BDI4JADE (II)

Sergio Alvarez

salvarez@cs.upc.edu

SID2022

https://kemlg.upc.edu

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA

JADE

Usando beliefs

- Cada capability incluye una base de creencias (BeliefBase):

```
// this es un SingleCapabilityAgent
Capability cap = this.getCapability();
BeliefBase beliefBase = cap.getBeliefBase();
```

- Los Beliefs incluidos en BDI4JADE no son persistentes, pero se puede extender la clase base `Belief` para guardarlos
- Los Beliefs pueden contener cualquier tipo de objeto `(K, V)`
- Clases básicas disponibles:
  - `TransientBelief<K,V>`: objeto a valor
  - `TransientBeliefSet<K,V>`: objeto a conjunto de valores
  - `TransientPredicate<K>`: objeto a booleano

Usando beliefs

```
// TransientBelief
Belief currentPosition = new TransientBelief("currentPosition",
                                             new int[]{0 , 0});

// TransientBeliefSet
Set objects = new HashSet();
objects.add("a");
objects.add("b");
objects.add("c");
Belief hasObjects = new TransientBeliefSet("hasObjects", objects);

// TransientPredicate
Belief playerAlive = new TransientPredicate("playerAlive", true);

// Add beliefs to capability's belief base
beliefBase.addBelief(currentPosition);
beliefBase.addBelief(hasObjects);
beliefBase.addBelief(playerAlive);

// Get all beliefs from all capabilities
System.out.println(this.getBeliefs());
```

Usando beliefs

- El *name* (de tipo K) identifica el belief dentro de un BeliefSet
- Si intentamos añadir dos beliefs con el mismo *name* (independientemente del valor), el agente fallará con una excepción:

```
beliefBase.addBelief(new TransientPredicate("playerAlive", true));
beliefBase.addBelief(new TransientPredicate("playerAlive", false));
```

> An undeclared exception was thrown - Caused by: Belief already exists exception: playerAlive = false

- Para actualizar un belief podemos usar `BeliefSet.updateBelief` o `BeliefSet.addOrUpdateBelief`:

```
Belief playerAlive = new TransientPredicate("playerAlive", true);
beliefBase.addOrUpdateBelief(playerAlive);
beliefBase.updateBelief("playerAlive", false);
```

Ejercicio III

- Crea un agente `Player` (`SingleCapabilityAgent`) que:
  - Reciba cuatro argumentos numéricos en la creación del agente, que llamaremos `CC`, `CD`, `DC` y `DD`
  - Su capability declare dos beliefs:
    - 1 belief con nombre `C` y con un valor lista con los valores `CC` y `CD`
    - 1 belief con nombre `D` y con un valor lista con los valores `DC` y `DD`
    - 1 belief set con nombre `history` y con valor vacío
  - Tenga un goal `minimizePlay`
  - El plan para resolver `minimizePlay` escriba por consola:
    - `C` si el valor mínimo del belief `C` es menor o igual al valor mínimo del belief `D`
    - `D` en cualquier otro caso
  - Este plan debe guardar el valor decidido (`C` / `D`) en `history`

Crear jerarquías de objetivos

- Los goals se pueden estructurar usando las clases `ParallelGoal` y `SequentialGoal`:

```
Set goals = new HashSet();
goals.add(new HelloGoal("1"));
goals.add(new HelloGoal("2"));
goals.add(new HelloGoal("3"));
goals.add(new HelloGoal("4"));
ParallelGoal parallelGoal = new ParallelGoal(goals);
this.addGoal(parallelGoal);
```

- `ParallelGoal` añade intentions para todos los subgoals
  - Sus planes se ejecutan concurrentemente
  - El `ParallelGoal` se cumple sólo cuando se cumplen **todos** los subgoals
- `SequentialGoal` añade un subgoal como intention sólo cuando el anterior se ha cumplido con éxito

Objetivos basados en creencias

- Podemos especificar un belief como objetivo del agente:
  - `BeliefValueGoal(name, value)`
  - `BeliefSetHasValueGoal(name, value in Set)`
  - `BeliefPresentGoal/BeliefNotPresentGoal(name)`
  - `BeliefNotNullValueGoal(name)`
  - `PredicateGoal(name, boolean)`
- Para crear planes que mapeen estos objetivos podemos usar los métodos estáticos de la clase `GoalTemplateFactory`:
  - `hasBeliefValue(name, value)`
  - `hasBeliefOfValueType(name, valueClass)`
  - `hasValueOfTypeInBeliefSet(name, valueClass)`
- El `PlanBody` debería instanciar `BeliefGoalPlanBody`
  - Más eficiente al ahorrar comprobaciones en el `BeliefSet`
  - Nos podemos saltar el `endState = SUCCESSFUL`

Objetivos basados en creencias

- Código en la constructora del agente:

```
// 1. Goal based on belief (predicate -> boolean value)
this.addGoal(new PredicateGoal("readyToStart", true));
// 2. Create goal template for matching all readyToStart predicate goals
//    with any value (true or false)
GoalTemplate goalTemplate = GoalTemplateFactory.hasBeliefValueOfType(
        "readyToStart", Boolean.class);
// 3. Create plan
Plan beliefPlan = new DefaultPlan(goalTemplate, ReadyPlanBody.class);
planLibrary.addPlan(beliefPlan);
```

- Código del PlanBody:

```
public class ReadyPlanBody extends BeliefGoalPlanBody {
    @Override
    protected void execute() {
        PredicateGoal goal = (PredicateGoal) this.getGoal();
        this.getBeliefBase().addOrUpdateBelief(
                new TransientPredicate("readyToStart", true)
        );
    }
}
```

Ejercicio IV

- En `Player`, reemplaza el goal `minimizePlay` por un `GoalTemplate` de manera que el objetivo sea conseguir que el belief `history` no esté vacío
  - Revisad el Javadoc de `GoalTemplate` para elegir el método estático correcto
  - Comprueba que el funcionamiento es el mismo que en el ejercicio IV
- Modifica `Player` para que el goal añadido sea un goal compuesto (decide si paralelo o secuencial) para separar el comportamiento en dos subobjetivos cuyos planes sean:
  - Decidir la jugada a hacer y registrarla en un belief
  - Obtener la jugada de ese belief y escribirla por consola

Objetivos basados en mensajes

- Podemos definir planes basados en un `MessageTemplate`:

```
Plan replyPlan =
        new DefaultPlan(MessageTemplate.MatchOntology("some-ontology"),
                ReplyPlanBody.class);
planLibrary.addPlan(replyPlan);
```

- El `PlanBody` recibe el mensaje usando la anotación `Parameter` de BDI4JADE:

```
public class ReplyPlanBody extends AbstractPlanBody {
    private ACLMessage msgReceived;

    @Override
    public void action() {
        ACLMessage reply = msgReceived.createReply();
        reply.setContent("reply");
        this.myAgent.send(reply);
        setEndState(Plan.EndState.SUCCESSFUL);
    }

    @Parameter(direction = Parameter.Direction.IN)
    public void setMessage(ACLMessage msgReceived) {
        this.msgReceived = msgReceived;
    }
}
```

Ejercicio V

- Modifica `Player` y el belief `history` para que cumpla con los siguientes objetivos y subobjetivos (definiendo una jerarquía de goals):
  - Registrarse en el DF con `serviceType == "player"`
  - Proponer jugada
    - Encontrar en el DF agentes con `serviceType == "player"` y registrar uno al azar
    - Decidir y registrar una jugada usando el criterio del ejercicio III
    - Enviar al agente registrado un mensaje INFORM con ontología "`play`", con la jugada registrada (`content == C o content == D`)
    - Esperar la respuesta, guardarla en el belief `history` y escribirla por consola
  - Esperar jugada
    - Recibir mensajes INFORM con ontología "`play`" y guardar la jugada en el `history`
    - Decidir y registrar una jugada (también en el `history`) usando el siguiente criterio:
      - La jugada recibida es `C`: si el valor de `CC` <= el valor de `DC`, entonces elegir `C`, en caso contrario, elegir `D`
      - La jugada recibida es `D`: si el valor de `CD` <= el valor de `DD`, entonces elegir `C`, en caso contrario, elegir `D`
    - Enviar una respuesta INFORM con ontología "`play`" con la jugada registrada
- Tened en cuenta que puede haber múltiples conversaciones, por lo que tendréis que definir los beliefs de manera que se puedan identificar las jugadas que guardáis (usando `AIDs` o `ACLMessage.getConversationId()`)

Monitorización de objetivos

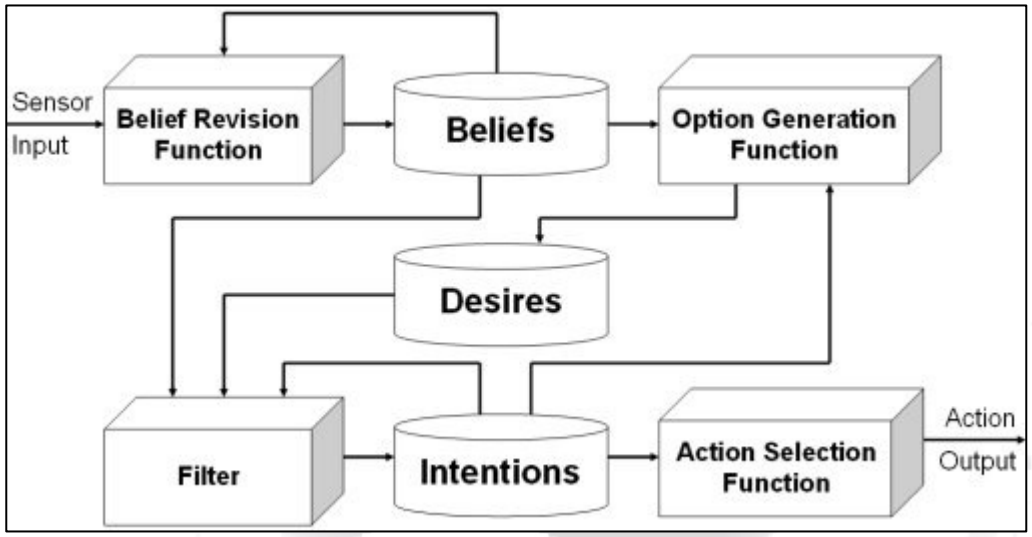
- Es posible definir un `GoalListener` y asociarlo al agente para recibir en tiempo real información sobre la consecución de los goals:

```
this.addGoalListener(new GoalListener() {
    @Override
    public void goalPerformed(GoalEvent goalEvent) {
        System.out.println(goalEvent);
    }
});
```

Controlar el ciclo BDI

Aunque BDI4JADE usa un ciclo BDI simplificado, es posible modificar las funciones de:

- Revisión de beliefs
- Generación de objetivos activos (options)
- Generación de intenciones activas (filtering)
- Selección de plan bodies (action selection)



Controlar el ciclo BDI

Aunque BDI4JADE usa un ciclo BDI simplificado, es posible modificar las funciones de:

- Revisión de beliefs
- Generación de objetivos activos (options)
- Generación de intenciones activas (filtering)
- Selección de plan bodies (action selection)

Comportamiento por defecto:

- No modifica
- Todos los objetivos contextualmente válidos
- Todos los objetivos generan intenciones
- Plan body aleatorio entre todos los que cumplen cada intención

Controlar el ciclo BDI

Aunque BDI4JADE usa un ciclo BDI simplificado, es posible modificar las funciones de:

- Revisión de beliefs
- Generación de objetivos activos (options)
- Generación de intenciones activas (filtering)
- Selección de plan bodies (action selection)

Interfaces a instanciar:

- `AgentBeliefRevisionStrategy`
- `AgentOptionGenerationFunction`
- `AgentDeliberationFunction`
- `AgentPlanSelectionStrategy`

Knowledge Engineering and Machine Learning Group

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Sergio Alvarez

salvarez@cs.upc.edu

2022