```
public class TestBDIAgent extends SingleCapabilityAgent {
    public TestBDIAgent() {
        Plan plan = new DefaultPlan(HelloGoal.class,
                HelloPlanBody.class);
        addGoal(new HelloGoal("there"));
        getCapability().getPlanLibrary().addPlan(plan);
    }
}
```

Ejercicio I

- Crea un agente que tenga como objetivo escribir el mensaje "`Hello world`"
- Añádelo a la plataforma y comprueba que funciona
- Modifica el `PlanBody` para que escriba, para `N=[0,9]`, "`Hello world: N`" (sin bucles)
- Añade el mismo goal dos veces (`addGoal`) y observa el comportamiento

Replanificación (I)

- Bajo ciertas condiciones, el agente replanificará siguiendo el ciclo de razonamiento:
  - Si el PlanBody acaba con `EndState == FAILED`

```
public void action() {
    if (planHasFailed) {
        setEndState(Plan.EndState.FAILED);
    } else {
        setEndState(Plan.EndState.SUCCESSFUL);
    }
}
```

  - En este caso, se escogerá otro Plan que cumpla con el mismo objetivo que tenía asignado el plan fallido

Replanificación (II)

- Bajo ciertas condiciones, el agente replanificará siguiendo el ciclo de razonamiento
  - Si el Plan retorna false en la sobreescritura de `isContextApplicable`:

```
Plan plan = new DefaultPlan(HelloWorldGoal.class,
                            HelloWorldPlanBody.class) {
    @Override
    public boolean isContextApplicable(Goal goal) {
        // Se puede ejecutar este plan en este contexto?
        return false;
    }
};
```

  - Esta comprobación se hace antes de elegir plan, nunca durante la ejecución del plan body
  - Si retorna `false`, se escogerá otro plan, si existe

Replanificación (III)

- Bajo ciertas condiciones, el agente replanificará siguiendo el ciclo de razonamiento
  - Si un goal desaparece del agente:

```
public void action() {
    AbstractBDIAgent agent = (AbstractBDIAgent) getAgent();
    agent.dropGoal(agent.getGoals().iterator().next());
}
```

  - Al desaparecer el objetivo, se interrumpe el plan body actual y tampoco habrá replanificación para dicho objetivo
  - Proactividad: también se pueden añadir objetivos, dinámicamente, desde los action()

Ejercicio II

- Crea un segundo `PlanBody` para el objetivo `HelloGoal` que escriba el mensaje `"Bye world: N"` para N=[0,9]
- Añade el `Plan` basado en este `PlanBody` al agente
- Comprueba cuántos `PlanBody` se llegan a ejecutar
- Haz que el primer `PlanBody` que se ejecuta tenga como `EndState FAILED`, ¿qué ocurre ahora?

Knowledge Engineering and Machine Learning Group

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Sergio Alvarez

salvarez@cs.upc.edu

2022