JADE - Movilidad

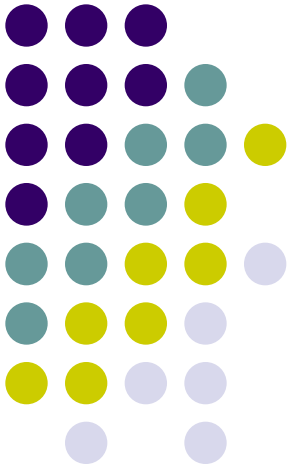
Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin

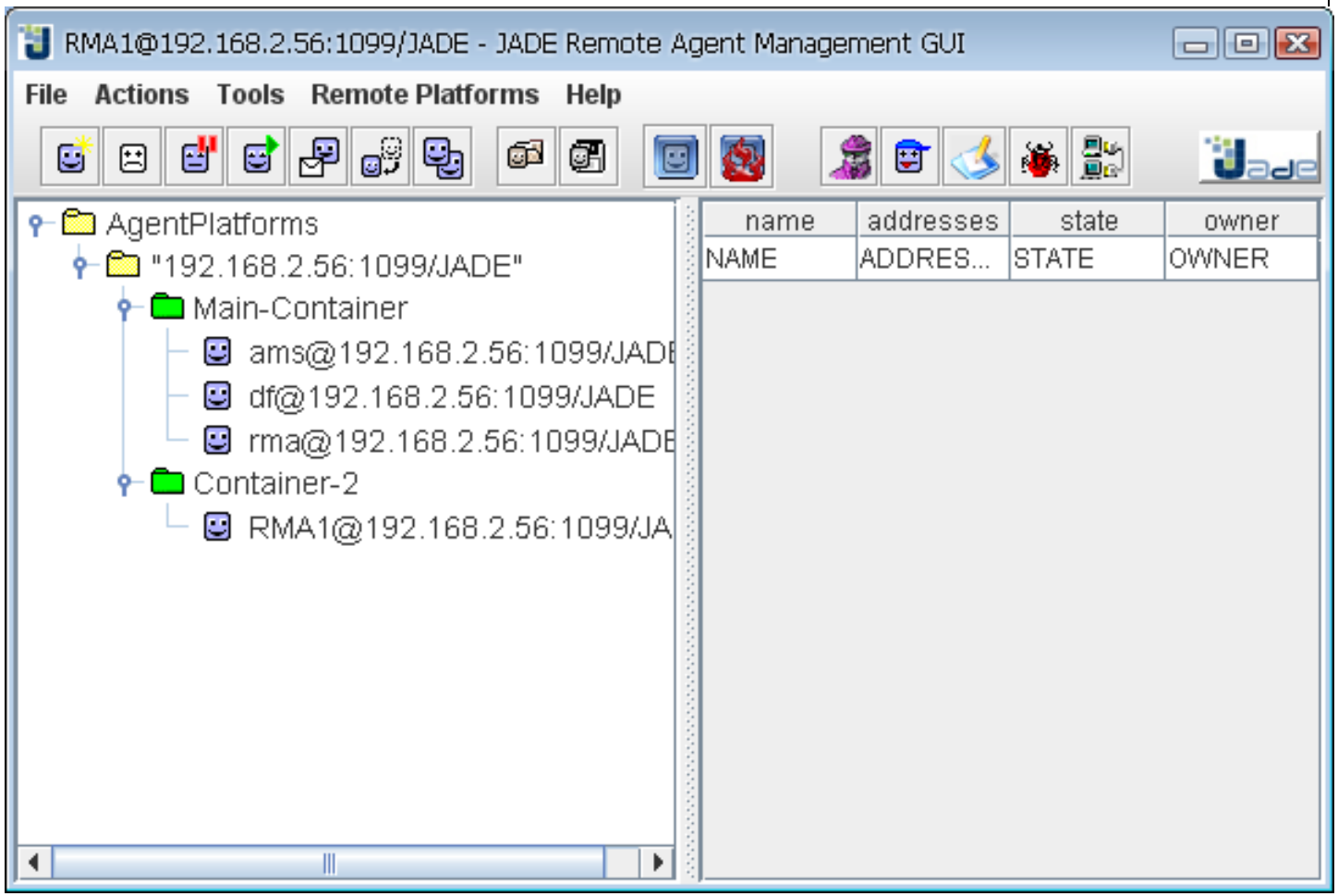
amontese@exa.unicen.edu.ar

ISISTAN –Fac. Cs. Exactas – UNICEN

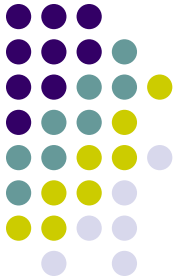
Tandil, Argentina



Movilidad desde RMA



Movilidad en JADE



- Movilidad intra-plataforma
- Estados del ciclo de vida del agente
  - En transito
- Ontología
  - `jade-mobility-ontology`
    - Conceptos y acciones necesarias para la movilidad.
    - move-agent, clone-agent, Mobile-agent-description…
  - `Agent.doMove(Location l)`
  - `Agent.doClone(Location l, String name)`
- `beforeMove(), afterMove(), beforeClone(), afterClone()`

Movilidad en JADE

- `Agent.doMove(Location l)`
  - `Location` es una interface abstracta.

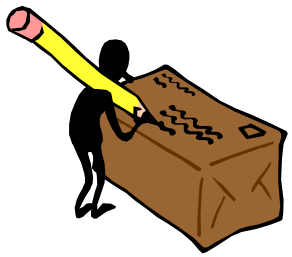
- El agente debe consultar por posibles `Location` al AMS.
  - Mediante REQUEST ACL
    - `WhereIsAgentAction`
      - `setAgentIdentifier(AID)`
    - `QueryPlatformLocationsAction`
  - MobilityOntology: `jade-mobility-ontology`
  - `SLCoded`

Envío de mensaje al AMS

```
ACLMessage req = new ACLMessage(ACLMessage.REQUEST);
req.addReceiver(getAMS());
req.setLanguage(codec.getName()); // SLCodec
req.setOntology(onto.getName());  // MobilityOntology

myAgent.getContentManager().fillContent(
        req,
        new Action(getAMS(),
                new QueryPlatformLocationsAction()));

myAgent.send(req);
```



Recepción de mensaje del AMS

```
ACLMessage resp = blockingReceive();

ContentElement ce = getContentManager().extractContent(resp);

Result result = (Result) ce;

jade.util.leap.Iterator it = result.getItems().iterator();

while (it.hasNext()) {
       loc = (Location)it.next();
}
...
myAgent.doMove(loc);
```

Agentes e inteligencia

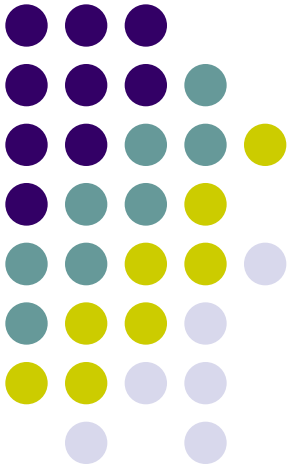
Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin

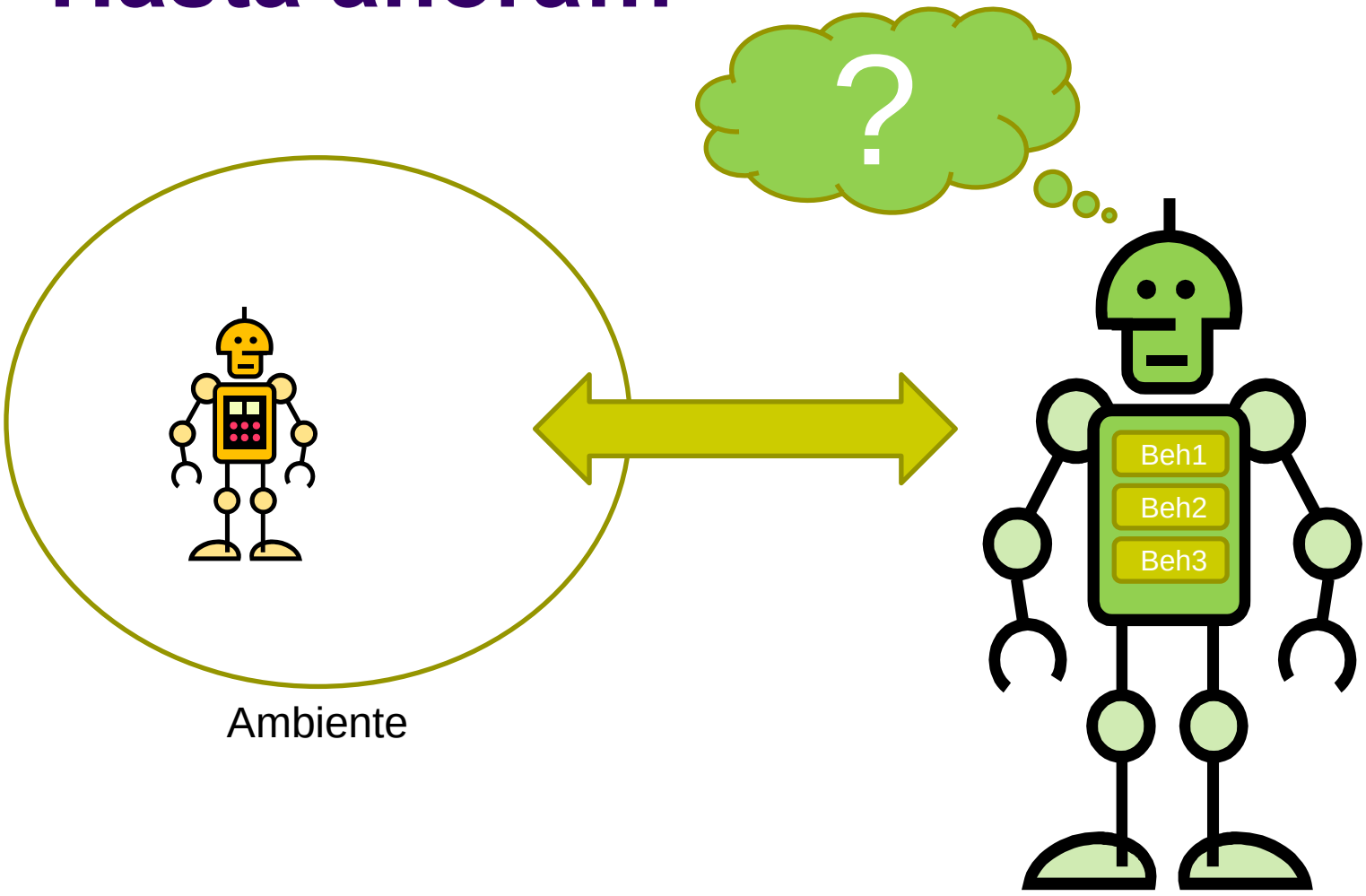
amontese@exa.unicen.edu.ar

ISISTAN –Fac. Cs. Exactas – UNICEN

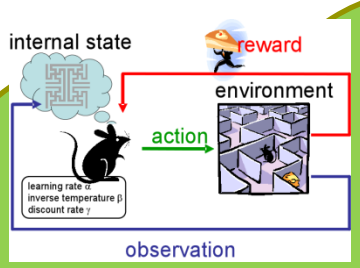
Tandil, Argentina



Hasta ahora…



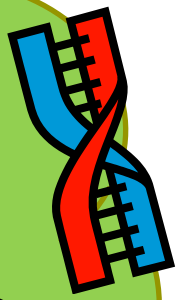
Aprendizaje y razonamiento



Reinforcement learning

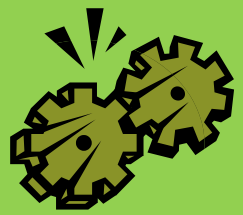
KDD

Planning



Programación evolutiva

Mucho más..



Razonamiento argumentativo

Motores de inferencia

Razonamiento basado en reglas

Jess - Java Expert System Shell

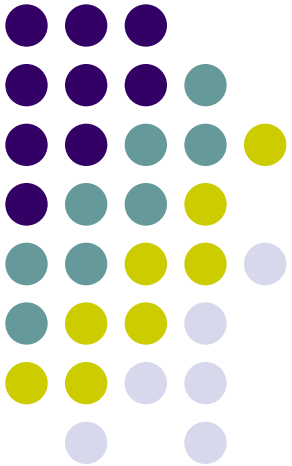
Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin

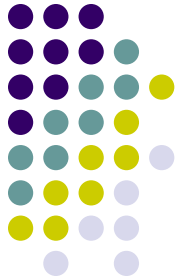
amontese@exa.unicen.edu.ar

ISISTAN –Fac. Cs. Exactas – UNICEN

Tandil, Argentina

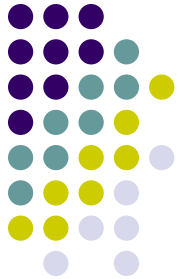


Jess - Java Expert System Shell



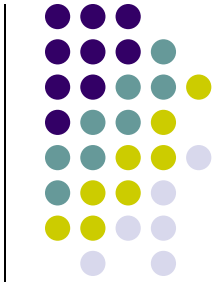
- Razonador basado en reglas para la plataforma Java.
  - Basado en el lenguaje de programación CLISP.
  - Provee programación basada en reglas.
    - Sistemas Expertos
  - Permite embeber la funcionalidad Jess dentro código Java.

Elementos básicos



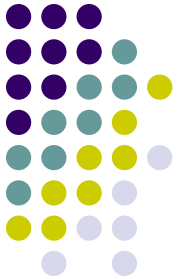
- Hechos
  - Representan conocimiento
- Reglas
  - Indican que acción debe realizarse cuando se cumple cierta condición.
- Preguntas
  - Realizan consultas sobre la base de conocimiento.

Elementos básicos



- Literales
  - `agente`
- Variables
  - `?a`
  - `(bind ?x "hola")`
  - `(bind $?estuche (create$ lapiz boli goma))`
  - `(defglobal ?*var* = primero)`
- Funciones
  - `(nombreFuncion parámetro1 parámetro2 …)`
  - ```
    (deffunction nombre_funcion (parámetros)
      expresiones
    )
    ```
  - ```
    (deffunction fact (?n)
            (if (= ?n 0) then 1
            else (* ?n (fact (- ?n 1)))
            ))
    ```
  - `(fact 5)`

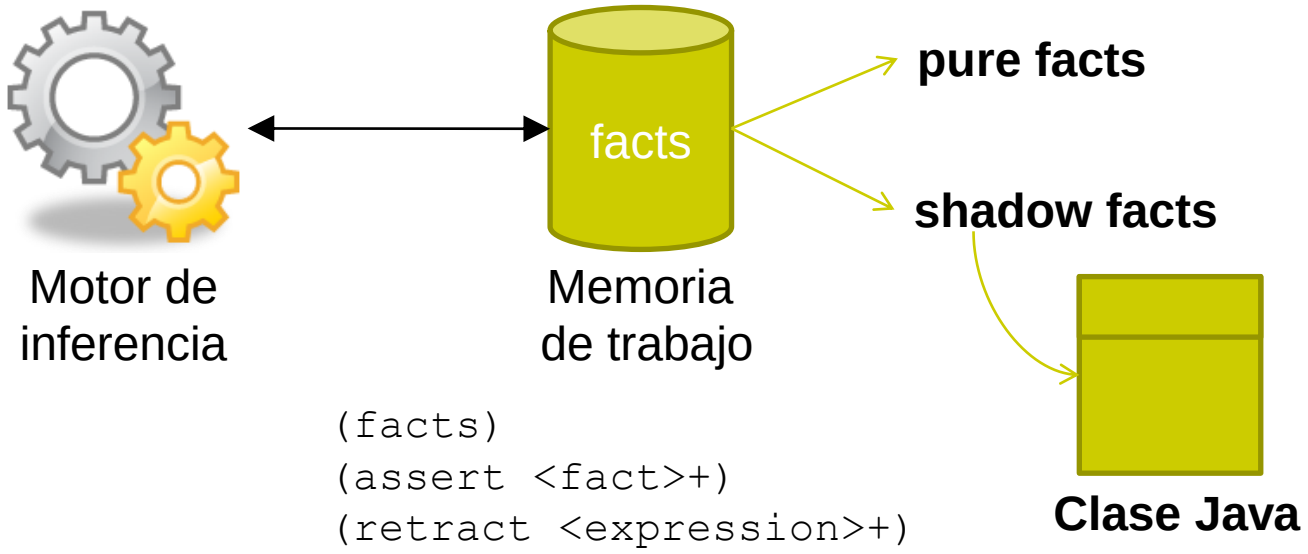
Elementos básicos



- Funciones predefinidas
  - `(++ <variable>) (-- <variable>)`
  - `(assert <fact>+)`
  - `(assert-string <string-expression>)`
  - `(batch <filename>)`
  - `(bind <variable> <expression>*)`
  - `(eval <lexeme-expression>)`
  - `(facts [<module name>])`
- Estructuras:
  - `if, while, for, try/catch`

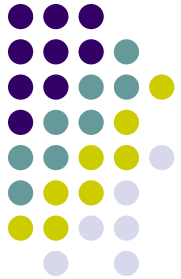
http://herzberg.ca.sandia.gov/jess/docs/70/functions.html

Motor de inferencia



- Ciclo de inferencia
  1. Se buscan todas las posibles reglas aplicables, es decir, que son compatibles con la memoria de trabajo. **Algoritmo RETE** (Algoritmo de Redundancia Temporal)
  2. Se selecciona una de las reglas según un orden de preferencia.
  3. Se aplica la regla seleccionada y se actualiza la memoria.

Hechos



- Hechos ordenados
  - ```
    Jess> (assert (vivienda ocupada))
          <Fact-0>
    Jess> (assert (puerta abierta)) <Fact-1>
    Jess> (facts)
          f-0 (MAIN:: ((vivienda ocupada))
          f-1 (MAIN:: ((puerta abierta))
          For a total of 2 facts in module MAIN.
    ```
- Hechos no ordenados
  - Template que declara la estructura (slots) de los hechos
    - ```
      (deftemplate automobile "A specific car." (slot make)
      (slot model) (slot year (type INTEGER)) (slot color
      (default white)))
      ```
- Shadow facts
  - ```
    public class Account implements Serializable {
          // Atributos y métodos interesantes
    }
    ```
  - ```
    (deftemplate Account (declare (from-class Account)))
    ```
  - ```
    (bind ?a (new Account))
    ```
    ```

Reglas

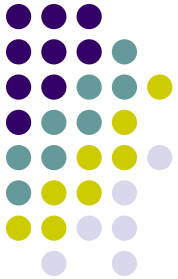
- Estructura IF-THEN
  - (defrule nombreRegla (condición) => (acción))

```
(defrule apagarLuces
        (and
                (vivienda vacia)
                (luz encendida))
=>
        (printout t "La vivienda esta vacia. Las luces deben
        estar apagadas." crlf)
        (retract-string "(luz encendida)")
        (assert (luz apagada))
)
```

```
Jess> (assert (vivienda vacia))
        <Fact-0>
Jess> (assert (luz encendida))
        <Fact-1>
Jess> (run)
        La vivienda esta vacia. Las luces deben estar
        apagadas.
```

- Usar **`test`** si se quiere disparar una regla basandose en una función

Integración Jess - JADE



- Motor de inferencia
  - Clase `jess.Rete`

  - **`batch(String)`**: equivalente a **(batch archivo)**, carga un archivo *.clp*.
  - **`run(), run(int)`**: equivalentes a **(run [integer])**, ejecuta el motor de inferencias.
  - **`reset()`**: equivale a **(reset)**.
  - **`clear()`**: equivale a **(clear)**, borra reglas, deffacts, defglobals, templates, facts… menos funciones.
  - **`assertFact(Fact)`**: equivale a **(assert (hecho))**, añade un hecho que debe estar definido de tipo Fact.
  - **`assertString("hecho")`**: a **(assert (hecho))**, añade un hecho que se pasa como String.
  - **`retract(Fact), retract(int)`**: equivalen a **(retract hecho)**, eliminan un hecho.
  - **`halt()`**: equivale a **(halt)**, detiene la ejecución de las reglas.
  - **`eval(String)`**: cuyo parámetro es el código JESS que se quiere ejecutar
  - **`listFacts()`**: devuelve todos los hechos de la memoria de trabajo en un Iterator
  - **`executeCommand(String)`**: ejecuta el comando pasado como parámetro.

De Jess a acciones

- Jess.Userfunction
  - Permite definir funciones que serán invocadas desde el código Jess.
    - `Rete.addUserfunction(Userfunction)`

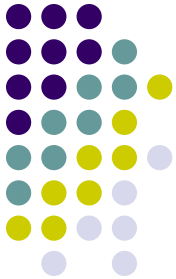
```
public class JessAddBehaviour implements Userfunction {

   public Value call(ValueVector arg0, Context arg1) throws
                                                  JessException {
      myAgent.addBehaviour(new JessBehaviour());
      return Funcall.TRUE;
   }

   public String getName() {
      return "addBehaviour";
   }
```

```
(defrule add-behaviour
   (and(c1)(c2))=>
   (addBehaviour)
   (retract-string "(c1)")
   (retract-string "(c2)"))
```

Ejemplo PELICULA

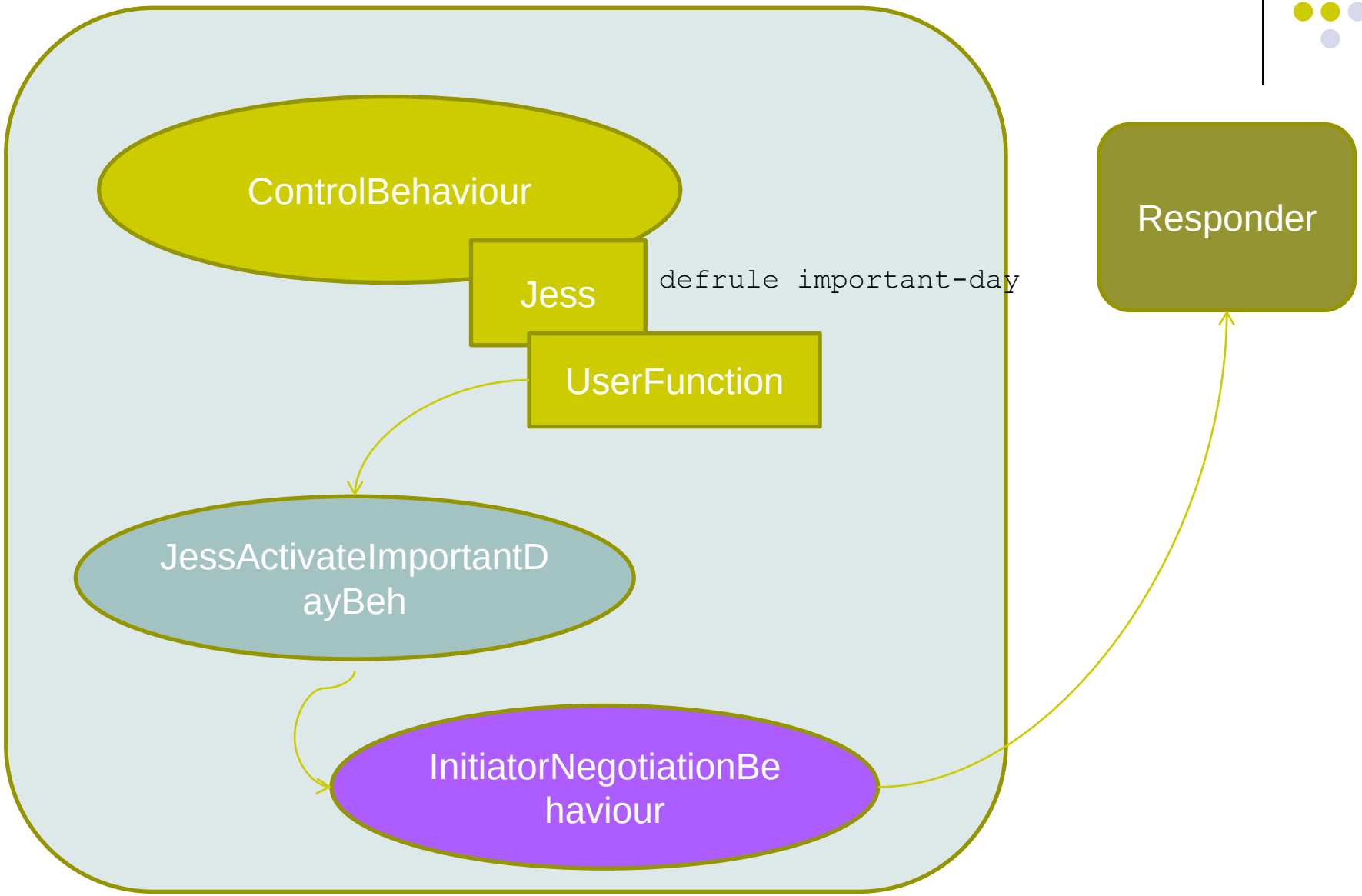


```
(deftemplate important-date (slot dd) (slot mm) (slot description)
   (slot name))
(deftemplate today (slot dd) (slot mm))

(assert (important-date (dd 18) (mm 07) (description cumple) (name
   Juana)))
(assert (important-date (dd 19) (mm 09) (description aniversario) (name
   Carola)))
(assert (important-date (dd 15) (mm 10) (description dia-de-la-madre)
   (name Susana)))

(defrule important-day
(and (today (dd ?dd) (mm ?mm))
   (important-date {dd == ?dd} {mm == ?mm} (description ?d) (name ?n)))
    =>
      (ActivateImportantDayBeh ?n)
 )
```

Ejemplo PELICULA



Ejemplo PELICULA

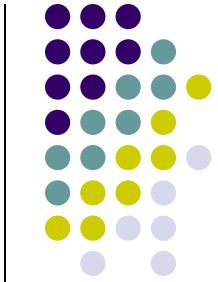
```
public class ControlBehaviour extends TickerBehaviour {

private Rete jess;
private List<Date> list;
private int i = 0;

public ControlBehaviour(Agent a, long tick, List<String> items) {
super(a, tick);
    jess = new Rete();
    list = new ArrayList<Date>();
        …
    try {
        String jessFile = "rules-anniversary.clp";
        FileReader fr = new FileReader(jessFile);
        Jesp j = new Jesp(fr, jess);
        jess.addUserfunction(new JessActivateImportantDayBeh(items));
        // Ejecuta el parser (sin mostrar datos por consola)
        j.parse(false);
    }
    catch (JessException je) {je.printStackTrace();}
    catch (FileNotFoundException je) {je.printStackTrace();}

}
```

Ejemplo PELICULA

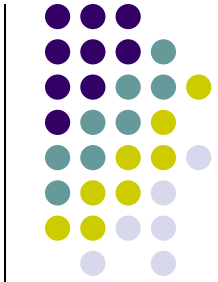


```
@Override
protected void onTick() {
   try {
      if (i < list.size()) {
        // Actualiza la fecha
        Calendar cal = Calendar.getInstance();
        cal.setTime(list.get(i));
        int mm = cal.get(Calendar.MONTH);
        int dd = cal.get(Calendar.DAY_OF_MONTH);

        jess.eval("(retract-string \"(today (dd ?d) (mm ?m))\")");
        jess.eval("(assert (today (dd " + dd + ") (mm " + mm + ")))");
        i++;
        jess.run();
      }
      else
        System.out.println("No hay mas fechas para cargar");
   }
   catch (JessException je) {je.printStackTrace();}

}
```

Ejemplo PELICULA



```
public class JessActivateImportantDayBeh implements Userfunction {
   private List<String> items;
   public JessActivateImportantDayBeh(List<String> items) {
      this.items = items;
   }

   public Value call(ValueVector jessArgs, Context context)
                                                  throws JessException {
      String name = ((Value)jessArgs.get(1)).
                        resolveValue(context).stringValue(context);
      myAgent.addBehaviour(new InitiatorNegotiationBehaviour(
                                 new AID(name, AID.ISLOCALNAME), items));
      return Funcall.TRUE;
}

   public String getName() {
      return "ActivateImportantDayBeh";
   }
}
```

Jess - Java Expert System Shell

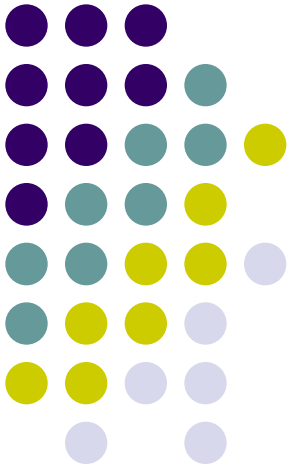
Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin

amontese@exa.unicen.edu.ar

ISISTAN –Fac. Cs. Exactas – UNICEN

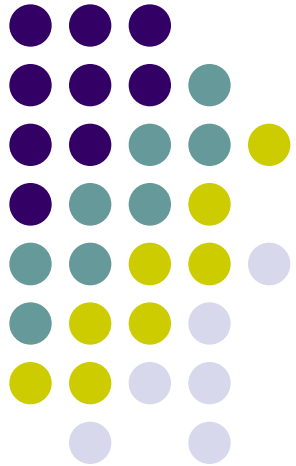
Tandil, Argentina



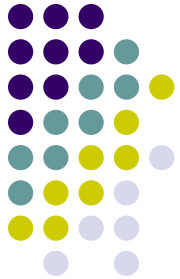
JADEX

Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin
amontese@exa.unicen.edu.ar
ISISTAN –Fac. Cs. Exactas – UNICEN
Tandil, Argentina



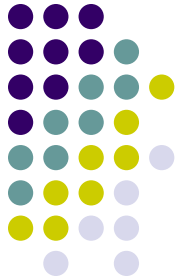
Introducción



- Framework para la creación de agentes BDI.
  - Basado en Java y XML.
  - Puede ser integrado a JADE.
- Motivación
  - Plataforma multiagente: requisitos
    - Apertura (Openness)
    - Middleware
    - Razonamiento

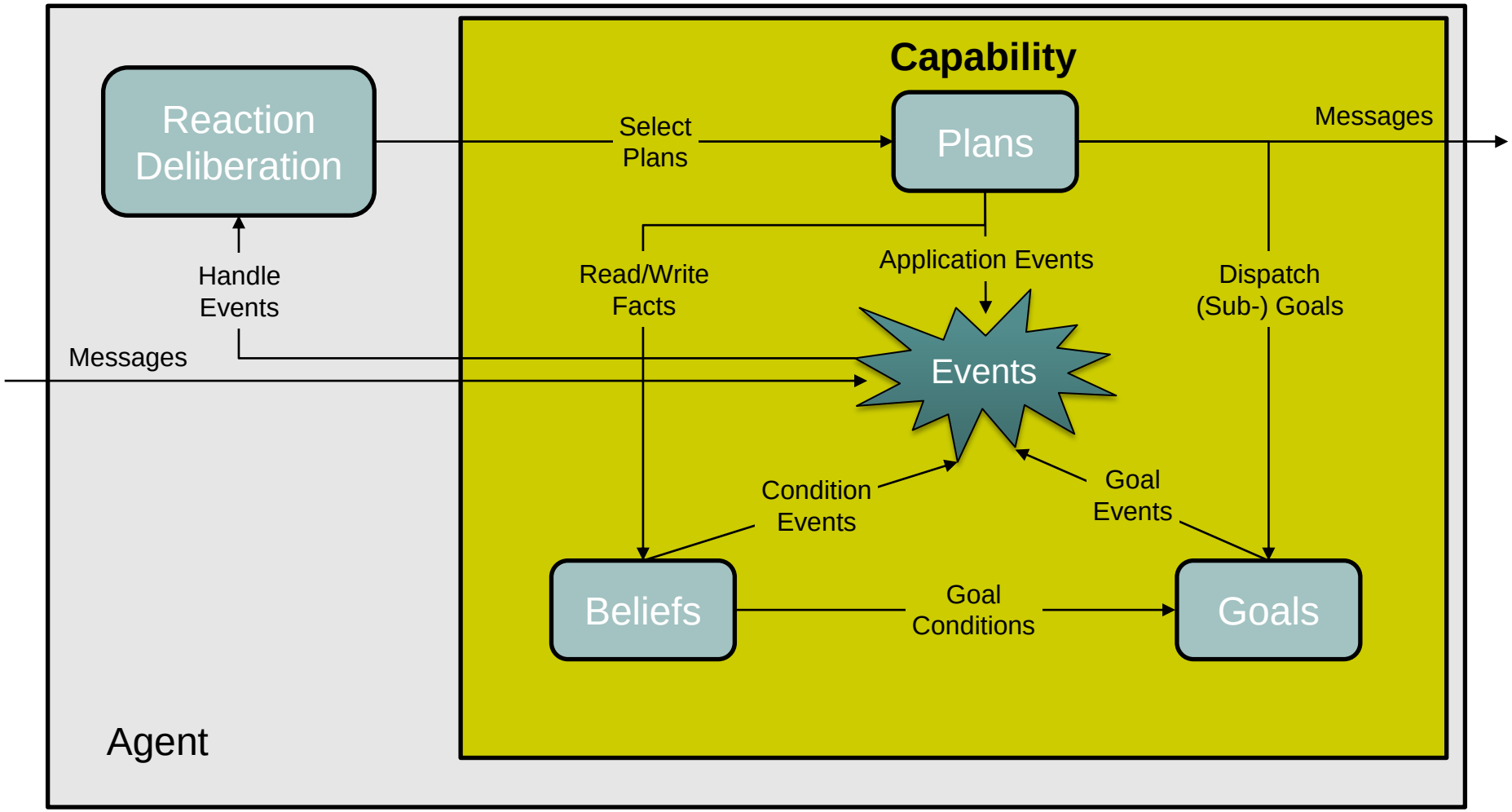
Dos tipos plataforma

El modelo BDI

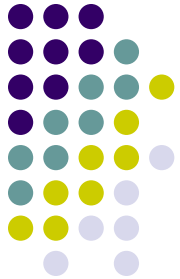


- Inspirada en la teoría del razonamiento práctico.
- Rao y Georgeff
  - Definen a las creencias, deseos e intenciones como actitudes mentales.
  - Proponen ciertas simplificaciones
    - Solamente las creencias son representadas explícitamente.
    - Los deseos son reducidos a eventos que son manejados por planes template predefinidos.
    - Las intenciones son representadas implícitamente por la pila de ejecución de planes.

Arquitectura abstracta de JADEX

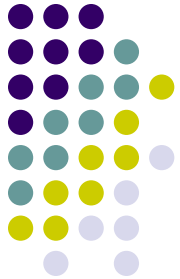


Creencias



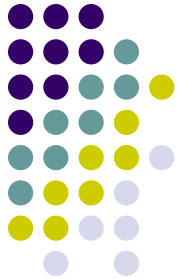
- Representación orientada a objetos.
- Dos tipos
  - Facts (Beliefs)
  - Sets of facts (Belief sets)
- OQL – lenguaje para realizar operaciones sobre la base de creencias
- Parte activa – cambios en la creencia, dispara eventos, crea/elimina objetivos.

Objetivos



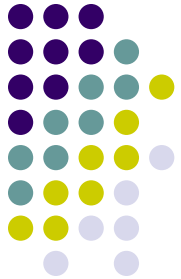
- Dirigen a acciones
- Objetivo actual
  - Implícitamente disponible como causa de los planes en ejecución.
- No se requiere consistencia entre objetivos
- Contexto de validez
- Ciclo de vida
  - Option – Active - Suspended

Tipos de objetivos



- Perform goal: especifica una acción que debe ser realizada.
- Achieve goal: especifica un estado del mundo al cual se debe llegar.
- Query goal: especifica un estado interno al cual se desea llegar.
- Maintain goal: especifica un estado deseado que debe restablecerse cada vez que sea modificado.

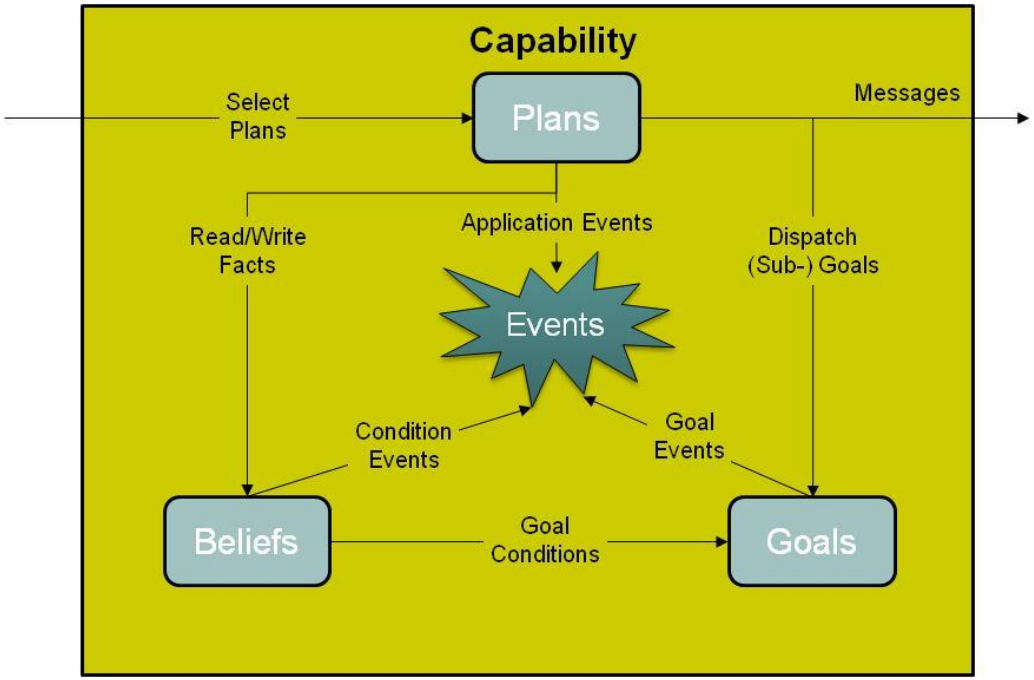
Planes



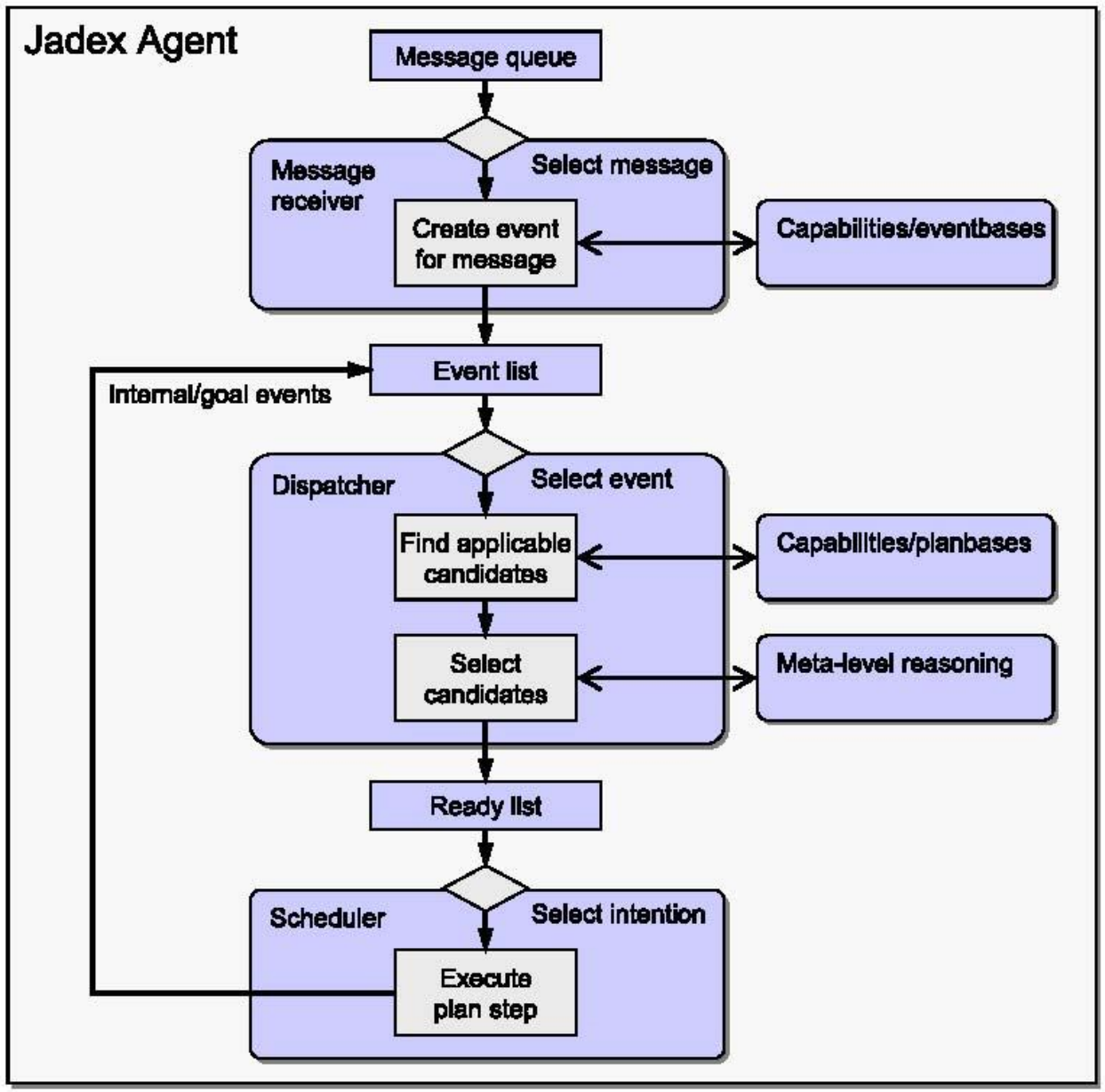
- Representan el comportamiento del agente.
  - Service plan – Passive plan
- Cabecera (head)
  - Especifica las circunstancias bajo las cuales el plan puede ser seleccionado.
- Cuerpo (body)
  - Provee un curso de acción predefinido que es ejecutado cuando el plan es seleccionado.

Capabilities

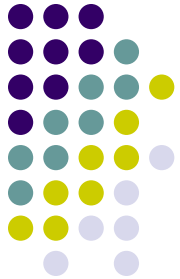
- Agrupan elementos de un agente BDI.
  - Permiten encapsular funcionalidad



Modelo de ejecución

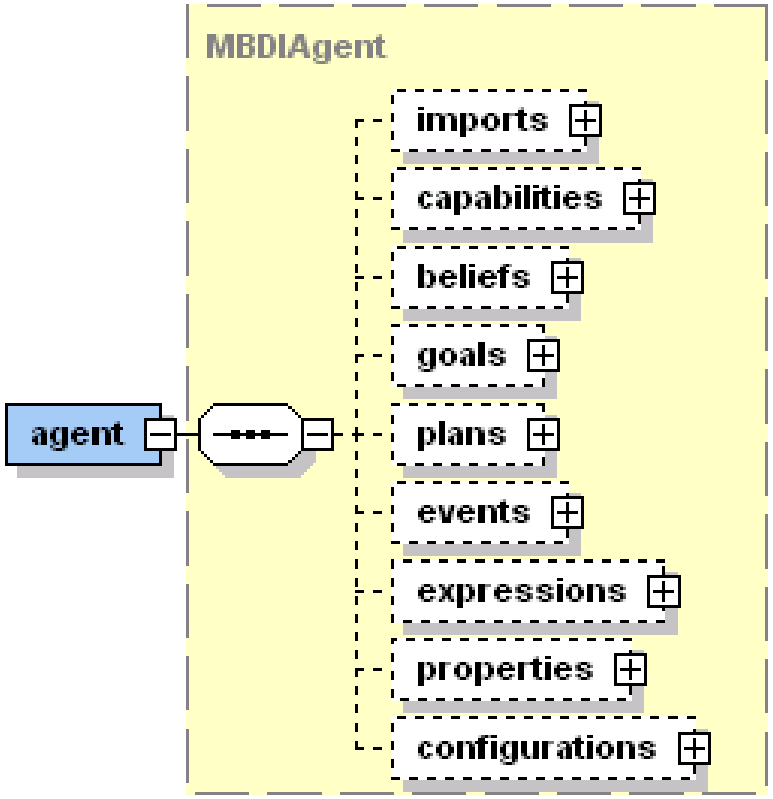


Lenguaje



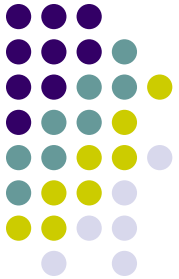
- Enfoque híbrido
  - Agent Definition File (ADF)
    - Especifica creencias, objetivos, planes.
    - Especificación estática del agente.
    - XML
  - Plan (Body)
    - Parte procedural del agente
    - Java.
    - Accede a las características BDI del agente mediante un API.

ADF



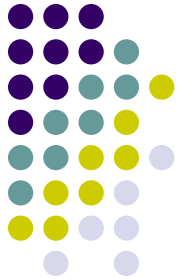

```
<agent
xmlns="http://jadex.sourceforge.net/jadex"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
        instance"
xsi:schemaLocation="http://jadex.sourceforge
        .net/jadex
        http://jadex.sourceforge.net/jadex-
        0.96.xsd"
name="TranslationE1"
package="jadex.tutorial">
<imports>
<import>java.util.*</import>
<import>jadex.bridge.fipa.*</import>
<import>jadex.commons.Tuple</import>
</imports>
…
</agents>
```

Beliefs



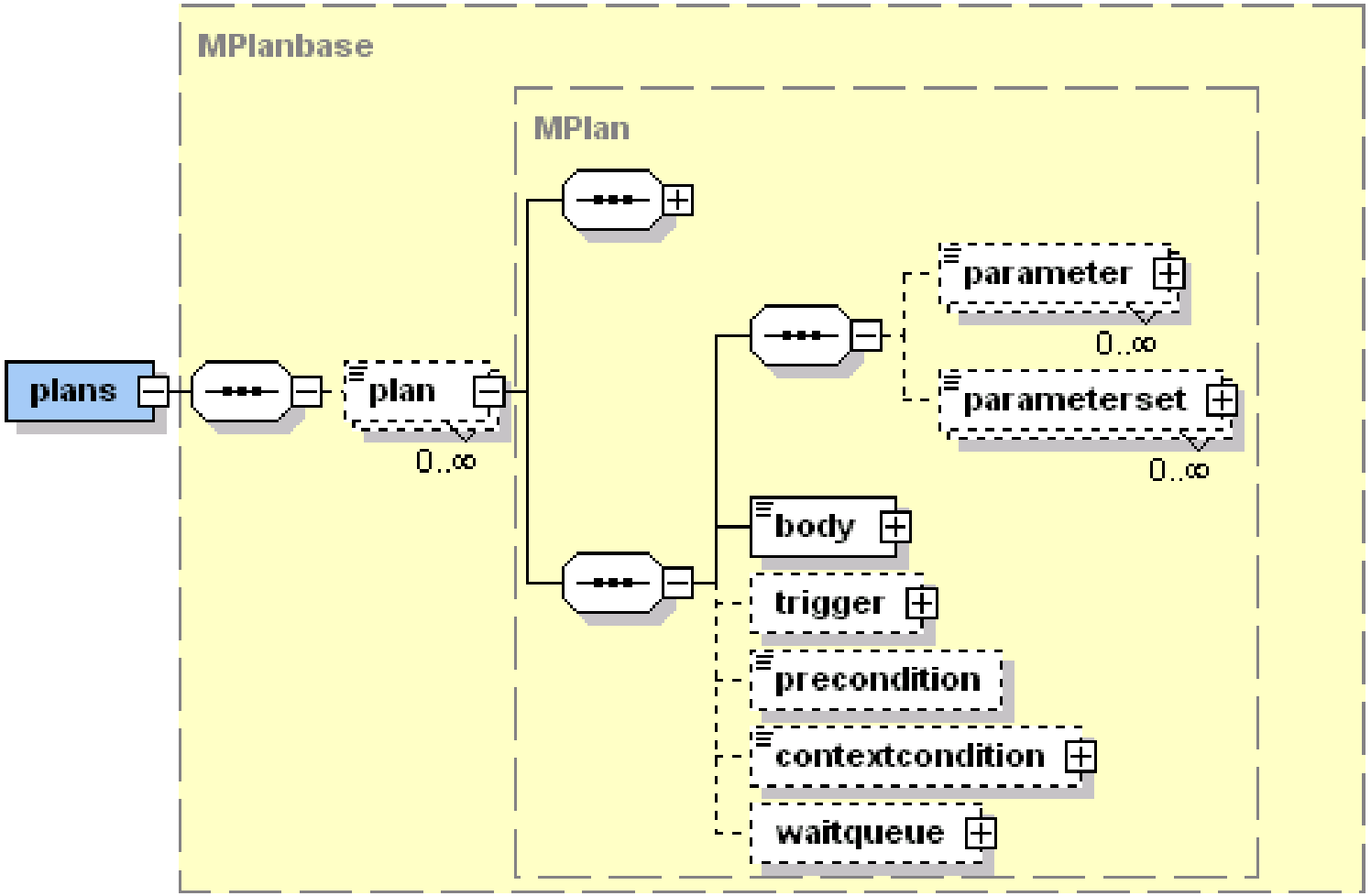
```
<beliefs>
    <belief name="egwords" class="Map">
          <fact>EnglishGermanTranslationPlanC1.getDictionary()</fact>
     </belief>
    ….
    <beliefset name="egwords" class="Tuple">
          <fact>new Tuple("milk", "Milch")</fact>
          <fact>new Tuple("cow", "Kuh")</fact>
          <fact>new Tuple("cat", "Katze")</fact>
          <fact>new Tuple("dog", "Hund")</fact>
    </beliefset>
    <beliefset name="efwords" class="Tuple">
          <fact>new Tuple("milk", "lait")</fact>
          <fact>new Tuple("cow", "vache")</fact>
          <fact>new Tuple("cat", "chat")</fact>
          <fact>new Tuple("dog", "chien")</fact>
    </beliefset>
</beliefs>
```

Goals



```
<goals>
   <achievegoal name="translate">
        <parameter name="direction" class="String" />
        <parameter name="word" class="String" />
        <parameter name="result" class="String" direction="out" />
   </achievegoal>
</goals>
```

Plans


MPlanbase
MPlan
plans
plan
0..∞
parameter
0..∞
parameterset
0..∞
body
trigger
precondition
contextcondition
waitqueue

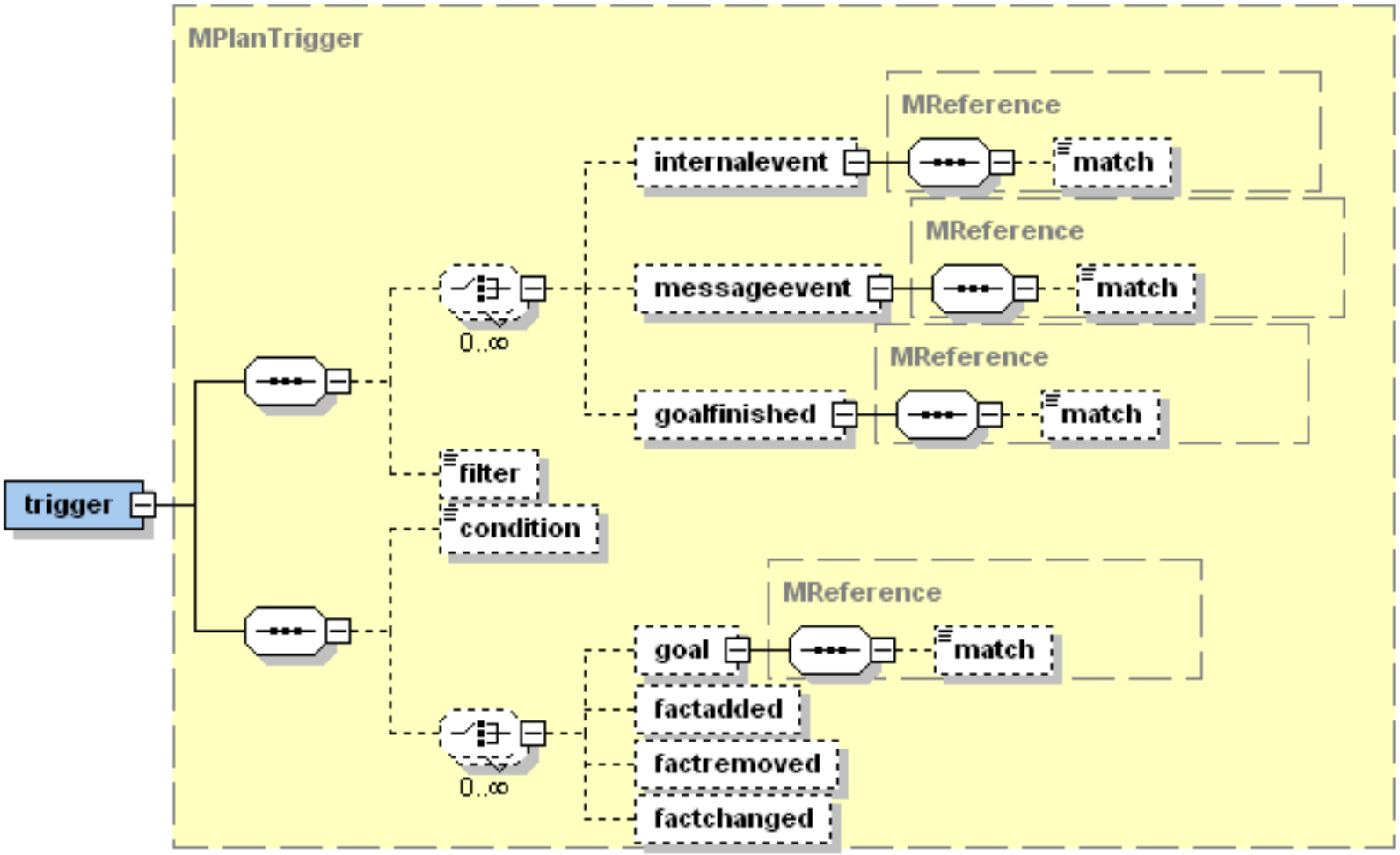
Plans

```
<plans>
    <plan name="process">
          <body class="ProcessTranslationRequestPlanE1" />
          <waitqueue>
                    <messageevent ref="request_translation" />
          </waitqueue>
    </plan>
    <plan name="egtrans">
          <parameter name="word" class="String">
                    <goalmapping ref="translate.word" />
          </parameter>
          <parameter name="result" class="String" direction="out">
                    <goalmapping ref="translate.result" />
          </parameter>
          <body class="EnglishGermanTranslationPlanE1" />
          <trigger>
               <goal ref="translate">
                    <match>"english_german".equals($goal.getParameter("direction").getValue())
                    </match>
               </goal>
          </trigger>
    </plan>
```

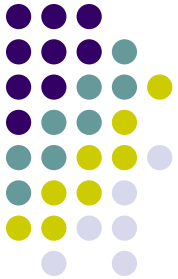
Service plan

Passive plan

Plan triggers



Cuerpo del plan



- Clase Java
  - Heredan de *jadex.bdi.runtime.Plan*
  - Método body()

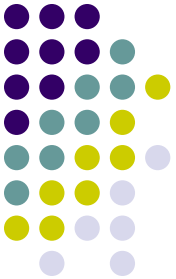
```
public class ProcessTranslationRequestPlanE1 extends Plan {

public ProcessTranslationRequestPlanE1() {
    System.out.println("Plan creado " + this);
}

public void body() {
    while(true) {
          IMessageEvent me =
          (IMessageEvent)waitForMessageEvent("request_translation");

          String request = (String)me.getParameter(SFipa.CONTENT).getValue();
          StringTokenizer stok = new StringTokenizer(request, " ");
          int cnttokens = stok.countTokens();
```

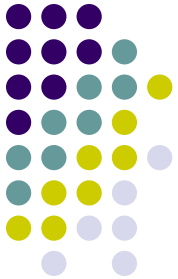
Cuerpo del plan



```
if(cnttokens == 3) {
     String action = stok.nextToken();
     String dir= stok.nextToken();
     String word = stok.nextToken();

     IGoal goal = createGoal("translate");
     goal.getParameter("direction").setValue(dir);
     goal.getParameter("word").setValue(word);
     try {
       dispatchSubgoalAndWait(goal);
     } catch(GoalFailureException e) {
     };
}
else {
      System.out.println("Request format not correct, needs:
#" + "action direction word1 [word2]");
}
```

Events



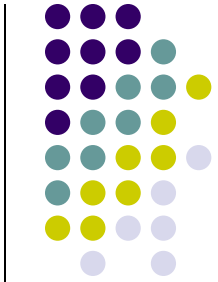
- Internal-events

```
<events>
  <internalevent name="gui_update">
    <parameter name="content" class="String"/>
  </internalevent>
</events>
```

```
public void body()
{
  String update_info;
  ...
  IInternalEvent event = createInternalEvent("gui_update");

  event.getParameter("content").setValue(update_info);
  dispatchInternalEvent(event);
  ...
}
```

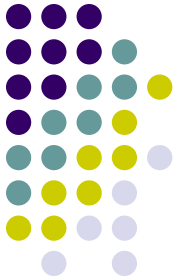
Events



- Message-event

```
<events>
   <messageevent name="request_translation" direction="receive"
   type="fipa">
      <parameter name="performative" class="String" direction="fixed">
               <value>SFipa.REQUEST</value>
      </parameter>
   </messageevent>
</events>
```

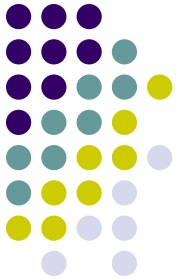
Expressions



- Permiten definir elementos propios del agente.
  - Hechos y creencias por defecto.
  - Consultas OQL.

```
<expressions>
   <expression name="query_egword">
      select one $wordpair.get(1)
      from Tuple
      $wordpair in $beliefbase.getBeliefSet("egwords").getFacts()
      where
      $wordpair.get(0).equals($eword)
   </expression>
…
</expressions>
```

Configurations



- Representa el estado inicial y final de un agente

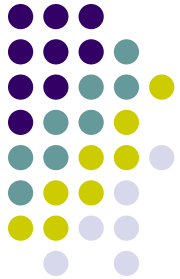
```
<configuration name="one">
      <plans>
        <initialplan ref="print_hello">
          <parameter name="text">"Hello World!"</parameter>
        </initialplan>
        <endplan ref="print_goodbye">
          <paramter name="text">"Goodbye World!"</parameter>
        </endplan>
      </plans>
    </configuration>
```

Configurations

```
<configurations>
   <configuration name="default">
        <plans>
              <initialplan ref="process" />
        </plans>
   </configuration>
</configura
```

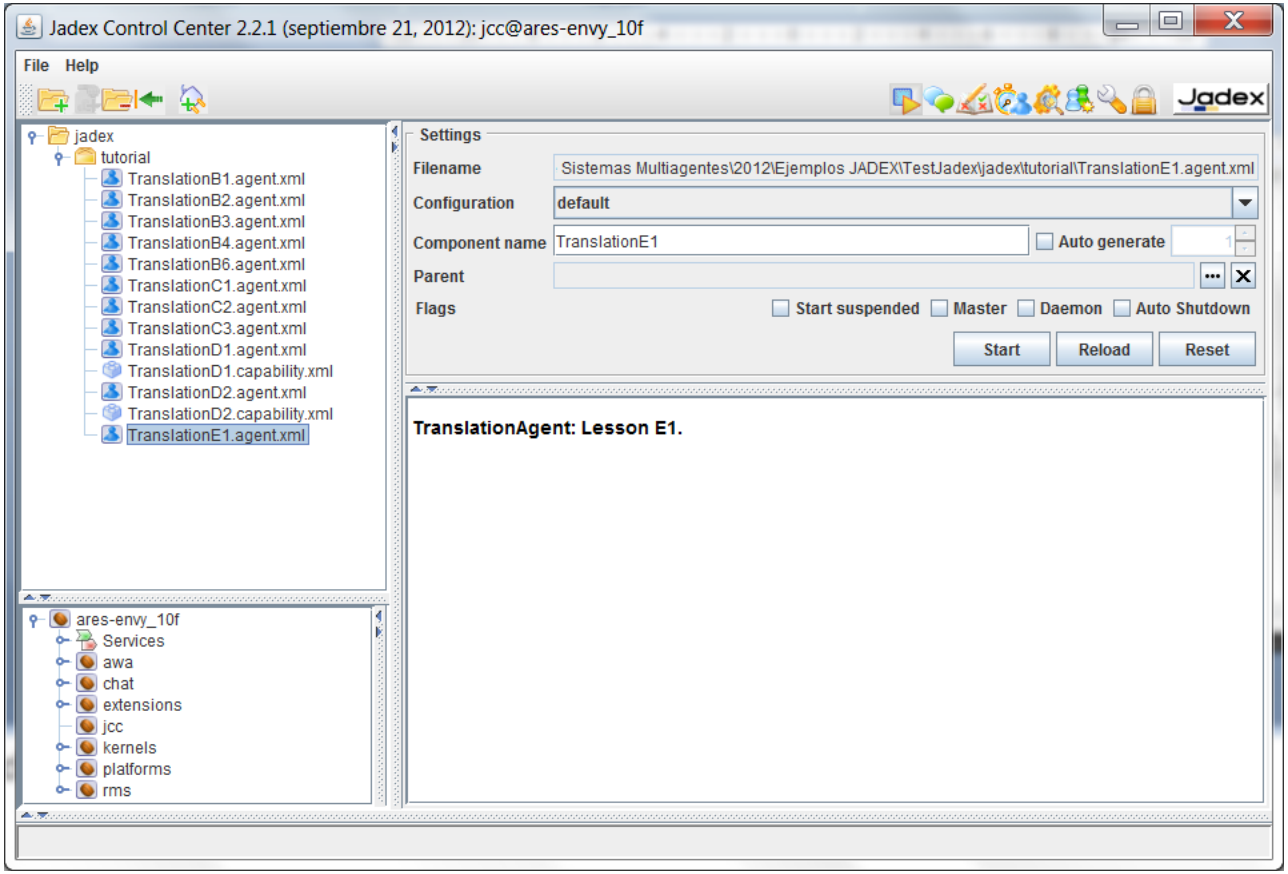
```
<plans>
   <plan name="process">
        <body class="ProcessTranslationRequestPlanE1" />
        <waitqueue>
              <messageevent ref="request_translation" />
        </waitqueue>
   </plan>
```

Pasive plan



```
public class EnglishGermanTranslationPlanE1 extends Plan {
   protected Iexpression queryword;
   public EnglishGermanTranslationPlanE1() {
        this.queryword= getExpression("query_egword");
   }
   public void body() {
        String eword = (String)getParameter("word").getValue();
        String gword = (String)queryword.execute("$eword", eword);
        if(gword!=null) {
               System.out.println("Translating from english to
   german:                          "+eword+" - "+gword);
               getParameter("result").setValue(gword);
        }
        else {
               System.out.println("Sorry word is not in database:
               "+eword);
        }
   }
}
```

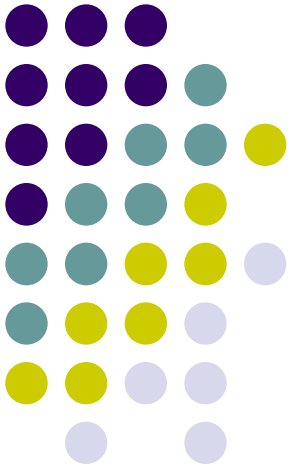
JCC (Jadex Control Center)



JADEX

Taller de sistemas multiagentes

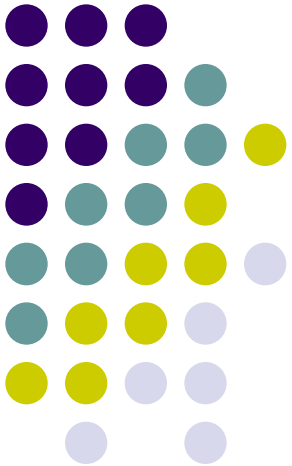
Dr. Ariel Monteserin
amontese@exa.unicen.edu.ar
ISISTAN –Fac. Cs. Exactas – UNICEN
Tandil, Argentina



Planificando cómo argumentar en un proceso de negociación

Taller de sistemas multiagentes

Dr. Ariel Monteserin
amontese@exa.unicen.edu.ar
ISISTAN –Fac. Cs. Exactas – UNICEN
Tandil, Argentina



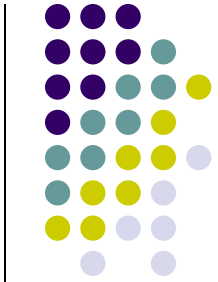
Habilidad Social

- Relaciones entre los agentes
  - Necesidad de interacción.
  - Tipos de interacción
    - Intercambio de información.
    - Coordinación.
    - Colaboración.
    - Negociación.

Negociación – Definición

- Forma de interacción en la cual un grupo de agentes, con intereses conflictivos y un deseo de cooperar, intentan alcanzar un acuerdo mutuamente aceptable en la división de recursos escasos (Rahwan et al., 2003a).

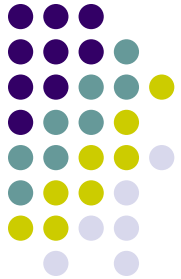
Negociación



- Esencia de la negociación → Intercambio de propuestas
  - Dificultad para comparar cuantitativamente dos propuestas.
  - Las propuestas no pueden influenciar la postura de negociación del oponente.

- Modelos de negociación basada en argumentación.

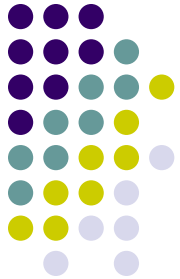
Negociación basada en argumentación



- Argumentos
  - Información adicional a las propuestas.

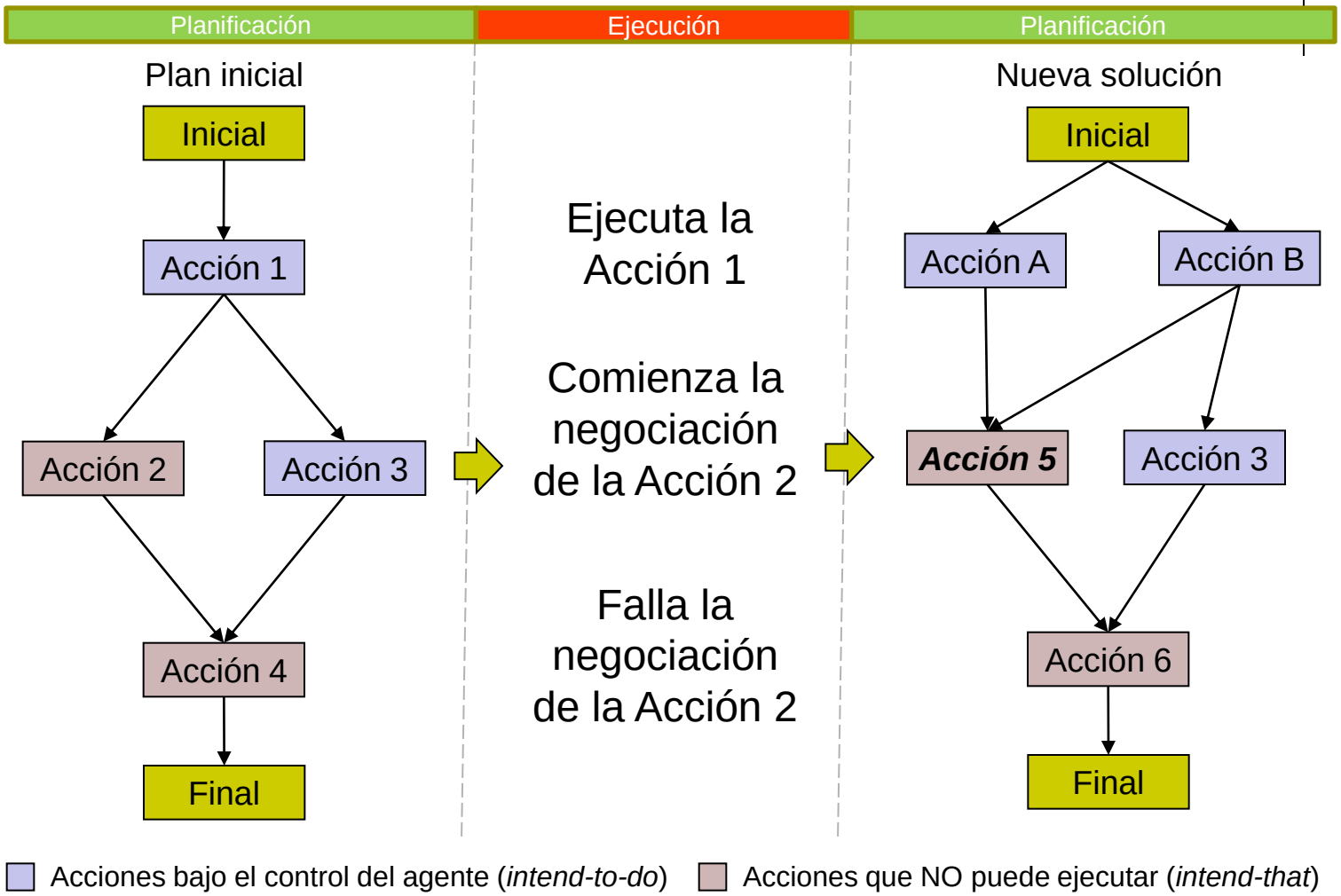
  - Permiten (Jennings et al., 1998):
    - (a) justificar su postura de negociación.
    - (b) influenciar la postura de negociación de otros agentes.

Negociación basada en argumentación - Tendencias

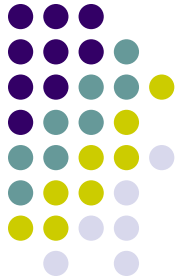


- Existen dos grandes tendencias en la literatura existente sobre negociación basada en argumentación (Rahwan et al., 2005).
  - Enfoques para adaptar lógicas dialécticas para argumentación rebatible embebiendo conceptos de negociación dentro de ellas (Amgoud et al., 2000; Parsons et al.,1998; Rueda et al., 2002).
  - Enfoques para extender *frameworks* de negociación para que permitan a los agentes intercambiar argumentos retóricos, como recompensas y amenazas (Kraus et al.,1998; Ramchurn et al., 2003a).

Enfoque tradicional

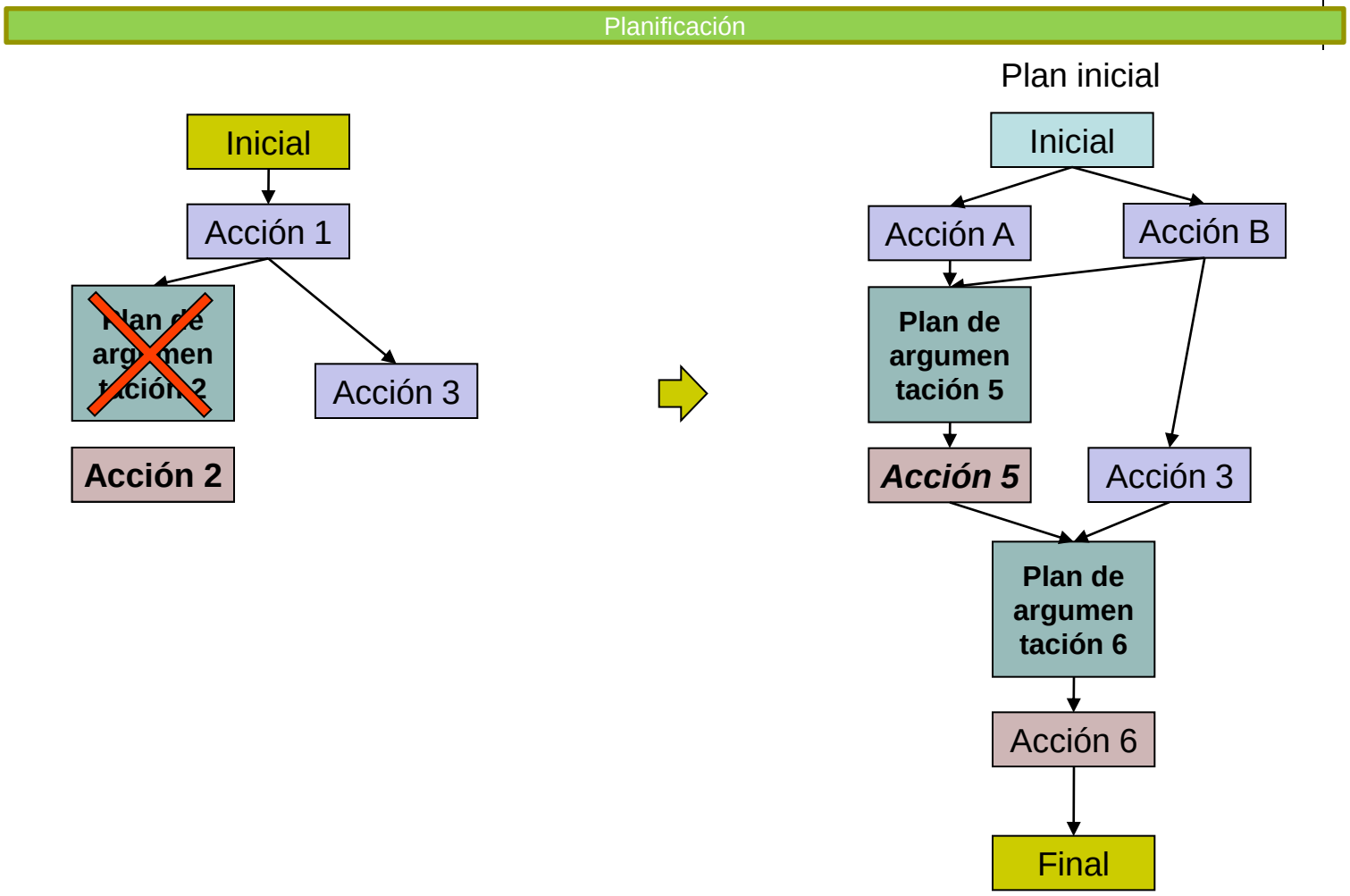


Solución propuesta

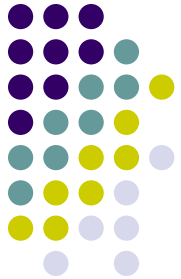


- Un algoritmo de *planning* puede ser utilizado para construir planes de argumentación que determinen los argumentos que un agente puede expresar durante una negociación.
  - Plan de argumentación.
    - Para la negociación autónoma (agente autónomo).
    - Para asistir a un usuario negociador (agente personal).
      - Modelo argumentativo del usuario.

Enfoque planificando la argumentación

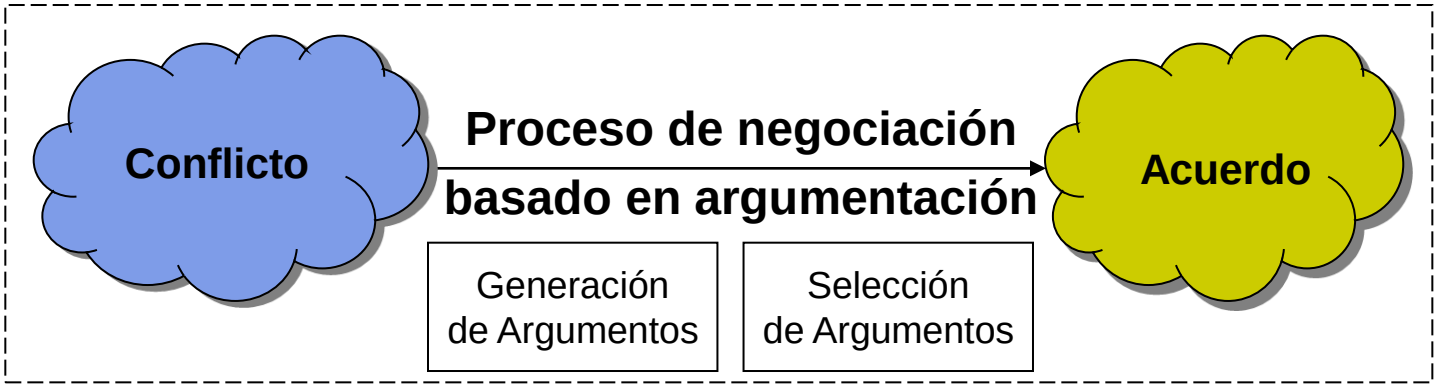


Plan de argumentación



- Secuencia de argumentos de orden parcial que permite a un agente alcanzar un acuerdo esperado cuando éste es expresado en una determinada situación conflictiva.

Escenario de la negociación basada en argumentación



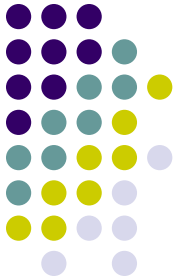
Información sobre el conflicto:

- Propia
- De los oponentes
- Contexto

Información sobre el acuerdo:

- Objetivos
- Intenciones

Generación de argumentos (1)



- Construcción de argumentos candidatos.
- Tipos de argumentos
  - Apelaciones (contraejemplo, práctica prevaleciente, interés propio).
  - Recompensas.
  - Amenazas.

Generación de argumentos (2)

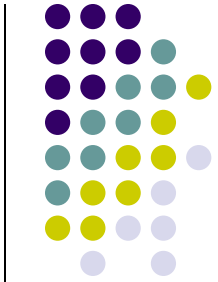
- Reglas para la generación de argumentos
  - (Kraus et al., 1998; Ramchurn et al., 2003)

- Si se cumplen las condiciones entonces el argumento puede ser generado.

IF
χ solicita la ejecución de una acción α a ψ &
ψ rechaza la propuesta porque α niega su objetivo γ &
χ conoce que ψ ha ejecutado una acción β &
haciendo β se negaba el mismo objetivo γ
THEN
χ solicita la ejecución de α a ψ con la siguiente justificación:
Realizá α porque ya has ejecutado β y β negaba γ

(a) Regla informal para la generación de argumentos basada en el framework de Kraus et al., 1998.

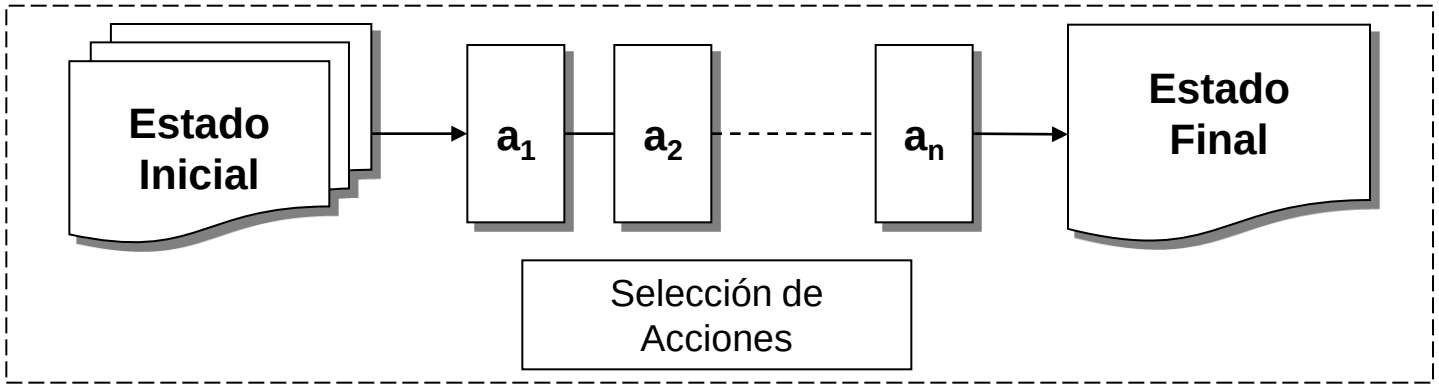
Selección de argumentos



- Dado el conjunto de argumentos candidatos, seleccionar el más adecuado.

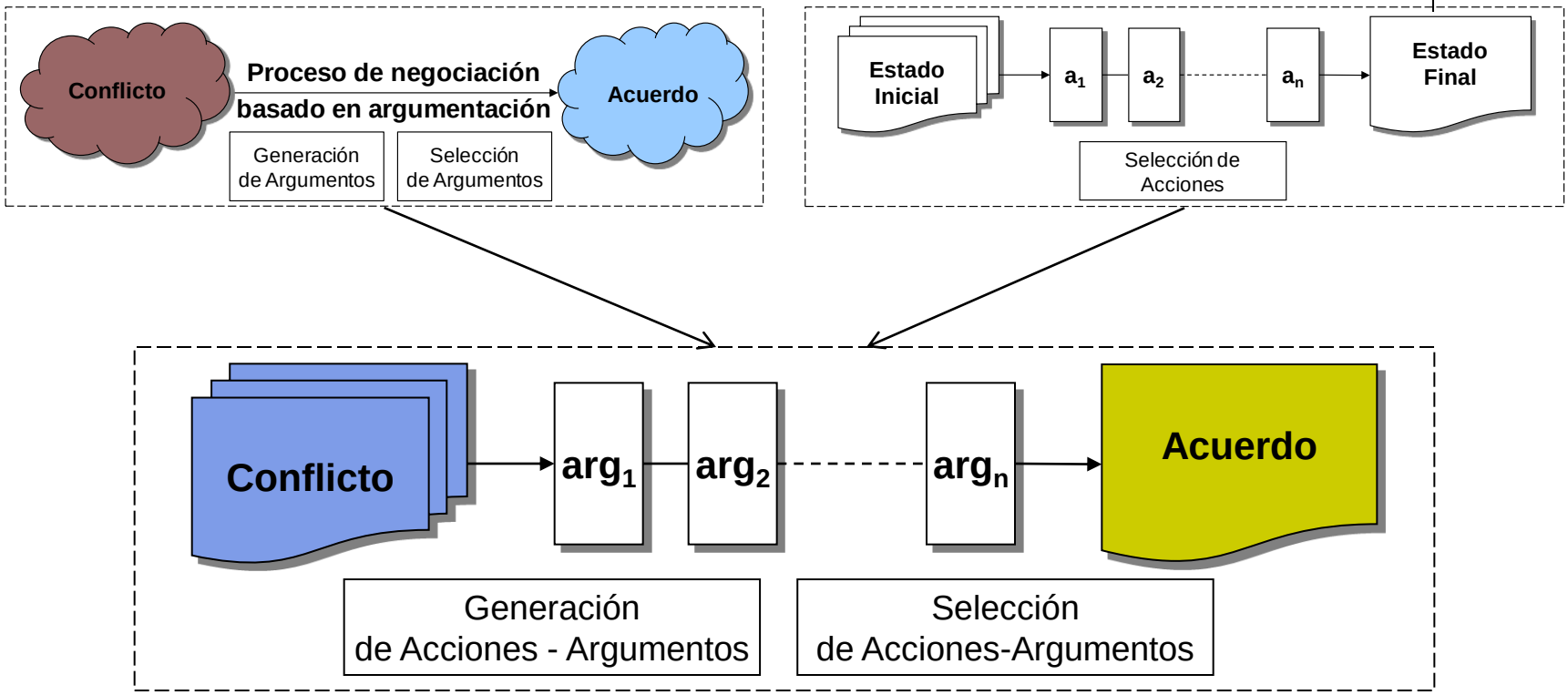
- Distintos criterios de selección:
  - Según la fuerza de los argumentos.
  - Según la confianza en el oponente.
  - Según la utilidad de la propuesta respaldada.

Planning



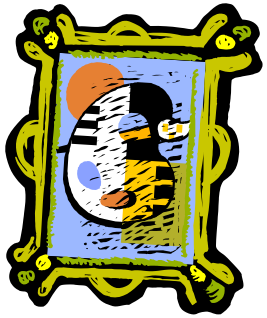
- Problema de planning definido por la tripla <*i*, *f*, *A*>.
  - Estado Inicial *i*: descripción del mundo.
  - Estado Final *f*: objetivo que se quiere alcanzar.
  - Acciones disponibles para la construcción del plan (*A*) las cuales poseen precondiciones y efectos.
- Mecanismo de selección de acciones.

El proceso de argumentación como un problema de planning



Escenario de la negociación

- Escenario de la negociación (Parsons y Jennings, 1996):
  - Agentes deben ejecutar tareas.
  - Recursos limitados.



Picty

Recursos:
• Pintura (*p1*)
• Destornillador (*sd1*)
• Tornillo (*s1*)
• Martillo (*h1*)
Creencias:
• Martillo + Clavo + Silla + Pintura => Pintura colgada
• Destornillador + Tornillo + Espejo => Espejo colgado
Objetivos:
• Colgar pintura

Mirry

Recursos:
• Espejo (*m1*)
• Clavo (*n1*)
• Adhesivo (*g1*)
Creencias:
• Martillo + Clavo + Espejo => Espejo colgado
Objetivos:
• Colgar espejo
• Guardar el adhesivo

Recursos:
• Silla rota (bc1)

Creencias:
• Adhesivo + Silla rota => Silla reparada

Objetivos:
• Reparar silla

Chairy

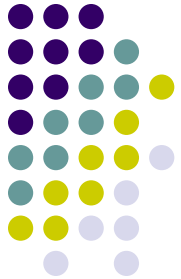
Conflictos

- Conflicto entre Picty y Mirry: es el conflicto original detallado en Parsons y Jennings (1996), Picty necesita el clavo que posee Mirry, y Mirry necesita el martillo que posee Picty.

- Conflicto entre Picty y Chairy: el primero necesita una silla para cumplir su objetivo, y el segundo debe repararla.

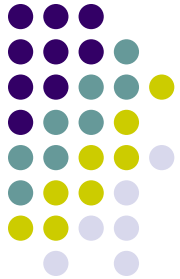
- Conflicto entre Chairy y Mirry: Chairy necesita el adhesivo para reparar la silla, y a su vez Mirry tiene como objetivo conservarlo.

Picty



- **Hechos propios:**
  - iam(picty).
- **Objetivos:**
  - • isgoal(picty, pictureHanging(p1)).
- **Creencias:**
  - *believe(picty, imply([has(picty, hammer(h1)), has(picty, nail(n1)), has(picty, chair(bc1)), has(picty, picture(p1))], do(picty, hangAPicture(picty, p1, h1, n1, bc1))))).*
    "Para poder realizar la acción 'colgar pintura' (hangAPicture(picty, p1, h1, n1, bc1)) es necesario contar con h1 (martillo), n1 (clavo), bc1 (silla) y p1 (pintura)."
  - *believe(picty, imply(do(picty, hangAPicture(picty, p1, h1, n1, bc1)), pictureHanging(p1))).*
    "Al ejecutar la acción hangAPicture(picty, p1, h1, n1, bc1) la pintura p1 quedará colgada".
  - *believe(picty, imply([has(mirry, mirror(m1)), has(mirry, screwdriver(sd1)), has(mirry, screw(s1))], do(mirry, hangAMirror(mirry, m1, sd1, s1))))).*
    "Para poder realizar la acción 'colgar espejo' (hangAMirror(mirry, m1, sd1, s1)) es necesario contar con m1 (espejo), sd1 (destornillador) y s1 (tornillo)".
  - *believe(picty, imply(do(mirry, hangAMirror(mirry, m1, sd1, s1)), mirrorHanging(m1))).*
    "Al ejecutar la acción hangAMirror(mirry, m1, sd1, s1) el espejo m1 quedará colgado."

Picty - Acciones



- *action(hangAPicture(Agent, Picture, Hammer, Nail, Chair),*
  - *[iam(Agent)],*
  - *[cando(Agent, hangAPicture), has(Agent, picture(Picture)), has(Agent, hammer(Hammer)), has(Agent, nail(Nail)), has(Agent, chair(Chair))],*
  - *[pictureHanging(Picture), not(has(Agent, nail(Nail)))]).*

- *action(giveResourceTo(AgentS, AgentD, Resource),*
  - *[isagent(AgentD), isagent(AgentS), notEqual(AgentS, AgentD)],*
  - *[has(AgentS, Resource),* ***do(AgentS, giveResourceTo(AgentS, AgentD, Resource))],***
  - *[has(AgentD, Resource), not(has(AgentS, Resource))]).*

Picty - Acciones

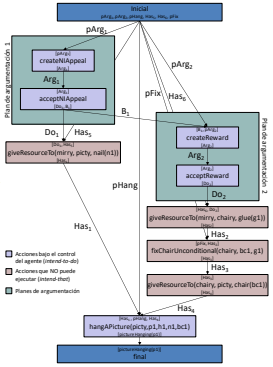
- *action(fixAChair(Agent, BrokenChair, Glue),*
  - *[isagent(Agent)],*
  - *[cando(Agent, fixAChair), has(Agent, brokenChair(BrokenChair)), has(Agent, glue(Glue)),* ***do(Agent, fixAChair(Agent, BrokenChair, Glue))****],*
  - *[has(Agent, chair(BrokenChair)), not(has(Agent, glue(Glue)))]).*

- Acciones incondicionales.

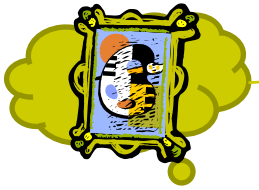
- Acciones para la generación de argumentos.

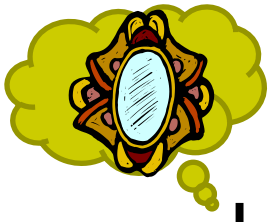
Escenario de la negociación

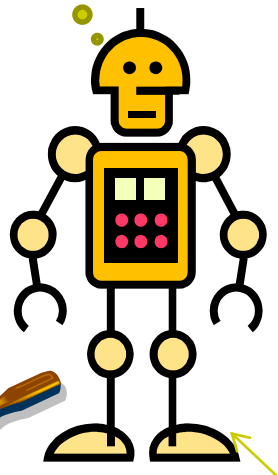
Planner

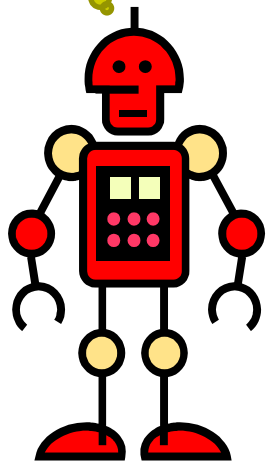


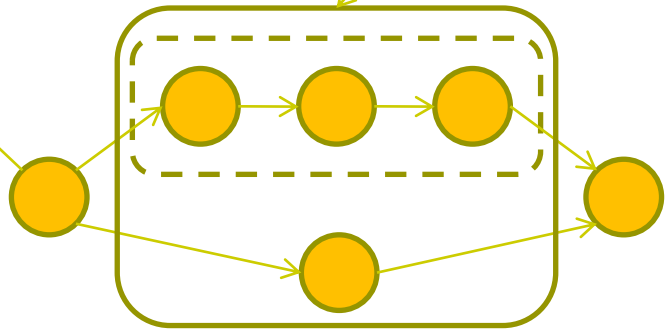
Plan





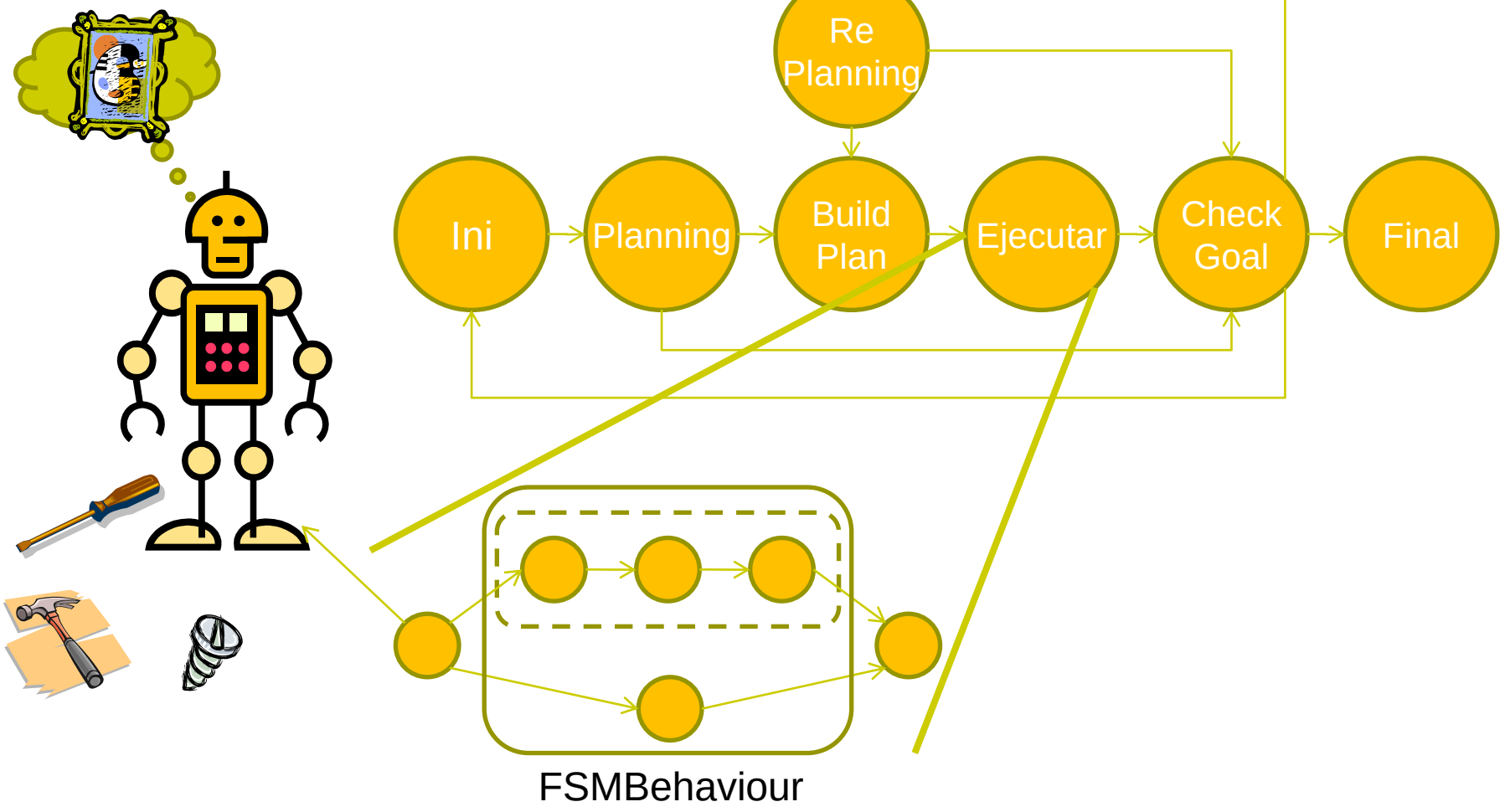




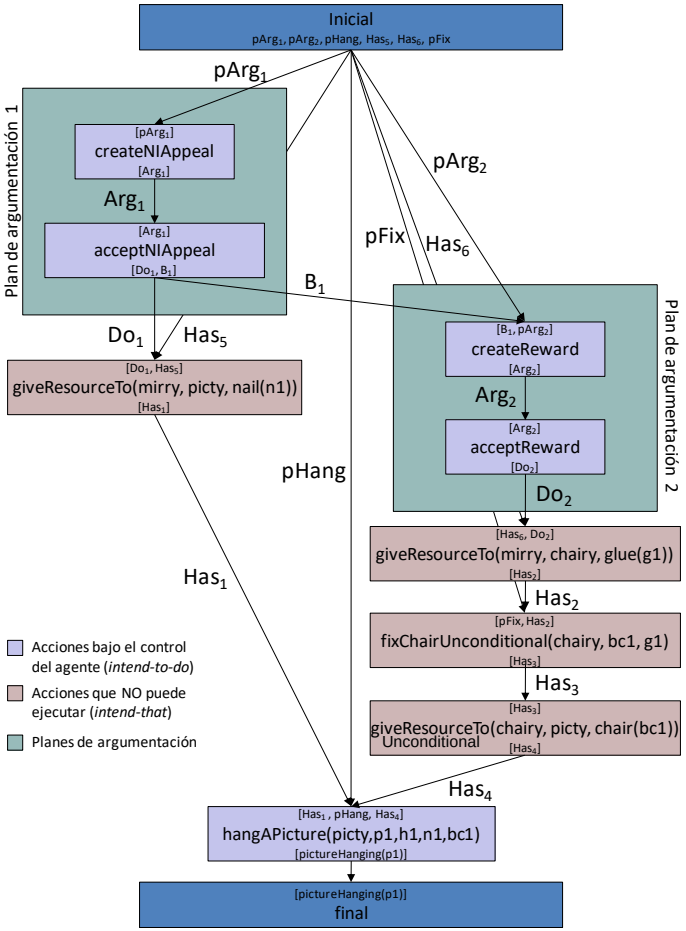


FSMBehaviour

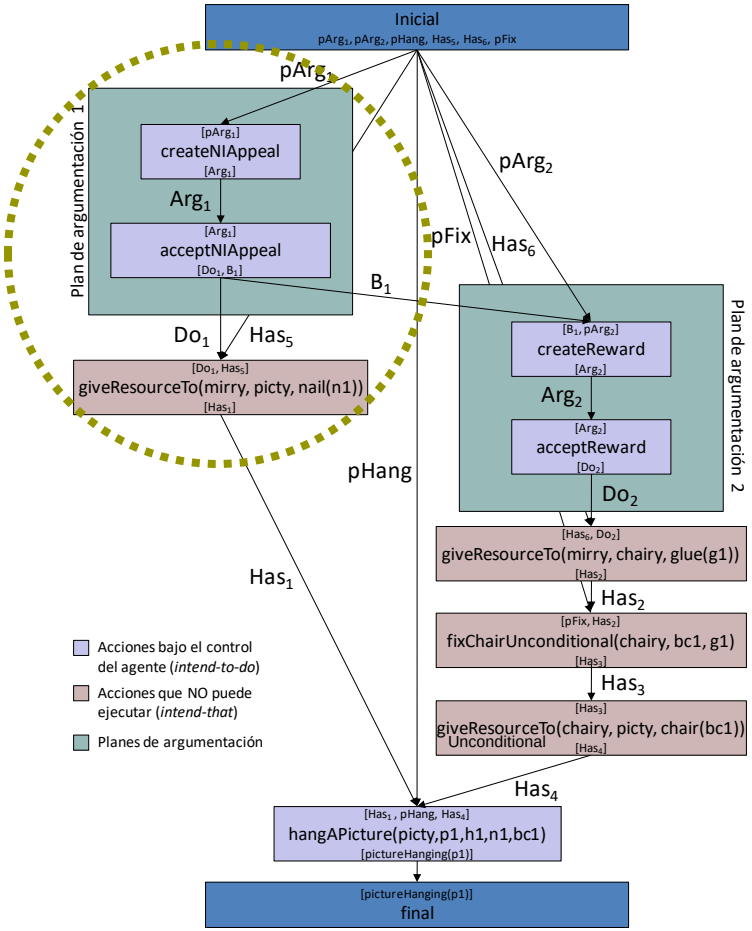
Escenario de la negociación

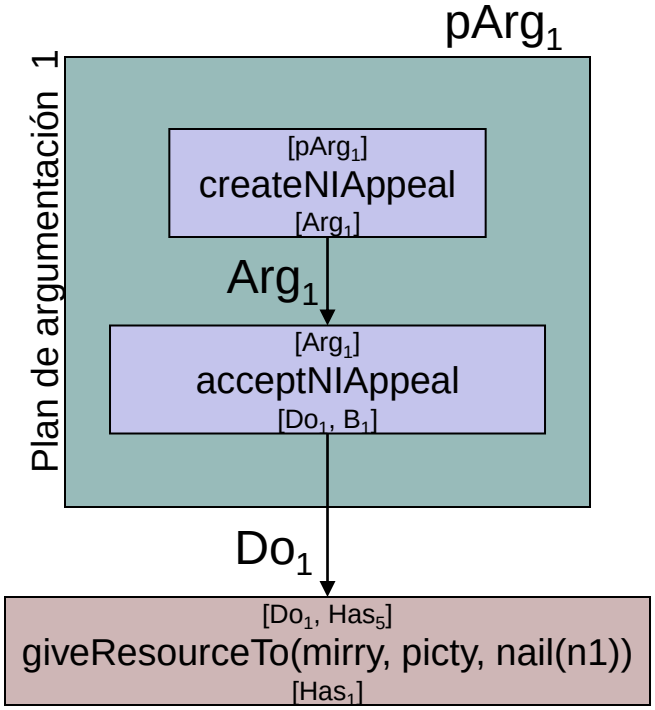


Plan general de *Picty*

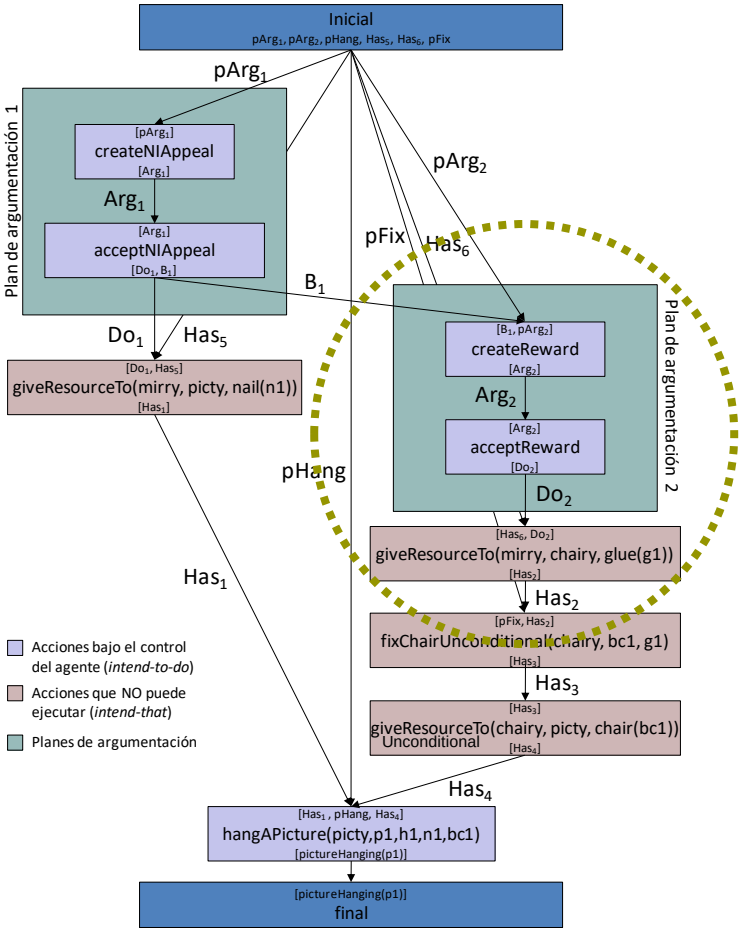


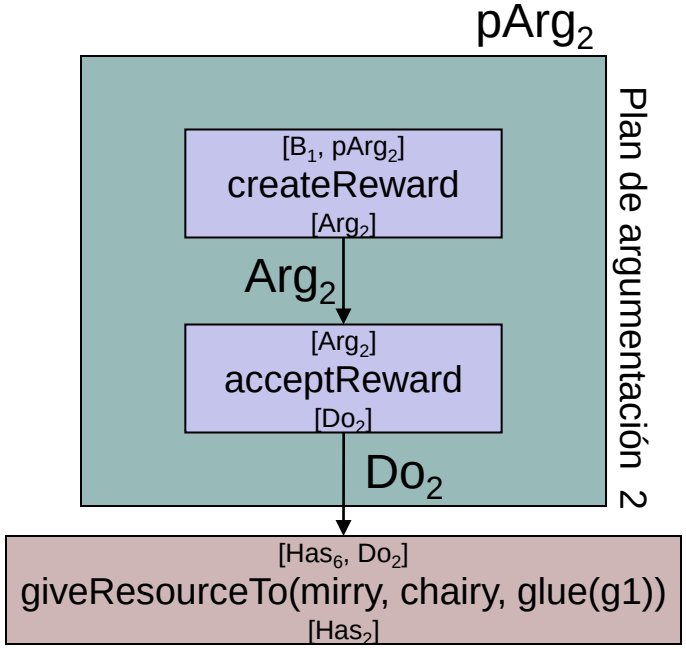
Plan general de *Picty*



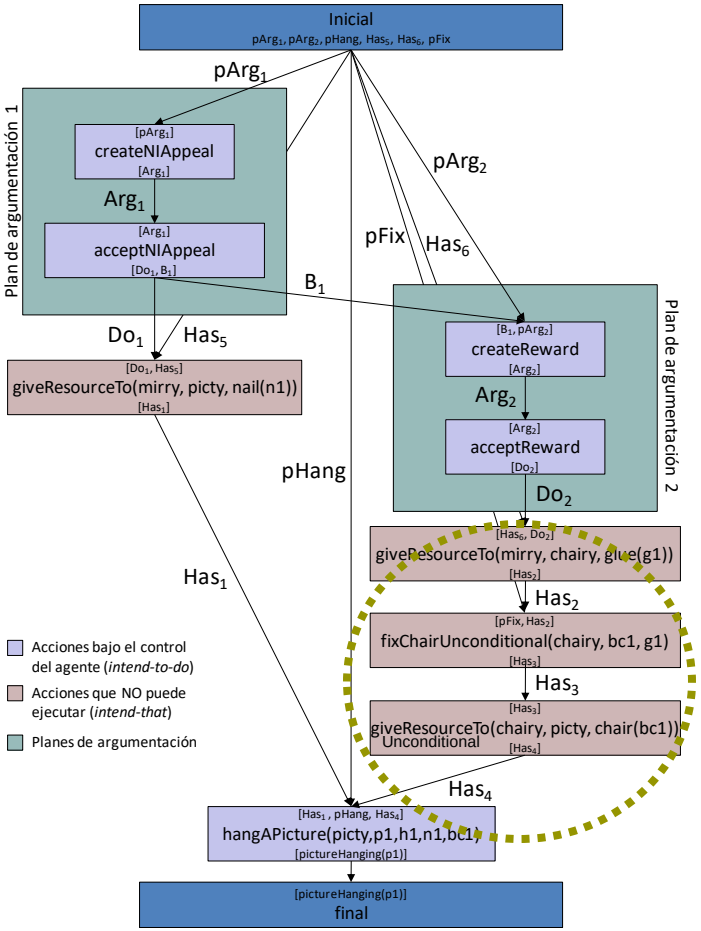


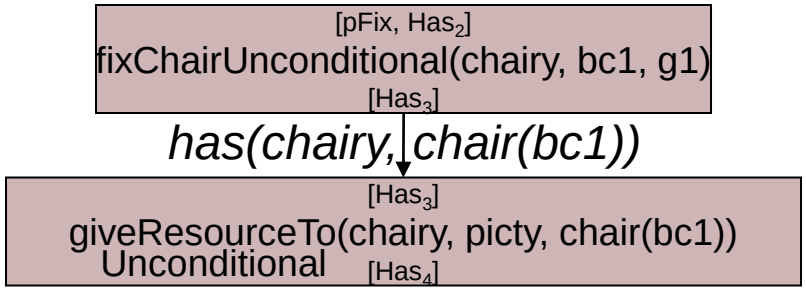
Plan general de *Picty*





Plan general de *Picty*

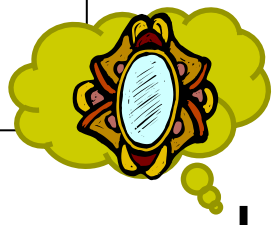


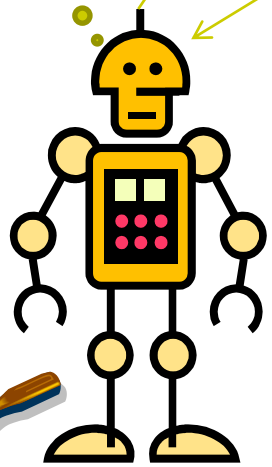


Escenario de la negociación

Javalog

```
appeal(Ue, Ur, do(Ur,Action),
        appeal(Ue, Ur,   do(Ur,
        Action), [pastpromise(Ur,
        Ue, do(Ur, Action))])):-
        iam(Ue), isagent(Ur),
        pastpromise(Ur, Ue,
        do(Ur, Action)).
```



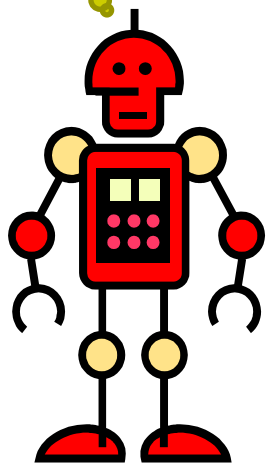


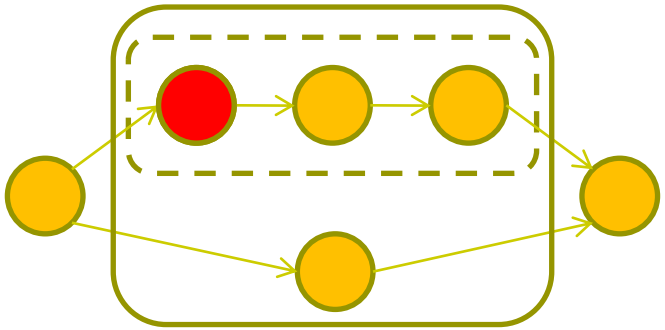
REQUEST

REFUSE

PROPOSE (Argumento)

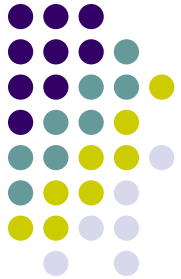
ACCEPT-PROPOSAL





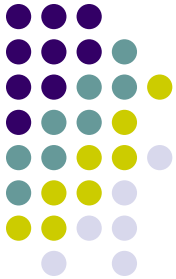
FSMBehaviour

Utilización de UCPOP desde JADE



- Agregar al proyecto Jade.
  - Como Externals Jars:
    - Librerías en Agucpop/libs y Agucpop/lib.
  - Como External Folder
    - Agucpop/libs/classes
  - Dentro de la carpeta “Proyecto”/src
    - Carpetas en Agucpop/src/
  - Dentro de la carpeta “Proyecto”
    - Carpeta Agucpop/config

Clase PlanningUcpop



- Provee lo necesario para la ejecución del algoritmo de planning.

```
PlanningUcpop pu = new PlanningUcpop(new JLInitialiceInterfaz(facts,
    actions, initials, finals));
```

- Métodos
  - `planning()`: ejecuta el algoritmo de planning.
  - `showPlanning()`: arma el String que representa el plan.
  - `getPlan()`: devuelve el plan resultante en formato String.
  - `Graph getGraphOrder()`: devuelve el grafo en donde el orden de ejecución de las acciones es representado.

Taller de Sistemas Multiagentes