Tecnologías Pregunta-Respuesta

Introducción a AgentSpeak

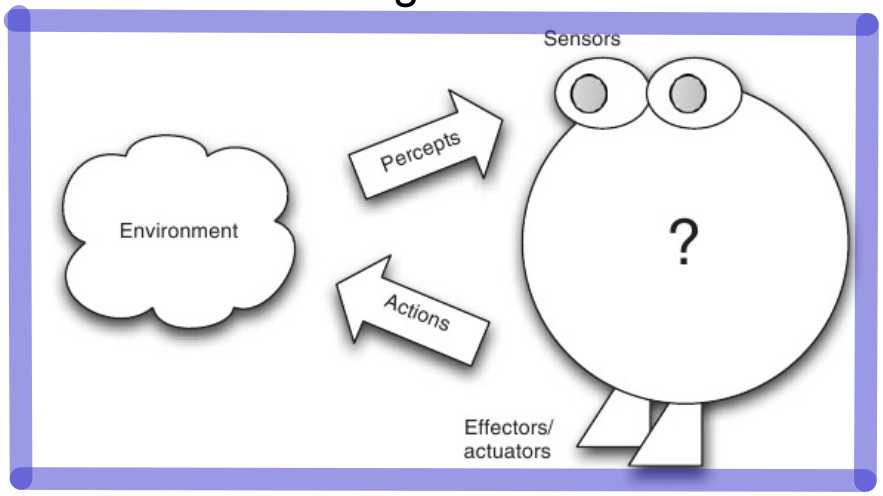
Departamento de Ingeniería de Sistemas Telemáticos
http://moodle.dit.upm.es

Sistemas de agentes

- Agente
  - Tratan de conseguir cumplir unos **objetivos**
  - **Comportamiento** reactivo basado en **planes** y razonamiento básico
- Sistemas multiagente (MAS)
  - Conjunto de agentes desplegados en el mismo **entorno**
  - **Cooperan o compiten** por conseguir sus objetivos

Sistemas de agentes

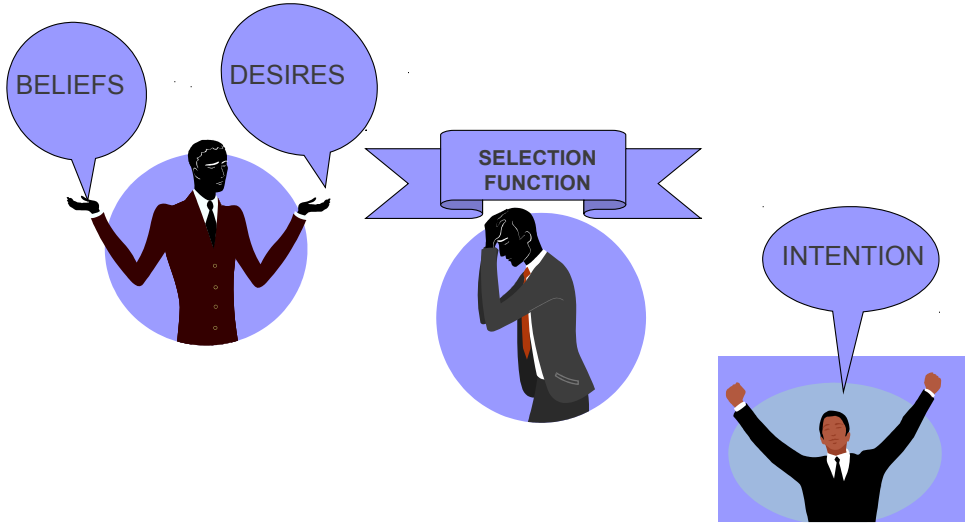
- Relación de los agentes con el entorno



Modelo BDI

- Belief (B)
  - Lo que el agente conoce. Representación del entorno.
  - Componente *informativo*
- Desire (D)
  - Lo que el agente busca conseguir: objetivos
  - Componente *motivacional*
- Intention (I)
  - El curso de acción actual
  - Componente *deliberante*

Modelo BDI



AgentSpeak

- Lenguaje que implementa el modelo BDI
  - Proporciona un **framework** elegante para programar agentes
  - Basado en reglas de primer orden
  - Traduce los conceptos teóricos

Conceptos en AgentSpeak

- El lenguaje está construido sobre tres conceptos
  - Beliefs
  - Goals
  - Plans

Beliefs

- Representación de lo que el agente conoce del entorno

```
weather (sunny)
temperature(25)
weather (london, cloudy)
~likes (john, rock_music)
travel(from(madrid), to(london), plane)
```

Anotaciones

- Dan información adicional a los beliefs

```
weather (sunny)[last_check (14,00)]
temperature(25)[last_check (14,00)]
weather (london, cloudy) [accuracy(0.65)]
~likes (john, rock_music) [degOfCert(0.7)]
travel(from(madrid),to(london))[by(plane)]
```


Fuentes de creencias

- Pueden provenir de tres fuentes
  - Sensores (percepción)

```
weather (sunny)[source(percept)]
```

  - Comunicación

```
weather (london, cloudy)
                [source(weaAg)]
```

  - Deducción (nota mental)

```
~likes (john, rock_music)
                [source(self)]
```


Goals

- Representan objetivos que el agente pretende conseguir
  - Logros (achievement goals)

```
!clean (room1)
!temperature (25)
```

  - Consultas (test goals)

```
?weather (W)
```


Plans

- Acciones básicas que el agente es capaz de llevar a cabo en su entorno

```
triggering_event : context <- body
```

  - Triggering event (te): evento al cual el plan constituye una reacción
  - Context: circunstancias en las cuales el plan puede llevarse a cabo
  - Body: conjunto de **acciones** que el agente llevará a cabo al ejecutar el plan

Plans

```
+!greet
    : true
    <- .print("hello world").
```


Tipos de acciones

- Las acciones constituyen el cuerpo (body) de un plan
  - Internas: definidas en AgentSpeak

```
.print("That is all folks")
.send(tom, tell, john(likes, music))
```

  - Externas: definidas en el entorno

```
open(curtains)
heat
```


Ejemplos de Planes

```
@leave1
+!leave(home)
    : weather(raining)
    <- !pick(umbrella);
       !go(door).

@leave2
+!leave(home)
    : weather(sunny)
    <- !pick(sunglasses);
       !go(door).
```

```
@leave3
+!leave(home)
    : true
    <- open(curtains);
       !leave(home).

@go_door
+!go(door)
    : not at(door)
    <- move_to(door);
       !go(door).
+!go(door): at(door)
    <- open(door).
```


Unificación de variables

- Las variables empiezan con mayúscula

```
+concert (A,V) : likes(A)
     <- !book_tickets(A,V).

+!book_tickets(A, V)
     :  ~busy(phone)
     <- call(V);
        …;
        !choose seats(A,V).
```

```
// beliefs
concert (beatles,
         concert_hall)
```

Rules

- Se usan para obtener conocimiento derivado del existente
  - Ejemplo
    - Si está soleado y hace más de 35 C hay alerta por altas temperaturas
    - Si esta cubierto y la temperatura es menor de 0 C hay alerta de nevadas

Rules

- Systaxis

```
head :- body.
```

```
hightemp_alert() :-
    weather(sunny) &
    temperature(T) & T > 35.
snow_alert() :-
    weather(cloudy) &
    temperature(T) & T < 0.
```


Rules

```
weather(madrid, sunny). /* Beliefs */
temperature(madrid, 31).

hightemp_alert(City) :-
    weather(City, sunny) &
    temperature(City, T) & T > 35.
/* Plan */
+!do : hightemp_alert(madrid) <- !stay(home)
+!do : true <- go(out).
```


Triggering events (te)

- `+b` Aparece una creencia
- `-b` Desaparece una creencia
- `+!g` Nueva meta
- `-!g` Desaparece (o fracasa) una meta
- `+?g` Nueva meta de tipo consulta
- `-?g` Desaparece una meta de tipo consulta

Cambios en las creencias

- Por percepción
  - El propio sistema las añade y elimina en función de la información de los sensores
- Por intención
  - Con el operador '+' o '-' en el cuerpo de un plan

```
+lier(ann) // añade lier(ann)[source(self)]
-lier(ann) // elimina lier(ann)[source(self)]
-+lier(alice) // vuelve a generar el te
```


Cambios en las creencias

- Por comunicación
  - Recepción de mensajes 'tell' o 'untell'

```
.send(tom, tell, lier(ann)) // by john
// añade lier(ann)[source(john)]

.send(tom, untell, lier(ann)) // by john
// elimina lier(ann)[source(john)]
```


Cambios en las metas

- Por intención

```
// añade meta !write(book)[source(self)]
!write(book)
```

- Por comunicación

```
.send(tom, achieve, open(door)) // by tom
// añade meta !open(door)[source(tom)]
```


Jason

- Entorno de ejecución de agentes
- Implementación del lenguaje AgentSpeak
- Íntegramente desarrollado en Java

jason.sourceforge.net/

Ejecución de Jason

- Descomprimir archivo descargado
- En la carpeta de descarga ejecutar

```
./bin/jason.sh
```

- Si hay problemas con variable java_home

```
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
```


Descriptor de proyecto

- Archivo con extensión mas2j

```
hello_world_project {
    infrastructure: Centralised
    environment: es.upm.dit.gsi.TestEnv
    agents: sample;
}
```


Definición de agentes

- Conjunto de reglas, planes y creencias
  - En archivo con extensión asl

```
/* Initial goals */
!greet.
/* Plans */
+!greet : true <- .print("hello world.").
```


Definición de agentes

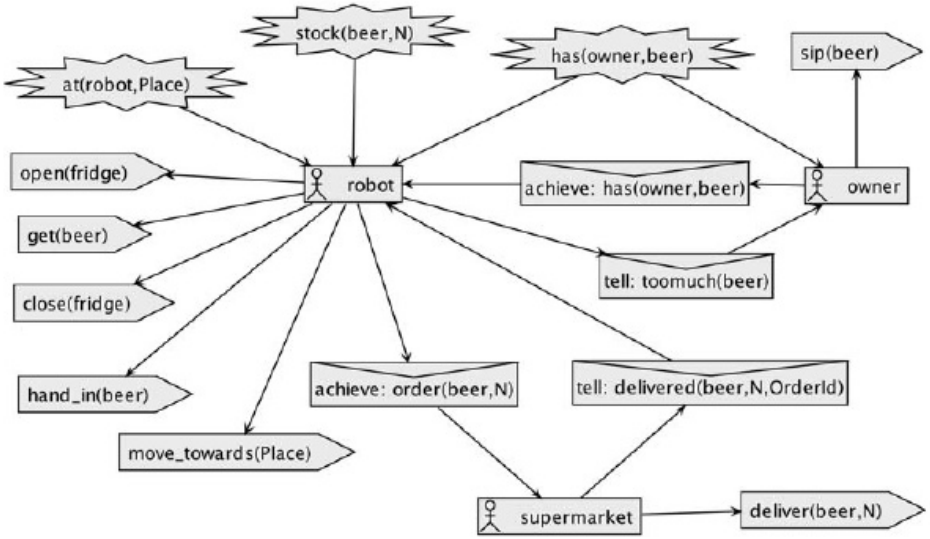
```
/* Initial goals */
!greet(john).
!greet(ann).

/* Plans */
+!greet(Name)
        : true
        <- .print("hello ", Name).
```

Ejemplo completo

"A domestic robot has the goal of serving beer to its owner. Its mission is quite simple, it just receives some beer requests from the owner, goes to the fridge, takes out a bottle of beer, and brings it back to the owner.

»However, the robot should also be concerned with the beer stock (and eventually order more beer using the supermarket's home delivery service) and some rules hard-wired into the robot by the Department of Health (in this example this rule defines the limit of daily beer consumption)".

Ejemplo completo



Owner Agent

```
/* Initial goals */
!get(beer).

/* Plans */
@g  +!get(beer) : true
      <- .send(robot, achieve, has(owner,beer)).
@h1 +has(owner,beer) : true
      <- !drink(beer).
@h2 -has(owner,beer) : true
      <- !get(beer).
```


Owner Agent

```
// while I have beer, sip
@d1 +!drink(beer) : has(owner,beer)
      <- sip(beer);
         !drink(beer).
@d2 +!drink(beer) : not has(owner,beer)
      <- true.

+msg(M)[source(Ag)] : true
      <- .print("Message from ",Ag,": ",M);
         -msg(M).
```

Supermarker Agent

```
last_order_id(1). // initial belief

// plan to achieve the the goal "order"
+!order(Product,Qtd)[source(Ag)] : true
       <-  ?last_order_id(N);
           OrderId = N + 1;
           -+last_order_id(OrderId);
           deliver(Product,Qtd);
           .send(Ag, tell,
           delivered(Product,Qtd,OrderId)).
```


Robot Agent

```
/* Initial beliefs */

// initially, I believe that there are some
   beers in the fridge

available(beer,fridge).


// my owner should not consume more than 10
   beers a day :-)

limit(beer,10).
```


Robot Agent

```
/* Rules */

too_much(B) :-
    .date(YY,MM,DD) &
    .count(consumed(YY,MM,DD,_,_,_,B),QtdB) &
    limit(B,Limit) &
    QtdB > Limit.
```


Robot Agent

- Procesado del pedido del *supermercado* y actualizar el *stock*

```
// when the supermarket finishes the order,
// try the 'has' goal again
@a1
+delivered(beer,Qtd,OrderId)[source(supermarket)]
        : true
        <-  +available(beer,fridge);
            !has(owner,beer).
```

Robot Agent

```
// when the fridge is openned, the beer stock is perceived
// and thus the available belief is updated
@a2 +stock(beer,0)
      :  available(beer,fridge)
     <- -available(beer,fridge).

@a3 +stock(beer,N)
      : N > 0 & not available(beer,fridge)
     <- +available(beer,fridge).
```


Robot Agent

- Planes para *desplazarse* por el entorno

```
@m1 +!at(robot,P) : at(robot,P)
      <- true.

@m2 +!at(robot,P) : not at(robot,P)
      <- move_towards(P);
         !at(robot,P).
```


Robot Agent

- Planes para *transportar* la bebida al dueño

```
@h1 +!has(owner,beer)
      : available(beer,fridge) & not too_much(beer)
      <-  !at(robot,fridge);
          open(fridge); get(beer);close(fridge);
          !at(robot,owner);
          hand_in(beer);
// remember that another beer will be consumed
          .date(YY,MM,DD); .time(HH,NN,SS);
          +consumed(YY,MM,DD,HH,NN,SS,beer).
```


Robot Agent

```
@h2 +!has(owner,beer)
   :  not available(beer,fridge)
   <- .send(supermarket, achieve, order(beer,5));
      !at(robot,fridge).
@h3 +!has(owner,beer)
   :  too_much(beer) &
      limit(beer,L)
   <- .concat("You cannot have mora than ", L,
        " beers a day! I'm sorry ",M);
      .send(owner,tell,msg(M)).
```

Ejercicios

- **E1:** Los planes h1, h2 y h3 se ejecutan cuando en el agente robot aparece la meta !has(owner, beer). (fácil)
  - Modifícalos para que sólo se ejecuten cuando el origen de la meta sea el propio agente.
  - ¿Crees que esto puede hacer que deje de funcionar el sistema? ¿Por qué?

Ejercicios

- **E2:** Mejora el código del supermarket agent para tener en cuenta el stock de cerveza. Supón un stock inicial de 20 cervezas y disminuye dicho stock con cada envío al robot. Si no quedan suficientes cervezas
  - Informa al robot de que no cerveza (fácil)
  - Envía todas las que queden (medio)

Ejercicios

- **E3:** ¿Qué ocurre si se intercambian de orden los planes m2 y m1? (fácil)

  En caso de que el código fuera

```
@m1 +!at(robot,P) : at(robot,P)
    <- true.
@m2 +!at(robot,P) : true
    <- move_towards(P);
!at(robot,P).
```

  ¿Qué ocurre si se cambian ahora de orden?

Ejercicios

- **E4:** Crea un nuevo agente supermarket con el mismo código del supemarket agent.

  Modíficalos para que nada más iniciar la ejecución le envíen al robot el precio de la cerveza

```
.send(robot,tell,price(beer,3))
```

  Modifica el código del robot para que compre la cerveza del supermercado más barato (medio)