Tema 1. Introducción a los sistemas multiagentes

Sistemas Multiagentes. Curso 2016/2017.

Universidad de Castilla-La Mancha

Septiembre de 2016

Índice

Tendencias

La **historia de la computación** ha venido marcada por **cinco** importantes **tendencias**:

1. Ubicuidad.
2. Interconexión.
3. Inteligencia.
4. Delegación.
5. Dirigido a los humanos.

Ubicuidad

- La **reducción** del **coste** en los sistemas de computación ha hecho posible que se **introduzcan sistemas** informáticos con capacidad de procesamiento en **lugares** y **dispositivos** donde hasta ahora **no** era **rentable** e incluso **inimaginable**.
- Este **tendencia** es **continua**, haciendo que la capacidad de **procesamiento** y por tanto, la **inteligencia** sea de un tipo **ubicuo**.

Interconexión

- Se ha pasado de un conjunto de **computadores aislados** a un conjunto de **computadores interconectados**.
- Ahora es **complejo encontrar** computadores, bien de uso comercial o académico, que **no** están **conectados** a Internet.
- La **norma** en computación **industrial** y **comercial** es el uso de sistemas **concurrentes** y **distribuidos**.
- La **computación** pasa a ser un proceso de **interacción**.

Inteligencia

- Existe una **tendencia** clara hacia la construcción de **sistemas más inteligentes**.
- Esto quiere decir, que cada vez **tareas** más **complejas** pueden ser **automatizadas** y **delegadas** a los **computadores**.
- Se tiene la posibilidad de construir **sistemas** de **computación** que puedan **tratar** con tareas que se consideraban **intratables** hace poco tiempo.

Delegación

- Las **posibilidades** de **"delegar"** son cada vez **mayores**. La **delegación** implica **ceder** el **control** de ciertas tareas **a sistemas** de computación.
- Estas **tareas** cada vez son cada vez más **críticas** en cuanto a seguridad.
- Por ejemplo, **sistemas de ayuda al pilotaje de aviones**. Se da la circunstancia, que en muchos casos, tiene **más peso** a la hora de tomar ciertas decisiones el **programa informático** que la propia experiencia de los pilotos.

Orientado al humano I

- Hay un **alejamiento progresivo** de una visión a la hora de **programar** vinculada a la **máquina**. Esto supone un **acercamiento** hacia conceptos y **metáforas** que representan de una manera más cercana la forma en la que las **personas entendemos** el **mundo**.
- Se ha pasado de una **interacción** con la **máquina** a través de **interruptores** y la necesidad de un **conocimiento interno** de la misma a un **control** de los dispositivos por parte del usuario mediante la manipulación de **iconos gráficos** que corresponden con programas y archivos.

Orientado al humano II

- De manera similar, la forma primera de programar era mediante **código máquina puro**, que requería de un conocimiento del funcionamiento interno de la máquina. Se ha pasado por **ensamblador**, paradigma **procedimental**, **TADs**, y más recientemente el paradigma de los **objetos**. Esto ha permitido una manera de programar más **orientada** al **humano**.

Retos I

- Con respecto a la **ubicuidad** y la **interconexión**, **no** se conocen técnicas que permitan **explotar** todo el **potencial** de procesadores **distribuidos**. Estas sólo se muestra **útiles** cuando el número de **procesadores** es relativamente **pequeño**. ¿Qué ocurre si tenemos 10^{10} procesadores?
- Si se quiere **incrementar** el **grado** de **delegación** e inteligencia, es necesario construir **sistemas** de **cómputo** que puedan **actuar** en nuestro nombre. Esto conlleva **dos implicaciones**: la **primera** es que los **sistemas** deben **operar** de manera **independiente**, **sin** intervenciones humanas **externas**. La **segunda**, es la necesidad de sistemas que sean capaces de actuar de tal manera que **representen** nuestros **intereses** de la **mejor manera** posible, mientras **interactúan** bien con **humanos**, bien con otros **sistemas**.

Retos II

- Las **tendencia** hacia una **mayor interconexión** y distribución es reconocida como clave y en las últimas décadas se ha **invertido** un **esfuerzo** importante en el **desarrollo** de de **herramientas** software y mecanismos que nos permiten construir **sistemas distribuidos** con facilidad y fiabilidad. Sin embargo, si unimos esto a la necesidad de que los sistemas **representen** lo mejor posible **nuestros intereses**, derivamos en un nuevo problema. Cuando un **computador** actuando en nuestro nombre debe **interactuar** con otro sistema que representa los **intereses** de un **tercero**, probablemente ocurra que estos intereses no sean los mismos. Por tanto, se hace necesario, dotar a estos sistemas con la habilidad de **cooperar** y alcanzar **acuerdos** con otros sistemas.

Aparición de los SMAs

Todo lo anterior **conduce** a la **aparición** de un nuevo campo denominado **Sistemas Multiagentes**:

- **Agente**: **computador** + **actuación** de manera independiente **representando** los intereses de su **propietario**.
- **Multiagentes**: varios **agentes** que **interactúan** entre ellos, por ejemplo, **intercambio** de **mensajes** en red y con capacidad de **cooperar**, **coordinarse** y **negociar**.

Objetivos de la asignatura

1. **Diseño de agentes**: **construir** agentes con capacidad de actuar de manera **autónoma** para llevar a cabo las **tareas** que le han sido **delegadas**.
2. **Diseño** de **sociedades** de **agentes**: construir **agentes** con **capacidad** de **interactuar** con otros, cuando estos otros, **no comparten** los mismos **intereses** u **objetivos**.

Cuestiones de inicio

1. ¿Cómo puede **existir** la **cooperación** en sociedades de agentes con **intereses particulares** (**self-interested**)?
2. ¿Qué tipos de **lenguajes** pueden utilizar los **agentes** para **comunicarse** bien con **personas** o con otros **agentes**?
3. ¿Cómo pueden **detectar** los propios **agentes** que están en una **situación** de **conflicto**?
4. ¿Cómo pueden **alcanzar acuerdos** con otros **guardando** su propio **interés** y sin entrar en **conflictos**?
5. ¿Cómo pueden los **agentes autónomos coordinar** sus **actividades** para de manera **cooperativa** alcanzar objetivos?

Índice

La sonda espacial

- Due to an unexpected system failure, a space probe approaching Saturn loses contact with its Earth-based ground crew and becomes disoriented. Rather than simply disappearing into the void, the probe recognizes that there has been a key system failure, diagnoses and isolates the fault, and correctly reorients itself in order to make contact with its ground crew.

Sistema de control de tráfico aéreo

- A key air-traffic control system at the main airport of Ruritania suddenly fails, leaving the flights in the vicinity of the airport with no air-traffic control support. Fortunately, autonomous air-traffic control systems in nearby airports recognize the failure of their peer, and cooperate to track and deal with all affected flights. The potentially disastrous situation passes without incident.

Búsqueda de vacaciones

- After the wettest and coldest (UK) winter on record, you are in desparate need of a last minute holiday somewhere warm and dry. After specifying your requirements to your personal digital assistant (PDA), it converses with a number of different Web sites, which sell services such as flights, hotel rooms, and hire cars. After hard negotiation on your behalf with a range of sites, your PDA presents you with a package holiday.

Interdisciplinaridad

Una vez **estudiados** estos **ejemplos**, se puede **afirmar** que el **campo** de los sistemas multiagentes es **multidisciplinar**, inspirándose en diversas áreas como son:

- Ciencias económicas.
- Filosofía.
- Lógica.
- Ecología.
- Ciencias sociales.

Índice

No son más que simples sistemas concurrentes/distribuidos

- **Teoría 1**. **No existe** una **diferenciación** de los **agentes** con los **sistemas concurrentes** que justifiquen su existencia, se deberán tener conocimientos en programación concurrente para controlar situaciones de **exclusión mutua**, evitando **interbloqueos**, etc.
- **Teoría 2**. Son una **subclase** de los **sistemas concurrentes** con dos **diferencias**: **autónomos** con capacidad de tomar decisiones y los agentes se mueven por su **propio interés**.

No es más que inteligencia artificial

- **Teoría 1**. Los **sistemas** multiagentes no son más que un **subcampo** de la **IA**. La **IA** está **relacionada** con amplios temas como son, **aprendizaje**, **planificación**, **visión** por computador, etc.
- **Teoría 2**. Los **sistemas multiagentes** son **99 %** ciencias de la computación y **1 %** IA. Para la gran **mayoría** de **aplicaciones**, **no** es **necesario** que el agente tenga **todas** las **capacidades** estudiadas en **IA**, incluso algunas como el aprendizaje pueden ser **no deseables**. Además, la IA clásica ha venido ignorando aspectos sociales como son la cooperación, la negociación y la toma de acuerdos.

No es más que teoría de juegos

- **Teoría 1**. Los **sistemas multiagentes** no son más que un **subcampo** de la **teoría** de **juegos**. Esta teoría se utiliza en **procesos** de **negociación**, y es una herramienta teórica utilizada de manera dominante en el análisis de sistemas multiagentes.
- **Teoría 2**. Muchos de los **conceptos** existentes en teoría de juegos se desarrollan **sin relación** alguna con la **computación**. Se indican de manera descriptiva y son a menudo **computacionalmente duros** (NP-completos).

Bibliografía

- Capítulo 1 del libro: **Introduction to Multiagent Systems**. Michael Woolridge. 2001. Ed. John Wiley & Sons, Inc.

Tema 1. Introducción a los sistemas multiagentes

Sistemas Multiagentes. Curso 2016/2017.

Universidad de Castilla-La Mancha

Septiembre de 2016

TEMA 2. AGENTES INTELIGENTES

Sistemas Multiagentes. Curso 2015/2016.

Universidad de Castilla-La Mancha.

Septiembre de 2015

Índice

Objetivos

- El **objetivo** principal de este tema es el de **definir** qué son los **agentes**.
- Además, se **estudiarán** aspectos asociados a la **construcción** de los mismos.
- En temas posteriores se estudiarán **aproximaciones específicas** en la **construcción** de agentes.
- Esta asignatura **no** trata de los **agentes individuales** sino de los sistemas **multiagentes**.

Definición de agente I

Definición

Un agente es un **sistema** de **computación** que está situado en un **entorno**, y que es capaz de actuar de manera **autónoma** en dicho entorno para alcanzar los **objetivos** para los que fue **diseñado**.

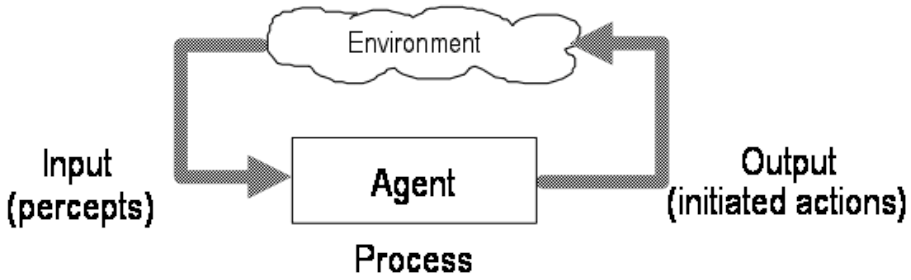


Figura 1 : Un agente en su entorno

Definición de agente II

Definición (ii)

Un agente es todo aquello que **percibe** su **entorno** mediante **sensores** y que responde o actúa en tal ambiente por medio de **efectores**

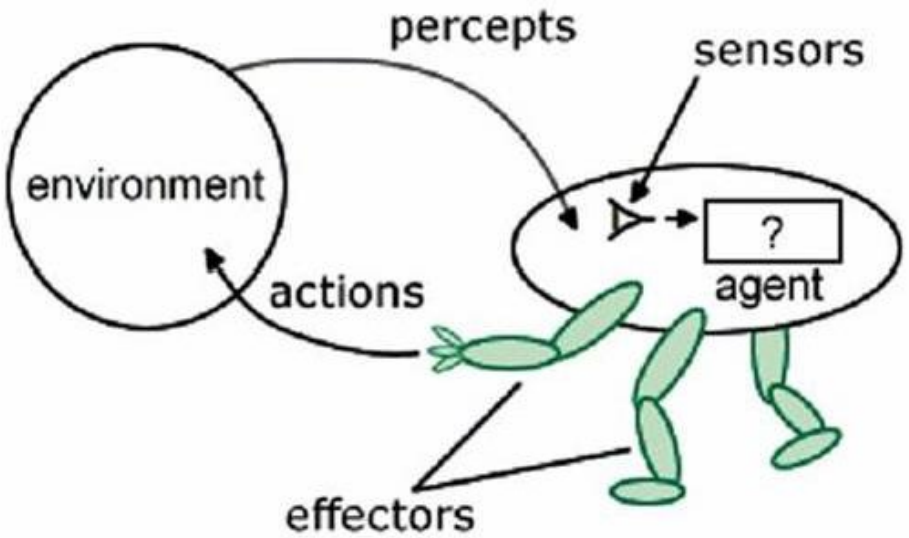


Figura 2 : Un agente en su entorno (ii). Sensores y efectores.

Definición de agente III

- En **dominios** de una **complejidad** razonable, el agente **no** puede tener un **control** absoluto sobre el **entorno**.
- Esto significa que la **misma acción** realizada dos veces bajo, a priori, las mismas circunstancias puede tener **efectos distintos**.
- Por tanto, los **agentes** incluso en los entornos más triviales deben estar **preparados** para **"fallar"**.
- Se puede resumir toda esta situación diciendo que se debe **asumir** que los **entornos** normalmente son **"no determinísticos"**.

Definición de agente IV

- Un **agente** puede **realizar** una serie de **acciones**. Se conoce como **"capacidad efectora del agente"** y se puede ver como la **habilidad** que tiene de **modificar** el **entorno**.
- El agente **no** puede **ejecutar** todas sus **acciones** en **todas** las **situaciones** posibles. Cada **acción** tiene vinculado un conjunto de **precondiciones**. Estas precondiciones determinan las **situaciones** posibles en las que pueden ser **aplicadas**.
- ¿Cuáles de las **acciones posibles** debe realizar el agente para **satisfacer** en mayor grado los **objetivos** para los que fue **diseñado**? Los agentes son **arquitecturas software** para la **toma** de **decisiones**.

Ejemplo de agente. Termostato

- El **termostato** tiene un **sensor** embebido en el **entorno** (habitación).
- El sensor produce dos **salidas** "temperatura **muy baja**" o "temperatura **correcta**".
- Las **acciones** que puede realizar el termostato son **"calentar"** (efecto no garantizado) y "**no calentar**".
- Las **reglas** para la toma de decisiones son:
  1. temperatura muy **baja** $\rightarrow$ **calentar**
  2. temperatura **correcta** $\rightarrow$ **no calentar**

Ejemplo de agente. Demonio Software

- Los **demonios** son **procesos** que se suelen ejecutar en **segundo plano** y que **monitorizan** un **entorno** software. Pueden verse como agentes.
- Por ejemplo, **xbiff**, continuamente comprueba si hay **correo entrante**. Cuando lo hay muestra un **icono** en el escritorio.
- La **información** del entorno la **recaba** mediante la ejecución de **órdenes** del **sistema** (p.e. ls).
- Las acciones que realiza son **"acciones software"**.

Ejemplos de sensores.

- **Temperatura**
- Luz (**color**)
- **Distancia**
- Sonido
- Posición
- **Orientación**
- Profundidad
- Presión
- Peso
- **Ph**
- Calor

Ejemplos de efectores.

- **Rotación** de un **Motor**.
- Apertura o cierre de **articulaciones**.
- **Pantalla**.
- Sistemas de **generación** de **voz**.
- Apertura o cierre de **válvulas**.
- **Movimiento** de **articulaciones**.

Ejercicio.

- ¿Diga qué **sensores** tienen los **humanos**?
- ¿Conoce algún **sensor** que tienen los **animales** pero no lo tienen los **humanos**?
- ¿Conoce algún **sensor** que tienen los **humanos** pero **no** lo tienen los **animales**?
- ¿Cuáles son los **efectores** de los **humanos**?
- ¿Conoce algún **efector** que tienen los **animales** pero no lo tienen los **humanos**?, ¿Viceversa?
- ¿Cuáles son los **sensores** y **efectores** de un agente **robótico**?

Ejercicio.

Diga si los siguientes **sistemas** son "**agentes**":

1. Un **reloj**.
2. Un **teléfono** móvil.
3. Un sistema de **control** de acceso a **personas**.
4. El sistema de **aire** acondicionado de un **automóvil**.
5. Una **lavadora** automática.
6. Un **software** para **entrenar** personas.
7. Un **software** para enseñar a **sumar**.
8. Un reloj **despertador**.
9. **Internet**.

Índice

Medida de rendimiento

- Evalúa **cómo** de **exitoso** ha sido un **agente**.
- Debe ser **objetiva**, impuesta por un **experto** y que **no** sea **aplicable** por igual a **todos** los **agentes**.
- **Ejemplo**: agente que se encarga de **limpiar** con una **aspiradora** un **suelo** sucio.
- ¿**Cuándo** se mide?

Agente racional I

- La **racionalidad** no es **omnisciencia**, **clarividencia**, ni **exitosa** necesariamente. Ejemplo: **cruzar** la **calle**.
- Un **agente** racional es aquel que **hace** lo **correcto**. ¿Qué **significa** esto?
- Como primera aproximación, se puede **afirmar** que lo **correcto** es aquello que permite al agente obtener el **mejor rendimiento**.

Agente racional II

- Para cada **conjunto** de **percepciones**, el agente **selecciona** la **acción** que **maximiza** su **rendimiento** basado en la información de la **percepción** y su propio **conocimiento** incorporado en tal agente.

Dependerá por tanto de:

- **Medida** del grado de **éxito**.
- **Secuencia** de **percepciones**.
- **Conocimiento** acerca del **medio**.
- **Acciones** que puede emprender.

Ejercicio.

Indique para los siguientes **"agentes"** qué medidas de **rendimiento** usaría:

1. **Alumno** de la asignatura de SMA.
2. **Práctica** número 1 de la asignatura de SMA.
3. **Docente** del curso de SMA.
4. Congresista.
5. **Personal** que recoge la **basura** en las noches.
6. **Software** para jugar **ajedrez**.
7. Google **noticias**.
8. **Software** que resuelve **laberintos**.

Agente autónomo

- Un **agente** es más **autónomo** en la medida en que su **comportamiento** se basa:
  - (+) en el **aprendizaje** y
  - (-) en el **conocimiento** incorporado.
- Si las **acciones** del agente se **basan** en un **conocimiento** integrado previamente, **no** es **autónomo**.
- Un **sistema** será **autónomo** en la medida en que su **conducta** está definida por su **propia experiencia**.

Índice

Percepciones

- El **comportamiento** de un **agente** depende de la **secuencia** de **percepciones** en un momento dado.
- Se puede **caracterizar** un **agente** elaborando una **tabla** de **percepciones** $\rightarrow$ **acciones**.

Mapeo ideal

Es aquel **mapeo** que **especifica** que tipo de **acción** deberá **emprender** un **agente** como respuesta a una determinada **secuencia** de **percepciones**.

Mapeo

- **Mapeo** de secuencias de **percepciones** para **acciones**:

Mapeo

$$percepcion \rightarrow accion$$

- **Mapeo** Ideal:

Mapeo Ideal

$$p_i \rightarrow a_i$$
$$p_1 \rightarrow a_1 \rightarrow p_2 \rightarrow a_2$$
$$p_1 \rightarrow a_1 \rightarrow p_2 \rightarrow a_2 \rightarrow p_3 \rightarrow a_3$$

- La **construcción** de la **tabla** percepción/acción puede llegar a tener 35^{100} **entradas** en el caso por ejemplo del **ajedrez**.

Ejemplo. Agente resolutor de SQRT en una calculadora

Percepción	**Acción**
1.0	1.000000000000000
1.1	1.048808848170152
1.2	1.095445115010332
1.3	1.140175425099138
1.4	1.183215956619923
1.5	1.224744871391589
1.6	1.264911064067352
1.7	1.303840481040530
1.8	1.341640786499874
1.9	1.378404875209022

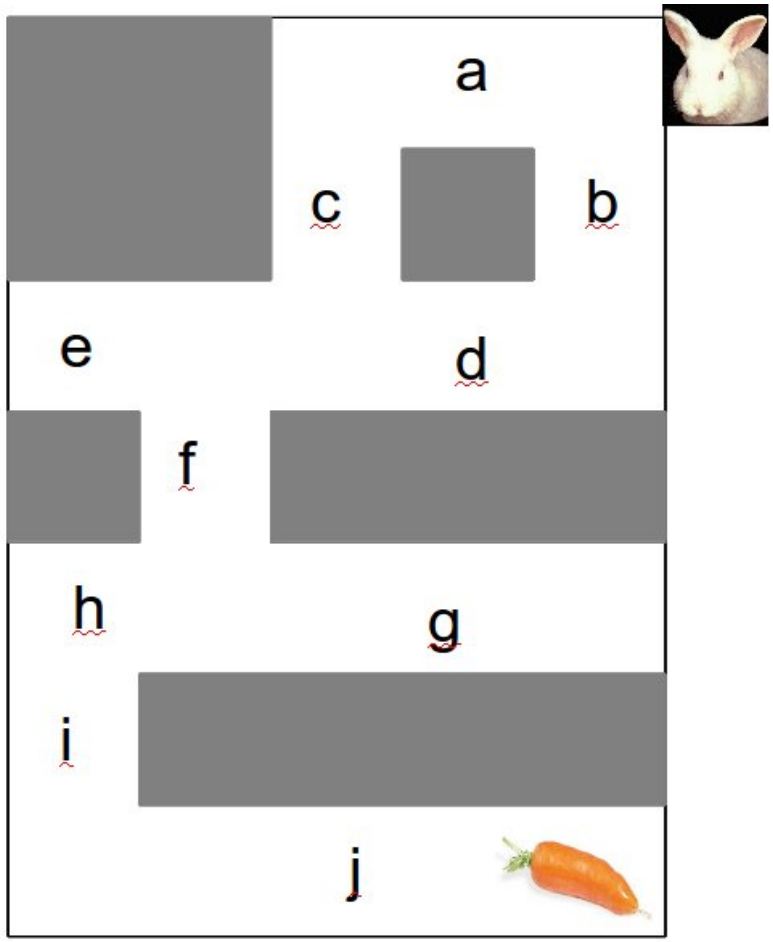
Ejemplo. Agente resolutor de SQRT en una calculadora

Programa que implementa el mapeo ideal

1: function SQRT(x)
2: z $\leftarrow$ 1.0
3: repeat until $\mid z^2 - x \mid < 10^{-15}$
4: $z \leftarrow z - (z^2 - x)/(2z)$
5: end
5: return z
6:end function

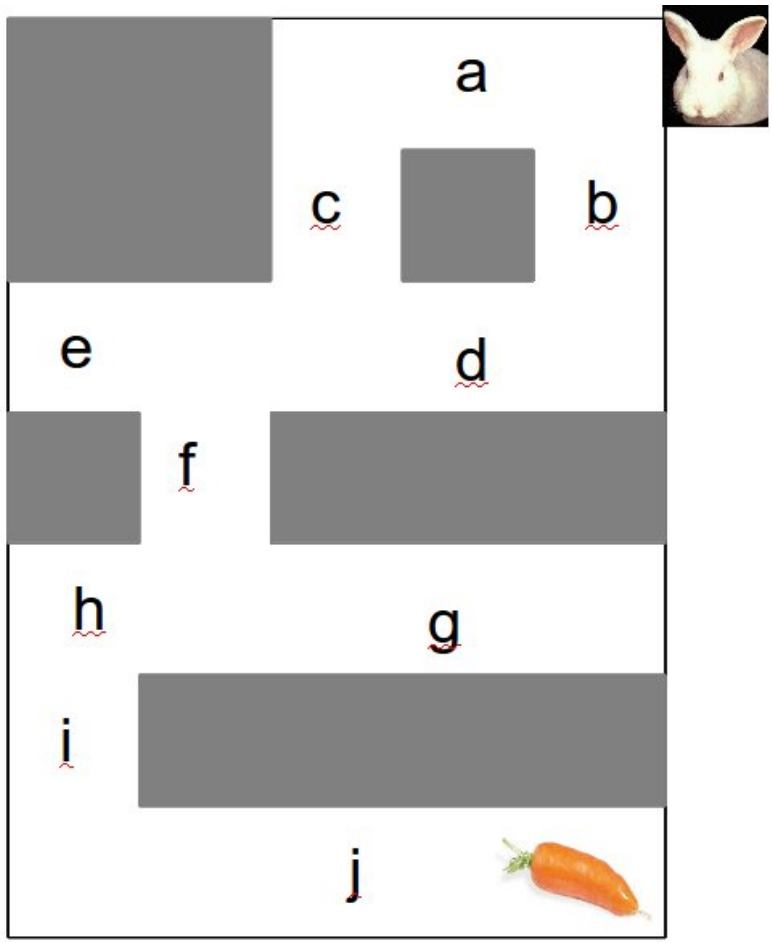
Ejercicio. Diseñar un agente para resolver el problema.

1. ¿Cuál es el **problema**?
2. ¿Cuáles son las **entradas** de la **tabla**?
3. ¿Se puede **diseñar** un **agente** que mediante una tabla *percepcion* $\rightarrow$ *accion* **resuelva** este **problema**?



Ejercicio

1. entrada → a
2. entrada → b
3. a → entrada
4. a → b
5. a → c
6. b → entrada
7. b → a
8. b → d
9. c → a
10. c → d
11. c → e
12. c → f
13. ...



Índice

Clasificación

- **Observable** vs. **Parcialmente observable**.
- **Determinista** vs. **No determinista/estocásticos**.
- **Episódico** vs. **Secuencial**.
- **Estático** vs. **Dinámico**.
- **Discreto** vs. **Continuo**.

Clasificación I

- **Observable**. El agente puede obtener **información completa**, **precisa** y **actualizada** del estado del entorno.
- **Determinista** vs. **No determinista/estocástico**. Un entorno **determinista** es aquel en el que cualquier **acción** tiene un único **efecto garantizado**.
- **Episódico**, cuando la **experiencia** del agente se divide en **episodios**. Cada episodio **consiste** en la **percepción** del agente y la **realización** de una única **acción** posterior. Si es **episódico**, es más **simple**. Ej: **Cadena** de **montaje**.
- **Estático** vs. **Dinámico**. Un entorno estático es aquel que permanece **inalterable salvo** por los **efectos** de las acciones del **agente**. En el dinámico puede haber **otros procesos** interactuando con el entorno.

Clasificación II

- **Discreto** vs. **Continuo**. Un entorno es discreto si existe un número **fijo** de **percepciones** que se pueden realizar sobre él.
- Mientras **más observable** es un entorno **más sencilla** es la **construcción** de **agentes** que operen de manera efectiva en él (**calidad** de la información).
- En un escenario **determinista** el agente puede asumir que todas sus **acciones** van a tener un **efecto** determinado. (escenarios **reales**, no deterministas).
- Los algoritmos de **planificación** en **IA**, asumen escenarios **estáticos** a la hora de ejecutar algoritmos de planificación. Los escenarios **reales** contienen múltiples **procesos** que se ejecutan de manera **concurrente** y por lo tanto existen **"interferencias"**. Una **precondición**, puede no cumplirse mientras que se ejecuta un proceso.

Clasificación III

- Un entorno **discreto** puede ser el juego del **ajedrez** ya que tiene un número finito de **estados** y uno **continuo**, la **conducción** de un **taxi**.
- Aunque los **computadores** son sistemas **discretos** y pueden simular con cierto **grado** de **precisión** un sistema **continuo** siempre dicha **representación** será **aproximada**.
- Por tanto, un **agente** (estados **discretos**) a la hora de **seleccionar** un acción en un **entorno** continuo se **basará** en información "**imprecisa**".
- En resumen, se puede decir que los entornos **más complejos** son aquellos que son **parcialmente observables**, **no deterministas**, **dinámicos** y **continuos**. Se conocen como **abiertos**.

Ejercicio

Entorno: **Observable**, **Determinístico**, **Episódico**, **Estático**, **Discreto**

- **Crucigrama**
- **Ajedrez** con reloj
- Póquer
- Backgammon
- **Taxi** circulando
- Diagnóstico **médico**
- Análisis de imagen
- Robot **clasificador**
- **Controlador** de refinería
- Tutor **interactivo** de inglés

Tipos de interacción agente-entorno I

- Originalmente se ha trabajado con los conocido **sistemas funcionales**: $f : I \rightarrow O$. (Ej. Compilador).
- Se pueden formular en base a un conjunto de **precondiciones** que representan que debe **ocurrir** para que el programa **funcione correctamente** y a un conjunto de **postcondiciones** representando el **estado** del **escenario** una vez el programa ha **finalizado**.
- Sobre todo, una de las **claves** de los sistemas **funcionales** es que éstos **terminan**. ¿Qué pasa si una **postcondición** deja de cumplirse? ¿O una precondición?

Tipos de interacción agente-entorno II

- En teoría de agentes, esto **puede no darse** si existen **otros** procesos **actuando** sobre el **entorno** (sistemas multiagentes).
- **No** todos los sistemas computacionales se **asocian** al paradigma **funcional**. Algunos puede considerarse como **reactivos**: aquellos que **responden rápidamente** a cambios en su **entorno** o que actúa directamente sobre el mundo antes que razonar de manera explícita sobre él. Son mucho **más complejos** de construir que los **funcionales**.
- Los sistemas **reactivos** no pueden ser **adecuadamente** descritos desde el **punto** de vista **racional** o funcional sino que se **describen** por su **comportamiento** a lo largo de su **ejecución**.

Índice

Estructura de los agentes inteligentes

- Un **propósito** de la **Inteligencia Artificial** es el **diseño** de un **programa** de **agente** (una función que mapee de percepciones a acciones).
- Este **programa** se **ejecutará** en algún **dispositivo** de **cómputo**, o arquitectura.

Agente

$$Agente = Arquitectura + Programa$$

- **Arquitectura**. Pone al alcance del programa las **percepciones** obtenidas mediante los **sensores**, lo **ejecuta** y **alimenta** el efector con **acciones** elegidas por el **programa** conforme se van generando.
- **Programa**. Es un **algoritmo** que recibe las **percepciones** del agente y **genera** una **secuencia** de acciones.

Programa de ejecución en el entorno

Programa

```
1: procedure Ejecutar-Ambiente(e, Funcion-Actualizacion, agentes, fin)
2:     Entradas: e, estado inicial del entorno.
3:               Función-Actualización, modifica el entorno
4:               agentes, conjunto de agentes
5:               fin, predicado para comprobar objetivos cumplidos
6:    repeat
7:        Para cada agente ∈ agentes
8:            PERCEPCION[agente] ← OBTENER_PERCEPCION(agente, e)
9:        end Para
10:       Para cada agente ∈ agentes
11:           ACCION[agente] ← PROGRAMA[agente](PERCEPCION[agente])
12:       end Para
13:       e ← Funcion-Actualizacion(ACCION[],agentes,e)
14:    until fin(e)
15:end procedure
```

PAMA/REAS

Antes de **diseñar** un programa de agente, hay que hacer la **descripción PAMA/REAS**.

- **Percepciones** / Sensores
- Acciones / **Actuadores**
- **Metas** / Rendimiento
- Ambiente / **Entorno**

Agente: sistemas de diagnóstico médico.

- **Rendimiento**: paciente **saludable**, reducción al mínimo de **costos**.
- **Entorno**: paciente, **hospital**.
- **Actuadores**: **preguntas**, pruebas, tratamientos.
- **Sensores**: síntomas, **evidencias** y respuestas del **paciente**.

Agente: robot clasificador de partes.

- **Rendimiento**: poner las **partes** en el contenedor correspondiente.
- **Entorno**: banda transportadora de partes.
- **Actuadores**: recoger partes y clasificarlas en contenedores.
- **Sensores**: píxeles de **intensidad** variable.

Agente: resolución del problema de las ocho fichas.

- **Rendimiento**: estado **final**.
- **Entorno**: **posición** de las fichas.
- **Actuadores**: movimiento de una **ficha**.
- **Sensores**: alguno de los **estados**.

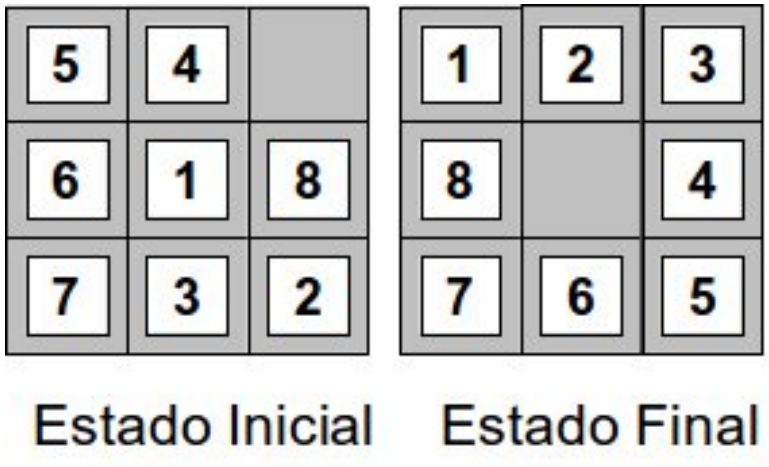


Figura 3 : Rompecabezas de las ocho fichas

Agente: conductor de un taxi

- **Rendimiento**:
- **Entorno**:
- **Actuadores**:
- **Sensores**:

Agente: conductor de un taxi

- **Rendimiento**: Viaje **seguro** y **rápido**, sin **infracciones**, cómodo, obtención **máxima** de **ganancias**.
- **Entorno**: **Caminos**, tráfico, **peatones**, clientes.
- **Actuadores**: Manejo del **volante**, **acelerar**, frenar, hablar con el **pasajero**.
- **Sensores**: Cámaras, **velocímetro**, acelerómetro, sistema **GPS**, sonar, **micrófono**.

Agente: sistema de análisis de imágenes satélite

- **Rendimiento**:
- **Entorno**:
- **Actuadores**:
- **Sensores**:

Criterios

- El **área** de agentes es bastante **heterogénea**, por ejemplo, existen diferentes **concepciones** del **modelo** de agente y muy **diferentes aplicaciones**.

Se podría dar una **clasificación** de los programas de agente atendiendo a los siguientes **criterios**:

- Sus **capacidades**: aprendizaje, **autonomía**, cooperación, etc.
- Su **función**: **ayuda** al usuario, **recolección** de información, **resolución** de problemas, etc.
- Su **arquitectura**: deliberativa o **reactiva**.
- Su **manera** de **organizarse**: **individual**, colectiva, sistemas **abiertos**, sistemas **cerrados**.

Programa de agente dirigido por tabla I

Programa

```
1: function AGENTE-CONDUCIDO-MEDIANTE TABLA (e)
2:         static: percepciones, secuencia inicialmente vacía
3:                 Tabla, una tabla, indexada por secuencia percepciones
4:         Añadir la percepción al final de todas las percepciones
5:         accion ← CONSULTA(percepciones, tabla)
6:         return accion
7:end function
```

Programa de agente dirigido por tabla II

- La **medición** del **rendimiento** no forma parte del **programa** (externo; raíz cuadrada)
- Juego del **ajedrez** necesitaría una **tabla** de 35^{100} **entradas**.
- La **elaboración** de tablas puede llever mucho **tiempo** al **diseñador**.
- Agente sin **autonomía** (cálculo de acción en tabla). Si el **ambiente** se modifica en algún **aspecto**, ¿Qué ocurriría?
- Si se implementa **aprendizaje** ¿Se tendría que realizar para cada **entrada** de la **tabla**? Costoso en **tiempo**.

Clasificación

- **Agentes reactivos (reflejo simple)**. Las **acciones** del agente se **establecen** en **función** a una **tabla** de percepción → acción.
- **Agentes con estado interno (bien informados/basados en modelos)**. Es un agente **reactivo**, pero que **almacena** sus **percepciones** anteriores, tiene memoria.
- **Agentes basados en objetivos (metas)**. Agente que **combina propiedades** de los dos anteriores, pero que tiene un **objetivo** a **alcanzar**. Necesita **buscar** el mejor **camino** y **planificar** la secuencia de **acciones**.
- **Agentes basados en utilidad**. Son aquellos **agentes** que tienen **múltiples objetivos** que cumplir, mide el grado de **satisfacción** en el cumplimiento de sus objetivos.

Agentes reactivos I

- Los agentes **reactivos**, o **reflex**, seleccionan sus **acciones** basados en su **percepción actual** del entorno, **ignorando** el **resto** de su **historia** perceptual.

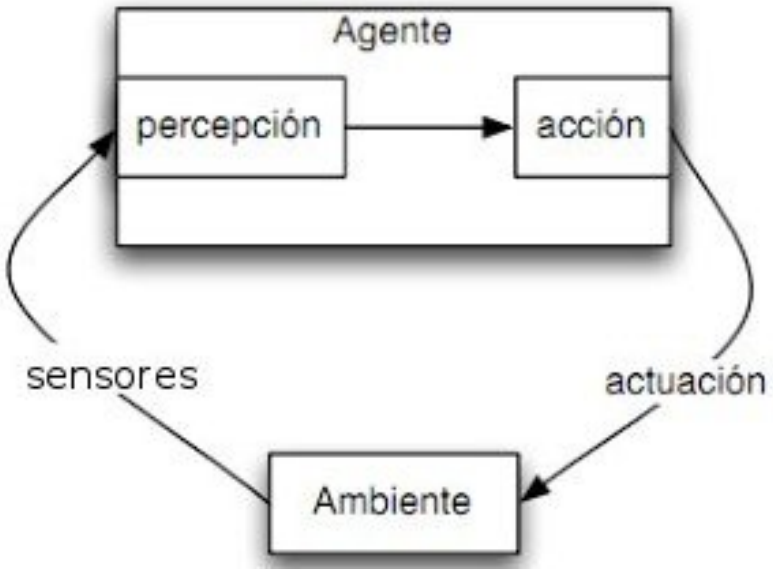


Figura 4 : Esquema funcional de un agente reactivo (i)

Agentes reactivos II

- **Usar** una **tabla** de consulta **explícita** está **fuera** de toda consideración.
- Sin embargo, es posible **resumir fragmentos** de **tabla** observando ciertas **asociaciones** entre **entradas/salidas** que se producen frecuentemente, y haciendo **reglas** de **condición/acción**, por ejemplo: "Si el **vehículo** de adelante está **frenando**, entonces empezar a **frenar**".
- Las **reglas** condición/acción tienen la forma: **IF condición THEN acción**.
- Se establece **correspondencia** entre la **percepción** dentro de un **conjunto** de **reglas** y se especifica la **acción** a tomar.

Agentes reactivos III

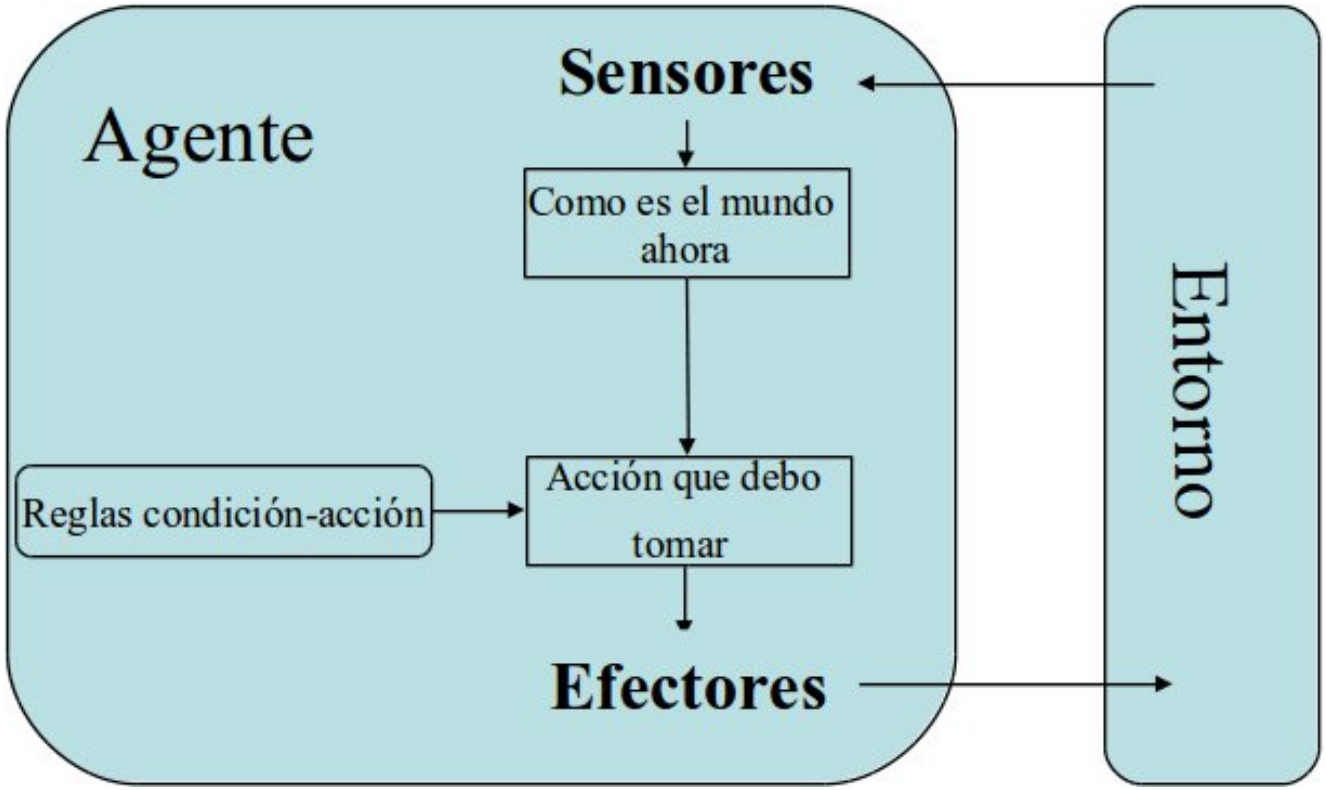


Figura 5 : Esquema funcional de un agente reactivo (ii)

Programa de agente reactivo

Programa

```
1: function AGENTE-REACTIVO(e)
2:          percepcion ← percibir(e)
3:          regla ← seleccionAcción(percepcion,reglas) (reglas predefinidas)
4:          accion← accionRegla(regla)
5:          return accion
6:end function
```

Limitaciones de los agentes reactivos

- Aunque hay **otras** maneras de **implementar** de **agentes** reactivos (arquitectura **subsumida**, redes de **comportamiento**, etc.), todos **comparten** una **limitación** formal: producen un **comportamiento racional** sólo si la decisión correcta puede **obtenerse** a partir de la **percepción actual** del agente.
- Esto es, su **comportamiento** es **correcto** si y sólo si, el **entorno** es **observable**.
- Sus **limitaciones** vienen dadas porque pueden tener **poco alcance**.

Ejemplo aspiradora

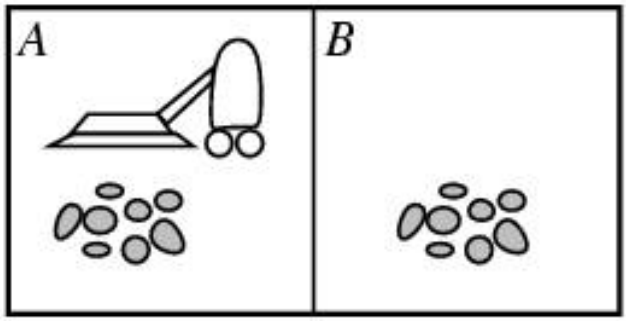


Figura 6 : Problema de la aspiradora

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean]$, $[A, Clean]$	$Right$
$[A, Clean]$, $[A, Dirty]$	$Suck$
⋮	⋮

Figura 7 : Tabla de percepciones/acción de la aspiradora

Programa de agente reactivo para el problema de la aspiradora

Programa

```
1: function AGENTE-REACTIVO-ASPIRADORA(e=[ubicación, estado])
devuelve una acción
2:          if (estado=SUCIO) then return SUCK
3:          else if ubicacion=A then return RIGHT
4:          else if ubicacion=B the return LEFT
5:end function
```

Agentes con estado interno I

- Un **agente** con estado interno **interactúa** con su **entorno**, tal y como se muestra en la siguiente figura:

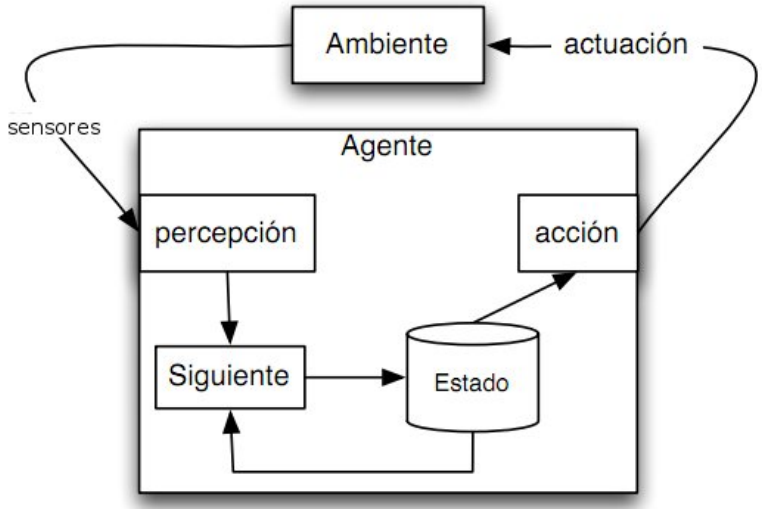


Figura 8 : Esquema funcional de un agente con estado interno (i)

Agentes con estado interno II

- También se conocen como **agentes** reactivos **basados** en **modelos** o Agentes **bien informados** de todo lo que pasa.
- El **agente** reactivo **funciona** sólo si se toma la **decisión adecuada** con base en la **percepción** de un momento dado.
- En **ocasiones** se requiere **mantener** cierto tipo de **estado interno** para estar en **condiciones** de poder **optar** por una **acción**.

Agentes con estado interno III

- Se puede **detectar** analizando una sola **imagen** si un vehículo **frena** si este dispone de **tercera** luz de **freno**.
- Sino, existe una **confusión** entre las luces de **frenado** y las luces de **alumbrado** traseras.
- Se necesita entonces analizar la imagen anterior para detectar **cambios**. La imagen anterior sería el **estado interno**.

Agentes con estado interno IV

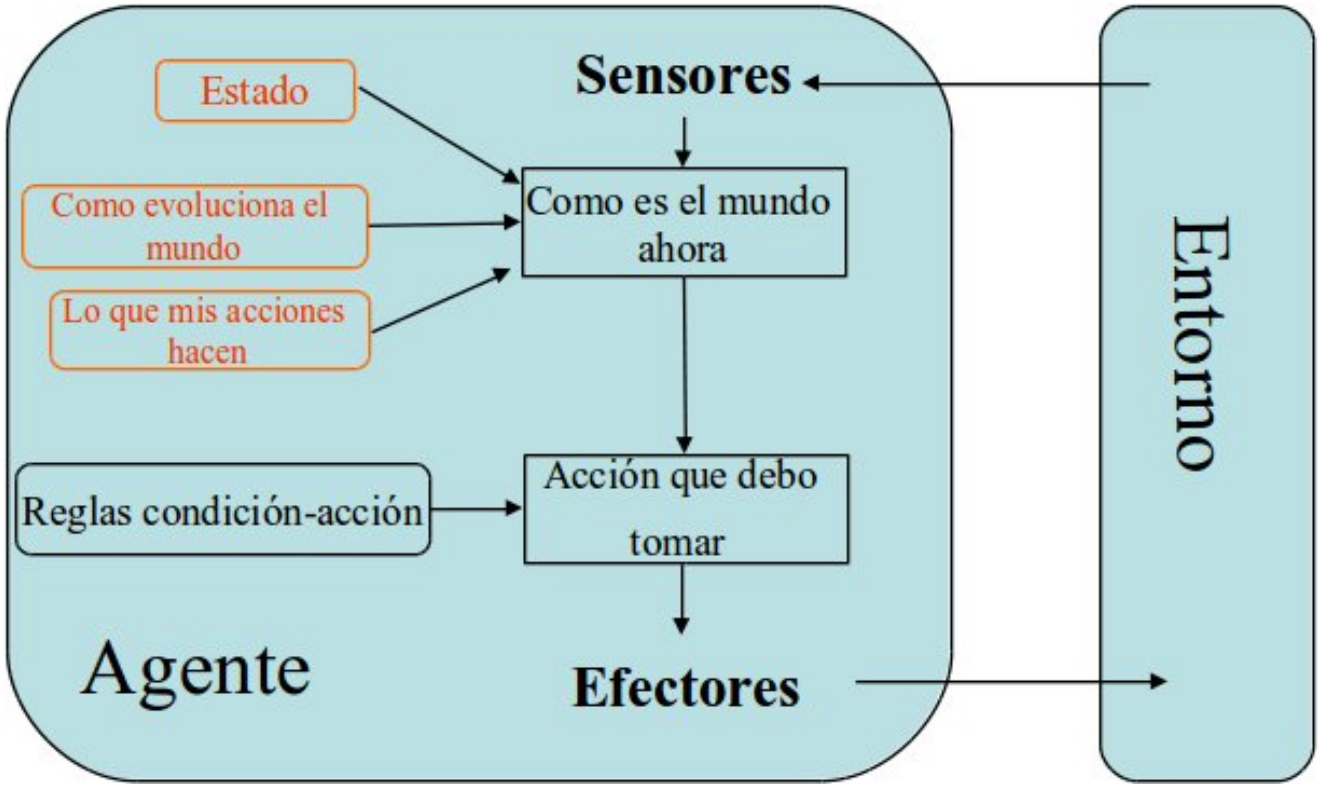


Figura 9 : Esquema funcional de un agente con estado interno (ii)

Agentes con estado interno V

- Aparte del **estado** en el **esquema** funcional aparecen "Cómo **evoluciona** el mundo" y "Lo que mis **acciones** hacen".
- Para un **cambio** de **carril** debo saber la **posición** de otros **vehículos** a partir de la **información** del espejo **retrovisor**.
- "Cómo **evoluciona** el mundo": **cambio** de carril si **detecto** un vehículo que no se **acerca**; No cambio de **carril** si el **vehiculo** se acerca.
- "Lo que mis **acciones** hacen": si cambio de carril queda **temporalmente** un hueco en el **carril** abandonado que puedo **aprovechar** si necesito de nuevo **cambiar**.

Programa de agente con estado

- Este programa es muy **parecido** al de un agente **reactivo**.
- El **estado** es una **descripción** del **mundo** y las **reglas** son del tipo **condición/acción**.

Programa

```
1: function AGENTE-CON-ESTADO(e)
2:          percepcion ← percibir(e)
3:          estado ← siguiente(estado,percepcion)
3:          regla ← seleccionAcción(estado,reglas) (reglas predefinidas)
4:          accion← accionRegla(regla)
5:          return accion
6:end function
```

Programa de agente con estado para el problema de la aspiradora

Programa

```
1: function AGENTE-ESTADO-ASPIRADORA(e=[ubicación, estado])
devuelve una acción
2:static last_A=∞ ,last_B=∞
3:      while(TRUE)
4:          last_A=++; last_B=++;
5:          if (estado=SUCIO) then
6:                  if (ubicacion=A) then last_A=0 else last_B=0;
7:                      return SUCK
8:          else if ubicación=A then
9:                  if (last_B > 3) return RIGHT else NoOP
10:         else if ubicación=B then
11                  if (last_A > 3) return LEFT else NoOP
12:     end while;
13:end function
```

Agentes basados en objetivos I

- También conocido como agente basado en **metas**.
- Para **decidir** qué hacer **no** basta con tener **información** acerca del **estado** que prevalece en el **ambiente**.
- **Además** del **estado** prevaleciente, se **requiere** cierto tipo de **información** sobre su **meta**.
- En el **ejemplo** del **Taxi**, ¿Hacia dónde nos **dirigimos** cuando hay un **cruce**?

Agentes basados en objetivos II

- La **búsqueda** y la **planificación** son **subcampos** de la **IA** que se ocupan de **encontrar** las **secuencias** de **acciones** que permiten alcanzar las **metas** de un agente.
- Este tipo de **agente** es **diferente** a los anteriores, debido a que implica **tomar** en **cuenta** el **futuro**.

Agentes basados en objetivos III

- Puede ser más **flexible** si **cambian** las **condiciones** o cambian las **metas**.
- Un **reactivo** simple: Si **frena** el coche anterior **freno** (**Tabla**).
- **Basado** en Objetivos: Si **frena** el coche anterior **freno** (Objetivo: **No chocar**).
- **¿Intensidad** de Frenado? **¿Cambio** de condiciones **climatológicas**?
- Si **cambio** el **destino** (meta u objetivo), en el **agente** reflejo **tendría** que cambiar toda la tabla de **acciones**.

Agentes basados en objetivos IV

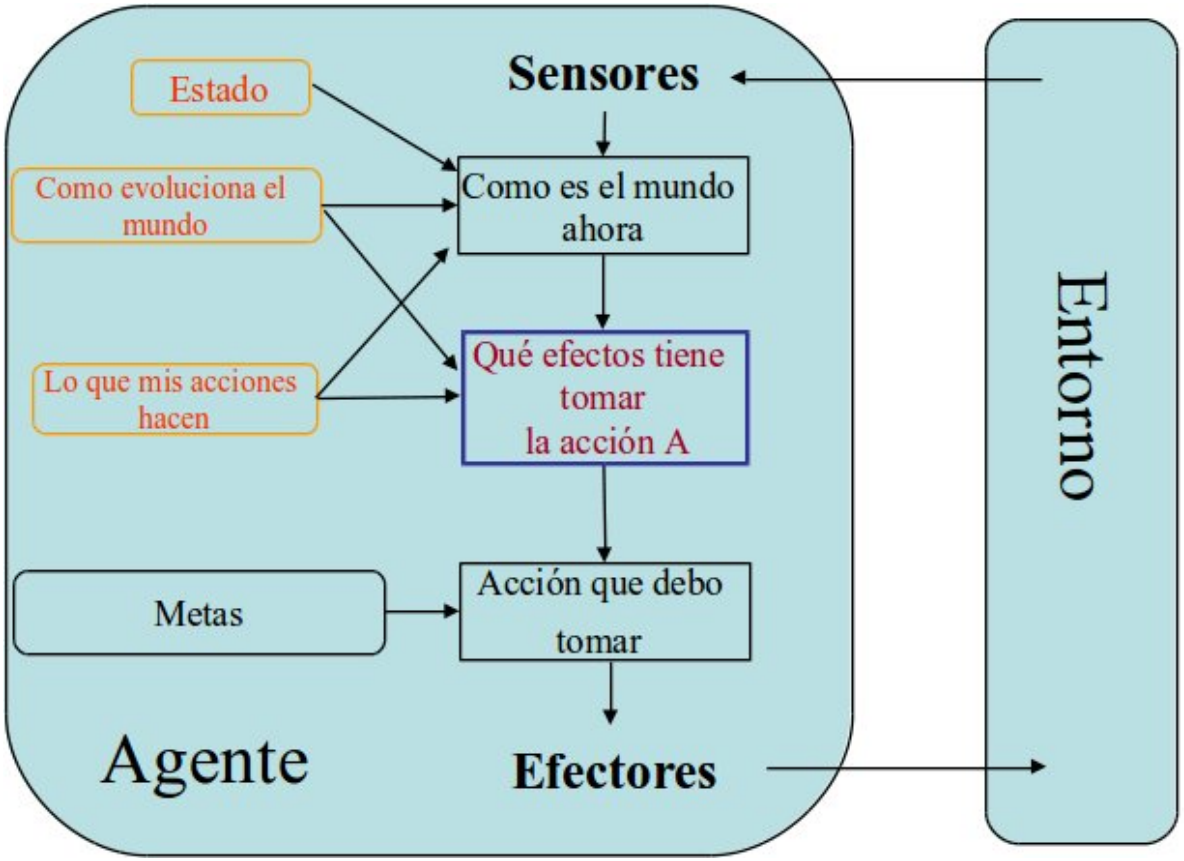


Figura 10 : Esquema funcional de un agente basado en objetivos

Agentes basados en objetivos V

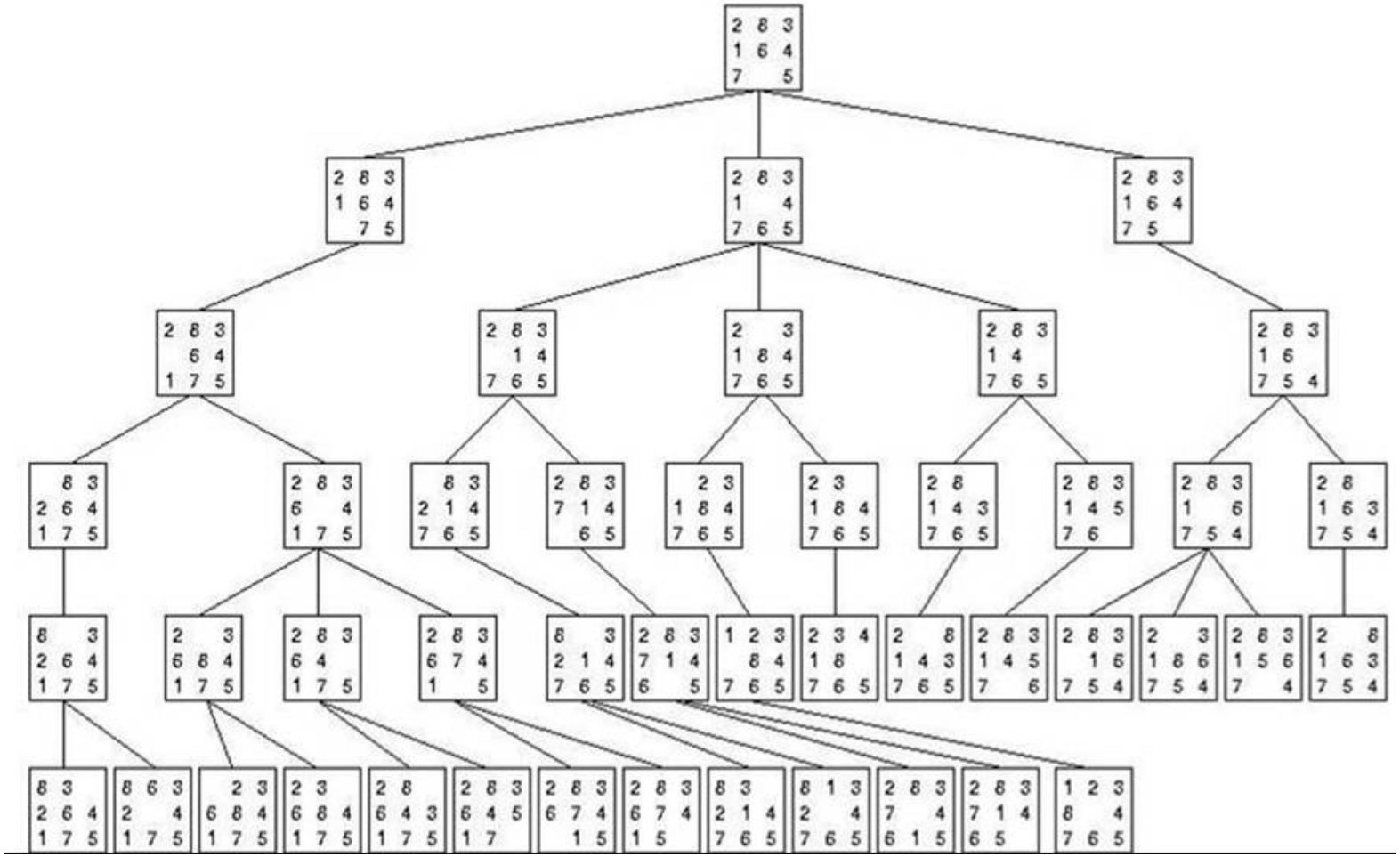


Figura 11 : Árbol de búsqueda del problema

Agentes basados en utilidad I

- Las **metas no** bastan por sí mismas para **generar** una **conducta** de alta **calidad**.
- Puede haber muchas **secuencias** de **acciones** que permitan **alcanzar** la **meta**, pero algunas ofrecen **más utilidad** que otras. Más **rápidas**, confiables, **seguras**, baratas, etc.
- La **utilidad** es una función que **correlaciona** un **estado** y un número real mediante el cual se caracteriza el **correspondiente** grado de **satisfacción (felicidad)**.

Agentes basados en utilidad II

El uso de una **función** de **utilidad** permite la toma de **decisiones** racionales en **dos** tipos de **casos** en los que las **metas** se encuentran con **problemas**:

- **Primero**: cuando el **logro** de algunas **metas** implica un conflicto y sólo **algunas** de ellas se pueden **obtener** (p.e. **velocidad** y **seguridad**). En estos casos la **función** de utilidad **definirá** cuál es el **compromiso** adecuado a optar.
- **Segundo**: cuando son **varias** las metas que el agente podría desear obtener, pero no existe la **certeza** de poder lograr **ninguna** de ellas. Entonces la **utilidad** permite **ponderar la posibilidad** de tener éxito teniendo en cuenta la **importancia** de las diferentes metas.

Agentes basados en utilidad III

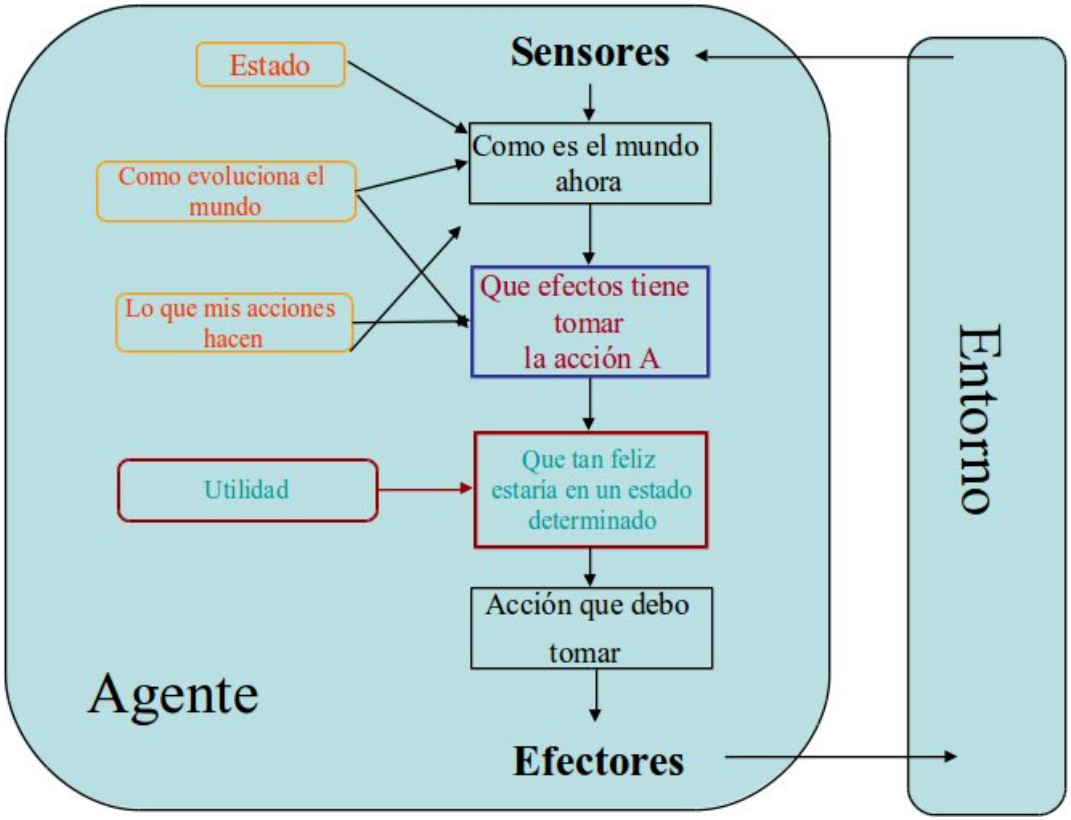


Figura 12 : Esquema funcional de un agente basado en utilidad

Ejercicio

Indique el **tipo** de **agente** para cada caso:

- Agente resuelve **laberintos**.
- Agente que devuelve la **raíz cuadrada** de un **número**.
- Agente que **conduce** un **automóvil**.
- Agente del **mundo** de los **wumpus**
- Agente que resuelve el **problema** de los **bloques**.

Índice

Agentes de interfaz I

- También se denominan **asistentes personales**.
- Se caracterizan principalmente por ser **autónomos** y tener capacidad de **aprendizaje**.
- A partir de la **monitorización** y **observación** de las **acciones** de un usuario, aprenden de sus necesidades y **proponen acciones**.
- **El aprendizaje se realiza**:
  - **Observando** e imitando al **usuario**.
  - Recibiendo **realimentación** positiva o negativa.
  - Por instrucción **directa**.
  - Pidiendo **ayuda** o consejo en otros **agentes**.
- **Objetivos**: Tener **interfaces** que puedan **acomodarse** a los **usuarios**, mejorar su uso, **facilitar** el **aprendizaje** a nuevos usuarios.

Agentes de interfaz II

- **Motivación**: **Tareas** pesadas y **laboriosas** pueden ser **delegadas** a agentes que de manera **proactiva** nos **faciliten** el trabajo.
- **Ventajas**: **Reducen** el **trabajo** al usuario, se pueden ir **adaptando** con el **tiempo** al usuario, además, de que el **conocimiento** adquirido puede ser **compartido** con otros usuarios/agentes.
- Un **ejemplo** puede ser el de **detectar** una **noticia** importante para un usuario y así **comunicárselo**.

Agentes móviles

- El **agente** puede **transitar** entre varias **máquinas** para utilizar **recursos** de los cuales no dispone en su propia máquina o simplemente para **evitar sobrecarga** en las comunicaciones.
- Son agentes capaces de **moverse** para realizar sus **tareas**.
- Aparte de la **movilidad**, deben ser **autónomos** y **cooperativos**.
- **Motivación**: nada obliga a que los **agentes** sean estáticos, de hecho la **movilidad** supone **ventajas**.
- **Ventajas**:
  - **Reducir** el **coste** de la comunicación.
  - Recursos locales **limitados**.
  - Facilitar la **coordinación**.
  - Computación **asíncrona**.
  - **Flexibilidad** en el diseño de sistemas **distribuidos**.

Agentes de información/Internet I

- Su **objetivo** es **recolectar información** a través de la red, **indexarla** y **ofrecérsela** al **usuario** cuando realiza una **consulta**.
- **Surgen** de la **necesidad** de **procesar** de manera automática la **información**.
- Se pretende **facilitar** el **acceso** a la **información** a los **usuarios** aprovechando la capacidad de los agentes.
- Son **agentes** de lo más diverso pues las **capacidades** necesarias entran dentro de las **áreas** de otros tipos de **agentes**.

Agentes de información/Internet II

- El mayor **énfasis** se encuentra en la manera de **representar** la **información** para poder **procesarla** de manera **automática**.
- Como **ejemplos** podemos tener los agentes **recomendadores**, comparadores, **agregadores**, buscadores, filtradores, **indexadores** de información, etc.
- **Ventajas**: **facilitan** al usuario **acceso** a **información** y datos **no procesables** manualmente.

Agentes colaborativos

- Se **caracterizan** principalmente por ser **autónomos** y **cooperativos**.
- Si tienen que **cooperar**, deben de disponer de un **lenguaje** de comunicación **bien definido** y **seguir** unos **protocolos** para la **interacción**, perfectamente establecidos.
- Se **basan** en la **idea** de que la **colaboración** logra que la capacidad del conjunto **supere** la suma de las **capacidades individuales**.
- **Motivación**:
  - **Problemas** muy **grandes** para ser resueltos por un único agente.
  - Posibilidad de **interconectar sistemas** existentes (wrapper).
  - **Solución** a **problemas distribuidos**.
  - Poder **disponer** de fuentes de **información distribuidas**.
  - Mejorar la **modularidad**, velocidad, **flexibilidad** y reusabilidad.

Agentes Heterogéneos/Sistemas Multiagentes

- **Motivación**: existen **sistemas** capaces de dar **servicios específicos** pero que **actúan** de manera **aislada**. Hay una **necesidad** de que todos estos **servicios** puedan **compartirse** y **cooperar** (composición de servicio).

Beneficios:

- Aplicaciones **individuales** pueden beneficiarse de los servicios que pueden obtener de su **interconexión** con otros **sistemas**.
- Aplicaciones **actuales** pueden ser **añadidas** fácilmente a estos **sistemas** (wrappers).
- Permite una **nueva** manera de **diseñar aplicaciones** (Agent based software design).

Otros tipos

- Agentes **consejeros**: este tipo de agente da **consejo** al **usuario** referentes a una **herramienta**, o un sistema de **diagnóstico** o ayuda.
- Agentes de **navegación**: estos **agentes** son utilizados para **navegar** en la **red**, su función **principal** es recordar **sitios** y **direcciones** de interés para el **usuario**.
- Agentes de **control**: Estos agentes proporcionan **información** de manera **eficaz** y **oportuna** para el usuario, en el momento en que ocurre un **evento**.
- Agentes de **recomendación**: Este **agente** posee una **base** de **datos** con **información** acerca de un tema de interés para un grupo, al hacer las **recomendaciones** se basan en **analogías** con otros usuarios de perfil **similar**.

bots I

El **término bot** ha llegado a ser usado para **substituir** al término **agente**, aunque este último término se sigue utilizando en **publicaciones académicas**.

- **Chatterbots**: usados para **chatear** en la Web (agentes **conversacionales**).
- **Annoybots**: para **reventar** chats.
- **Spambots**: usados para **generar spam** tras recopilación de **direcciones**.
- **Mailbots**: usados para eliminar spam o filtrar correos.

bots II

- **Spiderbots**: para recoger e **indexar** las **direcciones** de los sitios de **Internet** para que puedan ser **encontrados**. (Googlebot)
- **Infobots**: especializados en la **recolección** de **información**, Newsbot, Jobbots.
- **Knowbots**: buscan **información** en **bases** de datos **distribuidas** Shopbots, musicbots.

Índice

Consideración de un agente como inteligente

- **Reactividad**. Tendrán la capacidad de **percibir** su **entorno** y **responder** en tiempo acotado a aquellos **cambios** que ocurran para satisfacer los objetivos para los que fue diseñado.
- **Proactividad/Iniciativa**. Serán capaces de exhibir un **comportamiento dirigido** por **objetivos** tomando la iniciativa para alcanzar los **fines** para los que fue **diseñado**.
- **Habilidad Social**. Tendrán **capacidad** de **interaccionar** con otros agentes para **satisfacer** los **objetivos** para los que fue diseñado.

Proactividad y Reactividad I

- Sistemas **dirigidos** por **objetivos** se pueden implementar mediante procedimientos o funciones en **programación funcional**. Dada un situación (**precondición**) se realizan unas **acciones**.
- En el caso anterior, se **asume** que el **entorno no cambia** mientras que el procedimiento se ejecuta. Se puede **alterar** la **precondición** por la que el proceso llegó a ejecutarse y en este caso si ya **no** se quiere **alcanzar** el **objetivo** marcado por la función, ésta debería **dejar** de **ejecutarse**.
- En entornos **dinámicos**, el **agente** debe ser **reactivo**. Debe **responder** a **eventos** en el entorno, cuando éstos **afectan** a los **objetivos** del propio agente o a las **precondiciones** establecidas.

Proactividad y Reactividad II

- Se deben **balancear** los dos **enfoques** y esto puede llegar a ser **complejo** ya que por ejemplo, si tenemos un enfoque **continuamente** reactivo, nunca llegará a trabajar en alcanzar un objetivo el **suficiente tiempo** como para alcanzarlo.
- Es complejo diseñar agentes que puedan alcanzar un **equilibrio** correcto entre un comportamiento **dirigido** por **objetivos** y uno **reactivo**.

Habilidad Social

- **No** se puede entender por **habilidad** social a la **simple** capacidad de **intercambiar** información.
- El ser humano para **alcanzar** algunos **objetivos** debe **negociar** y **cooperar** con sus semejantes.
- Se debe tener la capacidad de **entender** y **razonar** sobre los objetivos de los otros, así como realizar **acciones** que les lleven a **cooperar** con nosotros para alcanzar los **objetivos** propuestos.

Otras propiedades atribuidas a los agentes

- **Continuidad temporal**: se considera un **agente** como un **proceso** que no puede tener fin y que se **ejecuta** de manera **continua**.
- **Adaptabilidad**: el agente al **aprender** puede cambiar su **comportamiento** a partir del **aprendizaje**.
- **Veracidad**: se asume que el **agente** no comunicará información **falsa** de manera **intencionada**.
- **Benevolencia**: el agente estará **dispuesto** a **ayudar** a otros agentes si esto no entra en **conflicto** con sus propios objetivos.

Bibliografía I

- Capítulo 2 del libro: **Introduction to Multiagent Systems**. Michael Woolridge. 2001. Ed. John Wiley & Sons, Inc.
- Capítulo 2 del libro: **Inteligencia Artificial**. Stuart Rusell, Peter Norving. 2004. Pearson Education.
- Tema de **Agentes Inteligentes** de la Asignatura de Sistemas Inteligentes, Samuel Oporto Díaz. Universidad Nacional de Ingeniería, Perú.
- Clasificación de los agentes: **Apuntes de Aplicaciones de la IA**. Javier Béjar. UP Cataluña.

Bibliografía II

- bots: **Artificial Intelligence - Agents and Environments**. William John Teahan. 2010. William John Teahan & Ventus Publishing ApS.
- **Arquitecturas de Agente** (Informe Técnico). Intelligent systems research group. UPM.

Fuentes de Figuras:

- Figura 2. `http://www.tibco.com/blog/2010/05/06/what-goes-on-inside-an-agent-by-james-odell-csc/` (última visita: 3/9/13).

- Figuras 4 y 9. `http://karenalduncin.blogspot.com.es/` (última visita: 5/9/13).

TEMA 2. AGENTES INTELIGENTES

Sistemas Multiagentes. Curso 2015/2016.

Universidad de Castilla-La Mancha.

Septiembre de 2015

TEMA 3. ARQUITECTURAS DE AGENTES

Sistemas Multiagentes. Curso 2015/2016

Escuela Superior de Informática Ciudad Real (UCLM)

Octubre de 2015

Índice

Objetivos

- Estudiar **representaciones formales** de todos los conceptos **estudiados** hasta ahora en la asignatura y **vinculados** con los **agentes** inteligentes.
- Estudiar las diferentes **arquitecturas** existentes que **permiten** definir la **conducta** del agente.

Índice

- En esta sección se pretende **establecer** una **formalización**, en la forma de una **arquitectura abstracta de agentes**, de todos los **conceptos** estudiados hasta ahora en la **asignatura** y vinculados con los agentes **inteligentes**.

- El **entorno** se define como un conjunto finito de estados discretos, definido como:

Entorno

$$E=\{e, e', ...\}$$

- Un **agente** puede ejecutar un conjunto finito de **acciones**:

Acciones

$$Ac=\{\alpha, \alpha', ...\}$$

- Una **ejecución** de un agente, **r**, en un entorno se define como una secuencia finita de estados y acciones intercalados:

Ejecución

$$r=\{e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_{u-1}} e_u\}$$

Ejecuciones

- Sea **R** el conjunto de todas las posibles secuencias finitas sobre **E** y **Ac**.
- Definimos R^{Ac} como el **subconjunto** de las **ejecuciones** que terminan en una **acción**.
- R^{E} como el **subconjunto** de las **ejecuciones** que terminan en un **estado** del entorno.
- Para **modelar** el efecto de una **acción** en el **entorno**, usamos una función de transición:

Función de transición

$$\tau : R^{Ac} \rightarrow \varphi(E)$$

- Si $\tau(r) = \emptyset$ para todo $r \in R^{Ac}$, se dice que el **sistema** ha **terminado** su **ejecución**.

Entorno y Agentes

- Un **entorno** se define como una tripleta:

Entorno

$$Env = \langle E, e_0, \tau \rangle$$

donde **E** es el conjunto de los posibles **estados** del entorno, $e_0 \in E$ es un estado **inicial** y τ es la función de **transición** de estados.

- Los agentes se **modelan** como **funciones** que mapean **ejecuciones** que terminan en un estado del entorno, a **acciones**:

$$Ag : R^E \rightarrow Ac$$

Sistema Agente

- Un **sistema agente** es una tupla conformado por un **agente** y un **entorno**.
- El conjunto de posibles ejecuciones del agente **Ag** en el entorno **Env** se denota como **R(Ag,Env)**.
- Una secuencia de la forma:

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, ...)$$

representa una **ejecución** del agente **Ag** en el entorno $Env = \langle E, e_0, \tau \rangle$ si y sólo si:

1. e_0 es el estado **inicial** de Env.
2. $\alpha_0 = Ag(e_0)$
3. $\forall u > 0$:

$$e_u \in \tau((e_0, \alpha_0, e_1, ..., \alpha_{u-1})) \text{ y } \alpha_u = Ag((e_0, \alpha_0, ..., e_u))$$

Programa de Entorno

- Un **programa básico de entorno** ilustra la relación de este y los agentes situados en él.

Entorno

```
1: procedure Entorno(e, τ, ags, fin) (e: Estado inicial del entorno)
2:         repeat
3:                 for all ag ∈ ags do (ags: Conjunto de agentes)
4:                         p(ag) ← percibir (ag ,e)
5:                 end for
6:                 for all ag ∈ ags do
7:                         accion(ag) ← ag (p(ag))
8:                 end for
9:                 e ← τ(∪_{ag∈ags} accion(ag)) (τ: función de transición)
10:         until fin(e)
11:end procedure
```

Percepción y Acción

- Sea **Per** un conjunto no vacío de **percepciones**. La función **percibir** se define como el mapeo del conjunto de estados del entorno **E** al conjunto de percepciones posibles **Per**:

$$percibir : E \rightarrow Per$$

- La función **acción** se define como el mapeo entre conjuntos de **percepciones** y el conjunto de **acciones** posibles del agente:

$$accion : Per \rightarrow Ac$$

- Un **agente** puede definirse ahora como la **tupla**:

$$Ag = \langle percibir, accion \rangle$$

Propiedades de la percepción

- Sea $e \in E$ y $e' \in E$, tal que $e \neq e'$ y **percibir(e)=percibir(e')**. Desde el punto de vista del agente, **e y e'** son **indistinguibles**.
- Dados dos estados del entorno $e, e' \in E$, **percibir(e)=percibir(e')** será denotado como $e \sim e'$.
- El entorno es **observable** para el agente, si y sólo si $\mid E \mid = \mid \sim \mid$ y entonces se dice que el agente es **omnisciente**. El número de percepciones **distintas** es igual al número de estados del entorno **distintos**.
- Si $\mid \sim \mid = 1$, entonces se dice que el agente **no** tiene **capacidad** de **percepción**, es decir, el entorno es percibido por el agente como si tuviera un estado único.

Estado interno

- La forma más **eficiente** de trabajar en un entorno **parcialmente observable** es llevando un **registro** de lo **percibido**, de forma que el agente tenga acceso mediante este registro, a lo que en cierto momento ya no puede percibir.
- Sea **I** el conjunto de **estados** internos **posibles** de un agente. Redefinimos la función **acción** para **mapear estados** internos a **acciones** posibles:

$$accion : I \rightarrow Ac$$

- La nueva función **siguiente**, mapea estados internos y percepciones a estados internos. Se usa para **actualizar** el estado interno del agente.

$$siguiente : I \times Per \rightarrow I$$

Objetivos

- Las **metas** u objetivos describen **situaciones deseables** para un agente, y se definen como cuerpos de conocimiento.
- Esta **concepción** de los **objetivos** está **relacionada** con el concepto de espacio de estados de un problema compuesto por un **estado inicial** del entorno, $e_0 \in E$; por un **conjunto de operadores** o acciones que el agente puede ejecutar para **cambiar** de **estado**; y un espacio de **estados deseables**.
- **Implícita** en la arquitectura del agente, está su **"intención"** de ejecutar las acciones que el "cree" le garantizan **satisfacer** cualquiera de sus **objetivos**.

Utilidad

- Una **utilidad** es un **valor numérico** que denota la **bondad** de un estado del entorno.
- Implícitamente, un agente tiene la "intención" de alcanzar aquellos estados que **maximizan** su **utilidad** a largo término.
- La **especificación** de una **tarea** en este enfoque corresponde simplemente a una función de utilidad:

$$u : E \rightarrow \Re$$

la cual asocia valores reales a cada estado del entorno.

- Por ejemplo, la **utilidad** para una ejecución **r** de un agente **filtro de spam**, puede definirse como:

$$u(r) : \frac{SpamFiltrado(r)}{SpamRecibido(r)}$$

Agentes óptimos I

- Si la función de utilidad tiene algún **límite superior**, por ej:

$$\exists k, k \in \Re \text{ tal que } \forall r \in R, u(r) \leq k$$

entonces es posible hablar de agentes que **maximizan** la **utilidad** esperada.

- Definamos **P(r|Ag,Env)**, como la probabilidad de que la ejecución **r** ocurra cuando el agente **Ag** esté situado en el entorno, entonces es evidente que:

$$\sum_{r \in R(Ag,Env)} P(r|Ag, Env) = 1$$

Agentes óptimos II

- Entonces el **agente óptimo** Ag_{opt} entre el conjunto de agentes posibles **AG** en el ambiente **Env** está definido como aquel que **maximiza** la utilidad esperada:

$$Ag_{opt} = arg \max_{Ag \in AG} \sum_{r \in R(Ag,Env)} u(r)P(r|Ag, Env)$$

Índice

Definicion

- Las arquitecturas de agentes **describen** los diferentes **módulos** que componen un **agente** y la forma en la que se **interconectan** entre los mismos para que éste **exhiba** una determinada **conducta**.
- Al **contrario** que otras **tecnologías** como pueden ser los **SBCs** (motor de **inferencia**, base de **hechos** y base de **conocimiento**) en los agentes se pueden encontrar una gran **diversidad** de **arquitecturas**.

Estructura Modular I

- **Módulo/s de interacción**: permitir la **comunicación** y **cooperación** entre el **agente** y su **entorno** (agentes, humanos, etc.). Los módulos de interacción proporcionan una **interfaz** de **entrada** y **salida** con el entorno.
- **Base/s de conocimiento**: tiene como objetivo el **mantenimiento** de una representación **interna** del **entorno** (si es necesaria).
- **Módulo/s de integración de información**: están **conectados** a los módulos de **interacción** y tienen como **función** principal la de la **integración** de la información **recibida** desde éstos en la base de **conocimiento** del **agente**.

Estructura Modular II

- **Módulo/s de interpretación de información**: trabajan sobre la base de **conocimiento** interpretando la nueva **información** existente, para el módulo de **planificación** pueda determinar las **acciones** a realizar.
- **Módulo/s de planificación**: deben construir un **plan**, es decir, la **secuencia** de **acciones** para conseguir los **objetivos** buscados por el agente.

Estructura Modular III

- **Módulo/s de acción**: están **conectados** a los módulos de **planificación** y ejecutan y **monitorizan** las acciones contenidas en los **planes**. Si alguna **acción** conlleva una interacción con el entorno, se utilizarán los **servicios** de los módulos de **interacción** correspondientes.
- **Módulo/s de cooperación**: responsable de **comunicarse** y **coordinarse** con otros agentes.

Jerarquías I

- Los **componentes** como ocurre en otras **arquitecturas** se estructuran en **capas** organizadas **jerárquicamente**.
- En cada capa se trata la **información** del entorno a **diferentes** niveles de **abstracción**, por lo que puede que en una misma **capa** no tengan porque estar los **módulos** indicados **anteriormente**.
- Las **arquitecturas** se clasifican, tomando como **criterio** su estructura en capas, en **horizontales** y **verticales**.

Jerarquías II

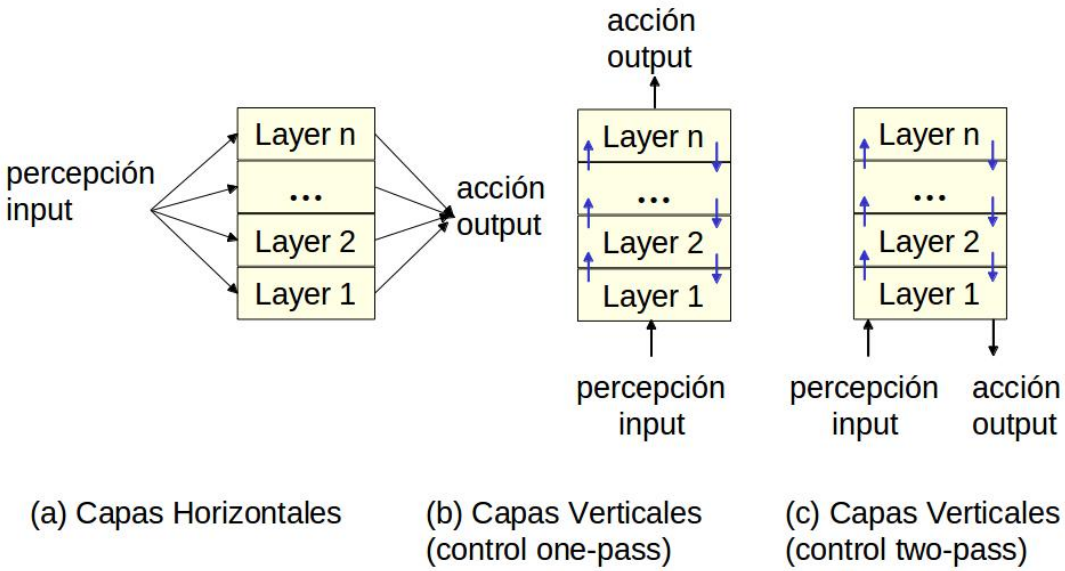


Figura 1 : Arquitecturas en Capas
(http://escritura.proyectolatin.org/inteligencia-artificial/arquitectura-de-agentes/)

Jerarquías III

- Las arquitecturas **horizontales** son **conceptualmente** muy **simples**: si un **agente** debe tener *n* **comportamientos** diferentes habría que **implementar** *n* capas.
- Pero como las capas **compiten** para recomendar **acciones** puede que el comportamiento global no sea **coherente** con lo **esperado**.
- Para **evitar** esto, se añade un **mediador** para la toma de **decisiones** sobre la capa que tiene el **control** del agente en un **instante** concreto.

Jerarquías IV

- La **introducción** de un mecanismo **mediador** provoca cuellos de **botella** y además a nivel de **diseño** se tendrían que tener en cuenta todas las **interacciones** posibles entre las diferentes **capas**.
- Las arquitecturas **verticales** pueden solucionar el **problema** anterior y se pueden **dividir** en dos tipos: arquitecturas verticales de **una pasada** y arquitecturas verticales de **dos pasadas**.
- En las de una pasada el **flujo** de **control** pasa de manera **secuencial** por cada **capa** y es la capa más **alta** la que genera la **acción** a realizar.

Jerarquías V

- En las de dos **pasadas**, el flujo de **control** atraviesa las capas hacia **arriba** y posteriormente hacia **abajo**. Cuando llega de nuevo a la capa más **baja** es cuando se genera la acción de **salida**.
- En las arquitecturas **verticales** se **reduce** la **complejidad** de las **interacciones** entre capas.
- Son menos **flexibles** que las **horizontales** ya que para que un **agente** tome una **decisión** es necesario que el **flujo** de control pase por **todas** las **capas**.

Jerarquías VI

- Son no **tolerantes** a fallos y si una capa **falla** lo harán también las **demás**.
- Las arquitecturas **horizontales** tienen la ventaja de que permiten un gran **paralelismo** entre capas, pero es necesaria una gran cantidad de **conocimiento** de **control** para **coordinar** las mismas.
- Las arquitecturas **verticales** es menor el **control** para coordinar las capas pero la capa que **interactúa** con los sensores debe ser más **compleja** y no son tolerantes a **fallos**.

Clasificación

Según el **comportamiento** que exhiben los agentes, tipo de **procesamiento** que emplean, se **clasifican** en:

- **Deliberativas** (también llamadas BDI).
- **Reactivas**.
- **Híbridas**.

Introducción I

- Las arquitecturas **deliberativas** son aquellas que utilizan modelos de **representación simbólica** del conocimiento.
- Los **agentes** que siguen esta arquitectura **parten** de un **estado inicial** y son capaces de **deliberar**, es decir, generar **planes** para alcanzar sus propios **objetivos**.
- Un agente deliberativo debe **disponer** de un **modelo simbólico** del mundo, representado **explícitamente**, para realizar un **razonamiento lógico** a partir de él y tomar las **decisiones** oportunas.

Introducción II

- La principal **ventaja** de esta **arquitectura** es que el **conocimiento** es más fácil de entender y codificar por el ser **humano**.
- Las **desventajas** que presenta son la **dificultad** para **traducir** el mundo **real** a un modelo **simbólico** adecuado y preciso, y el elevado **coste** temporal, necesario a veces para trabajar con este modelo.

Introducción III

- Sin duda, la arquitectura deliberativa **BDI (Belief, Desire, Intention)** es la más **estudiada** y posiblemente la arquitectura deliberativa más **extendida**.
- La arquitectura **BDI**, cuyos acrónimos en español significan **Creencia, Deseo e Intención**, **combina** un sólido modelo **filosófico** del razonamiento humano y una **semántica** abstracta, lógica y elegante.
- Desde su establecimiento en la década de los 80, prácticamente ha permanecido **inalterable**. Los tres **componentes básicos** de esta arquitectura son las creencias, los deseos u **objetivos**, y las intenciones o **planes**.

Introducción IV

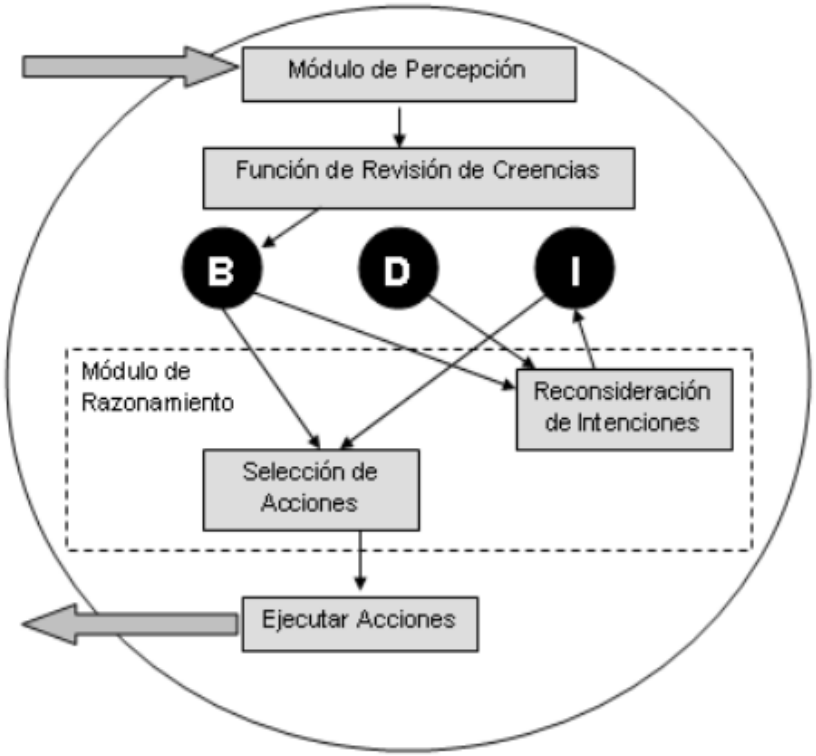


Figura 2 : Esquema básico de la arquitectura BDI

Creencias

- Las **creencias** representan el **conocimiento** de los **agentes** sobre el **mundo**.
- En términos computacionales, son una forma de **representar** el **estado** del mundo, ya sea mediante el **valor** de una **variable**, una **BD** relacional o expresiones **simbólicas**.
- Las creencias son **esenciales** debido al **dinamismo** y la **visión local** del mundo.
- Dado que las creencias pueden representar **información imperfecta** del mundo, su **semántica** subyacente debe obedecer a una **lógica** de **creencias**, aunque la representación **computacional** no necesite ser puramente **lógica** o simbólica.

Deseos y planes

- Un **deseo** (objetivo) representa un **estado final** deseado.
- En términos informáticos, un **objetivo** para un agente será **alcanzar** cierto **valor** en una **variable**, un registro, o **cumplir** una **expresión** simbólica representada en alguna formalización lógica.
- Con las **creencias** y los **objetivos no** es **suficiente** para poder **modelar** un **sistema** capaz de operar en **entornos** dinámicos e **inciertos**: si hemos decidido sobre un curso de acción o (plan), y el mundo cambia ligeramente, deberíamos **cuestionarnos** si seguir con el plan o replantearlo. La respuesta es que el sistema necesita acometer los planes y sub-objetivos que **planifica**, pero además debe ser capaz de **reconsiderarlos** en momentos cruciales.

Intenciones

- Estos **planes** acometidos o formas de **proceder** se llaman **intenciones**, y representan el **tercer** componente de la arquitectura **BDI**.
- Para **determinar** el grado de **persistencia** de una **intención** (su **resistencia** a ser **cambiada** ante **percepciones** de cambio en el mundo), se han definido **tres estrategias** de dedicación:
  1. Dedicación **total** (a ciegas): el agente **seguirá** con su **intención actual** hasta que crea haberla alcanzado.
  2. Dedicación **firme**: El agente seguirá con su **intención** actual hasta que crea haberla **alcanzado**, o bien crea que **no puede** ser **lograda**.
  3. Dedicación **abierta**: Se mantendrá la **intención** solamente mientras el agente la **considere** viable.

Implementación del razonamiento práctico I

- Algoritmo **global** de un agente con **razonamiento** práctico

Agent Control Loop Version 1

1. while true
2. **observe** the world;
3. update internal world **model**;
4. **deliberate** about what **intention** to achieve next;
5. use means-ends reasoning to get a **plan** for the **intention**;
6. **execute** the **plan**
7. end while

Implementación del razonamiento práctico II

- El agente primero intenta **comprender** cuáles son las opciones **disponibles**.
- Escoge entre estas **opciones**. Estas serán las **intenciones**.
- El algoritmo está **redactado** a un nivel muy **alto**.
- De manera **progresiva** cada opción se va **descomponiendo**: por ejemplo, la deliberación se divide en un **Generador de Opciones** (Deseos) y un **Filtrado de Opciones** (Intenciones)

Implementación del razonamiento práctico III

Agent Control Loop Version 2

1. initialize-state();
2. while true do
3. options := option-generator(event-queue);
4. selected-options := deliberate(options);
5. update-intentions(selected-options);
6. execute();
7. get-new-external-events();
8. drop-successful-attitudes();
9. drop-impossible-attitudes();
10. end-while

Plataformas para programación de agentes BDI I

- **JASON**. `http://jason.sourceforge.net/`. Plataforma y lenguaje para el desarrollo de agentes BDI basado en AgentSpeak(L).
- **JADEX**. `https://vsis-www.informatik.uni-hamburg.de/vsis/research/lookproject/27`. Plataforma para el desarrollo de agentes BDI y agentes dirigidos por objetivos.
- **2APL**. `http://www.cs.uu.nl/2apl/`. Plataforma que proporciona mecanismo para la implementación de agentes cognitivos basados en la arquitectura BDI.

Plataformas para programación de agentes BDI II

- **PRACTIONIST**. `http://practionist.eng.it/`. Marco de desarrollo construido sobre la teoría de razonamiento práctico con soporte para el desarrollo de agentes BDI.
- **GOAL**. `http://ii.tudelft.nl/trac/goal`. GOAL es un lenguaje de programación de agentes para la programación de agentes "racionales".

Plataformas para programación de agentes BDI III

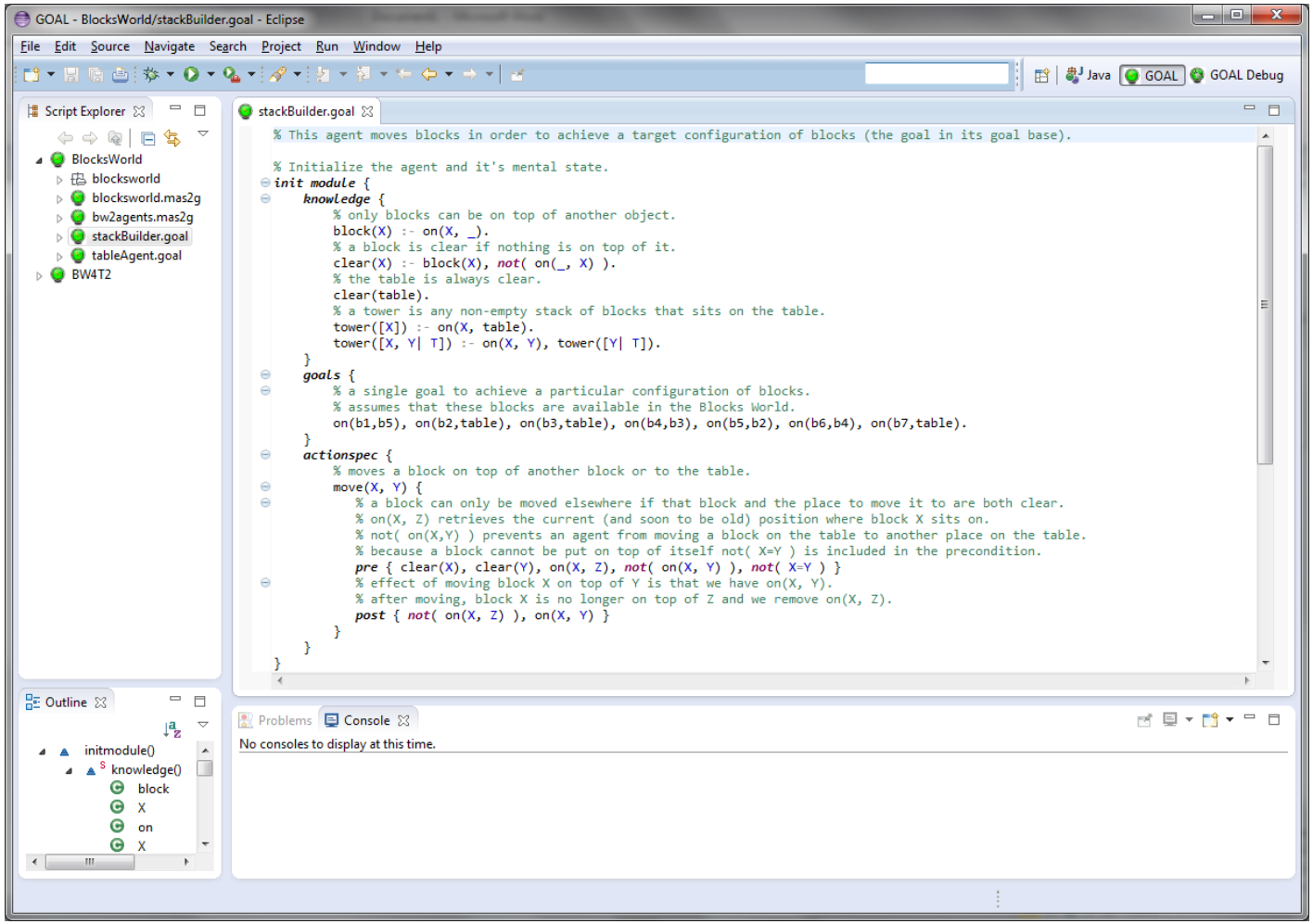


Figura 3 : IDE GOAL para ECLIPSE

Reactivas I

- Las arquitecturas **reactivas** implementan la forma de **actuar** de los agentes como un mecanismo de **correspondencia** directa **percepción** y **acción** (reglas situación $\rightarrow$ acción).
- Se basan en **mecanismos** de respuesta ante **estímulos**.
- A **diferencia** de las arquitecturas **deliberativas**, carecen de modelo **simbólico** y por tanto no emplean **ningún** mecanismo de **razonamiento**.
- Su principal **ventaja** es la mayor **efectividad**, al prescindir de un mecanismo de razonamiento **complejo**.

Reactivas II

- Por contra, presentan el **inconveniente** de que el **comportamiento** del agente es **fijo**, y este **no** puede **aprender** y razonar a partir de la información que recibe.
- Están **compuestas** por unos módulos de **interacción** con el entorno y los **módulos** de **competencia**.
- Un módulo de **competencia** debe ser ser **responsable** de una tarea claramente **definida** y deben tener las **capacidades** necesarias para **procesar** sus **tareas**. Puede que **implementen** una tarea **no completa**.
- La información desde los módulos de **interacción** a los de **competencia** no reciben ningún tipo de **procesamiento** (no traducción **simbólica**).

Arquitectura de subsunción I

- Esta arquitectura **define** una serie de **capas** conectadas a **sensores** que transmiten información en **tiempo real**, de forma que las **capas** componen una **jerarquía** de **tareas** en la que los niveles inferiores tienen **más control** sobre los niveles superiores.
- Aunque los **módulos** de competencia podrían **operar** de forma **autónoma**, existen **mecanismos** que permiten a los de más bajo **nivel** inhibir o **modificar** entradas y salidas de los módulos **superiores**.

Arquitectura de subsunción II

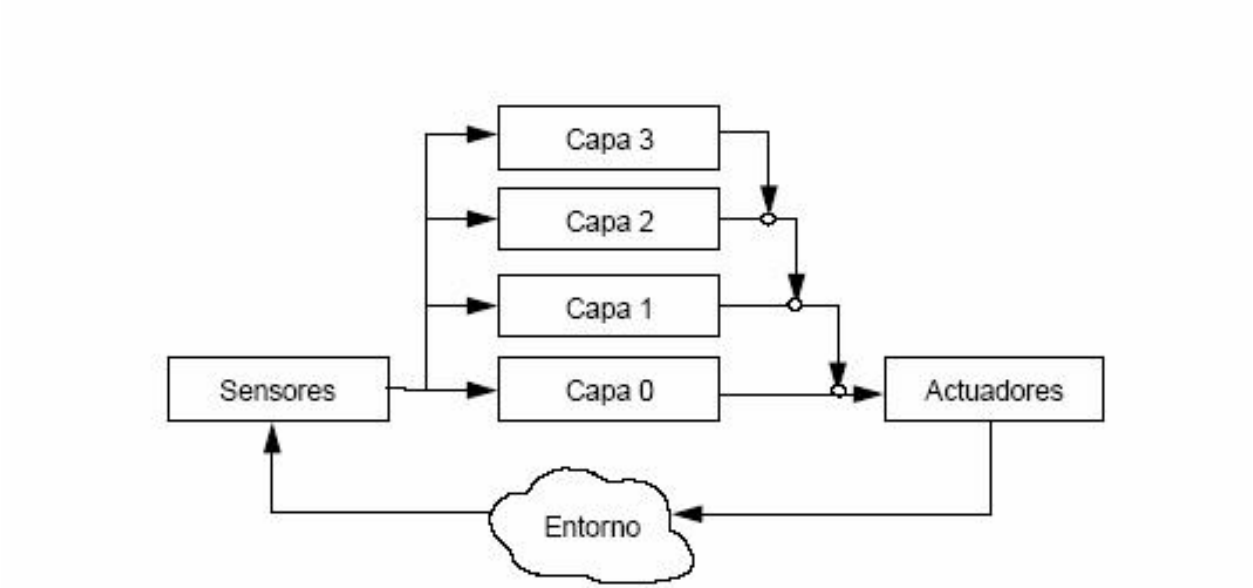


Figura 4 : Esquema básico de un agente en arquitectura de subsunción

Ejemplo: explorador marciano de Steels I

- Un conjunto de **robots** tienen que recoger **minerales** cuya **localización** no se conoce de antemano y llevarlas a una nave **nodriza**. Los **minerales** se encuentran en **cúmulos**.
- Solución mediante uso de una **arquitectura** de subsunción **cooperativa**.
- La **cooperación** no usa **comunicación** directa.

Ejemplo: explorador marciano de Steels II

- La comunicación se realiza a través del **entorno**: campo gradiente de la **señal** generada por la nave nodriza, partículas **radioactivas** que pueden recoger o tirar y **detectar** los robots al pasar.
- Las **reglas** se asocian a dos tipos de **comportamientos** que se pueden ejecutar en paralelo: comportamientos de manejo de **objetos** y comportamientos de movimiento **organizados** de acuerdo a una jerarquía de subsunción.

Ejemplo: explorador marciano de Steels III

1. IF (detecta un obstáculo) THEN (cambiar de dirección)
2. IF (traslada muestras) AND (está en la base) THEN (soltar muestras)
3. IF (traslada muestras) AND (no está en la base) THEN (desplazarse hacia la base por el gradiente)
4. IF (detecta una muestra) THEN (recoger la muestra)
5. IF (verdadero) THEN (desplazarse aleatoriamente por el gradiente lejos de la base)

Ejemplo: explorador marciano de Steels IV

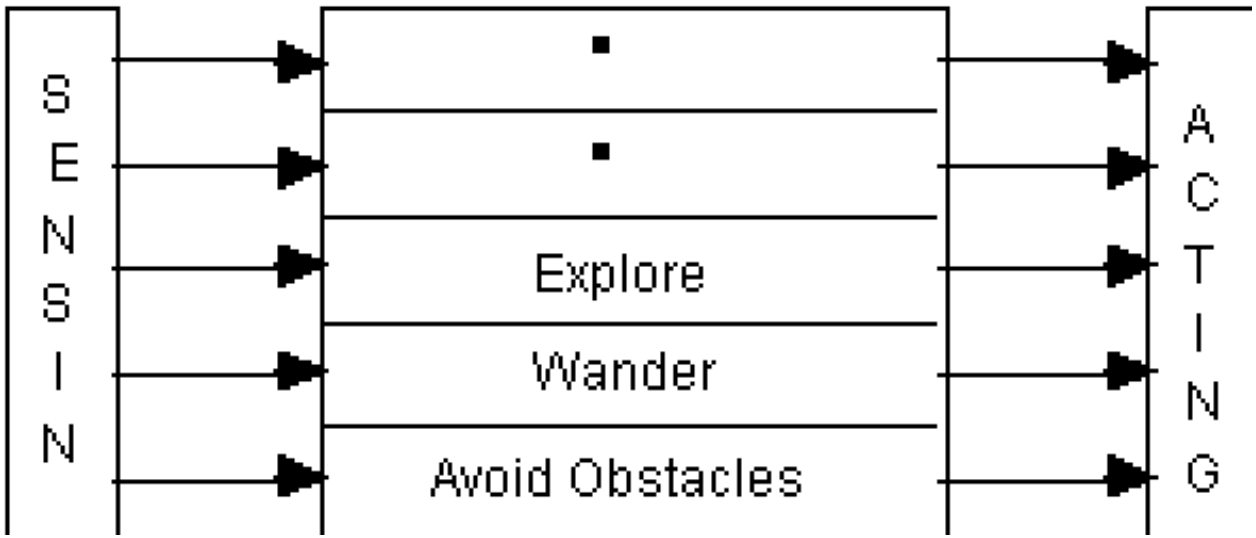


Figura 5 : Esquema básico explorador marciano

Híbridas I

- Para intentar **solventar** las **limitaciones** de las arquitecturas **deliberativas** y **reactivas**, se han propuesto arquitecturas híbridas, que **combinan** aspectos de **ambos** modelos.
- Los **agentes** en arquitecturas **híbridas** presentan dos **subsistemas**: uno **deliberativo**, que utiliza un modelo simbólico para generar planes, y otro **reactivo**, centrado en reaccionar ante eventos en el entorno.

Híbridas II

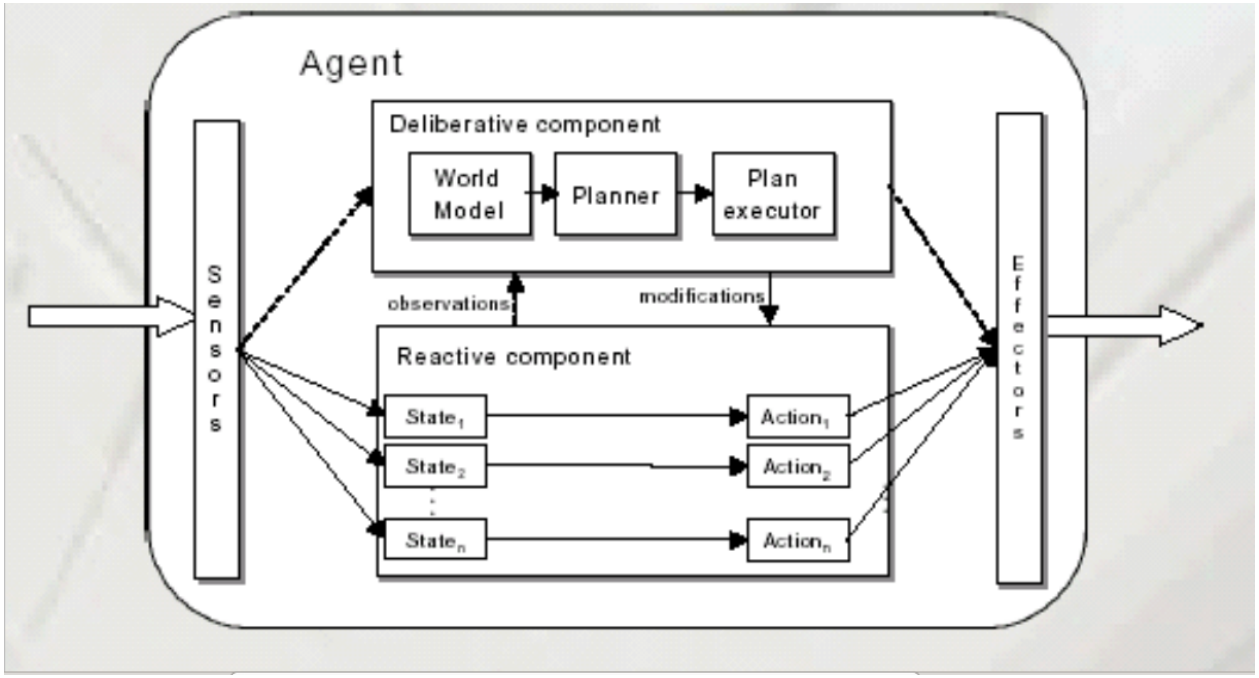


Figura 6 : Esquema básico de una arquitectura híbrida

División por aplicaciones

- **Deliberativos**: agentes software, control de procesos dinámicos complejos, recopilación de información y simulación.
- **Reactivos**: agentes hardware y autónomos (robots), juegos.
- **Híbridos**: robots, sistemas de control, simulación, cooperación entre compañías de transporte, planificación.

Idoneidad

- Las arquitecturas **deliberativas** son más **adecuadas** para agentes que necesitan de **planificación** y mecanismos de **razonamiento** a largo plazo como algo **básico**.
- Las **reactivas** son más adecuadas para agentes que se **encuentran** en entornos con cambios **constantes** y en los que ante estos **cambios** hay que responder de manera **rápida**.
- Cuando el **entorno** es **desconocido** y cambiante encontramos las **limitaciones** de las **deliberativas**, sin capacidad de una respuesta **rápida**, y de las **reactivas**, en las que no se puede **implementar** aprendizaje. Una arquitectura **híbrida** es la que mejor se adapta a este caso.

Bibliografía I

- Arquitectura Abstracta en el Capítulo 2 del libro: **Introduction to Multiagent Systems**. Michael Woolridge. 2001. Ed. John Wiley & Sons, Inc.
- Arquitecturas concretas y multiagentes (FIPA): **Desarrollo de un Sistema Multi-Agente para automatizar Procesos de Consenso en Problemas de Toma de Decisión en Grupo**. Iván Palomares Carrascosa. Director: Luis Martínez López. 2010. Universidad de Jaén.
- **Agentes Inteligentes y Sistemas Multiagentes.** Tutorial MICAI 2006. Dr. Leonardo Garrido. Centro de Sistemas Inteligentes. Universidad de Monterrey (México).

TEMA 3. ARQUITECTURAS DE AGENTES

Sistemas Multiagentes. Curso 2015/2016

Escuela Superior de Informática Ciudad Real (UCLM)

Octubre de 2015

TEMA 4. COMUNICACIÓN.

Sistemas Multiagentes

Escuela Superior de Informática - Universidad de Castilla-La Mancha

Noviembre de 2015

Índice

Índice

Importancia de la comunicación

- La **comunicación** entre agentes es la **llave** para obtener todo el **potencial** del **paradigma** de agentes, al igual que el **desarrollo** del lenguaje **humano** fue la **llave** para el desarrollo de la **inteligencia** y de la **sociedad**.
- Los **agentes** emplean un **lenguaje** de comunicación **(ACL - Agent Communication Language)** para **comunicar** información y **conocimiento**.
- Las **sociedades** de agentes pueden **realizar** tareas que los agentes **individuales** no pueden.
- Los sistemas multiagente (**MAS**) están **orientados** a la resolución **distribuida** de problemas. Sólo es **posible** si los agentes tienen la **capacidad** de comunicación sobre la que **establecer** estrategias de **cooperación**.

Índice

Arquitecturas de pizarra

- **Zona** de trabajo **común** en la que se encuentra la **información** a **compartir**.
- No hay **comunicación** directa entre los **agentes**.
- Pueden **existir** agentes con tareas de control **específicas**.
- Pueden existir **varias** pizarras.
- **Inconveniente**: comunicación **centralizada**.

Paso de mensajes

- La **comunicación** se establece **directamente** entre dos **agentes** (emisor y receptor).
- Pueden existir agentes **facilitadores**.
- **Ventaja**: flexibilidad.

Modelos

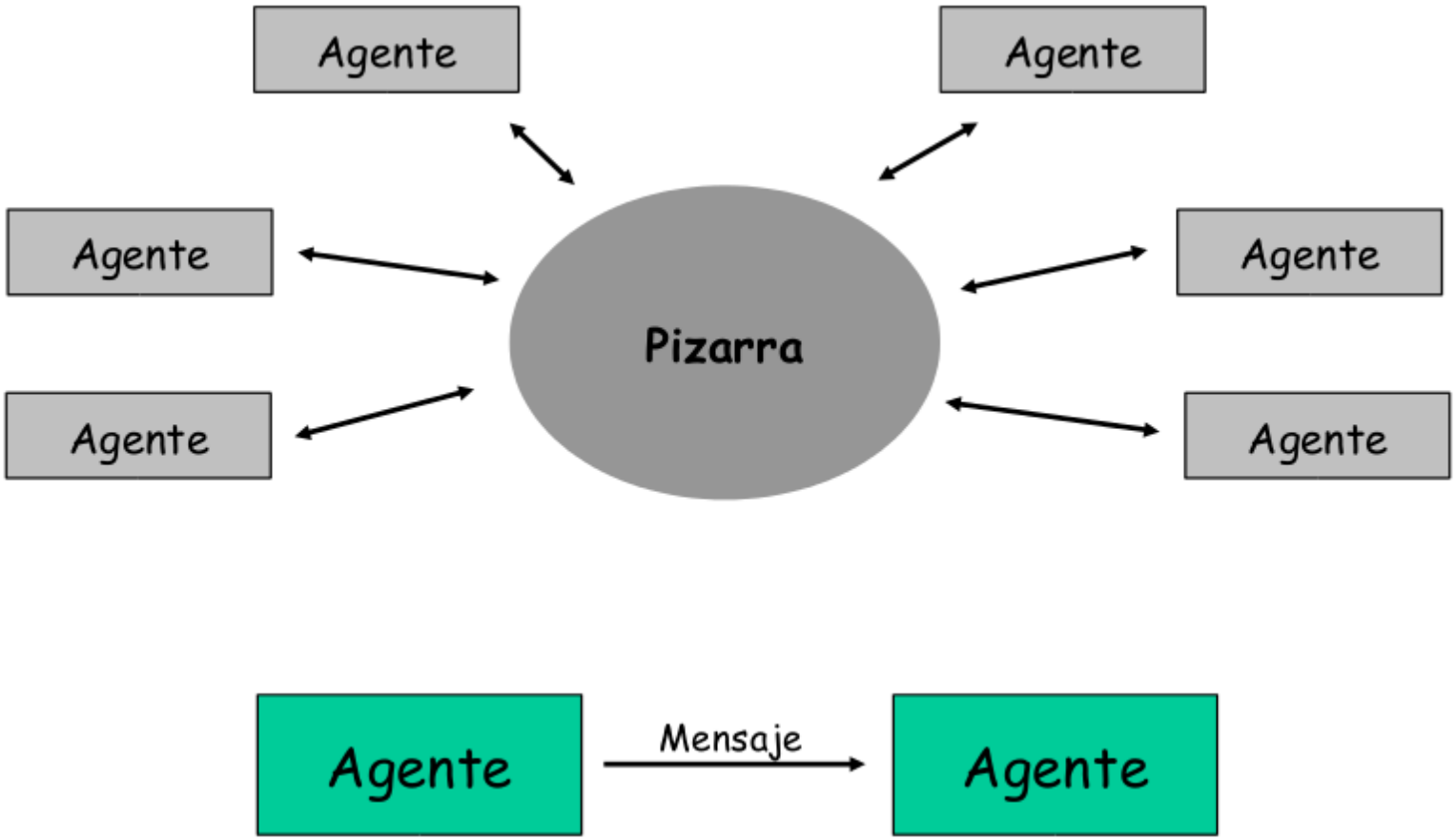


Figura 1 : Modelos de Comunicación

Índice

- Está **desarrollada** por **lingüistas** para ayudar en la **comprensión** del **lenguaje** humano.
- Estudia el **lenguaje** como **acción**.
- Los **hablantes** no sólo **expresan** sentencias **ciertas** o **falsas**.
- Los **hablantes** realizan **actos** del habla: **informan**, preguntan, **sugieren**, prometen...
- Cada **expresión** conlleva un **acto** del habla
- **Identificar** el **acto** del habla es **imprescindible** para una correcta **comunicación**.

- Es la **base** teórica en la **comunicación** entre **agentes**.
- En un **acto** del **habla** se encuentran tres **acciones**:
  1. **Locución**. Contexto del **hecho** físico del acto del habla.
  2. **Ilocución**. **Intención** con la que se realiza.
  3. **Perlocución**. **Acto** que ocurre como **resultado**.
- **Ejemplo**: "Cierra la ventana".
  1. **Locución**: identificación del **emisor**, del **receptor**, de la **ventana**...
  2. **Ilocución**: el emisor quiere que el **receptor** cierre la ventana.
  3. **Perlocución**: el **receptor** cierra (o no) la **ventana**.

La hipótesis F(P)

Todo **acto** del habla consiste en una **fuerza** F aplicada a una **proposición** P.

- **Ilocución** $\rightarrow$ **Sentencia**
- Información $\rightarrow$ La ventana está abierta.
- Pregunta $\rightarrow$ ¿Está abierta la ventana?
- Orden $\rightarrow$ Abre la ventana.
- Petición $\rightarrow$ Podrías abrir la ventana.
- Promesa $\rightarrow$ Te prometo que abriré la ventana.
- Oferta $\rightarrow$ Abriré la ventana si tu quieres.

Índice

Organización de estandarización

- **OMG (Object Management Group)**:
  http://www.omg.org
- **KSE (Knowledge Sharing Effort)**:
  http://www.cs.umbc.edu/kse/
- **FIPA (Foundation for Intelligent and Physical Agents)**:
  http://www.fipa.org

- **Basado** en la Teoría de los **Actos** del **Habla**.
- Un **mensaje** en FIPA ACL representa la **intención** de realizar alguna **acción** (acto comunicativo).
- Un mensaje en FIPA ACL tiene una **sintaxis** similar a un mensaje en **KQML**.
- El **primer** elemento de la lista es el **identificador** del acto **comunicativo** (obligatorio).
- El **resto** de la lista son pares **parámetro-valor** sin un orden predefinido (opcionales):

(inform) :**sender** gestor-bolsa :**receiver** agente1 :**in-reply-to** accion-telefonica :**content** (Precio Telefonica 20) :**language** prolog :**ontology** IBEX-35

Parámetros

Atributo	Significado
:content	Contenido del mensaje
:sender	Identidad del Emisor del mensaje
:receiver	Identidad del Receptor del mensaje (un agente o una lista de agentes)
:language	El nombre del lenguaje de representación empleado en el atributo :content
:ontology	El nombre de la ontología utilizada en el atributo :content
:reply-with	Etiqueta para la respuesta (si es que el emisor la espera)
:in-reply-to	La etiqueta esperada en la respuesta
:protocol	Identificador del protocolo de interacción que se está utilizando
:conversation-id	Identificador de una secuencia de actos comunicativos que forman parte de una misma conversación
:reply-to	Agente al que han de ser enviadas las respuestas (si no es el emisor)
:reply-by	Indicación del tiempo en el que se quiere que se responda al mensaje

Actos Comunicativos

Nombre	Paso de Información	Solicitud de Información	Negociación	Ejecución de Acciones	Manejo de Errores
accept-proposal					
agree					
cancel					
cfp					
confirm					
disconfirm					
failure					
inform					
inform-if					
inform-ref					
not-understood					
propose					
query-if					
query-ref					
refuse					
reject-proposal					
request					
request-when					
request-whenever					
subscribe					

Envío y recepción de mensajes

- El paso de mensajes es **asíncrono**.
- Hay una **cola** asociada a cada **agente** y por tanto **compartida** por todos los **comportamientos**.
- Se puede **leer** el primer **mensaje** de la **cola** o recorrerla hasta encontrar el **primer** mensaje que cumpla algún **criterio**.
- Se pueden establecer **mecanismos** de **sincronización** mediante el envío y recepción de mensajes (**bloqueo** del **receptor**).
- Los mensajes son **objetos** de clase jade.lang.acl.**ACLMessage**

Métodos Principales I

- **setPerformative(int)**: el int representa el tipo de acción **performativa**.
- **getPerformative()**: devuelve el **int** representando la performativa del **mensaje**.
- **createReply()**.
- **addReceiver(AID)**.
- **getAllReceiver()**: retorna el **iterador** sobre los **receptores**.

Métodos Principales II

- **setContent(String)**.
- **getContent()**.
- **block()**: **bloquea** un comportamiento del agente hasta que llegue un mensaje (finaliza ejecución **comportamiento**).
- **blockingReceive()**: bloquea **instantáneamente todos** los **comportamientos** del agente hasta recibir un **mensaje**. **Retorna** el mensaje recibido.

Selección de mensajes entrantes I

- Se utilizará la clase **MessageTemplate** que permite definir **filtros** sobre los **atributos** del mensaje.
- Es el **parámetro** utilizado en los métodos **receive** y **blockingReceive**.
- Los **métodos** de esta clase **devuelven** un objeto de tipo **MessageTemplate**.
- **MatchPerformative**(int), **MatchSender**(AID), **MatchConversationID**(String).

Selección de mensajes entrantes II

- **MatchOntology**(String), **MatchProtocol**(String), **MatchLanguage**(String), **MatchContent**(String), **MatchReplyWith**(String).
- **and**(Template1, Template2), **or** (Template1, Template2), **not**(Template).
- **match**(ACLMessage): devuelve TRUE si el **parámetro** cumple con el filtro **definido** en el MessageTemplate.

Páginas amarillas (DF Agent) I

- Mediante su utilización los agentes pueden **publicar** los **servicios** que **proporcionan**. Así otros **agentes** podrán acceder a los mismos.
- Los agentes **interactúan** con el **DF** intercambiando mensajes **ACL**.
- Jade **facilita** esta tarea mediante los **métodos** implementados en la clase **DFService**.
- El agente debe **proporcionar** al DF una **descripción**, incluyendo su AID, los **protocolos**, lenguajes y **ontologías** que el resto de **agentes** necesitan conocer para **interactuar** con él; y la lista de **servicios** publicados.

Páginas amarillas (DF Agent) II

- Para cada **servicio** se proporciona una **descripción**, incluyendo: tipo de servicio, **nombre**, protocolos, lenguajes y **ontologías**; y una serie de **propiedades** específicas del **servicio**.
- Antes de **finalizar** su ejecución el **agente** debe eliminar del **DF** sus servicios.
- Para realizar las **acciones** anteriores Jade **proporciona** los siguientes métodos:
  - static DFAgentDescription **register**: **registra** los servicios de un **agente** en el DF.
  - static void **deregister**: **elimina** del registro del DF los **servicios** del agente.

Páginas amarillas (DF Agent) III

- Los **servicios** se definen con los siguientes métodos de la clase **ServiceDescription**:
  - void **setName**: modifica el nombre del servicio.
  - void **setOwnership**: modifica el propietario del servicio.
  - void **setType**: modifica el tipo de servicio.
  - void **addLanguages**: añade lenguaje del servicio.
  - void **addOntologies**: añade ontología del servicio.
  - void **addProtocols**: añade protocolo del servicio.
  - void **addProperties**: añade propiedades del servicio.

Páginas amarillas (DF Agent) IV

- La **descripción** del agente se modifica con los siguientes métodos de la clase **DFAgentDescription**:
  - void **setName**: modifica el AID de la descripción del agente.
  - void **addServices**: añade el servicio pasado por parámetro a la descripción del agente.
  - void **removeServices**: elimina el servicio pasado por parámetro a la descripción del agente.
  - void **addLanguages**: añade lenguajes que el agente entiende.
  - void **addOntologies**: añade ontologías que el agente entiende.
  - void **addProtocols**: añade protocolos que el agente entiende.
- Un agente que busca **servicios** debe proporcionar una plantilla de descripción de la clase **DFAgentDescription**.

Páginas amarillas (DF Agent) V

- El resultado de la **búsqueda** es la lista de todas las **descripciones** que encajan con la plantilla **proporcionada** (los campos especificados en la plantilla están presentes en la descripción con los mismos valores, los campos no **cubiertos** no se comprueban).
- Para **realizar** las acciones **anteriores** jade proporciona diversos **métodos** de búsqueda que se encuentran en la clase **DFService**. En el ejemplo que vamos a estudiar se utiliza el método **search**(Agente, Descripción).

Páginas blancas (AMS Agent) I

- **Garantiza** que cada agente en la **plataforma** tenga un **único** nombre.
- Se encarga de **proporcionar** los servicios de páginas **blancas** y ciclo de vida, y de mantener el directorio de los **identificadores** de agentes (AID: Agent Identifier) y su **estado**.
- Cada **agente** debe registrarse con el **AMS** para obtener un **AID** válido, esta operación en JADE la realizan los agentes de manera **automática** en el agente AMS por defecto.

Páginas blancas (AMS Agent) II

- Para **acceder** a los **servicios** del agente AMS hay que importar la clase **AMSService**.
- Esta clase **contiene** los siguientes **métodos**:
  - static void **register**: registra al agente en el AMS. Tanto esta operación como la operación deregister se realizan **automáticamente** en JADE cuando se **ejecutan** los métodos **setup**() y **takeDown**() respectivamente, por lo tanto no suelen ser usados normalmente.
  - static void **deregister**: elimina el **registro** del agente en el AMS.
  - static void **modify**: modifica los datos del agente en el AMS.
  - static AMSAgentDescription[] **search**: devuelve la **descripción** de los agentes **registrados** en el agente AMS.

Índice

Conversación - Protocolo

- Para poder **establecer** una **conversación** entre agentes es necesario **definir** previamente el **protocolo** que van a **seguir** durante la **conversación**.
- Un **protocolo** de interacción es una **descripción** detallada del **tipo** y **orden** de los **mensajes** involucrados en una **conversación** entre **agentes**.
- Un **agente** puede **participar** simultáneamente en **múltiples** diálogos con diferentes **agentes** y con diferentes **protocolos** de interacción.
- Dentro del estándar **FIPA** (utilizando **AUML**) se encuentran **definidos** algunos de los **protocolos** más extendidos.

Utilización de protocolos

- Para utilizar un **protocolo** en una **conversación**, los agentes deben escribir el **nombre** del protocolo a utilizar en el parámetro **:protocol**.
- Un protocolo **termina** cuando se alcanza el **último mensaje** del protocolo o se **elimina** el **nombre** del protocolo del **parámetro** :protocol.
- Cuando un agente **no conoce** un determinado **protocolo** tiene que devolver un mensaje **refuse** explicando el motivo por el que **rechaza** la **comunicación**.
- Si durante el **seguimiento** de un protocolo, un agente recibe un **mensaje** no contemplado en el mismo tiene que **devolver** un mensaje **not-understood**. Está **prohibido** responder un not-understood con otro not-understood para no caer en **bucles infinitos**.

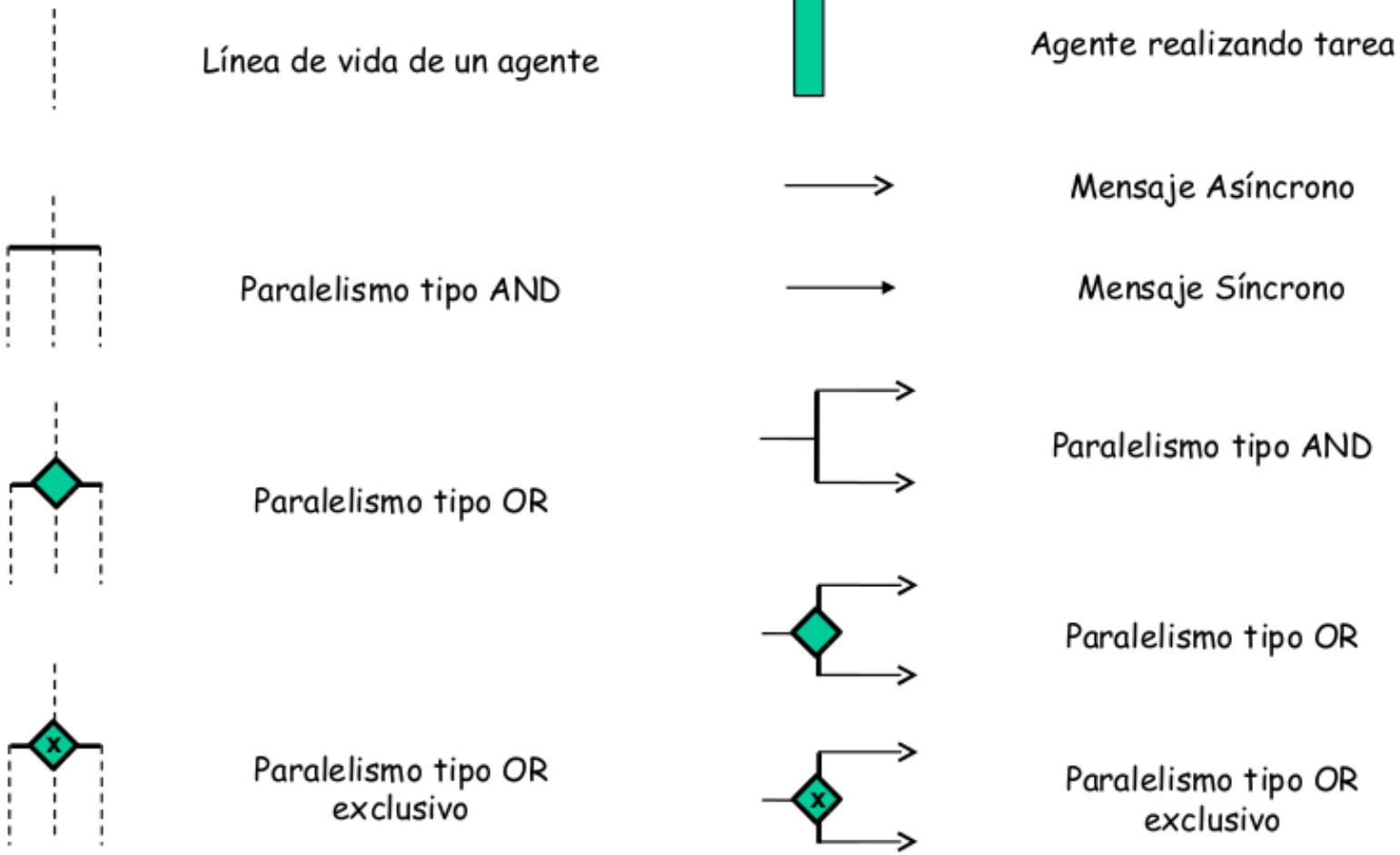
AUML

- **AUML** = Agent Unified Modelling Language.
- **UML** es **insuficiente** para **modelar** sistemas **multiagente**. **Comparados** con los **objetos**, los agentes son **activos** ya que actúan por **razones** que emergen de ellos **mismos**. Entre otros aspectos, necesitamos **modelar** los **protocolos** de **interacción** entre agentes.
- **Diagramas de Protocolo**: Similar a los **diagramas** de **interacción** utilizados en **UML**.

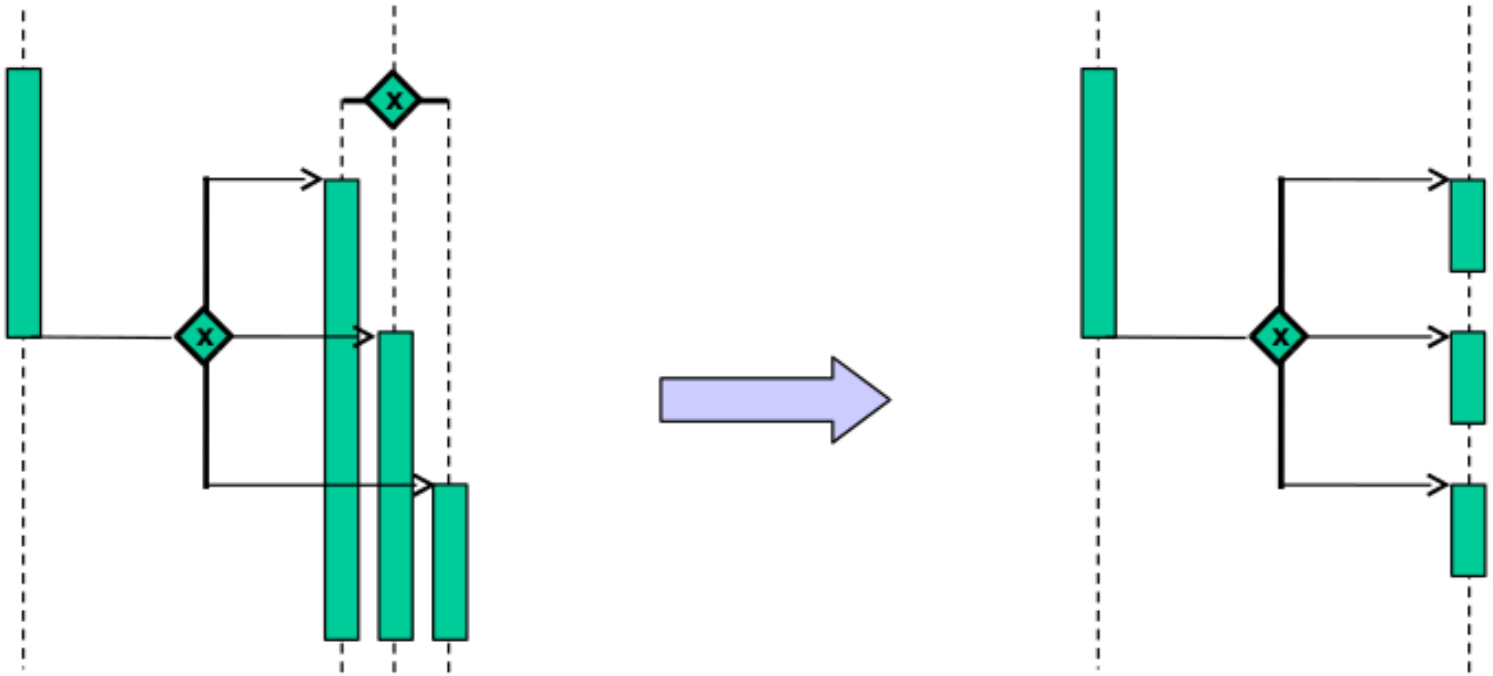
Diagramas de protocolo

- Un **Diagrama de Protocolo** (DP) tiene dos **dimensiones**: la vertical representa el **tiempo** y la horizontal representa el **rol** de los agentes **involucrados** en la **interacción**.
- Un DP **especifica** la secuencia de las **comunicaciones** por lo que **contiene** una **colección** de **mensajes** parcialmente **ordenados** entre un agente **emisor** y un agente **receptor**.
- Un **DP** también permite especificar **protocolos** de interacción **anidados**.

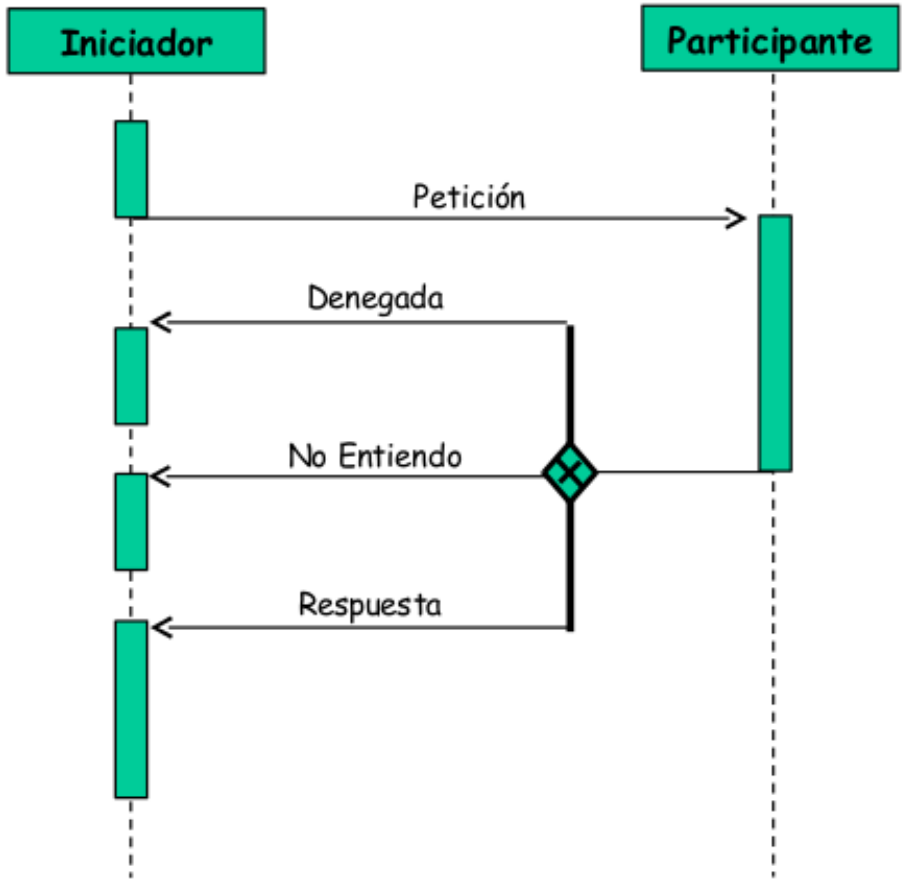
Elementos básicos de un DP



Posibilidad de abreviar el DP



Ejemplo de DP



El paquete jade.proto I

- En los **protocolos** de **comunicación** se define el rol de **Initiator** y el rol de **Responder**.
- Jade ofrece a través de **jade.proto** unas clases de **comportamiento** para ambos roles.
- En este **paquete** se agrupan todas las **clases** como comportamientos para la **implementación** de los protocolos de **comunicación** FIPA.
- Existen cuatro parejas (**initiator** y **responder**) de clases principales con las **cuales** se podrán **implementar** la mayoría de los **protocolos**.

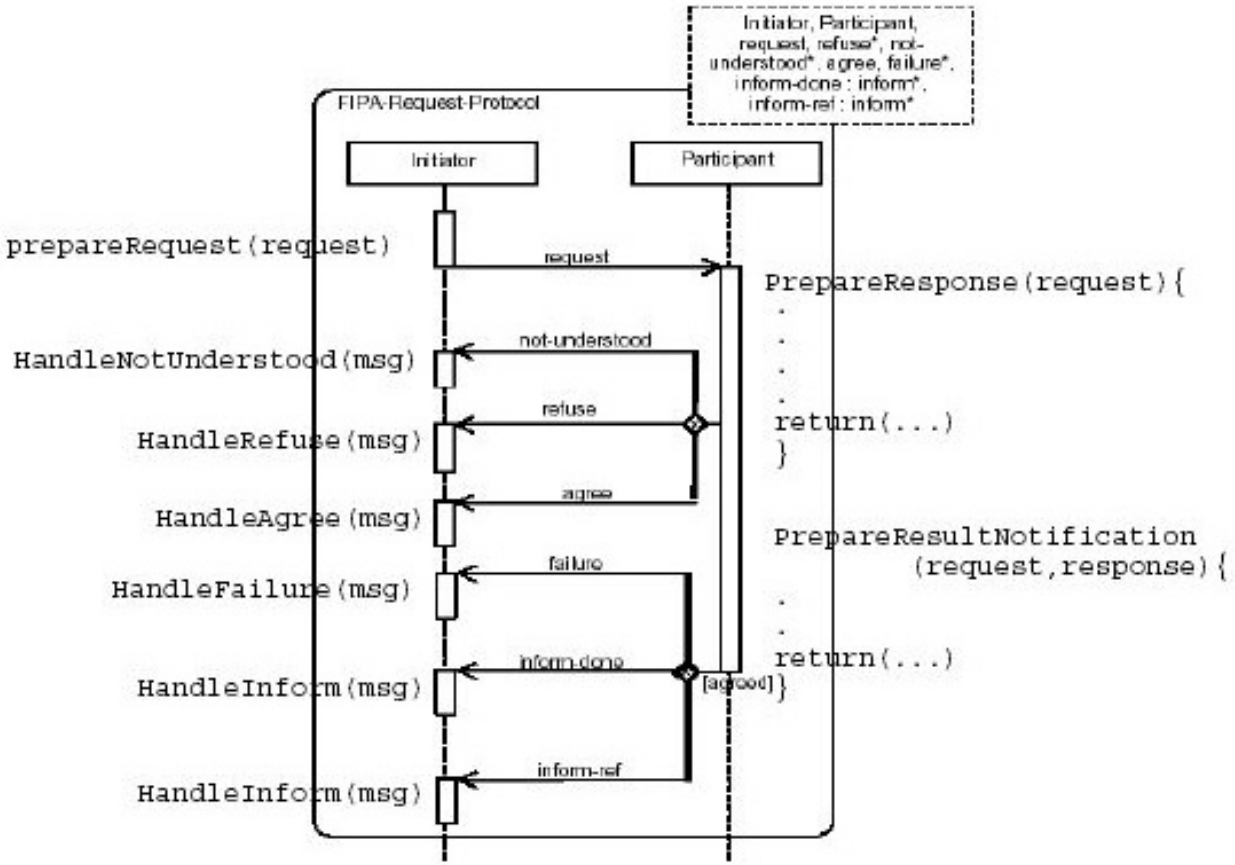
El paquete jade.proto II

Comportamientos	Protocolos FIPA

• **AchieveREInitiator**
• **AchieveREResponder**
  ○ SimpleAchieveREInitiator
  ○ SimpleAchieveREResponder
  ○ IteratedAchieveREInitiator
  ○ SSIteratedAchieveREResponder
	FIPA-Request
	FIPA-Query
	FIPA-Recruiting
	FIPA-Request-When
	FIPA-Brokering

• **ContractNetInitiator** • **ContractNetResponder** ○ SSContractNetResponder	FIPA-Contract-Net
• **SubscriptionInitiator** • **SubscriptionResponder**	FIPA-Subscribe FIPA-Request-Whenever
• **ProposeInitiator** • **ProposeResponder**	FIPA-Propose

El paquete jade.proto III



Bibliografía I

- Capítulo 8 del libro: **Introduction to Multiagent Systems**. Michael Woolridge. 2001. Ed. John Wiley & Sons, Inc.
- Transparencias de **Lenguajes** de **comunicación** entre **agentes**. Agentes Inteligentes. UPM. `http://www.sia.eui.upm.es/isa/doku.php?id=asignaturas:agentes_inteligentes`
- **Comunicación**: `http://programacionjade.wikispaces.com/Comunicacion`.

TEMA 4. COMUNICACIÓN.

Sistemas Multiagentes

Escuela Superior de Informática - Universidad de Castilla-La Mancha

Noviembre de 2015

TEMA 5. Ontologías.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Noviembre de 2015

Índice

Índice

Definición General

- La **RAE** define el **término** ontología como la "parte de la **metafísica** que trata del ser en **general** y de sus propiedades **trascendentales**".
- Sin embargo, en el campo de la **informática**, "**no** ha de ser considerada como una entidad **natural** que se descubre sino como un recurso **artificial** que se **crea**" (Mahesh, 1996).

Ontología y Conceptualización

El **sinónimo** más habitual de **ontología** es **conceptualización**:

- Modelo **abstracto** de algún **fenómeno** del mundo del que se identifican los **conceptos** que son **relevantes**.
- Hace **referencia** a la necesidad de **especificar** de forma **consciente** los distintos **conceptos** que conforman una **ontología**.
- Indica que la **especificación** debe representarse por **medio** de un **lenguaje** de representación **formalizado**.
- **Refleja** que una ontología **debe**, en el **mejor** de los **casos**, dar cuenta de **conocimiento** aceptado, como **mínimo**, por el **grupo** de personas que deben **usarla**.

Necesidad de utilización

- Una **ontología** es la **"especificación** de una **conceptualización"**, la **descripción** de los **conceptos** y **relaciones** entre ellos, que pueden formar **parte** del **conocimiento** de un agente o una **sociedad** de **agentes**.
- La **necesidad** de utilizar **ontologías** viene dada por la **complejidad** inherente a las **aplicaciones** desarrolladas en el contexto de los Sistemas **Multi-Agente**, que hace que se presenten las siguientes **dificultades**:
  1. **Abundancia** de **comunicación** entre agentes.
  2. **Interoperabilidad** de sistemas y **plataformas**.
  3. Problemas **semánticos**.

Comunicación

- Para que los **agentes** se **comuniquen** entre ellos, deben **compartir** el mismo **idioma**, **vocabulario** y **protocolos**.
- Al seguir las **recomendaciones** del estándar **FIPA**, JADE ya aporta un cierto grado de **coincidencia** al usar los **actos** comunicativos **FIPA** y su lenguaje de contenido **SL (Semantic Language)**, que determinan la **forma** en que los **mensajes** son **intercambiados** por los agentes.
- Sin embargo, será **necesario** definir **ontologías** específicas, con su propio **vocabulario** y **semántica** del contenido de los mensajes **intercambiados** por los agentes.

Comunicación I

JADE proporciona **tres** formas distintas de llevar a cabo la **comunicación** entre agentes:

- La forma más **básica** es utilizar **cadenas** para representar el **contenido** de los mensajes. Esto es **conveniente** cuando el contenido de los mensajes son datos **atómicos** pero no en el caso de **conceptos abstractos**, **objetos** o **datos estructurados**, ya que en estos casos, se **tendrían** que realizar **parseos** sobre las **cadenas** para acceder a sus diversas partes.
- Otra forma es utilizar objetos **serializables** de Java, que **transmitirían** directamente el **contenido** de los mensajes. Este es el **método** más **conveniente** para **aplicaciones locales** donde todos los agentes son implementados en Java. Un inconveniente es que estos mensajes **no** son **entendibles** para las **personas**.

Comunicación II

- El tercer método consistiría en **definir** los **objetos** que van a ser **transferidos** como **extensión** de las clases **predefinidas** por JADE que pueden **codificar/decodificar** los mensajes a un formato **FIPA estándar**. Esto permite que los agentes de JADE puedan **interoperar** con **otros** sistemas de **agentes**.
- Una **ontología** en JADE, se define de forma que los **agentes** se **comuniquen** utilizando el **tercer** método descrito.
- El soporte **JADE** para ontologías **incluye** las **clases** para **trabajar** con éstas y con los **lenguajes** de **contenido**.
- Los lenguajes de **contenido** tienen que ver con la **representación** interna del contenido de los **mensajes** ACL.
- Las **ontologías** tienen que ver con la **semántica** de los **mensajes** que se intercambian y su **chequeo**.

Comunicación III

- Es decir, mediante el uso de **ontologías** incorporamos **contenido semántico**, y no sólo datos, como hasta ahora, a los **mensajes** que se **intercambian** los agentes.
- Pero teniendo en cuenta que las **ontologías** se definen en base a **objetos** de **Java**, necesitamos un modo de **encapsular** o codificar la **semántica** de esos objetos dentro de **mensajes ACL**, y hacer el **proceso** contrario, de **descodificación**, en la recepción del **mensaje**.
- Con ese **propósito** llegamos a los **lenguajes** de contenido (**LEAP** y **SL**) y todo el **soporte** Jade para el manejo de **ontologías**.

Soporte de JADE para ontologías

- **Jade** incorpora, en el paquete **jade.content**, soporte (**codecs**), para los **siguientes** lenguajes de **contenido**:
- El **lenguaje SL** es legible por los **humanos** y codifica las **expresiones** como **string**.
- El **lenguaje LEAP no** es **legible** por los humanos y es **byte-encoded**.
- Una ontología es una **instancia** de la clase **jade.content.onto.Ontology** en la cual se definen los **Schemas**, conjuntos de elementos que definen la **estructura** de los **predicados**, las **acciones** de los agentes y **conceptos** relevantes al dominio del problema.

Soporte de JADE para ontologías

- **Predicados**: **expresiones** sobre el **estado** de **mundo**. Se **utilizan** típicamente en mensajes **INFORM** y **QUERY-IF**, no en REQUEST.
- **Acciones** de los agentes: expresiones que indican **acciones** que pueden **realizar** los **agentes**. Típicamente se utilizan en **mensajes** de tipo **REQUEST**.
- **Conceptos**: **expresiones** que representan **objetos**, representan una **estructura** con varios **atributos**. No aparecen **aislados** en los mensajes sino **incluidos** en otros elementos.
- **Otros** elementos: **primitivas** (elementos **atómicos** como números o cadenas de caracteres), **agregaciones** (conjuntos, listas de otros términos), **expresiones** (identifican las entidades para las que se **cumple** un **predicado**), **variables**.

Esquemas

- Los **esquemas** son instancias de las clases **PredicateSchema**, **AgentActionSchema** y **ConceptSchema** incluidas en el paquete **jade.content.schema**. Para cada uno de los **elementos** que definamos en la **ontología** se deberá de crear una **clase asociada**.

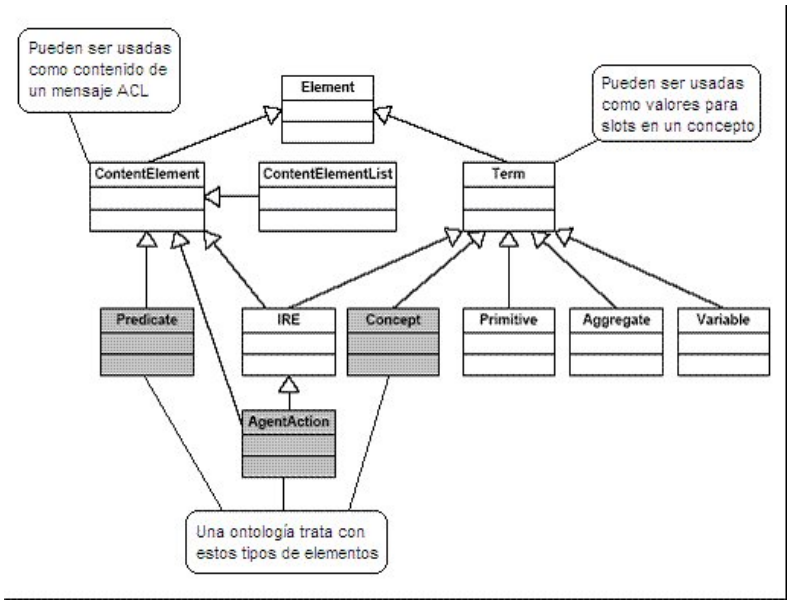


Figura 1 : Modelo de contenido de JADE.

Índice

Ontología JADE I

- Una **ontología** en JADE es una **instancia** de la clase **jade.content.onto.Ontology**.
- En ella se **añaden** los **esquemas** que definen la **estructura** de los **tipos** de predicados, acciones y conceptos **relevantes** en el **dominio** en cuestión.
- Estos esquemas son **instancias** de las clases **PredicateSchema**, **AgentActionSchema** y **ConceptSchema**, que se incluyen en el paquete **jade.content.schema**.
- Estas **clases** tienen **métodos** para poder declarar **slots** que definen la **estructura** de cada **tipo** de predicado, acción y concepto.

Ontología JADE II

- Como una **ontología** es básicamente una **colección** de **esquemas** que normalmente no **varían** a lo largo del **ciclo** de **vida** de un **agente**, es recomendable **declarar** la ontología siguiendo el patrón **singleton**, de modo que **sólo** se pueda crear un **objeto** de esa **clase**.
- Esto nos permite **compartir** la misma **ontología** (y todos los esquemas incluidos) **entre** todos los **agentes** que se estén ejecutando en la **JVM**.

Ejemplo de la frutería.

Supongamos una **ontología** para una **tienda** de **frutas**, que podemos utilizar para enviarse **ofertas** entre **agentes**.

- **Conceptos**: representan las **entidades** que forman parte de la ontología, en este caso la ontología tendrá un **concepto** que será "**Frutas**".
- **Predicados**: son **expresiones** que **relacionan** a los **conceptos** para decir algo. Son **necesarios** porque en un **mensaje** nunca podremos enviar **conceptos**, sino que tendremos que enviar **predicados** o **acciones**. En el ejemplo usaremos un predicado "**Oferta**" que indicará que un **agente** oferta una determinada **fruta**.
- **Acciones**: son **acciones** que pueden llevar a **cabo** los **agentes**. Para la **ontología** de **frutas** una acción será "**Comprar**" que **indicará** a un agente que **debe** comprar una determinada **fruta**.

Ejemplo de la frutería.

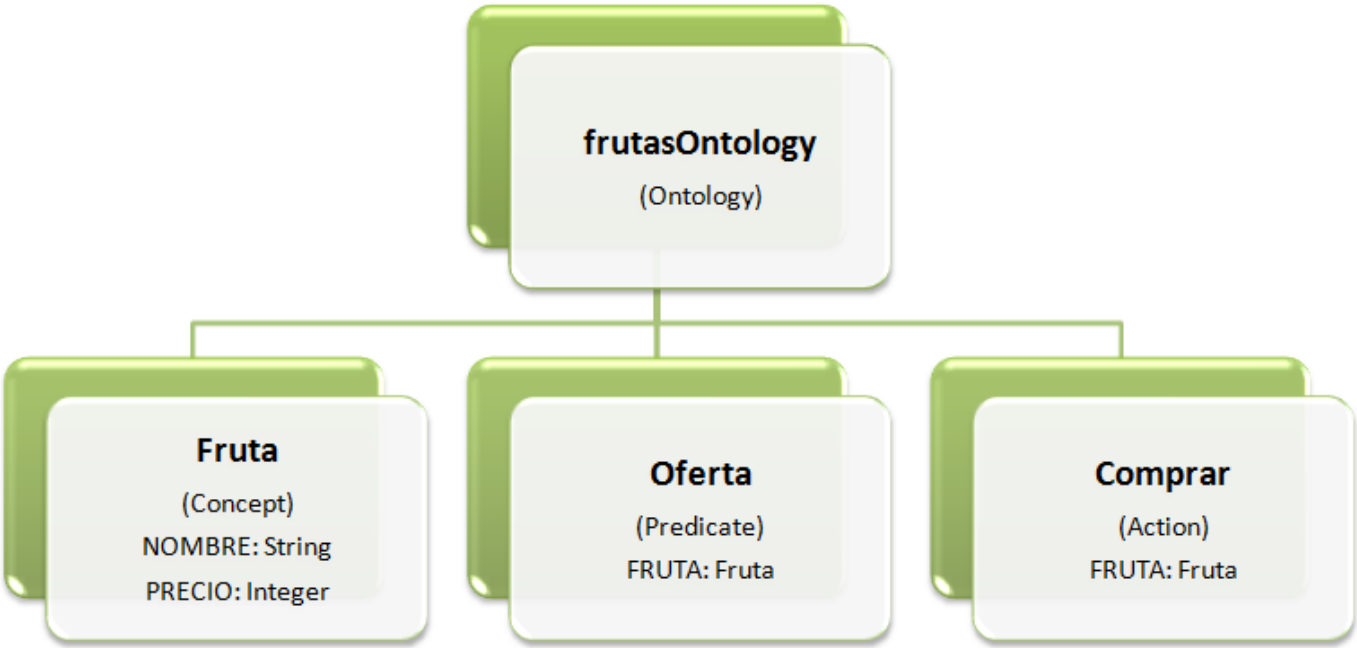


Figura 2 : Ontología para una tienda de frutas.

Ejemplo de la frutería.

- Imaginemos dos **agentes** que **comparten** esta **ontología**, **Comprador** y **Vendedor**, y que se **intercambian** mensajes de **ofertas**.
- Si **Vendedor** desea **comunicar** una oferta de **pimientos rojos** a 1 euro el kilo, esa **información** se guarda como un **objeto** de la clase **Oferta**, que tenemos que **convertir** a **formato ACL** para realizar el **acto comunicativo**.
- En el **receptor** se realizará el **proceso contrario**, a saber: se **recibe** un **mensaje** en formato **ACL** y se almacena **internamente** como un objeto de la clase **Oferta**.

Conversión y Operaciones.

- La **conversión** y las **operaciones** de chequeo son llevadas a cabo por un **objeto** que gestiona el **contenido**.
- Este **objeto** se denomina **ContentManager** y está **incluido** en el paquete **jade.content**.
- Cada **agente** en jade **posee** un **ContentManager** al que puede **acceder** usando el método **getContentManager()**.
- Esta clase **proporciona** todo los **métodos** de **transformación** de un objeto a un **string** para introducirlo en un **slot** de un **ACLMessage** y viceversa.

Conversión y Operaciones.

- La **conversión** la realiza a partir de la **ontología** definida y un **codec** de un lenguaje de **contenido** (instancia de la clase Codec del paquete **jade.content.lang**).
- La **ontología** permite **validar** la **información** desde un punto de vista **semántico** y el codec realiza la **conversión** a **string** según las reglas **sintácticas** del **lenguaje** de contenido.
- **JADE** define dos **lenguajes** de contenidos predefinidos.
- Lenguaje **SL**: **legible** para las personas y codifica las **expresiones** como string (**jade.content.lang.sl**) y Lenguaje LEAP: **no legible** y codifica en **byte-encoded** (**jade.content.lang.leap**).
- El **último paso** consiste en **registrar** el **lenguaje** de contenido y la **ontología** en el método **setup()** de los **agentes**.

Bibliografía I

- Utilización de Ontologías. http://www.ibspan.waw.pl/~gawinec/abc/2007/labs/lab2-onto.pdf.
- Tutorial Ontologías TILAB. http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf
- Código Frutería + ejercicios seminarios. http://programacionjade.wikispaces.com/Ontologías.
- Jade Implementation Short Guide. http://www.cs.uu.nl/docs/vakken/mas/Pract/JADEtutorial.pdf.

TEMA 5. Ontologías.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Noviembre de 2015

TEMA 6. Agentes Móviles.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Noviembre de 2015

Índice

Índice

Introducción

- Los **agentes** inteligentes son capaces de **ofrecer** un comportamiento **"inteligente"**: razonamiento, planificación, **aprendizaje**, etc.
- Los **agentes móviles** (AM) tienen la capacidad de **moverse** por distintos **nodos** de una **red** (una o más veces).
- Para la comunidad de IA la **movilidad** no es un **atributo** de los agentes.
- Para la **comunidad** de AM la **movilidad** es el atributo **principal**.

Índice

Agente móvil

- Agente que realiza sus **tareas** no sólo en la **computadora** de su **propietario**, sino también en **otras** en la **red**.
- Busca **información** en **beneficio** de su propietario.
- Puede **negociar** y cerrar **tratos** en su nombre.
- Puede **utilizar** servicios **remotos**.

Agente móvil

- Un **agente** móvil tiene **capacidad** para decidir a qué **servidores** moverse.
- Hay instrucciones **explícitas** para que el agente pueda **parar** su **ejecución**, **migrar** a otro **nodo** (preservando su estado), y continuar su **ejecución**
- Puede **moverse** a **uno** o más **servidores**.
- Es una **extensión** del modelo **cliente-servidor**. Los clientes **envían** parte de ellos al **servidor** (o a varios servidores) para **ejecutarse**.

Taxonomía del código móvil

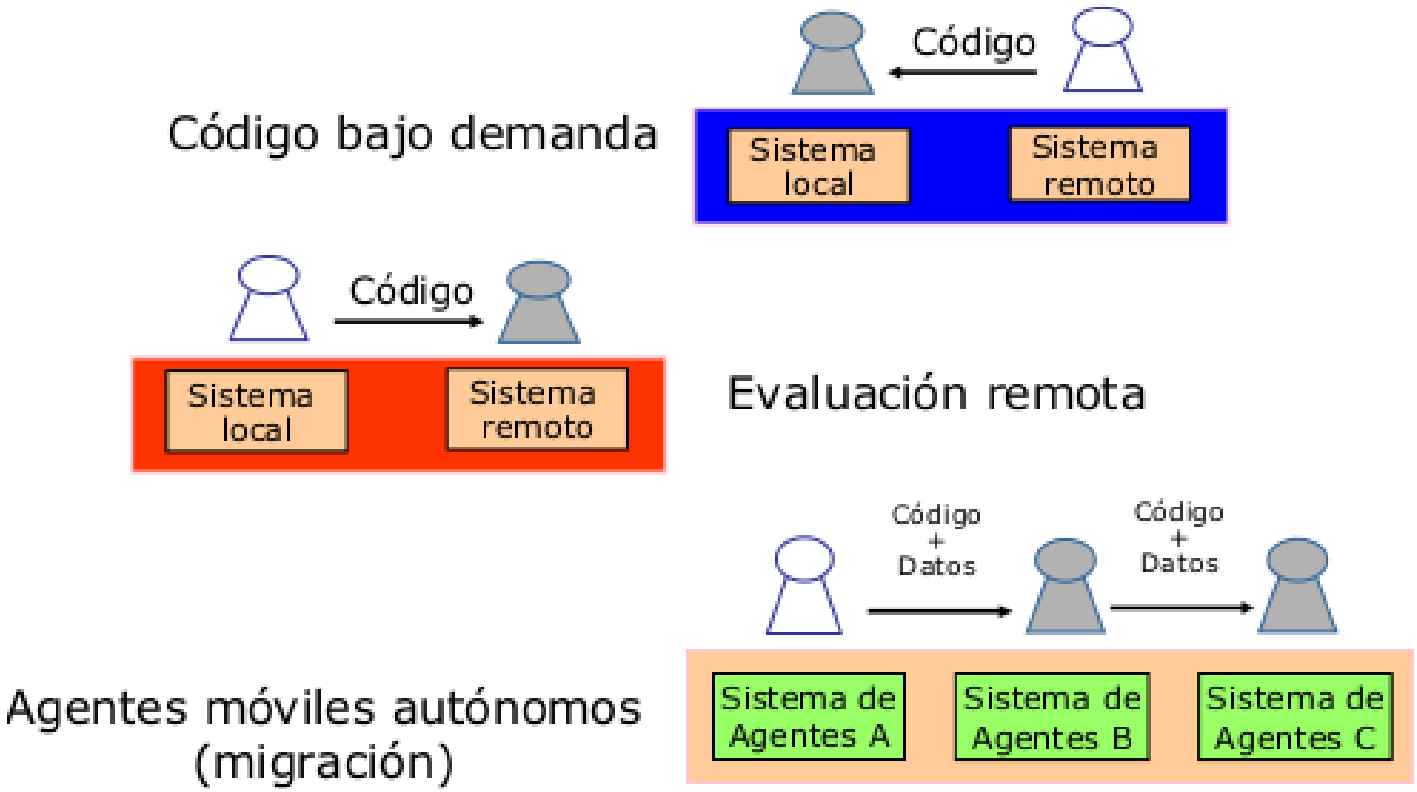


Figura 1 : Código móvil

Taxonomía del código móvil

Dependiente de la aplicación:

- **applet**: aplicación **cargada** por la **red** para ejecutar **localmente**.
- **servlet**: agente que es **cargado** en un lugar **remoto** donde será **activado** como un **servidor**.
- **extlet**: agente que es **cargado** en un lugar **remoto** para extender las **capacidades** del **receptor**.
- **deglet**: agente en el que se **delega** la autoridad para **realizar** una tarea **determinada** (acabada la tarea, desaparece).
- **netlet**: agente en el que se delega la **autoridad** para realizar una tarea **permanente** (esto es, nunca termina).
- **piglet**: agente **malicioso** que supone **riesgo** de **seguridad**

AM vs. RPC: Paradigmas de programación distribuida

Arquitecturas cliente-servidor clásicas:

- **Sockets**.
- RPC, Java **RMI**.
- **Middleware**: DCE, **CORBA**, DCOM.

Programación remota:

- Ejecución **remota** de trabajos (IBM Remote Job Entry, años 70).
- **Telecarga** de funciones a **bases** de **datos** (funciones a los datos).
- **Compartición** de **recursos**, balance de carga, etc..
- Java **applets**.
- **Agentes móviles**.

Cuestiones de Seguridad

- **Autenticación** del **usuario**: **¿Quién envía** el agente móvil?
- **Autenticación** del **servidor** o entorno de ejecución de agentes: ¿No **caerá** el **agente** en una **trampa**?
- **Derechos** de **ejecución** de **agentes** de un usuario en un **servidor**: ¿Pueden **ejecutarse** los agentes? ¿Qué **funciones** pueden realizar?
- **Capacidad** del agente para **pagar** por los **servicios** utilizados: **Teleclicks** (General Magic).
- **Detección** de **virus**: ¿Se puede **confiar** en el agente?

Cuestiones de Seguridad

- **Control** de **ejecución** de las **tareas**: la **localización** y el **estado** de un agente pueden ser **desconocidos**.
- **Gestión** de **fallos**: **¿Quién** detecta los fallos?
- **Eficiencia** (el código interpretado suele ser lento).
- **Heterogeneidad** de **sistemas** de AM.
- **Sobrecarga** de la **transferencia** del **código**: tiene que **compararse** con las **interacciones**.
- Acceso a **servicios** existentes (**Internet**, CORBA, etc.)

Aplicaciones

- **Servicios** de **información** en **Internet**: **recuperación** y **extracción** de información de múltiples lugares, **búsqueda** y **filtrado** de la información control de **cambios** y **difusión** de información.
- **Comercio electrónico**: **mercado** de servicios electrónico y **negociación**.
- **Equipos** móviles y **PCs** en el hogar: conexiones **intermitentes** y **bajo ancho** de **banda**.
- **Redes** públicas de **telecomunicaciones**: provisión de **servicios** bajo **demanda** y descentralización del control y **gestión** de **redes**.
- Procesamiento **paralelo**.
- Gestión de procesos (**workflow**).
- **Juegos** (**agentes** que representan **jugadores**).

Índice

Soporte para la movilidad de JADE I

- La **movilidad** de un agente es la **habilidad** para que este **migre** o haga una copia de sí mismo (**clon**) a través de uno o múltiples **nodos** de una red.
- El **soporte** de movilidad en **JADE** consiste en un conjunto de **clases**, **métodos** y una **ontología** específica de movilidad (MobilityOntology).
- Las clases y métodos permiten a un agente **ejecutar** las **acciones** requeridas por sí mismo o por el **AMS** (Agent Management System).
- JADE soporta solo **movilidad** dentro de la misma **plataforma**. Esto quiere decir que un agente **móvil** se puede mover a través de **contenedores** pero está **limitado** a una sola plataforma.

Soporte para la movilidad de JADE II

- Existe un **proyecto** de la Universidad autónoma de Barcelona que implementa el servicio de movilidad inter-plataforma de movilidad llamado IPMS (**Inter-Platform Mobility Service**), el cual puede permitir **migrar** de una **plataforma** a otra **distinta**.
- Cada **instancia** de ejecución en **JADE** se llama **contenedor**. El conjunto de todos los contenedores se llama plataforma y proporciona una capa **homogénea** que **esconde** los **agentes**, la aplicación desarrollada, la **complejidad** y los contenedores.
- Por lo tanto, tenemos que: se **involucran** varios **hosts**, cada host tiene su **contenedor** y la **migración** puede suceder a **petición** del propio agente.

Ventajas

- Proceso **independiente** y **asíncrono**: cuando un agente migra **no** es **necesario** que siga en **contacto** con su **propietario** para realizar su cometido. Un agente puede **migrar** y realizar una **tarea** complicada y cada cierto tiempo **informará** sobre los **resultados**.
- **Tolerancia** a **fallos**: cuando se detectan **problemas**, los agentes móviles pueden **migrar** a otra **plataforma** y probar suerte allí sin que se **detenga** la **ejecución**.
- **Apropiados** para **conjuntos** grandes de **datos**: los agentes se **desplazan** hacia donde se **encuentran** los datos y no al revés, ahorrando en **transferencia** de información y ganando en **procesamiento**.

Inconvenientes

- **Escalabilidad** y rendimiento: aunque **reducen** el tráfico en **red**, generan un **incremento** de carga de **procesamiento**. Esto es porque son generalmente **programados** en lenguajes **interpretados**.
- **Seguridad**: el uso de agentes móviles puede traer **problemas** de **seguridad**. Cualquier **código** móvil debe ser revisado **cautelosamente**.
- **Portabilidad** y **estandarización**: Los agentes móviles no pueden inter-operar sino siguen unos **estándares** de **comunicación**.

Movilidad Inter-Plataforma (IPMS)

- El **modelo** de Movilidad Inter-Plataforma (IPMS) es un **complemento** de Jade que se ha creado para permitir la **movilidad** de agentes entre **plataformas**.
- La idea es **utilizar** mensajes **FIPA-ACL** como medio de **transporte**.
- Estos **mensajes** son **enviados** entre los **AMS** de las plataformas **finales**.
- Se **especifican** dos **acciones** en la **ontología**, **move** y **power-up**.
- La primera representa el **movimiento** del código del agente, y la segunda la **activación** del agente una vez **completada** la **migración** inter-plataforma.

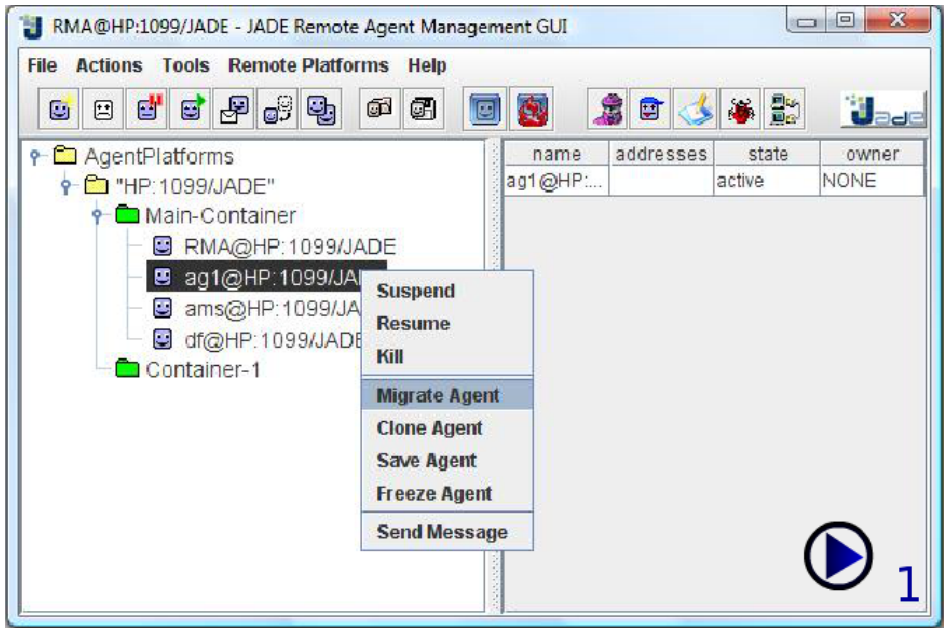
Movilidad Inter-Plataforma (IPMS)

Adicionalmente, algunos conceptos de la ontología son **mobile-agent-description** y **mobile-agent-profile** que contienen toda la información del **agente**. Estos conceptos aseguran la **compatibilidad** entre plataformas. El **proceso** es el siguiente:

- Se envía un **Request** a la plataforma **destino** esperando respuesta.
- Si obtenemos un **Inform** el agente puede **serializarse** y enviarse dentro de un mensaje a la **plataforma** destino.
- Si ocurre algún **problema** durante este traspaso de información se **desharán** todos los **cambios**.

Movilidad Inter-Plataforma (IPMS)

- La **ventaja** de utilizar mensajes **ACL** es que **no** es necesario **abrir** otro canal de **comunicación** entre **plataformas**.
- La desventaja es que el **rendimiento** no es particularmente alto debido al proceso de **codificación** y **decodificación** del propio **mensaje** ACL.



Ejecución en plataforma en host remoto

```
java jade.Boot -host hostRemoto -container agente1:Agente
```

Movilidad de código y estado de ejecución

- Un **agente** puede **dejar** de **ejecutarse** sobre una máquina, migrando sobre diferentes nodos **remotos** (sin la necesidad de tener el código del agente instalado en ese nodo).
- Podrá **reiniciar** su **ejecución** en el **punto** en que fue **interrumpido** (realmente, JADE implementa una forma de movilidad no tan débil, porque la **pila** y el **registro** del programa no pueden ser **salvados** en **Java**).
- Esta **funcionalidad** permite, por ejemplo, **distribuir** la carga **computacional** al tiempo de ejecutar los movimientos del agente, a las **máquinas** menos cargadas sin algún **impacto** en la **aplicación**.

Métodos en Agent para movilidad y clonación

Requieren de ciertos **parámetros** (objetos de tipo **Location**) porque los agentes necesitan **conocer** su **localización** para ello se debe incluir el siguiente **import**:

```
import jade.core.*;
```

- doMove()
- **beforeMove**()
- afterMove()
- **doClone**()
- beforeClone()
- **afterClone**()

doMove()

doMove(Location destino):

- Este método **recibe** como parámetro un **objeto** de tipo jade.core.**Location**, que representa el **destino** requerido donde debe migrar el agente.
- jade.core.Location es una **interfaz** abstracta, usada porque las **aplicaciones** de agentes no pueden **crear** sus propias **localizaciones**.
- Para **obtener** el objeto **Location** es necesario hacer una **petición** al **AMS**.
- Esta puede ser del tipo **WhereIsAgentAction** o **QueryPlatformLocationsAction**.

doMove()

WhereIsAgentAction:

- Es una clase que **contiene** un método **setAgentIdentifier(AID)**.
- Recibe como **parámetro** el **identificador** del **agente** del que se quiere obtener el objeto **Location**.
- Este indicará la **localización** del **container** de dicho agente.
- Esto se puede **obtener** mediante el método **getAgentIdentifier()**.

QueryPlatformLocationsAction:

- Devuelve los objetos **Location** de los **containers** de todos los **agentes** que estén **disponibles**.
- Por tanto, no **necesitará** recibir ningún **parámetro**.

doClone()

doClone(Location destino,string nombre):

- Este **método** recibe dos **parámetros**, uno es el **destino** al que debe **migrar** el nuevo **clon** del **agente**, y el otro un **string**.
- Este será el **nombre** que se le dará a dicho **clon**, **distinto** del **original**.

Gestión de Recursos I

- El hecho de **mover** un agente implica **enviar** su **código** y **estado** a través de la red.
- Algunos de los **recursos** usados por el **agente** móvil se **moverán** con el, mientras que **otros** serán **desconectados** antes de salir, y **conectados** de nuevo, una vez que **lleguen** al **destino**.
- **beforeClone**() y **afterClone**() permiten al agente realizar **tareas específicas** antes y después de **clonarse**. Esto se utilizaría, por ejemplo, en el caso de **agentes** que son **plataformas** o contenedores **dependientes** como la **ontología** y el **lenguaje**. Por esta razón, **después** de que el agente se **mueva**, se debe **registrar** otra vez el lenguaje y la ontología.

Gestión de Recursos II

- **beforeMove()**: se invoca en la **localización** de **partida** antes de **enviar** el **agente** a través de la **red** (con el **scheduler** de **comportamientos** ya parado)
- **afterMove**(): se invoca en la **localización** de **destino** tan pronto como el **agente** llega y se **identifica** en el lugar (el **scheduler** aún no reiniciado)

MobilityOntology

- Para la **movilidad** y **comunicación** entre los agentes necesitaremos una **ontología** que proporcione una **descripción** del sistema además de las **acciones** que se pueden llevar a cabo.
- JADE **proporciona** esta **ontología** mediante la clase jade.domain.**MobilityOntology**.
- Esta ontología **no** se **ajusta** a ninguna **especificación FIPA**. Según el equipo de desarrollo de JADE, la actualizarán tan pronto como aparezca una especificación FIPA **apropiada**.
- Se utiliza la **ontología** de **movilidad** para decir a un agente que se mueva a un **Location** en concreto.

```
import jade.domain.mobility.*;
```

MobilityOntology. Conceptos. I

- **mobile-agent-description**: describe un **agente móvil** que va a algún **sitio**. Se representa por la **clase** interna MobilityOntology.MobileAgentDescription, la cual tiene varios métodos **get** y **set** de acuerdo con las **reglas** para clases de la ontología JADE.
- **mobile-agent-profile**: describe el **entorno** de **programación** necesario para el agente **móvil**. Se representa a través de la clase interna MobilityOntology.MobileAgentProfile.
- **mobile-agent-system:** describe el **tiempo** de **ejecución** usado por el agente móvil. Se representa mediante la clase interna MobilityOntology.MobileAgentSystem.

MobilityOntology. Conceptos. II

- **mobile-agent-language**: describe el **lenguaje** de **programación** usado por el **agente móvil**. Se representa mediante la clase interna MobilityOntology.MobileAgentLanguage
- **mobile-agent-os**: describe el conjunto de **operaciones** del **sistema** necesarias para el agente **móvil**. Se representa mediante la clase interna MobilityOntology.MobileAgentOS
- **location**: describe el **lugar** a **donde** un agente puede **ir**. Se representa mediante la clase interna MobilityOntology.Location.

MobilityOntology. Acciones.

- **move-agent**: acción de **mover** un **agente** de un lugar a **otro**. Se representa mediante la **clase** interna MobilityOntology. MoveAction
- **clone-agent**: la acción que representa es la de hacer una **copia** de un **agente**, posiblemente **ejecutado** en **otra ubicación**. Se representa mediante la **clase** interna MobilityOntology.CloneAction
- **where-is-agent:** la acción de **responder** con la **localización** donde un agente se está **ejecutando**. Se representa mediante la clase interna MobilityOntology.WhereIsAgent.
- **query-platform-locations**: la acción de **responder** con una lista de las **ubicaciones** de todas las **plataformas**. Se representa mediante la clase interna MobilityOntology. QueryPlatformLocations.

- El **AMS** provee algunas **extensiones** que dan soporte a la **movilidad** de agentes y es capaz de realizar las **acciones** presentes en el jade-mobility-ontology.
- Cada acción **relacionada** con la **movilidad** se puede **solicitar** al **AMS** a través del protocolo **FIPA-request**, con jade-mobility-ontology como valor del campo **ontology** y **FIPA-SL0** como valor **language**.
- Un **comportamiento** típico para un agente móvil será **preguntar** al **AMS** por **localizaciones** (bien la lista completa o bien a través de varias acciones where-is-agent), luego el **agente** será capaz de **decidir** si, cuando y a donde migrar.

```
import jade.domain.JADEAgentManagement.*;
```

Acciones para la movilidad I

- **move-agent**: Esta **acción** toma un **mobile-agent-description** como su parámetro. **Mueve** el **agente** identificado por los **slots** name y address del mobile-agent-description al lugar indicado en el **slot destination**.
- Mediante la **ontología** JADE, podemos añadir **movilidad** a los agentes **sin** tener que **componer** mensajes **ACL**.
- Para ello, en primer lugar, el agente debe crear un objeto de tipo **MoveAction**, rellenar sus argumentos con un **MobileAgentDescription**, que a su vez se rellenará con el **nombre** y **dirección** del agente que queremos mover (él mismo u otro agente) y con el objeto **Location** para el destino.

Acciones para la movilidad II

- Tras esto, con una simple **llamada** al método **Agent. getContentManager().fillContent(..,...)** podemos convertir el objeto **MoveAction** en un **String** y escribirlo dentro del **slot** content de un mensaje ACL **request** apropiado.
- **clone-agent**. Es **similar** a la acción **move-agent**, pero en este caso tiene un argumento más de tipo **String**, en el que se pasa el **nuevo nombre** del agente que resulta del **proceso** de **clonación**.

Acciones para la movilidad III

- El **AMS** también soporta otras acciones **relacionadas** con la **movilidad** y que están definidas en **JADEManagementOntology**.
  - **where-is-agent**. Solo tiene como argumento el **AID** del agente que se quiere **localizar**. Tiene como resultado el lugar del agente, que se **coloca** en el **slot** content del mensaje **ACL inform**.
  - **query-platform-locations**. Esta acción **no** toma **argumentos**. Su resultado es un **conjunto** de todos los objetos **Location** disponibles actualmente en la plataforma **JADE**.

Bibliografía I

- Research Group on Intelligent Agents - Engineering and Applications. UCM. `http://grasia.fdi.ucm.es/main/`
- Soporte para la movilidad en JADE. `http://programacionjade.wikispaces.com/Movilidad`

TEMA 6. Agentes Móviles.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Noviembre de 2015

TEMA 7. INTERACCIÓN.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Diciembre de 2015

Índice

Índice

- En un sistema **multiagente**, los agentes **individuales** van a tener que ser **capaces** de **interaccionar** entre **ellos** para obtener **soluciones** de **problemas** que el propio **agente** no es capaz de **encontrar**.

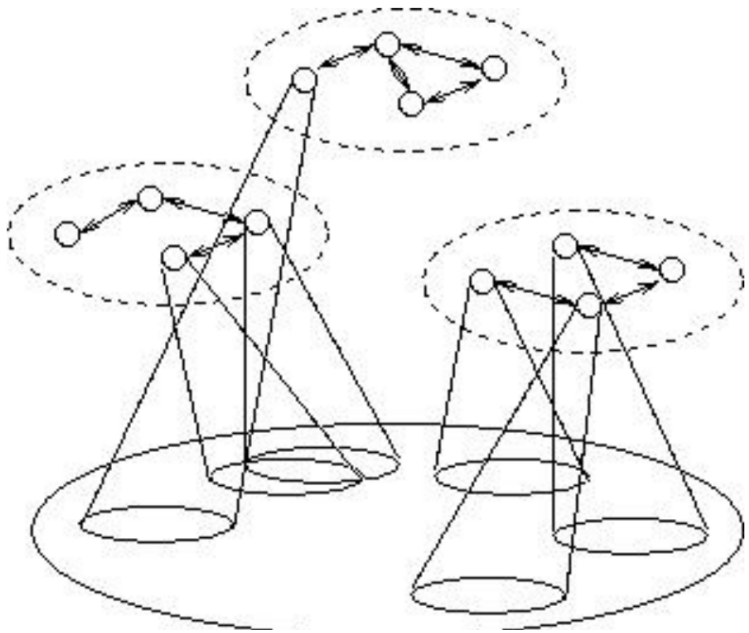


Figura 1 : Esfera de influencia de un agente

Predicción

- Un **agente** debe **prever** las **acciones** de los otros **agentes** en su propia tarea de **planificación** y cómo puede **influir** en las **acciones** de otros agentes en **beneficio** de sus propios **objetivos**.
- Los **efectos** de las **acciones** de otros agentes sobre nuestro agente pueden ser **favorables**, **neutras** o **perjudiciales** para los **objetivos** de nuestro agente.

Tipos de interacción

- **Cooperativa** (**CMAS** "Cooperative Multi-Agent Systems")
- **Competitiva** (**SMAS** "Self-Interested Multi-Agent Systems")

Negociación

- **Función de utilidad**: **diferencia** entre el **beneficio** de **conseguir** un **objetivo** y el **coste** invertido para **lograrlo**.
- **Espacio de transacciones**: **acción** que un **agente** hace y que lleva **asociada** una **utilidad**.
- **Estrategias y Protocolos de Negociación**: **reglas** que gobiernan la **negociación**, incluyendo **cómo** y **cuándo finaliza** la misma.

Índice

Definiciones I

- Se asume que se dispone de **dos agentes**: $Ag = i, j$.
- Cada **agente** tiene su **propio interés** sobre cómo es el entorno.
- Sea $\Omega = \{\omega_1, \omega_2, ...\}$ el conjunto de las "**salidas**" sobre las que el agente tiene **preferencias**.
- Se **capturan** las **preferencias** usando funciones de **utilidad**:

Utilidad

$$u_i : \Omega \rightarrow \Re$$
$$u_j : \Omega \rightarrow \Re$$

Definiciones II

- Las funciones de **utilidad** conducen a **ordenamientos** de las **preferencias** sobre las **salidas**:

Ordenamiento

$$\omega \succcurlyeq \omega' : u_i(\omega) \geq u_i(\omega')$$
$$\omega \succ \omega' : u_i(\omega) > u_i(\omega')$$

¿Qué es la utilidad?

- La **utilidad** no es **dinero** (pero es una **analogía** útil).

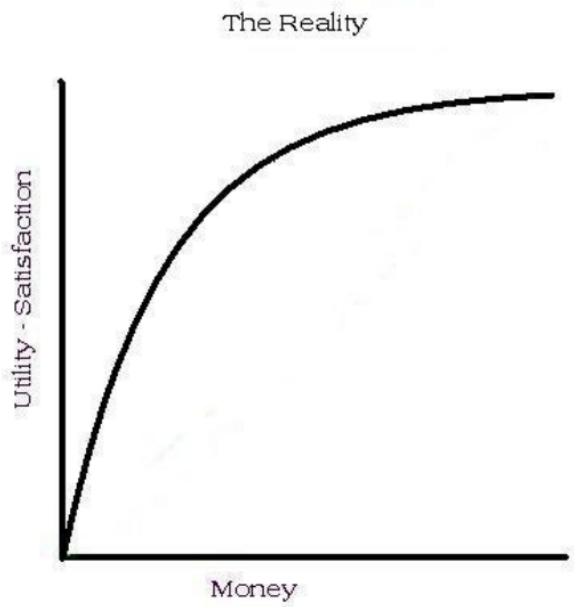


Figura 2 : Relación típica entre utilidad y dinero

Interacciones competitivas y suma cero

- Cuando las **preferencias** de los **agentes** son diametralmente **opuestas** tenemos **escenarios** estrictamente **competitivos**.
- Encuentros suma cero son **aquellos** donde la **suma** de las **utilidades** es cero:

$$u_i(\omega) + u_j(\omega) = 0 \text{ para todo } \omega \in \Omega$$

- Suma cero implica **estrictamente competitivo**.
- Los **encuentros** suma cero son muy **raros** en la vida **real**... Pero las **personas tienden** a **actuar** en muchos escenarios como si lo **fueran**.

Índice

- Se **necesita** de un **modelo** del **entorno** en el que actuarán estos **agentes**.
- Los **agentes** escogen **simultáneamente** una **acción** a realizar, y como **resultado** de estas **acciones** seleccionadas, se produce un **salida** Ω.
- La **salida** real depende de la **combinación** de las **acciones**.
- Se **asume** que cada **agente** sólo puede realizar **dos acciones**: **C** (Cooperar) y **D** (Defect-traicionar).
- El **comportamiento** del entorno está dado por una **función** de **transformación** de **estados**:

Comportamiento

$$\tau : Ac\ X\ Ac \rightarrow \Omega$$

Ejemplo de funciones de transformación

Entorno sensible a acciones de los dos agentes

$$\tau(D,D) = \omega_1 \; \tau(D,C) = \omega_2 \; \tau(C,D) = \omega_3 \; \tau(C,C) = \omega_4$$

Entorno no influenciado por ningún agente

$$\tau(D,D) = \omega_1 \; \tau(D,C) = \omega_1 \; \tau(C,D) = \omega_1 \; \tau(C,C) = \omega_1$$

Entorno controlado por j

$$\tau(D,D) = \omega_1 \; \tau(D,C) = \omega_2 \; \tau(C,D) = \omega_1 \; \tau(C,C) = \omega_2$$

Agentes racionales

- Se supone el **caso** en que ambos **agentes** pueden **influenciar** la **salida**:

$$u_i(\omega_1) = 1 \; u_i(\omega_2) = 1 \; u_i(\omega_3) = 4 \; u_i(\omega_4) = 4$$
$$u_j(\omega_1) = 1 \; u_j(\omega_2) = 4 \; u_j(\omega_3) = 1 \; u_j(\omega_4) = 4$$

- **Simplificando** de la **notación**:

$$u_i(D,D) = 1 \; u_i(D,C) = 1 \; u_i(C,D) = 4 \; u_i(C,C) = 4$$
$$u_j(D,D) = 1 \; u_j(D,C) = 4 \; u_j(C,D) = 1 \; u_j(C,C) = 4$$

- Por tanto, las **preferencias** del **agente** i son:

$$C,C \succcurlyeq_i C,D \succ_i D,C \succcurlyeq_i D,D$$

- C es la opción **racional** para i: **prefiere** todas las **salidas** que obtiene haciendo C sobre todas las que obtiene **haciendo** D.

Matrices de retribución

- Este **escenario** se puede **caracterizar** como una **Matriz de Retribución** o payoff matrix.
- El **agente i** es el jugador de las **columnas**.
- El **agente j** es el jugador de las **filas**.

	defect	coop
defect	1 1	4 1
coop	1 4	4 4

Figura 3 : Matriz de retribución

Índice

Tipos de comportamiento

En la literatura relacionada con **teoría de juegos** ya **no** se habla de **acciones** sino de **estrategias**.
¿Cómo se **comportará** un **agente** en cualquier **escenario** dado?

- Estrategia **dominante**.
- Estrategia de **equilibrio** de **Nash**.
- Estrategia **Pareto Optimal**.
- Estrategias que **maximicen** el bien **común**.

Estrategia dominante

- Dada **cualquier estrategia** particular **s** (C o D) para el **agente i**, habrá cierto **número** de posibles **salidas**.
- Se dice que s_1 **domina** s_2 si cada **salida posible** de i jugando s_1 es **preferible** sobre cada **salida posible** jugando s_2.
- Un agente **racional nunca** jugará una estrategia **dominada**.
- Luego, al **decidir** que **hacer**, se podrían **borrar** las estrategias **dominadas**.
- **Desafortunadamente** no siempre hay una estrategia **no dominada única**.

Equilibrio de Nash

- Dos **estrategias** s_1 y s_2 están en **equilibrio de Nash**:
  1. El **agente i** juega s_1 y el **agente j** no puede hacer **nada mejor** que jugar s_2
  2. Bajo la suposición de que el **agente j** juega s_2 y el **agente i** no puede hacer **nada mejor** que jugar s_1
- **Ninguno** de los **agentes** tiene algún **incentivo** para **desviarse** de un equilibrio de Nash.
- Desafortunadamente:
  1. **No** todo escenario de **interacción** tiene un **equilibrio** de Nash.
  2. Algunos **escenario** de interacción tienen **más** de un **equilibrio** de Nash.

Optimalidad de Pareto

- Se dice que una **salida** es **Optimal** de **Pareto** si **no** hay otra **salida** que haga que uno de los agentes **mejore** sin hacer que el otro **empeore**.
- Si una salida es **optimal** de Pareto, entonces, al menos **uno** de los agentes se **rehusará** a **moverse** de ella. (Debido a que **empeorará** su situación).
- Si una **salida** ω **no** es optimal de Pareto, entonces hay una **salida** ω' que hace que todo el mundo tan **feliz**, o más feliz, que ω.
- En este caso. los agentes "**razonables**" estarán de **acuerdo** en moverse a ω' (Aun cuando no me **beneficio** directamente de ω', tu puedes **beneficiarte** sin que yo **sufra**).

Bienestar Social

- El Bienestar Social de una salida ω es la **suma** de las **utilidades** que cada **agente** obtiene de ω:

$$\sum_{i \in Ag} u_i(\omega)$$

- Puede **pensarse** como "la cantidad **total** de **dinero** en el **sistema**".
- Como un **concepto** de **solución**, puede ser **apropiado** cuando todo el **sistema** (todos los agentes) tienen un dueño **único** (todo el **beneficio** del sistema es **importante**, no el de los **individuos**).

Índice

El dilema del prisionero

Dos **hombres** has sido **acusados** en conjunto de un **crimen** y están **presos** en celdas separadas, **sin** poder **comunicarse**. Se les dice:

- Si uno **confiesa** y el **otro no** lo hace, el que confesó es **liberado**; mientras que el otro pagará **tres años**.
- Si ambos **confiesan**, cada uno pagará **dos años**.
- **Ambos** saben que si **ninguno** confiesa, serán **encarcelados** por **un año**.
- **Confesar** se considera "**Traicionar** al otro prisionero".
- **No confesar** se considera "**Cooperar** con el otro prisionero".

Matriz de retribución

- Si ambos **traicionan**, los dos son **penalizados** por la **traición mutua**.
- Si *i* **coopera** y *j* **traiciona**, *i* obtiene la **retribución** de lo **traicionado** de 1, mientras que *j* obtiene 4.
- Si *j* **coopera** e *i* **traiciona**, *j* obtiene la **retribución** de lo **traicionado** de 1, mientras que *i* obtiene 4.
- **Recompensa** por cooperación mútua.

	defect	coop
defect	2 2	1 4
coop	4 1	3 3

Figura 4 : Matriz de retribución dilema del prisionero

¿Qué deben hacer?

- La **acción** racional **individual** es **defect** ya que garantiza un **payoff** no mayor que 2, mientras que **cooperar** garantiza un **payoff** de a lo sumo 1.
- Luego **defect** es la **mejor respuesta** para todas las **estrategias** posibles: si **ambos** agentes hacen **defect**, obtienen **payoff**=2.
- Pero la **intuición** dice que esta **no** es la **mejor** salida: seguramente ellos deben **cooperar** y ambos **obtener** un payoff de **3**.

Concepto de solución

- **No** hay una estrategia **dominante**
- *(D, D)* es el único **equilibrio** de **Nash**
- **Todas** las **salidas**, excepto **(D,D)** son **optimales** de Pareto.
- **(C, C)** maximiza el **bienestar social**

- Esta **paradoja** aparente es el **problema fundamental** de las **interacciones** multiagentes. Parece implicar que la **cooperación** no ocurrirá en **sociedades** de agentes **egoístas**.
- **Ejemplos** del mundo **real**: tratados de **reducción** de **armas** nucleares, etc.
- El **dilema** del **prisionero** es **simétrico**.
- ¿Podemos **recuperar** la **cooperación**?

Argumentos que recupera la cooperación

- De este **análisis** algunos han **concluido**: la **noción** de **racionalidad** de la teoría de juegos es **errónea** y el **dilema** ha sido **formulado** de manera **errónea**.
- **Argumentos** para **recuperar** la **cooperación**: no todos somos **Maquiavelo**; El otro **prisionero** es mi **mellizo**; La **sombra** del **futuro**...

El dilema del prisionero iterado

- Una **respuesta**: **jugar más** de una **vez**. Si se sabe que se **enfrentará** al **oponente** de nuevo, el **incentivo** para hacer defect **desaparecerá**.
- La **cooperación** es la **decisión racional** en el dilema del **prisionero** repetido **infinitamente**.

Inducción para atrás

- **Supongamos** que **ambos saben** que el **dilema** se **juega** exactamente *n* **veces**.
- En la **ronda n-1**, tendrán un **incentivo** para **traicionar**, para ganar el poquito de **recompensa extra**...
- Eso hace la **ronda** *n-2* la **ultima** "real", luego, también habrá un **incentivo** para **traicionar**. Este es el problema de la **inducción** para **atrás**.
- Al jugar el **dilema** del **prisionero** con un número **finito** y predeterminado la **mejor** estrategia es la **traición**.

El torneo de Axelrod

- **Supongamos** que jugamos el **dilema** del **prisionero** iterado en **contra** de un grupo de **diferentes** oponentes.
- ¿Qué **estrategia** deberíamos escoger para **maximizar** la **recompensa** general?
- **Axelrod** (1984) investiga este **problema**, con un torneo por computadoras para programas que **jueguen** el dilema del prisionero.

Estrategias para el torneo

- **ALLD**: always **defect**.
- **TIT-FOR-TAT**:
  En la ronda $u = 0$, **cooperar**.
  En la ronda $u > 0$, hacer lo que hizo el **oponente** en la **ronda** $u - 1$.
- **TESTER**: En la primera ronda hacer **defect**. Si el oponente se **venga**, jugar TIT-FOR-TAT; en caso contrario **intercalar cooperación** y **traición**.
- **JOSS**: Como TIT-FOR-TAT, excepto que se hace **defect periódicamente**.

Recetas para tener éxito en el torneo

Axelrod sugiere las siguientes **reglas**:

- **No** ser **envidioso**: **no** jugar como si fuera **suma-0**.
- Ser **bueno**: **comenzar** cooperando y **devolver** la **cooperación**.
- Vengarse **apropiadamente**: **penalizar** siempre la **traición**, usando una fuerza "**moderada**".
- No **guardar** la **envidia**: siempre **devolver** la **cooperación** de **inmediato**.

Juego de la gallina

- En el **juego**: **frenar** es **cooperar** y seguir **recto** es **traicionar**.
- La **diferencia** con el dilema del prisionero es que la **traición mutua** es la salida **más temida**.

	defect	coop
defect	1 1	2 4
coop	4 2	3 3

Figura 5 : Matriz de retribución problema de la gallina

Conceptos de Solución

- **No** hay una **estrategia dominante**
- Los **pares** de **estrategias** *(C, D)* y *(D, C)* son equilibrios de **Nash**.
- **Todas** las salidas, **excepto** *(D, D)*, son **optimales** de **Pareto**
- **Todas** las salidas, excepto *(D, D)*, **maximizan** el **bienestar** social.

Índice

Otros juegos simétricos de 2x2 I

Dadas las **4 posibles salidas de juegos** (simétricos) de **cooperación/traición**, hay **24** posibles **ordenamientos** de las **salidas**.

Domina la cooperación

$$CC \succ_i CD \succ_i DC \succ_i DD$$

Deadlock. Lo mejor para hacer siempre es traicionar

$$DC \succ_i DD \succ_i CC \succ_i CD$$

Dilema del prisionero

$$DC \succ_i CC \succ_i DD \succ_i CD$$

Otros juegos simétricos de 2x2 II

Gallina

$$DC \succ_i CC \succ_i CD \succ_i DD$$

Stag hunt

$$CC \succ_i DC \succ_i DD \succ_i CD$$

Bibliografía I

- Capítulo 6 del libro: **Introduction to Multiagent Systems**. Michael Woolridge. 2001. Ed. John Wiley & Sons, Inc.

TEMA 7. INTERACCIÓN.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Diciembre de 2015

TEMA 8. MODELOS DE NEGOCIACIÓN.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Diciembre de 2015

Índice

Índice

Objetivos y tipos

Objetivo: determinar (las condiciones de) un **acuerdo** entre, al menos, **dos agentes**.

- **Subastas**: adjudicar **productos** y tareas a través de un "mercado". n **participantes**, pero **transacción** final entre dos.
- **Regateo**: llegar a un **acuerdo** entre todos los **participantes**.
- **Argumentación**: **resolver** (supuestos) **conflictos** a través del **debate**.

Acuerdos

- ¿Cómo se alcanza un **acuerdo** en una sociedad de **agentes individualistas**?
- **Negociación** + **Argumentación** = Acuerdos.
- Los **protocolos** [Rosenschein y Zlotkin, 1994] definen las **reglas** del **encuentro** para todos los participantes en una **negociación**.
- Dado un **protocolo** particular ¿Cómo podemos **diseñar** una **estrategia** que los agentes puedan usar cuando **negocian**?

Propiedades de los protocolos

- **Garantizar** el **éxito**. Se logra un acuerdo.
- **Maximizar** el **bienestar** social. Suma de **utilidades**.
- Eficiencia de **Pareto**. Si un agente **mejora**, otro **empeora**.
- Racionalidad **individual**. No hay de otra.
- **Estabilidad**. Todos reciben **estimulo** (Equilibrio Nash).
- **Simplicidad**. Todos **entienden**.
- **Distribución**. Nada es **centralizado**, comunicación **mínima**.

Índice

Estructura

- **Mecanismo** estructurado para forjar **acuerdos**.
- **Protocolo** semi-distribuido, con **diferentes** roles: 1 **subastador** ofrece un bien y N **subasteros** (licitadores) que desean ese bien.
- El **subastador** desea **maximizar** el **precio** del bien (vía el protocolo) y los **licitadores** desean **minimizarlo** (vía la estrategia).
- **Estrategias**: "**pujas**" de los **subasteros** y precio inicial, etc., del **subastador**
- **No** muy **frecuentes** en la realidad, pero sí bastante **populares** en **Comercio Electrónico** (p.e. eBay)

Valoraciones

- Si el **bien** tiene el mismo **valor** para todos los **licitadores**, decimos que su **valor** es **común**.
- Si el bien tiene diferentes **valores** para diferentes **licitadores**, decimos que su valor es **privado**. Ej: el ultimo **dólar** que se gastó **Michael Jackson**.
- **Correlacionada**. **Combinación** de las **anteriores**. Ej: Marchantes de arte.

Ganador y ofertas

- Determinación del **ganador**: Precio **primero** vs. Precio **segundo**.
- **Conocimiento** de ofertas: **Abiertas** vs. **Selladas** (sobre cerrado).
- **Ofertas**: **Un tiro** vs. Ascendentes / **Descendentes**.

Subasta inglesa

Inicio:

- El **subastador** ofrece un producto a un **precio inicial** (precio de **reserva**: usualmente por **debajo** de un precio **mínimo** privado).

Apuestas:

- Los **agentes** van ofertando **precios** (ninguna, una, o varias veces).
- Cada **oferta** tiene que **superar** todas las **anteriores**.
- El ciclo de **apuestas** termina cuando **no hay más ofertas**.

Adjudicación:

- Si la última **oferta** alcanza el precio **mínimo** (privado) del **subastador**, el **producto** es **adjudicado** al agente de la **oferta** más **alta**
- De lo contrario **no** se **vende** el **producto** (el **subastador** tiene la **última** palabra)

Subasta inglesa entre agentes: Protocolo FIPA

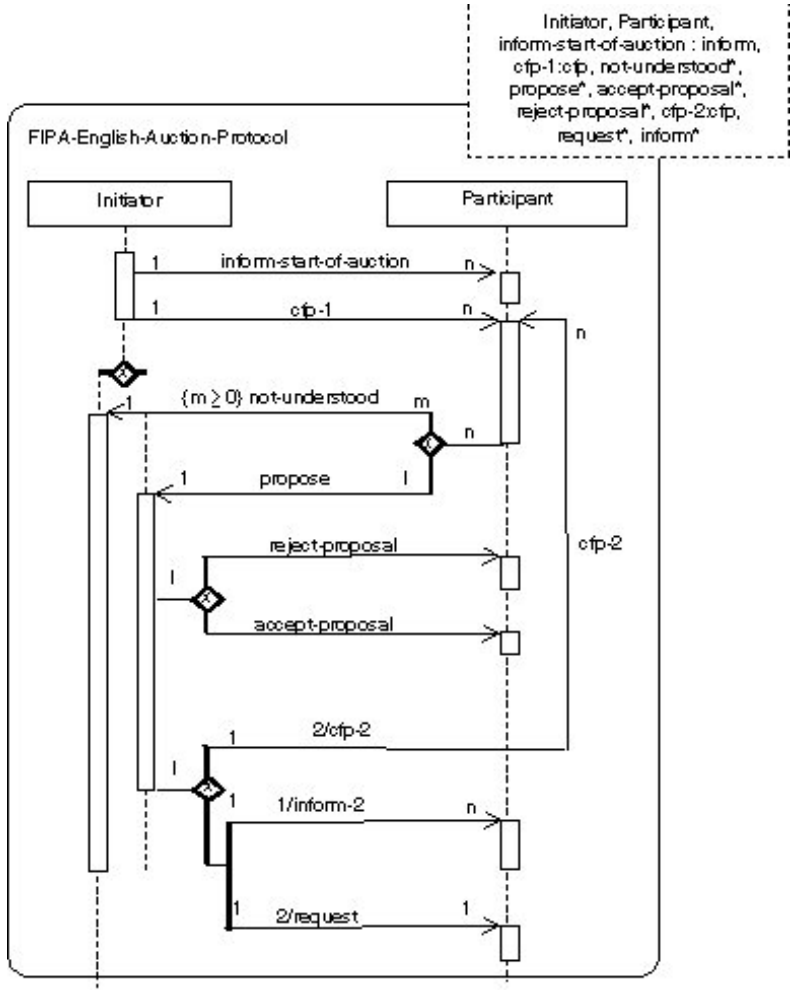


Figura 1 : Esquema UML de subasta inglesa

Subasta inglesa entre agentes: Protocolo FIPA

En los subastas de agentes, los participantes **no** están físicamente **presentes** en una **sala de subasta**:

- **reject-proposal**: pueden llegar **pujas ilegales**, p.ej. por retardos en la red.
- **cfp**: anunciar cada nueva **ronda de pujas** con el precio actual.
- **inform**: informar a todos los **participantes** sobre el **resultado** de la **subasta**.
- **request**: **requerir** que el **ganador** realice la **transacción**.

Estrategia y propiedades

- La **estrategia dominante** parece ser **ofertar sucesivamente pequeñas cantidades** sobre la **oferta actual** hasta que la oferta alcance el **precio** de la **valoración actual** del bien y entonces **retirarse**.
- ¿Qué sucede cuando el **valor** del bien **subastado no** es bien **conocido**?
- ¿Debería estar **contento** el **ganador** por obtener el bien por un **precio** igual o menor al **valor privado**? ¿O debería **preocuparse** de que los demás **licitadores** decidieron **retirarse**?
- Cuando se da el **caso** de que el **ganador sobrevalora** el bien, se habla de la **maldición del ganador**.

Subasta holandesa

Inicio:

- El **subastador** ofrece una **cantidad** de un **producto** a un precio inicial (usualmente por encima de un **precio mínimo privado**).

Apuestas:

- Cada **tiempo** (Dt) **disminuye** el **precio** en una cantidad (D$).
- Cada **oferta** especifica la **cantidad** del producto a comprar al **precio actual**.
- El subastador **determina** el **final** de la **subasta** (o bien porque **toda** la **cantidad** ha sido **adjudicada**, o bien porque se alcanza el **precio mínimo privado**).

Adjudicación:

- La **adjudicación** de cada oferta a los **subasteros** es **directa**.
- El subastador **informa** del **final** de la **subasta**.

Estrategia y propiedades

- **No** hay **estrategia** dominante.
- La **maldición del ganador** también está **presente**.

Subasta holandesa

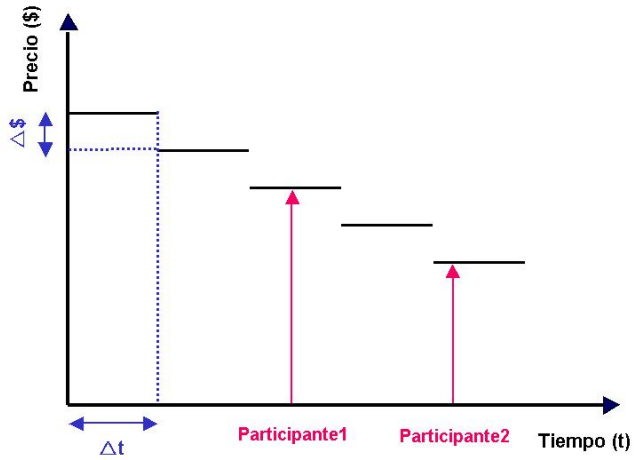


Figura 2 : Subasta holandesa

Subasta holandesa entre agentes: protocolo FIPA

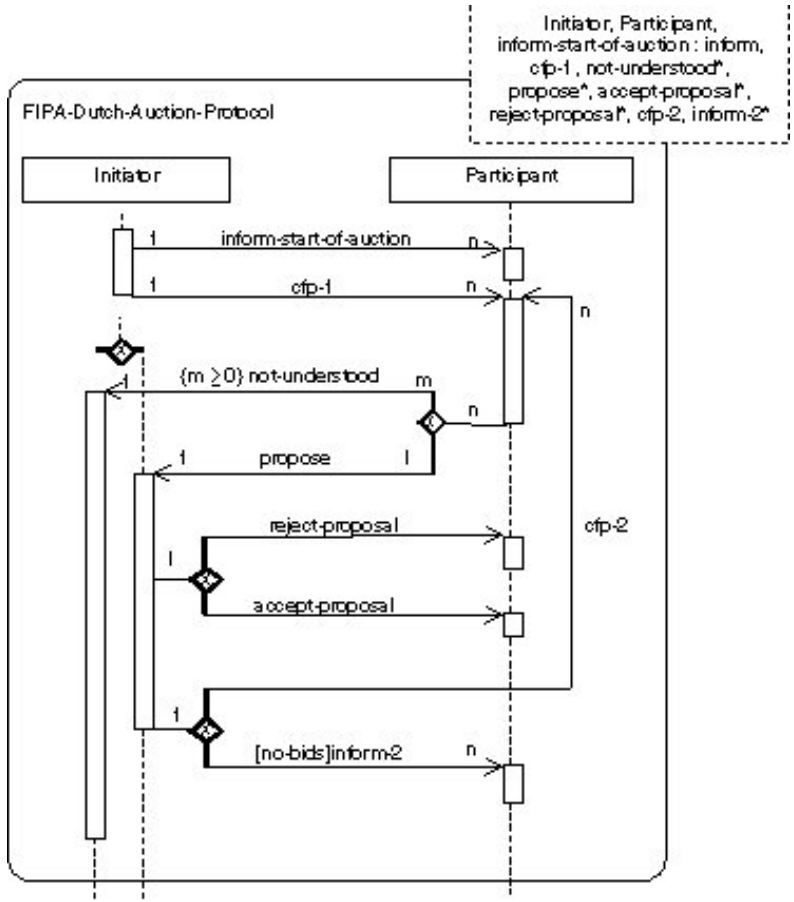


Figura 3 : Esquema UML de Subasta holandesa

Subasta holandesa entre agentes: protocolo FIPA

- **reject-proposal**: si llegan **pujas ilegales** o a la vez.
- **accept-proposal**: ya que la **adjudicación** es **directa**, la aceptación de una **puja** implica la **realización** de la **transacción**.
- **inform**: del **final** de la **subasta**.

Subastas one-shot, sellada

Subasta **primer precio** o a **un tiro**.

- **Sólo** hay una **oportunidad** para hacer **ofertas**.
- **No** hay **rondas** subsecuentes y el bien se **adjudica** al **agente** que haya hecho la **oferta mayor**.
- El **ganador** paga el **precio** de su oferta. **No** hay por tanto, **oportunidad** para que los otros **licitadores**, ofrezcan **ofertas** más altas por el **bien**.
- La **estrategia dominante** es ofrecer **menos** que el **verdadero valor** del bien ¿Qué tanto menos? Depende de lo que **ofrezcan** los otros **agentes**.
- **No** hay **solución** general para este caso.

Subastas de Vickrey

- Es la **menos usada** y quizá la **menos intuitiva** de todas las subastas vistas.
- Este es el **caso** de una **subasta** de **segundo precio**, **sellada**.
- Eso significa que **solo** hay una **ronda** de **negociación**, en la cual cada **licitador** propone una **sola oferta**.
- El bien se **adjudica** al **agente** que hizo la **mayor oferta**, pero **paga** el **precio** de la **segunda mejor oferta**.

Subastas one-shot

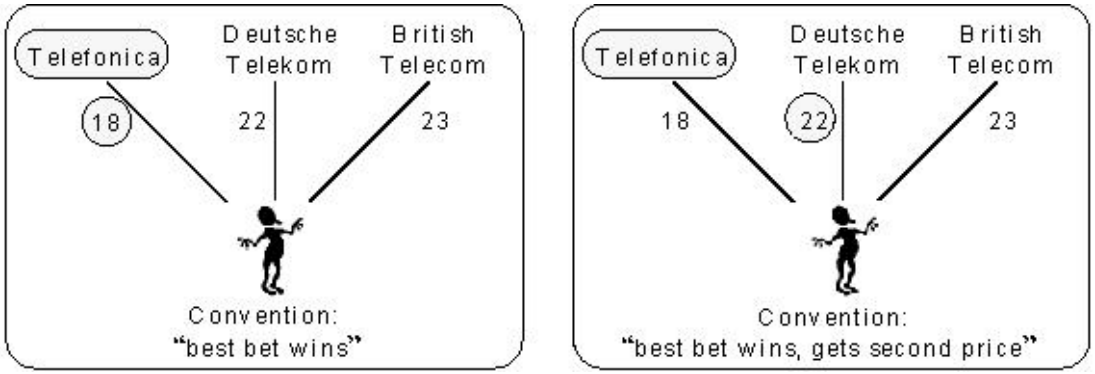
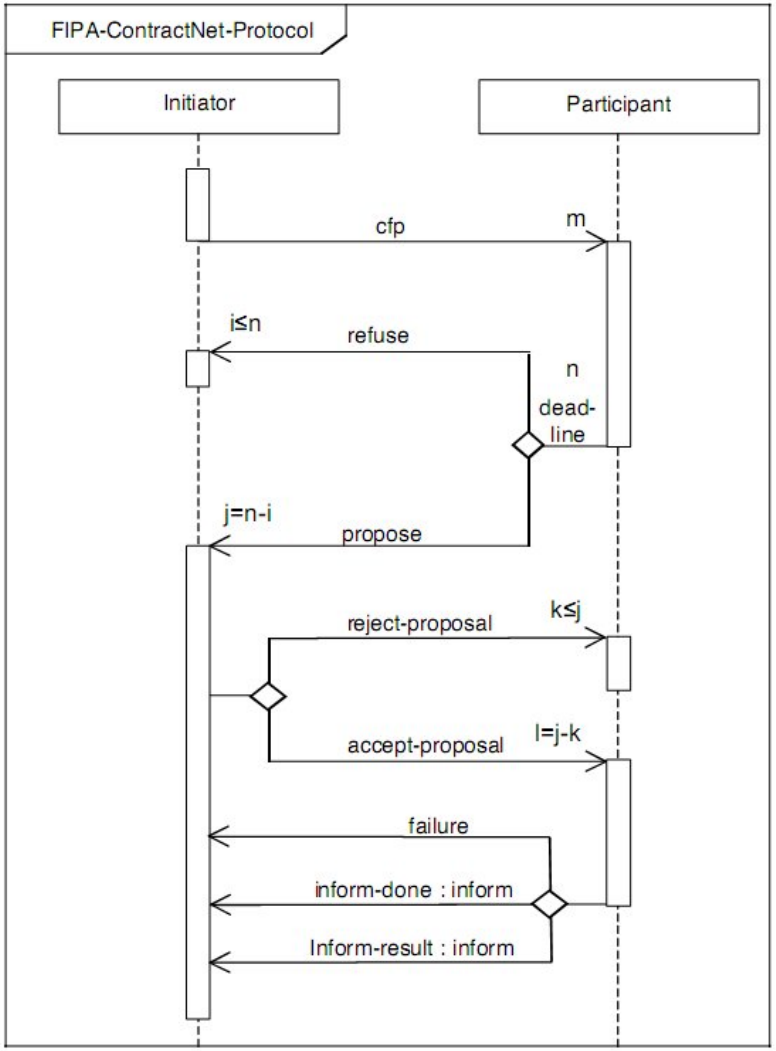


Figura 4 : Subasta a primer precio y a segundo precio

Estrategia y propiedades

- Tienen una verdadera **estrategia dominante**: **ofertar** el **verdadero valor** del bien.
- Supongamos que **ofertamos** más que el **valor real** del bien. En ese caso, el **bien** nos será **adjudicado**, pero se corre el riesgo de adquirir el bien a un **precio superior** a su valor **real**. Si se **gana** en esas circunstancias, la **compra** fue una **pérdida**.
- Supongamos que **ofertamos menos** que el **valor real** del bien. Se **reduce** la **posibilidad** de **ganar** el bien. Pero **aún ganando** en ese caso, haber **ofrecido** menos no tiene efecto pues el **precio a pagar** es el de la **segunda mejor oferta**.

Subastas one-shot entre agentes: protocolo FIPA



Subastas one-shot entre agentes: protocolo FIPA

- **No** existen **protocolos** específicos para **subastas** one-shot en **FIPA**.
- Sin embargo, dichas **subastas** se pueden **"simular"** sobre la base de la **"Red de Contratos"**.
- El protocolo Red de Contratos (**Contract Net**) es un mecanismo **"clásico"** de la **IA Distribuida** para la asignación de **tareas**.

Subastas de venta y de compra

Subastas de venta:

- 1 **vendedor**, n **compradores**.
- Ejemplos: subasta **inglesa** y **holandesa** "tradicionales"

Subastas de compra:

- 1 **comprador**, n **vendedores**.
- **Variaciones** de las **subastas** descritas: subasta **inglesa** de precio **descendente** y subasta **holandesa** de precio **ascendente**.

Tipos de protocolos de subasta I

Tipos de ofertas:

- **Abierto (open-cry)**: los subasteros **conocen mutuamente** sus **ofertas**.
- **Privado/cerrado (sealed-bid)**: los **subasteros** sólo **conocen** sus **propias** ofertas

Proceso de ofertas:

- **Una vuelta (one-shot)**: los subasteros sólo dan una oferta
- **Directa (forward):** el precio de las ofertas va **ascendiendo**
- **Inversa (reverse)**: el precio de las ofertas va **descendiendo**

Proceso de adjudicación:

- ¿Qué **oferta** se usa para **determinar** el **precio** que ha de **pagar** el ganador? (first-price, second-price, ...)

Tipos de protocolos de subasta II

Ejemplos:

- **Subasta inglesa** (tradicional): first-price, open-cry y forward.
- **Subasta holandesa** (tradicional): first-price. open-cry y reverse.
- **Subasta Vickrey**: one-shot, second-price y sealed-bid

Ganancia esperada (subastador)

- Para **licitadores neutrales al riesgo**, la **ganancia** esperada es probablemente **idéntica** en los **cuatro** casos de **subasta** discutidos. Esto es, el **subastador** puede esperar una **ganancia similar** en todos los casos.
- Para **licitadores con aversión al riesgo**, las subastas **holandesa** y de **primer precio, sellada**, llevan a **ganancias** más altas para el **subastador**.
- Para **subastadores con aversión al riesgo**, funcionan mejor las **subastas Vickrey e inglesa**.

Mentiras y colusiones

- **Colusión**. Los **licitadores** pueden **aliarse** para obtener un **precio bajo** del bien. Solución: **modificar** el **protocolo** para que los **licitadores** no se **conozcan** entre ellos, aunque esto **no** es **posible** en **subastas abiertas**.
- **Honestidad del subastador**. En las **subastas Vickrey**, se puede **mentir** acerca del **monto** de la **segunda mejor oferta**, haciendo pagar más de esta forma al ganador. Firmar las ofertas, de forma que el **ganador** pueda **verificar** el resultado; o contratar a un tercero, **confiable**, para **procesar** las ofertas. En subastas **abiertas**, no hay **forma** que el subastador **mienta**.
- **Falsos** licitadores en las **subastas** inglesas.

Ejemplo Subasta inglesa I

Listado 1: Subasta inglesa

```
1
2  Agent container Main-Container@JADE-IMTP://climber1 is ready.
3  englishauction: New good - Roga - on the auction
4  englishauction: Roga for starting price 200
5  englishauction: Nobody wants to buy Roga for starting price 200
6  englishauction: New good - Hvost - on the auction
7  englishauction: Hvost for starting price 20
8  bidder2: I buy Hvost for 20
9  englishauction: Hvost for 20 to bidder2
10 bidder1: I buy Hvost for 30
11 englishauction: Hvost for 30 to bidder1
12 bidder2: I buy Hvost for 40
13 englishauction: Hvost for 40 to bidder2
14 bidder1: I buy Hvost for 50
15 englishauction: Hvost for 50 to bidder1
16 bidder2: I buy Hvost for 60
17 englishauction: Hvost for 60 to bidder2
18 englishauction: Hvost is selled to bidder2 for 60
19 englishauction: New good - Nos kabana - on the auction
20 englishauction: Nos kabana for starting price 20
21 bidder1: I buy Nos kabana for 20
22 englishauction: Nos kabana for 20 to bidder1
23 bidder2: I buy Nos kabana for 30
24 englishauction: Nos kabana for 30 to bidder2
```

Ejemplo Subasta inglesa II

```
25 bidder1: I buy Nos kabana for 40
26 englishauction: Nos kabana for 40 to bidder1
27 bidder2: I buy Nos kabana for 50
28 englishauction: Nos kabana for 50 to bidder2
29 bidder1: I buy Nos kabana for 60
30 englishauction: Nos kabana for 60 to bidder1
31 bidder2: I buy Nos kabana for 70
32 englishauction: Nos kabana for 70 to bidder2
33 bidder1: I buy Nos kabana for 80
34 englishauction: Nos kabana for 80 to bidder1
35 bidder2: I buy Nos kabana for 90
36 englishauction: Nos kabana for 90 to bidder2
37 bidder1: I buy Nos kabana for 100
38 englishauction: Nos kabana for 100 to bidder1
39 englishauction: Nos kabana is selled to bidder1 for 100
40 englishauction: New good - hobot afrikanskogo slona - on the auction
41 englishauction: hobot afrikanskogo slona for starting price 20
42 bidder2: I buy hobot afrikanskogo slona for 20
43 englishauction: hobot afrikanskogo slona for 20 to bidder2
44 englishauction: hobot afrikanskogo slona is selled to bidder2 for 20
45 englishauction: Auction is finished
```

Ejemplo Subasta holandesa I

Listado 2: Subasta holandesa

```
1  Agent container Main-Container@JADE-IMTP://climber1 is ready.
2  dutchauction: Message 'New Good Roga for 300' send to bidder2
3  dutchauction: Message 'New Good Roga for 300' send to bidder1
4  dutchauction: Message 'New price 290 for Good Roga' send to bidder2
5  dutchauction: Message 'New price 290 for Good Roga' send to bidder1
6  dutchauction: Message 'New price 280 for Good Roga' send to bidder2
7  dutchauction: Message 'New price 280 for Good Roga' send to bidder1
8  dutchauction: Message 'New price 270 for Good Roga' send to bidder2
9  dutchauction: Message 'New price 270 for Good Roga' send to bidder1
10 dutchauction: Message 'New price 260 for Good Roga' send to bidder2
11 dutchauction: Message 'New price 260 for Good Roga' send to bidder1
12 dutchauction: Message 'New price 250 for Good Roga' send to bidder2
13 dutchauction: Message 'New price 250 for Good Roga' send to bidder1
14 dutchauction: Message 'New price 240 for Good Roga' send to bidder2
15 dutchauction: Message 'New price 240 for Good Roga' send to bidder1
16 dutchauction: Message 'New price 230 for Good Roga' send to bidder2
17 dutchauction: Message 'New price 230 for Good Roga' send to bidder1
18 dutchauction: Message 'New price 220 for Good Roga' send to bidder2
19 dutchauction: Message 'New price 220 for Good Roga' send to bidder1
20 dutchauction: Message 'New price 210 for Good Roga' send to bidder2
21 dutchauction: Message 'New price 210 for Good Roga' send to bidder1
22 dutchauction: Message 'New price 200 for Good Roga' send to bidder2
23 dutchauction: Message 'New price 200 for Good Roga' send to bidder1
24 bidder2: Message 'I get Roga for 200' send to dutchauction
```

Ejemplo Subasta holandesa II

```
25 dutchauction: Message 'You get: Roga for 200' send to bidder2
26 dutchauction: Message 'Good Roga is selled for 200' send to bidder2
27 dutchauction: Message 'Good Roga is selled for 200' send to bidder1
28 bidder1: Message 'I get Roga for 200' send to dutchauction
29 dutchauction: Message 'New Good Hvost for 500' send to bidder2
30 dutchauction: Message 'New Good Hvost for 500' send to bidder1
31 dutchauction: Message 'New Good Nos kabana for 500' send to bidder2
32 dutchauction: Message 'New Good Nos kabana for 500' send to bidder1
33 dutchauction: Message 'New price 490 for Good Nos kabana' send to bidder2
34 dutchauction: Message 'New price 490 for Good Nos kabana' send to bidder1
35 dutchauction: Message 'New price 480 for Good Nos kabana' send to bidder2
36 dutchauction: Message 'New price 480 for Good Nos kabana' send to bidder1
37 dutchauction: Message 'New price 470 for Good Nos kabana' send to bidder2
38 dutchauction: Message 'New price 470 for Good Nos kabana' send to bidder1
39 dutchauction: Message 'New price 460 for Good Nos kabana' send to bidder2
40 dutchauction: Message 'New price 460 for Good Nos kabana' send to bidder1
41 dutchauction: Message 'New price 450 for Good Nos kabana' send to bidder2
42 dutchauction: Message 'New price 450 for Good Nos kabana' send to bidder1
43 dutchauction: Message 'New price 440 for Good Nos kabana' send to bidder2
44 dutchauction: Message 'New price 440 for Good Nos kabana' send to bidder1
45 dutchauction: Message 'New price 430 for Good Nos kabana' send to bidder2
46 dutchauction: Message 'New price 430 for Good Nos kabana' send to bidder1
47 dutchauction: Message 'New price 420 for Good Nos kabana' send to bidder2
48 dutchauction: Message 'New price 420 for Good Nos kabana' send to bidder1
49 dutchauction: Message 'New price 410 for Good Nos kabana' send to bidder2
50 dutchauction: Message 'New price 410 for Good Nos kabana' send to bidder1
51 dutchauction: Message 'New price 400 for Good Nos kabana' send to bidder2
```

Ejemplo Subasta holandesa III

```
52 | dutchauction: Message 'New price 400 for Good Nos kabana' send to bidder1
53 | dutchauction: Message 'New price 390 for Good Nos kabana' send to bidder2
54 | dutchauction: Message 'New price 390 for Good Nos kabana' send to bidder1
55 | dutchauction: Message 'New price 380 for Good Nos kabana' send to bidder2
56 | dutchauction: Message 'New price 380 for Good Nos kabana' send to bidder1
57 | dutchauction: Message 'New price 370 for Good Nos kabana' send to bidder2
58 | dutchauction: Message 'New price 370 for Good Nos kabana' send to bidder1
59 | dutchauction: Message 'New price 360 for Good Nos kabana' send to bidder2
60 | dutchauction: Message 'New price 360 for Good Nos kabana' send to bidder1
61 | dutchauction: Message 'New price 350 for Good Nos kabana' send to bidder2
62 | dutchauction: Message 'New price 350 for Good Nos kabana' send to bidder1
63 | dutchauction: Message 'New price 340 for Good Nos kabana' send to bidder2
64 | dutchauction: Message 'New price 340 for Good Nos kabana' send to bidder1
65 | dutchauction: Message 'New price 330 for Good Nos kabana' send to bidder2
66 | dutchauction: Message 'New price 330 for Good Nos kabana' send to bidder1
67 | dutchauction: Message 'New price 320 for Good Nos kabana' send to bidder2
68 | dutchauction: Message 'New price 320 for Good Nos kabana' send to bidder1
69 | dutchauction: Message 'New price 310 for Good Nos kabana' send to bidder2
70 | dutchauction: Message 'New price 310 for Good Nos kabana' send to bidder1
71 | dutchauction: Message 'New price 300 for Good Nos kabana' send to bidder2
72 | dutchauction: Message 'New price 300 for Good Nos kabana' send to bidder1
73 | bidder1: Message 'I get Nos kabana for 300' send to dutchauction
74 | dutchauction: Message 'You get: Nos kabana for 300' send to bidder1
75 | dutchauction: Message 'Good Nos kabana is selled for 300' send to bidder2
76 | dutchauction: Message 'Good Nos kabana is selled for 300' send to bidder1
77 | dutchauction: Message 'New Good hobot afrikanskogo slona for 500' send to bidder2
78 | dutchauction: Message 'New Good hobot afrikanskogo slona for 500' send to bidder1
```

Ejemplo Subasta holandesa IV

```
 79 | dutchauction: Message 'New price 490 for Good hobot afrikanskogo slona' send to bidder2
 80 | dutchauction: Message 'New price 490 for Good hobot afrikanskogo slona' send to bidder1
 81 | dutchauction: Message 'New price 480 for Good hobot afrikanskogo slona' send to bidder2
 82 | dutchauction: Message 'New price 480 for Good hobot afrikanskogo slona' send to bidder1
 83 | dutchauction: Message 'New price 470 for Good hobot afrikanskogo slona' send to bidder2
 84 | dutchauction: Message 'New price 470 for Good hobot afrikanskogo slona' send to bidder1
 85 | dutchauction: Message 'New price 460 for Good hobot afrikanskogo slona' send to bidder2
 86 | dutchauction: Message 'New price 460 for Good hobot afrikanskogo slona' send to bidder1
 87 | dutchauction: Message 'New price 450 for Good hobot afrikanskogo slona' send to bidder2
 88 | dutchauction: Message 'New price 450 for Good hobot afrikanskogo slona' send to bidder1
 89 | dutchauction: Message 'New price 440 for Good hobot afrikanskogo slona' send to bidder2
 90 | dutchauction: Message 'New price 440 for Good hobot afrikanskogo slona' send to bidder1
 91 | dutchauction: Message 'New price 430 for Good hobot afrikanskogo slona' send to bidder2
 92 | dutchauction: Message 'New price 430 for Good hobot afrikanskogo slona' send to bidder1
 93 | dutchauction: Message 'New price 420 for Good hobot afrikanskogo slona' send to bidder2
 94 | dutchauction: Message 'New price 420 for Good hobot afrikanskogo slona' send to bidder1
 95 | dutchauction: Message 'New price 410 for Good hobot afrikanskogo slona' send to bidder2
 96 | dutchauction: Message 'New price 410 for Good hobot afrikanskogo slona' send to bidder1
 97 | dutchauction: Message 'New price 400 for Good hobot afrikanskogo slona' send to bidder2
 98 | dutchauction: Message 'New price 400 for Good hobot afrikanskogo slona' send to bidder1
 99 | dutchauction: Message 'New price 390 for Good hobot afrikanskogo slona' send to bidder2
100 | dutchauction: Message 'New price 390 for Good hobot afrikanskogo slona' send to bidder1
101 | dutchauction: Message 'New price 380 for Good hobot afrikanskogo slona' send to bidder2
102 | dutchauction: Message 'New price 380 for Good hobot afrikanskogo slona' send to bidder1
103 | dutchauction: Message 'New price 370 for Good hobot afrikanskogo slona' send to bidder2
104 | dutchauction: Message 'New price 370 for Good hobot afrikanskogo slona' send to bidder1
105 | dutchauction: Message 'New price 360 for Good hobot afrikanskogo slona' send to bidder2
```

Ejemplo Subasta holandesa V

```
106 dutchauction: Message 'New price 360 for Good hobot afrikanskogo slona' send to bidder1
107 dutchauction: Message 'New price 350 for Good hobot afrikanskogo slona' send to bidder2
108 dutchauction: Message 'New price 350 for Good hobot afrikanskogo slona' send to bidder1
109 dutchauction: Message 'New price 340 for Good hobot afrikanskogo slona' send to bidder2
110 dutchauction: Message 'New price 340 for Good hobot afrikanskogo slona' send to bidder1
111 dutchauction: Message 'New price 330 for Good hobot afrikanskogo slona' send to bidder2
112 dutchauction: Message 'New price 330 for Good hobot afrikanskogo slona' send to bidder1
113 dutchauction: Message 'New price 320 for Good hobot afrikanskogo slona' send to bidder2
114 dutchauction: Message 'New price 320 for Good hobot afrikanskogo slona' send to bidder1
115 dutchauction: Message 'New price 310 for Good hobot afrikanskogo slona' send to bidder2
116 dutchauction: Message 'New price 310 for Good hobot afrikanskogo slona' send to bidder1
117 dutchauction: Message 'New price 300 for Good hobot afrikanskogo slona' send to bidder2
118 dutchauction: Message 'New price 300 for Good hobot afrikanskogo slona' send to bidder1
119 bidder2: Message 'I get hobot afrikanskogo slona for 300' send to dutchauction
120 dutchauction: Message 'You get: hobot afrikanskogo slona for 300' send to bidder2
121 dutchauction: Message 'Good hobot afrikanskogo slona is selled for 300' send to bidder2
122 dutchauction: Message 'Good hobot afrikanskogo slona is selled for 300' send to bidder1
123 dutchauction: Auction ended
```

Índice

Características y escenarios

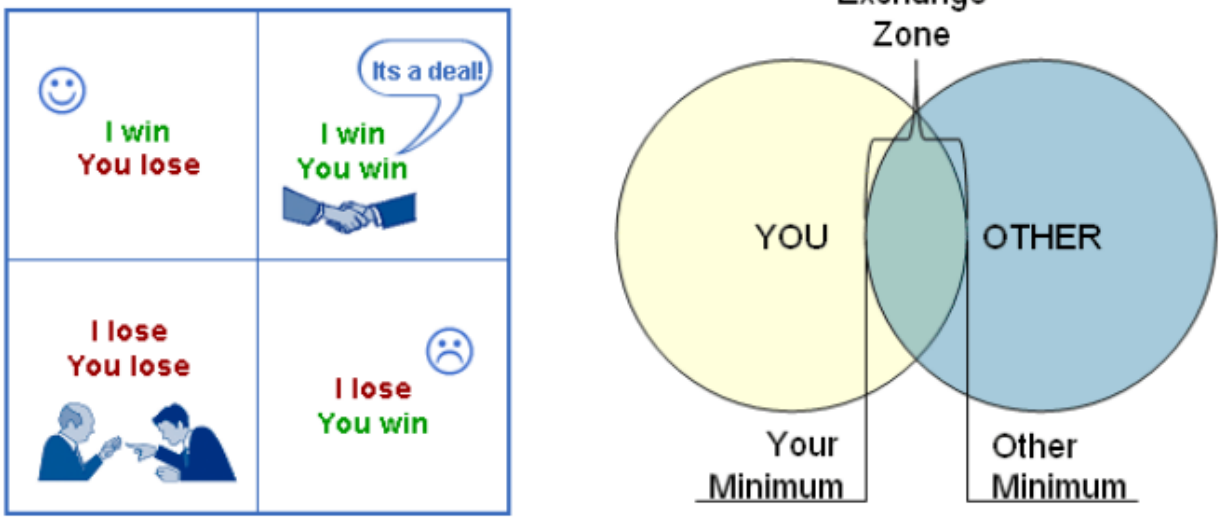


Figura 6 : Concepto de negociación

Características

- Las **subastas** están vinculadas únicamente a la **colocación** de **bienes**: se necesitan de **técnicas** más **completas** para alcanzar **acuerdos**.
- Posibilidad de **forjar** acuerdos **globales** ("creíbles") entre n **agentes**.
- Todos los **agentes** pueden **beneficiarse** de un **acuerdo**.
- Pero hay una **diferencia** de **opinión** con respecto a las **características** del **acuerdo** (qué acuerdo elegir).

Escenario

- **Conjunto (espacio) de negociación**: **todos** los **posibles acuerdos** a los que se pueden **llegar**. Ejemplo: todos los **precios** entre las **expectativas iniciales** de un **comprador** y un **vendedor**
- **Protocolo de negociación**: **reglas** que **determinan** el proceso de **negociación**. **¿Cómo**, **cuándo**, y **qué** ofertas se pueden hacer? **¿Cuándo** termina la **negociación** y cuál es el **resultado**? Ejemplo: "**No** se puede *empeorar* una **oferta** ya **hecha**".
- **Estrategia de negociación**: Cómo **elegir** entre las **diferentes** acciones que **permite** el protocolo Ejemplo: **"Mejorar** mi última **oferta** en un 10 % cada **5 minutos**".

Objetivos de diseño

Diseño de estrategias.

- **Racionalidad**: maximizar las ganancias esperadas.
- **Eficiencia**: minimizar el coste computacional para determinar una acción

Diseño de protocolos.

- **Distribución**: evitar que haya un "cuello de botella".
- **Convergencia**: garantizar que se llega a un acuerdo (o desacuerdo) en tiempo finito.
- **Simplicidad**: fomentar que se llegue rápido a un acuerdo (o desacuerdo).
- **Eficiencia**: si se llega a un acuerdo, este no "desperdicia" utilidad.
- **Estabilidad**: motivar a los agentes para elegir estrategias con características deseadas (est. dominantes, eq. Nash).

Oferta y contraoferta

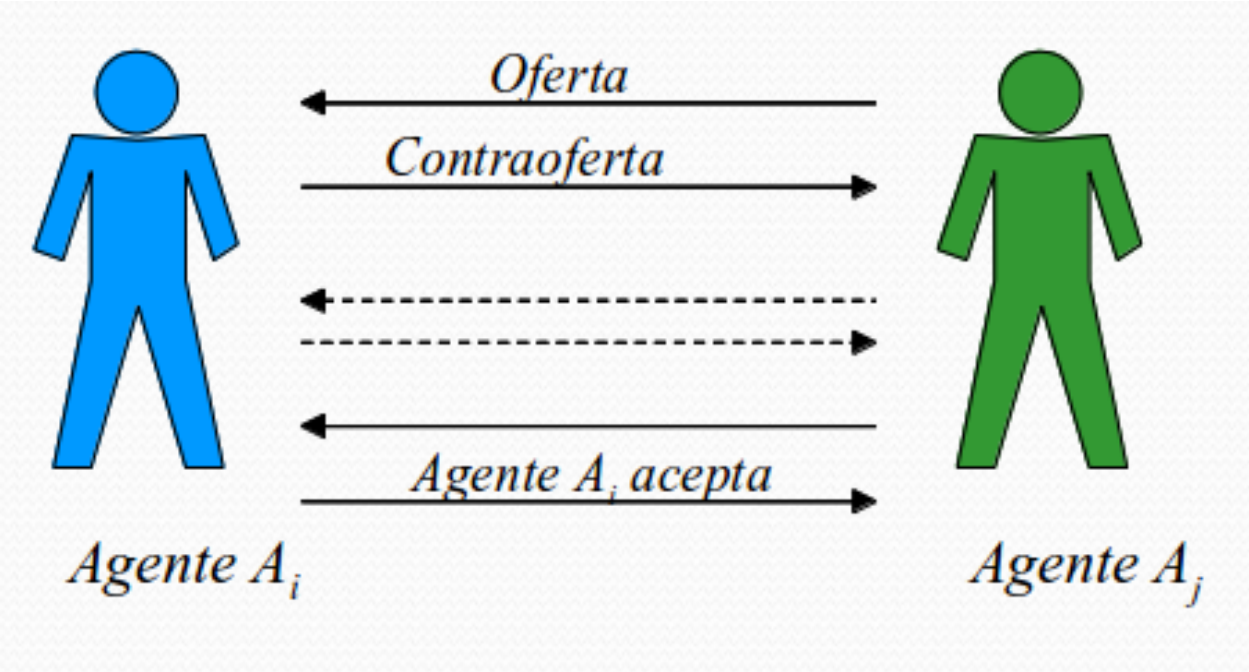


Figura 7 : Negociación como proceso de oferta y contraoferta

Oferta y contraoferta

- Que existan **reglas** vinculadas al **protocolo**, **no** quiere decir que **siempre** se tenga que llegar a un **acuerdo**.
- Los **agentes** pueden **continuamente rechazar** ofertas.
- Si **no** se **alcanza** un **acuerdo**, se dice que el resultado es el **acuerdo conflicto** (Θ).
- Se **realizan** las **siguientes suposiciones**:
  1. El **desacuerdo** es el **peor resultado**. Ambos agentes **prefieren** cualquier **acuerdo** a ninguno.
  2. Los agentes buscan **maximizar** la **utilidad**.
- Con este **modelo básico**, se llegan a resultados "**extraños**".

Oferta y contraoferta: división del pastel.

- Este **problema** se **modela** como un **recurso** con **valor 1**, que puede ser dividido en dos partes.
- Cada **parte** está **entre** 0 y 1.
- La **suma** de las dos **partes** es **1**.
- El **conjunto** de posibles **acuerdos** es:

$$\{(x, 1-x) : 0 \leq x \leq 1\}$$

- Si eres el **Agente 1**, ¿Qué **acuerdo** ofrecerías?

Oferta y contraoferta: división del pastel.

- Se **asume** que **sólo** hay una **ronda** de **negociación**, **juego** del **ultimátum**.
- El **Agente 1** tiene **todo** el **poder** de **decisión**.
- Si el **Agente 1** propone (1, 0), es una **opción mejor** que el **acuerdo conflicto** para el **Agente 2**.
- El **agente no** puede tener **mejor opción** que ésta. Tenemos el **equilibrio** de **Nash**.

Oferta y contraoferta: división del pastel.

- Si **existen dos rondas**, ahora **Agente 2** tiene todo el **poder** de **decisión**.
- **Cualquier** cosa que **proponga** el **Agente 2**, será **rechazado** por el **Agente 1**.
- El **Agente 2**, propone **(0, 1)**.
- Como antes, esto es **mejor** para el **Agente 1** que el **acuerdo conflicto** y por tanto se **acepta**.
- Esto va a **ocurrir** cada vez que haya un **número fijo de rondas**.

Oferta y contraoferta: división del pastel.

- ¿Qué **ocurre** en el caso de que haya **infinitas rondas** de **negociación**?
- El **Agente 1**, utiliza siempre la **estrategia** de **proponer (1, 0)** y siempre se **rechazará** cualquier **oferta** del Agente 2.
- ¿Cómo **responderá** el **Agente 2**?

Oferta y contraoferta: división del pastel.

- Si el **Agente 2**, **rechaza** la **oferta**, **nunca** habrá **acuerdo**. Se **finaliza** por tanto en el **acuerdo conflicto**.
- Por lo tanto, el **Agente 2** debería **aceptar**.
- **No** tiene **sentido** no aceptar en la **primera ronda**.
- De hecho, para **cualquier** (x, 1 - x) que el **Agente 1** proponga, la **inmediata aceptación** supone el **equilibrio de Nash** una vez que el **Agente 2**, conoce la **estrategia** a seguir por el **Agente 1**.
- Existe un **número infinito** de **equilibrios** de **Nash** (debilidad).
- Todos son optimales de Pareto.
- PROBLEMA DE LOS **JUGADORES IMPACIENTES**.
- BOULWARE, CONCEDER (**HEURÍSTICAS**)

TREX en el juego del ultimátum

Figura 8 : T.REX on the ultimatum game

Concesiones Mutuas

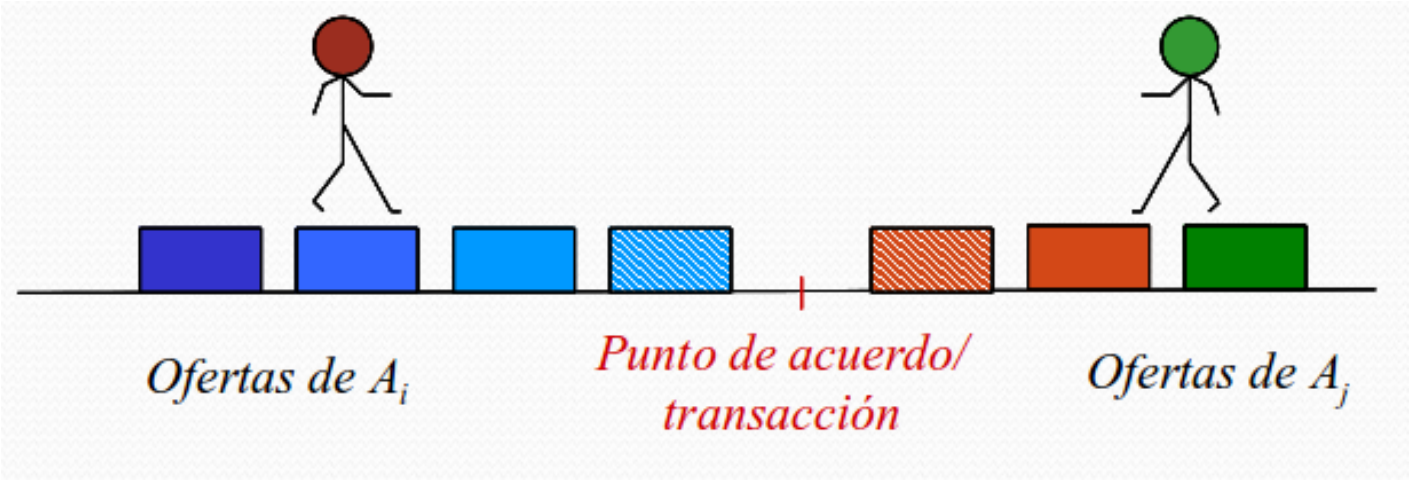


Figura 9 : Regateo como proceso de concesiones mutuas

Protocolo de concesiones monótonas (PCM)

- La **negociación** se realiza por **rondas**.
- En la **ronda 1**, cada **agente** propone **simultáneamente** un **trato** del conjunto de **negociación**.
- Se llega a un **acuerdo**, si un agente **considera** que el **trato propuesto** por el otro es al menos **tan bueno** (para él) como el **suyo**.
- Si **no** hay **acuerdo**, se realiza una **nueva** ronda de **propuestas**. En la ronda $u+1$, ningún agente **puede** realizar una **propuesta peor** que en la ronda u.
- Si **ningún** agente **cede**, la **negociación** termina en **desacuerdo**.
- **Estrategias**: ¿Con qué propuesta **empezar**? ¿**Cuándo** (en qué ronda) hay que **ceder**? ¿**Cuánto** hay que **ceder**?

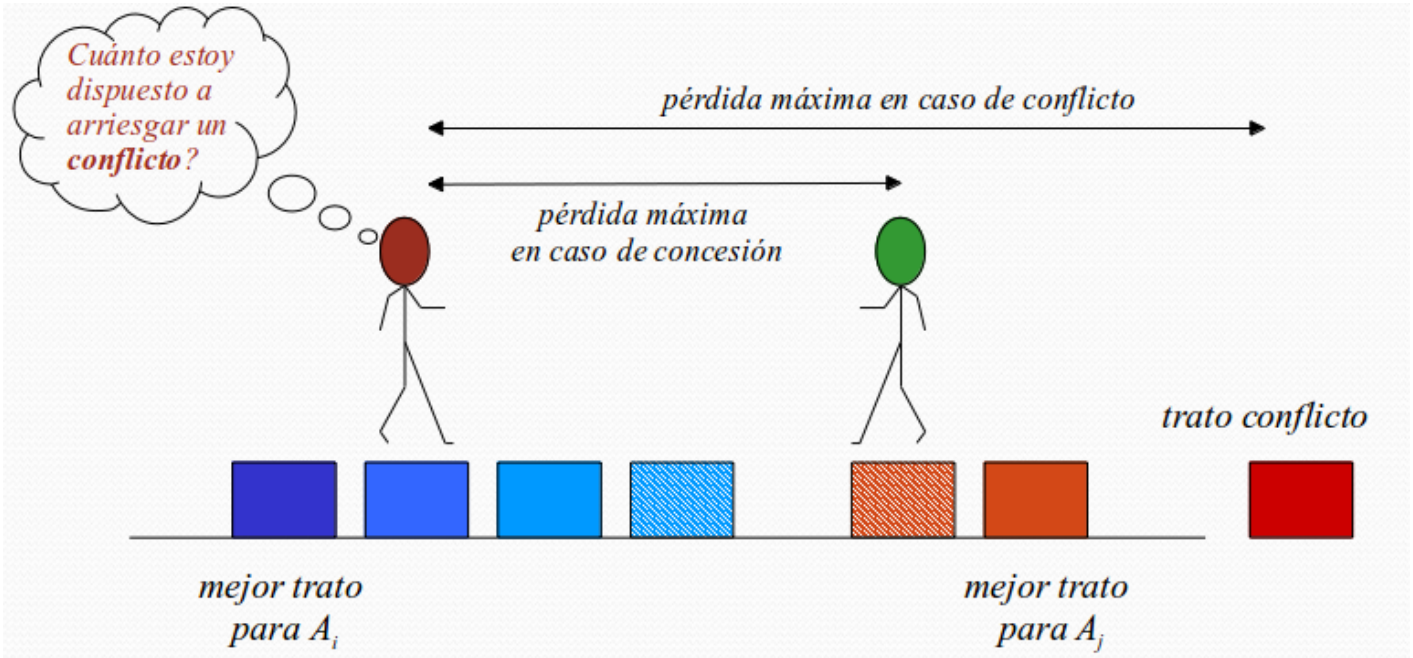
La propuesta inicial

- Leo Baekeland sold the rights to his invention, Velox photographic printing paper, to Eastman Kodak in 1899. It was the first commercially successful photographic paper and he sold it to Eastman Kodak for $1 million. Baekeland had planned to ask $50.000 and to go down to $25.000 if necessary, but fortunately for him, Eastman spoke first. (Asimov, 1982)

El factor riesgo

Idea de estrategia:

- **Empezar** con el **trato** más **favorable** para uno mismo.
- **Determinar** cuándo (y cuánto) **ceder** dependiendo de **cuánto** se puede **perder** en caso de **conflicto** (riesgo).



La estrategia de Zeuthen

Disposición para arriesgar en el conflicto:

- **Riesgo(A,t)** : pérdida **relativa** máxima si el **agente** A **cede** en la ronda t.

$$\frac{\textit{Perdida Maxima de A si cede (y acepta la oferta de B)}}{\textit{Perdida Maxima de A (si no cede y se llega a un conflicto)}}$$

- **Idea**: el **agente** con el **menor riesgo** (el que realizaría la **menor** perdida **relativa** máxima) debería **ceder**.

La estrategia de Zeuthen

- Calcular el propio **Riesgo(A,t)** y el del contrario (**Riesgo(B,t)**).
- Si el propio **riesgo** es igual o más **pequeño** que el del **contrario**, entonces hacer la **oferta mínima suficiente**.
- **Suficiente**: **cambia** la **balanza** de **riesgos** (después el **contrario** tiene el **menor riesgo**).
- **Mínima**: elegir la **oferta** que minimice la propia **pérdida** de **utilidad**.
- De lo **contrario**, **no ceder** (repetir la misma oferta).

Zeuthen: propiedades positivas

- **Distribución**: si. No se requiere un "**árbitro**" centralizado.
- **Convergencia**: si. Se llega a un **acuerdo** o **desacuerdo** en un número **finito** de **rondas**.
- **Eficiencia**: si. **Ninguna** oferta es **dominada** por otra (sólo se eligen del **conjunto** de **negociación**).

Zeuthen: propiedades negativas

- **Simplicidad**: no. Se **exploran** casi todos las **posibles** ofertas del **conjunto** de **negociación**.
- **Estabilidad**: no. Si un **agente** sabe que su **contrario** juega la **estrategia** de **Zeuthen**, puede **aprovecharse** de ello. Aunque **este caso** sólo se da en la **última ronda**.

La estrategia de Zeuthen

- Si no se está en la **última ronda**, y el **agente A** juega la estrategia de **Zeuthen**, la mejor **opción** para el **agente B** es **jugarla** también:
  1. $Riesgo(A, t) > Riesgo(B, t)$ y B no cede: se llega a un **conflicto** y, por tanto, saca **beneficio mínimo**.
  2. $Riesgo(A, t) \geq Riesgo(B, t)$ y B no cede, o hace una **concesión insuficiente**. Agente A no hará **concesiones** hasta que B ceda más. De lo contrario, se llega a un **conflicto** (situación a).
  3. $Riesgo(A, t) \geq Riesgo(B, t)$ y B hace una **concesión suficiente** pero **no mínima**. Agente B podría haber **obtenido** más **cediendo menos** (con la **concesión mínima tampoco** se habría llegado a un **conflicto**)
  4. $Riesgo(A, t) < Riesgo(B, t)$ y B hace una concesión. Agente B podría haber **obtenido** si no hubiera **cedido** menos (porque A cede en todo caso)

La estrategia de Zeuthen

- Si se está en la **última ronda**, y los niveles de **riesgo** de A y B son **iguales**, B puede aprovecharse del hecho de que sabe que A juega la estrategia de **Zeuthen**, y **no ceder**.
- **Solución**: en la estrategia extendida de Zeuthen, si se da el anterior caso, se tira una **moneda para determinar quién ha de ceder**.

Ejemplos I

Imagine that you have three children, each of whom needs to be delivered to a different school each morning. Your neighbour has four children, and also needs to take them to school. Delivery of each child can be modelled as an indivisible task. You and your neighbour can discuss the situation, and come to an agreement that it is better for both of you (for example, by carrying the other's child to a shared destination, saving him the trip). There is no concern about being able to achieve your task by yourself. The worst that can happen is that you and your neighbour won't come to an agreement about setting up a car pool, in which case you are no worse off than if you were alone.

You can only benefit (or do no worse) from your neighbour's tasks. Assume, though, that one of my children and one of my neigbours's children both go to the same school (that is, the cost

Ejemplos II

of carrying out these two deliveries, or two tasks, is the same as the cost of carrying out one of them). It obviously makes sense for both children to be taken together, and only my neighbour or I will need to make the trip to carry out both tasks.

Ejemplos

- Un **grupo** de **agentes** puede **redistribuir** tareas entre sí (sin efectos secundarios). Pueden **beneficiarse** si llegan a un **acuerdo**, pero cada uno **prefiere** un **acuerdo diferente**.
- **Repartición de correo**: Varios **repartidores** de correo han de **entregar cartas** en diferentes **partes** de la **ciudad**. Cada **repartidor** quiere **minimizar** el **camino** que tiene que recorrer, y una forma de hacerlo es **intercambiar** cartas con sus **compañeros**.
- **Consultas en Bases de Datos**: varios **agentes** tienen **acceso** a una Base de Datos común, y cada uno ha de **realizar** una serie de **consultas**. Podrán **coordinar** sus (sub-)queries para **maximizar** la **eficiencia** de sus consultas (Join, Proyección, Unión, Intersección, ...).

Definición

- Un **dominio orientado a tareas (DOT)** es una tripleta *(T, Ag, c)* tal que:
- T es un conjunto finito **tareas**.
- $Ag = \{A_1, A_2, ..., A_n\}$ es un conjunto finito de **agentes**.
- $c : \wp(T) \rightarrow R^+$, $c(\emptyset) = 0$, es una **función monótona creciente** que define el **coste** para **ejecutar** cualquier **subconjunto** de **tareas**.
- Quedarse **quieto** no cuesta **nada** ($c(\emptyset) = 0$), cuantas más tareas se **ejecutan**, más **coste** se **genera** (c es monótona creciente) y el coste de **ejecutar** cada **subconjunto** de tareas no **depende** de quién las **lleva** a cabo (**situación idealizada**).

Utilidad de tratos

Dado un **DOT** con dos **agentes** $(T, \{A_1, A_2\}, c)$:

- Un **encuentro** dentro del **DOT** es un **vector** (T_1, T_2) tal que para todo k, $T_k \subseteq T$.
- Un trato $d = (D_1, D_2)$ en un encuentro (T_1, T_2) es una **redistribución** de **tareas** entre **agentes**, tal que $D1 \bigcup D2 = T1 \bigcup T2$
- El trato $\Theta = (T_1, T_2)$ se llama trato **conflicto** y modela que no hay **acuerdo** entre agentes (**autonomía**).

Ejemplo.

- dominio: $(\{a, b\}, \{1, 2\}, c)$
- encuentro: $(\{a\}, \{a, b\})$ función de coste c:
- $c(\emptyset) = 0$
- $c(\{a\}) = 1$
- $c(\{b\}) = 1$
- $c(\{a, b)\} = 3$

- Posibles tratos:
- $(\{a\}, \{b\})$
- $(\{b\}, \{a\})$
- $(\{a, b\}, \emptyset)$
- $(\emptyset, \{a, b\})$
- $(\{a\}, \{a, b\})$ $\rightarrow$ Trato conflicto
- $(\{b\}, \{a, b\})$
- $(\{a, b\}, \{a\})$
- $(\{a, b\}, \{b\})$
- $(\{a, b\}, \{a, b\})$

Índice

Introducción

- Muchas veces un **conflicto** de **interés** es sólo **aparente** porque a un agente le **falta información** o porque a un agente **no llegar** a una **conclusión** cierta ya que ha sacado una conclusión **equivocada** de su **información**
- La **argumentación** es el proceso de intentar **convencer** a los **demás** de la **veracidad** de un **hecho**.

Modos de argumentación

- **Modo lógico**: "Si **aceptas** A y que A **implica** B, entonces debes **aceptar** B"
- **Modo emocional**: "¿Cómo te **sentirías** si eso te **sucediera** a ti?"
- **Modo visceral**: "¡Cretino!"
- **Modo "Kisceral"**: ¡Esto es un **dogma** cristiano!

Modelo lógico de argumentación

Definición: un argumento $\Delta \vdash$ (Sentencia, Razones) consta de tres partes:

- Δ es un conjunto de **fórmulas** lógicas (probablemente inconsistente).
- **Sentencia** es una formula lógica conocida como **conclusión**.
- **Razones** es un **conjunto** de formulas lógicas tal que: Razones $\subseteq \Delta$; y **Sentencia** se puede derivar de **Razones**.

Modelo lógico de argumentación

Definición: Sean (f_1, G_1) y (f_2, G_2) **argumentos apoyados** en alguna Δ, entonces:

- (f_2, G_2) puede ser **rechazado** (atacado) de dos formas:
  1. (f_1, G_1) **refuta** (rebuts) (f_2, G_2) si $f_1 \rightarrow \neg f_2$
  2. (f_1, G_1) **mina** (undercuts) $(f2, G2)$ si $f1 \rightarrow \neg y2$ para algún $y \in G_2$

  Los dos casos se **resumen** bajo el término **ataque**..

Sistemas de diálogo

- Un **diálogo** entre dos agentes es una serie de **argumentos intercalados**: el agente A intenta **convencer** al agente B de la **conclusión** de su primer **argumento**. El agente B se **defiende**, rebatiéndolo o rebajándolo.
- Cada **paso** del **diálogo** se denomina **movimiento**.
- Formalmente, un **diálogo** es una secuencia **finita** y no vacía de **movimientos** $(m_0, m_1, ..., m_k)$.
- Un diálogo **termina** cuando **no** hay más movimientos **posibles**.

Tipos de Diálogo

	Tipo	Inicio	Meta	Ánimo
I.	**Persuasión**	conflicto de opiniones	resolver un asunto	convencer al otro
II.	**Negociación**	conflicto de intereses	hacer un trato	obtener lo mejor
III.	**Pregunta**	ignorancia general	aumentar el conocimiento	buscar una prueba
IV.	**Deliberación**	necesidad de una acción	alcanzar una decisión	influir
V.	**Extracción de información**	ignorancia personal	mejorar el conocimiento	aumentar el conocimiento personal
VI.	**"Eristics"**	conflicto / antagonismo	buscar una acomodación	
VII	**Mixto**	varios	varios	varios

Figura 10 : Tipos de diálogo

Otros mecanismos de argumentación

- **Protocolos de Votación** (voting mechanisms): cada agente tiene una relación de **preferencia** sobre un conjunto de posibles **acuerdos**. Se elige una **alternativa** según un **protocolo** de **votación**.
- **Formación de coaliciones**: agentes pueden formar coaliciones (**subgrupos**) y **cooperan** con los **miembros** de su **coalición**, y **compiten** con los demás **cuestiones**: ¿Cuál es la **estructura** de coaliciones?, ¿Cuál es el **valor** de cada coalición?, ¿Cómo se **reparte** el valor de una **coalición** entre sus **miembros**?
- **Mecanismos de mercado**: búsqueda **distribuida** para el equilibrio **general** de un mercado.

Bibliografía I

- Capítulo 7 del libro: **Introduction to Multiagent Systems**. Michael Woolridge. 2001. Ed. John Wiley & Sons, Inc.
- Sistemas Multiagente. Sesión 2: **Coordinación - Negociación y Argumentación**. Curso 2010/2011. Ramón Hermoso Traba.

TEMA 8. MODELOS DE NEGOCIACIÓN.

Sistemas Multiagentes

Universidad de Castilla-La Mancha

Diciembre de 2015