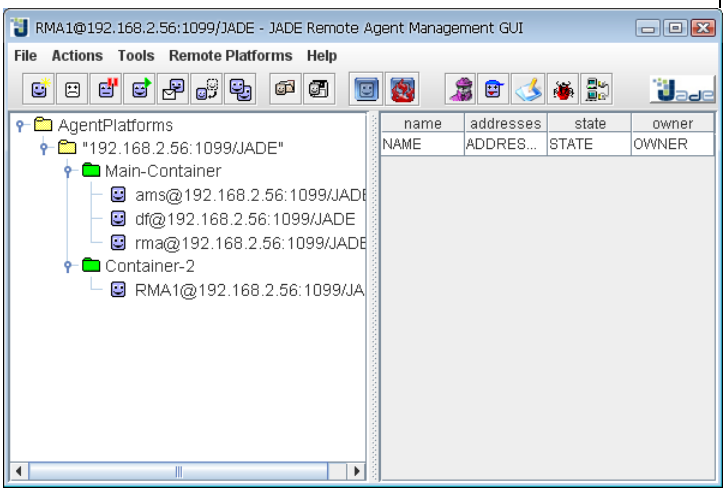
JADE - Movilidad

Taller de sistemas multiagentes

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(), afteClone()`

Movilidad en JADE

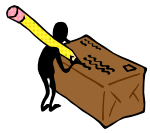
- `Agent.doMove(Location l)`
  - `Location` es una interface abstracta.
- El agente debe consultar por posibles `Location` al AMS.
  - Mediante REQUEST ACL
    - `WhereIsAgentAction`
      - `setAgentIdentifier(AID)`
    - `QueryPlatformLocationsAction`
  - MobilityOntology: `jade-mobility-ontology`
  - `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(mt);

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

Result result = (Result) ce;

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

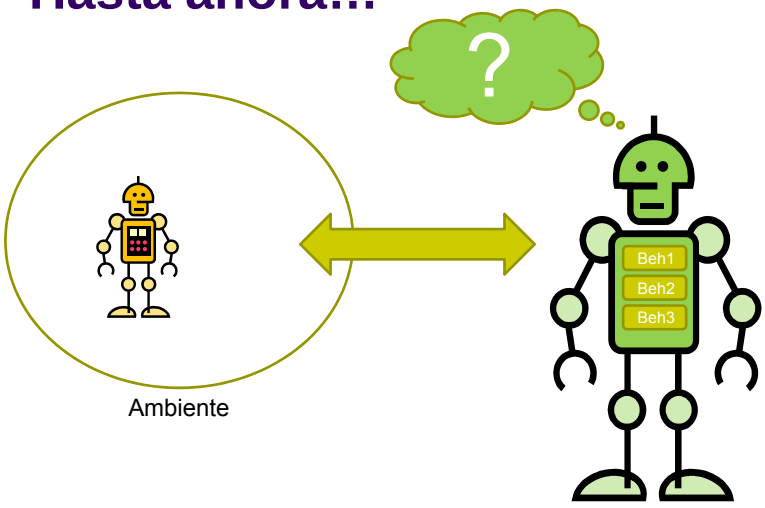
while (it.hasNext()) {
       loc = (Location)it.next();
}
...
myAgent.doMove(loc);
```

Agentes e inteligencia

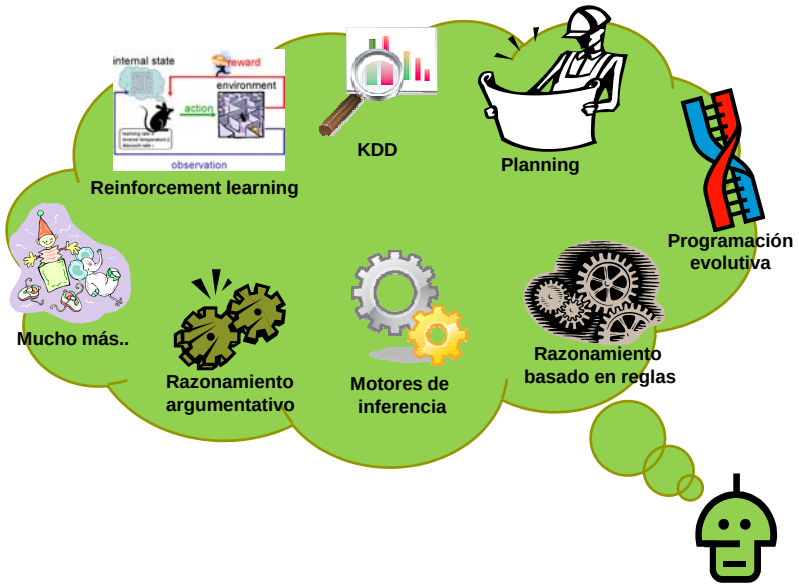
Taller de sistemas multiagentes

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

Hasta ahora…



Aprendizaje y razonamiento



Jess - Java Expert System Shell

Taller de sistemas multiagentes

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

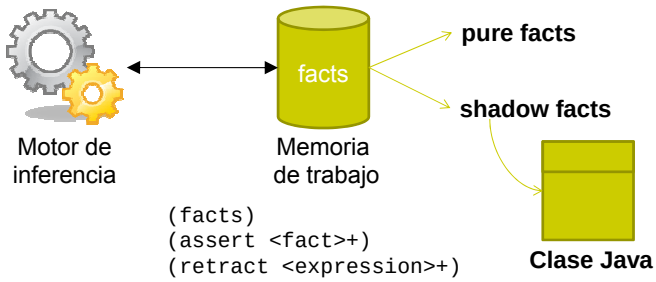
Jess - Java Expert System Shell

- Razonador basado en reglas para la plataforma Java.
  - Basado en el lenguaje de programación CLIPS.
  - 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.

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.
```

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

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)"))
```

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.

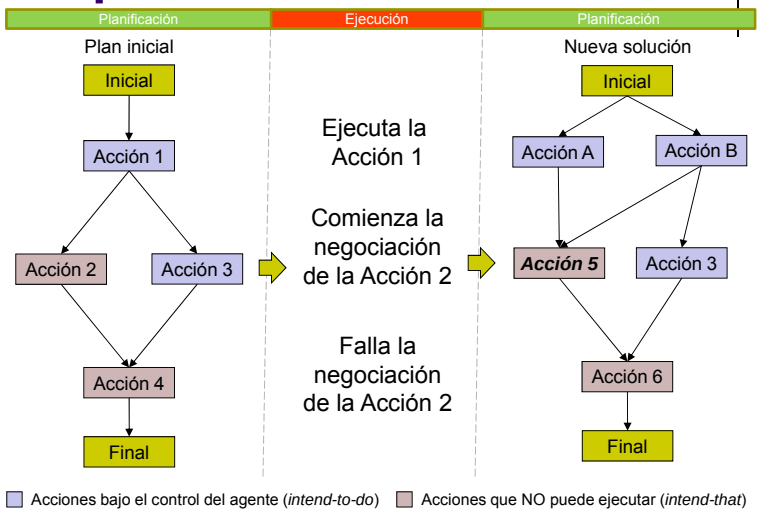
Negociación basada en argumentación

- 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).

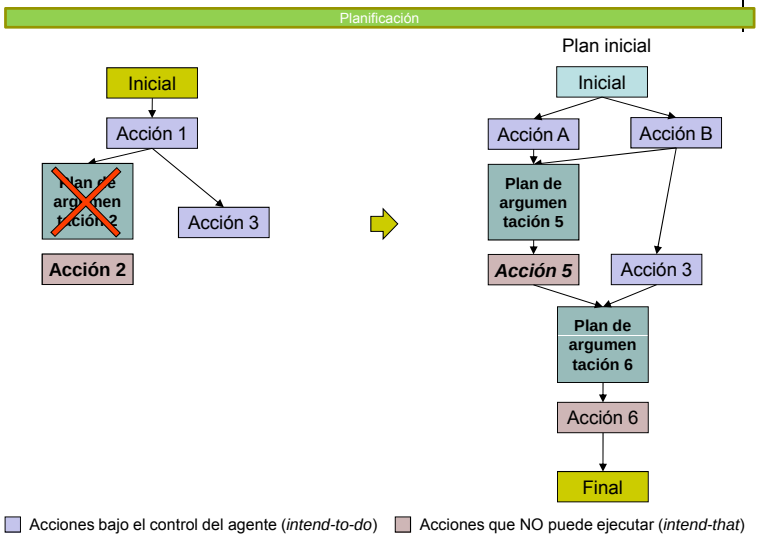
Enfoque tradicional



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.

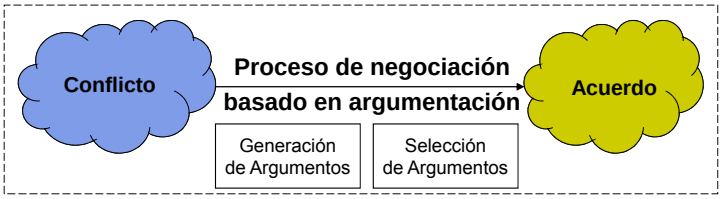
Enfoque planificando la argumentación



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.

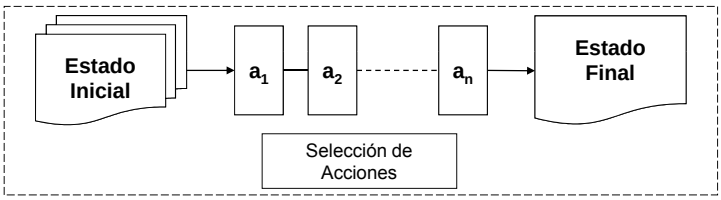
IF
χ solicita la ejecución de una acción α a ψ &
ψ rechaza la propuesta porque α niega su objetivo γ &
χ conoce que ψ ha ejecutado una acción β &
haciendo β se negaba el mismo objetivo γ
THEN
χ solicita la ejecución de α a ψ con la siguiente justificación:
Realizá α porque ya has ejecutado β y β negaba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

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

Selección de argumentos

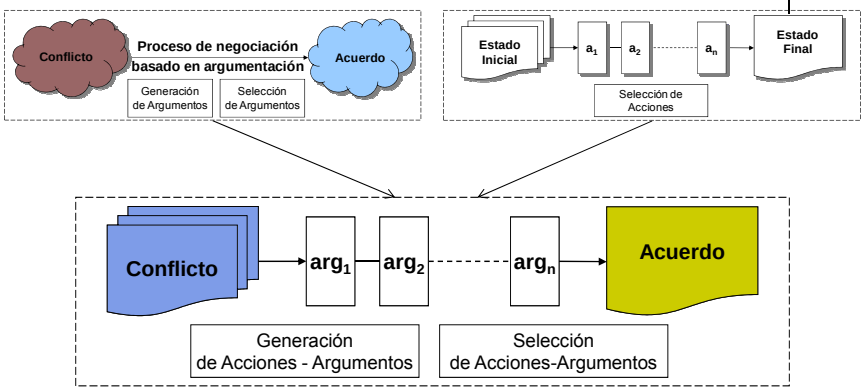
- Dado el conjunto de argumentos candidatos, seleccionar el más adecuado.
- Distintos criterios de selección:
  - Según la fuerza de los argumentos.
  - Según la confianza en el oponente.
  - Según la utilidad de la propuesta respaldada.

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

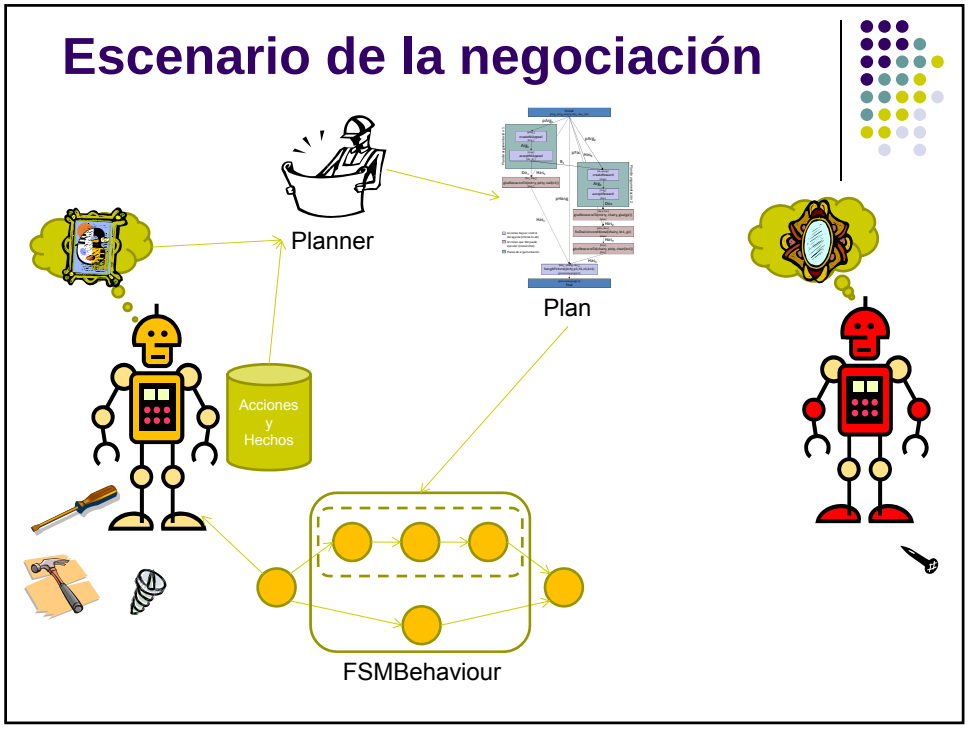
- *action(hangAPicture(Agent, Picture, Hammer, Nail, Chair),*
  - *[iam(Agent)],*
  - *[cando(Agent, hangAPicture), has(Agent, picture(Picture)), has(Agent, hammer(Hammer)), has(Agent, nail(Nail)), has(Agent, chair(Chair))],*
  - *[pictureHanging(Picture), not(has(Agent, nail(Nail)))]).*

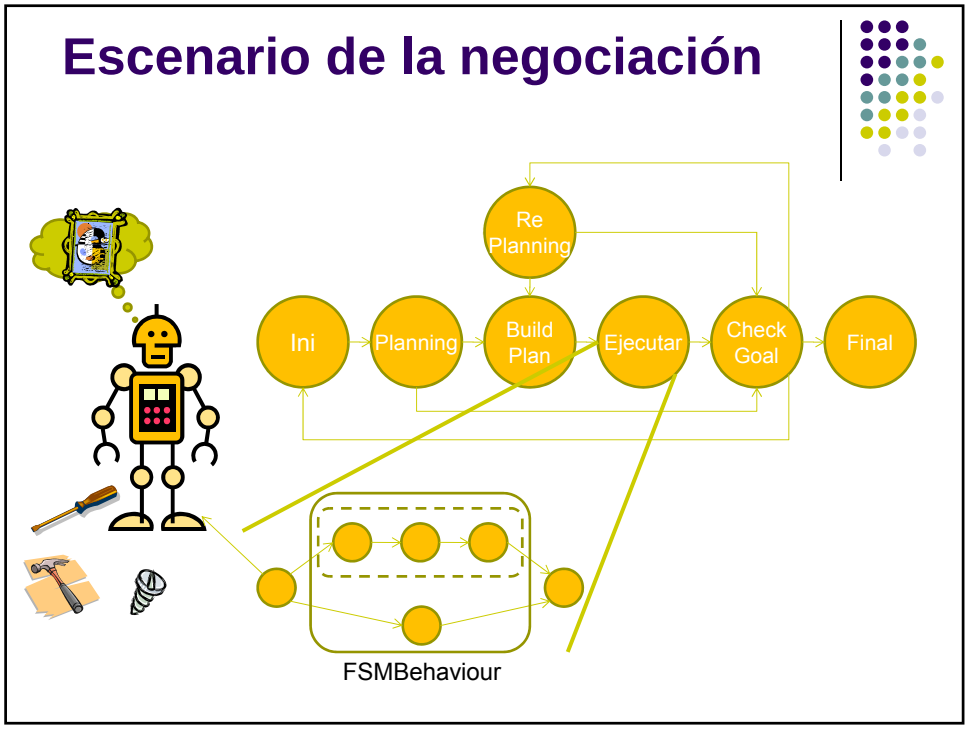
- *action(giveResourceTo(AgentS, AgentD, Resource),*
  - *[isagent(AgentD), isagent(AgentS), notEqual(AgentS, AgentD)],*
  - *[has(AgentS, Resource),* ***do(AgentS, giveResourceTo(AgentS, AgentD, Resource))],***
  - *[has(AgentD, Resource), not(has(AgentS, Resource))]).*

Picty - Acciones

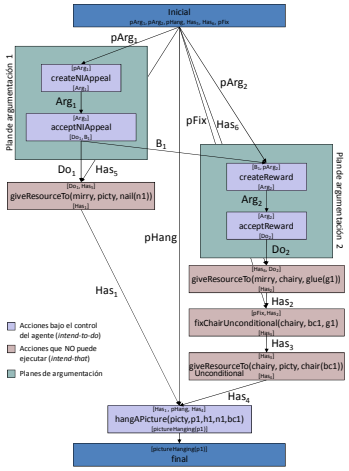
- *action(fixAChair(Agent, BrokenChair, Glue),*
  - *[isagent(Agent)],*
  - *[cando(Agent, fixAChair), has(Agent, brokenChair(BrokenChair)), has(Agent, glue(Glue)),* ***do(Agent, fixAChair(Agent, BrokenChair, Glue))****],*
  - *[has(Agent, chair(BrokenChair)), not(has(Agent, glue(Glue)))]).*

- Acciones incondicionales.
- Acciones para la generación de argumentos.

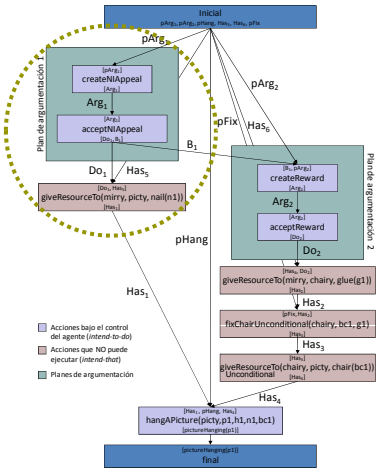

Escenario de la negociación
Planner
Plan
Acciones
y
Hechos
FSMBehaviour

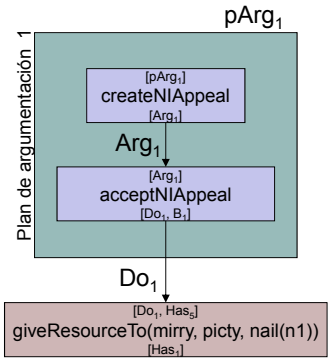

Escenario de la negociación
Re
Planning
Ini
Planning
Build
Plan
Ejecutar
Check
Goal
Final
FSMBehaviour

Plan general de *Picty*

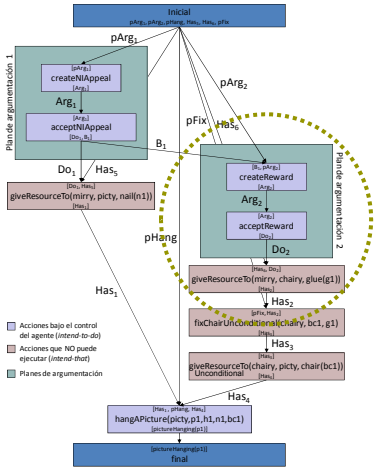


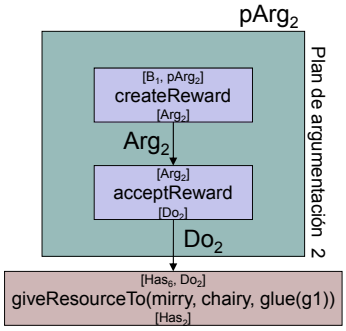
Plan general de *Picty*



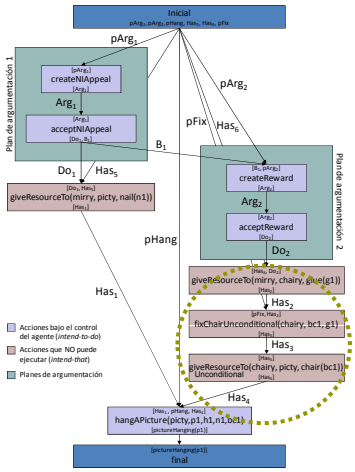


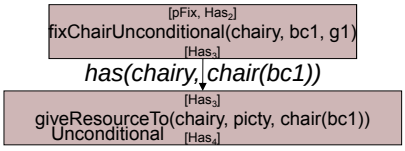
Plan general de *Picty*



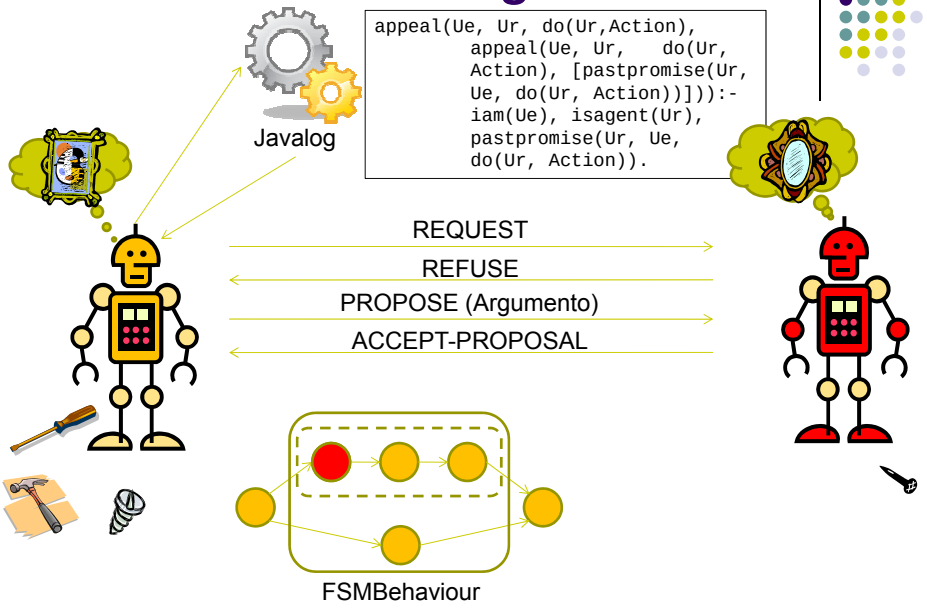


Plan general de *Picty*





Escenario de la negociación

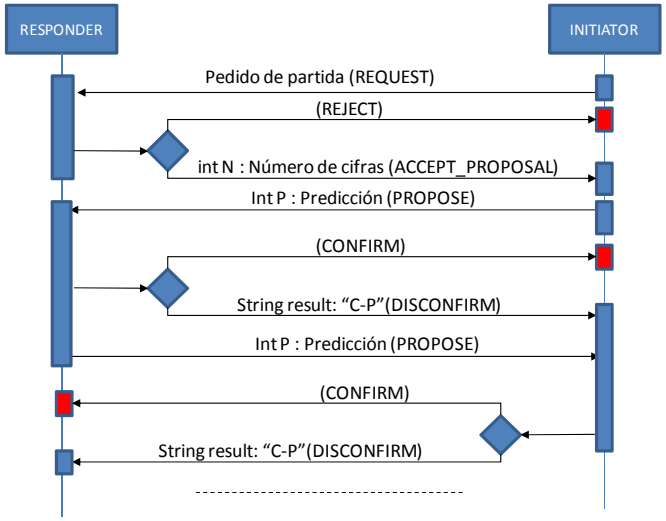


Trabajo Final

Taller de sistemas multiagentes

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

Protocolo MasterMind



FSMBehaviour para MasterMind