

Tema 1.2:

Arquitecturas de

Agente.

Authors: Vicent Botti, Vicente Julián

Tema 2: Agentes inteligentes y sistemas multiagente

- **Agentes: Modelos. Arquitecturas.**
- Sistemas Multiagente: Capacidad Social.

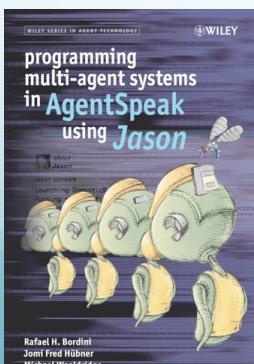
Bibliografía



Agentes software y sistemas multi-agente
Juan Pavón / José L. Pérez
2004, 352p.

Agreement Technologies

Series: Law, Governance and Technology Series, Vol. 8
Ossowski, Sascha (Ed.)
2013, XXXV, 645 p. 133 illus.
Springer Verlag
eBook ISBN 978-94-007-5583-3
Hardcover ISBN 978-94-007-5582-6



Programming Multi-Agent Systems in AgentSpeak using Jason
Rafael H. Bordini, Jomi Fred Hübner, Michael Wooldridge
2007
Wiley Series in Agent Technology
ISBN: 978-0-470-02900-8

Definición de Agente



Agentes software

¿Qué son?

¿Por qué otro paradigma?

¿Otra moda tecnológica?

¿Qué hay de nuevo?

¿Cómo se construyen?

¿Por dónde empezar?

¿Hasta dónde podemos llegar?

¿Nuestro *áller-ego* en la Red?

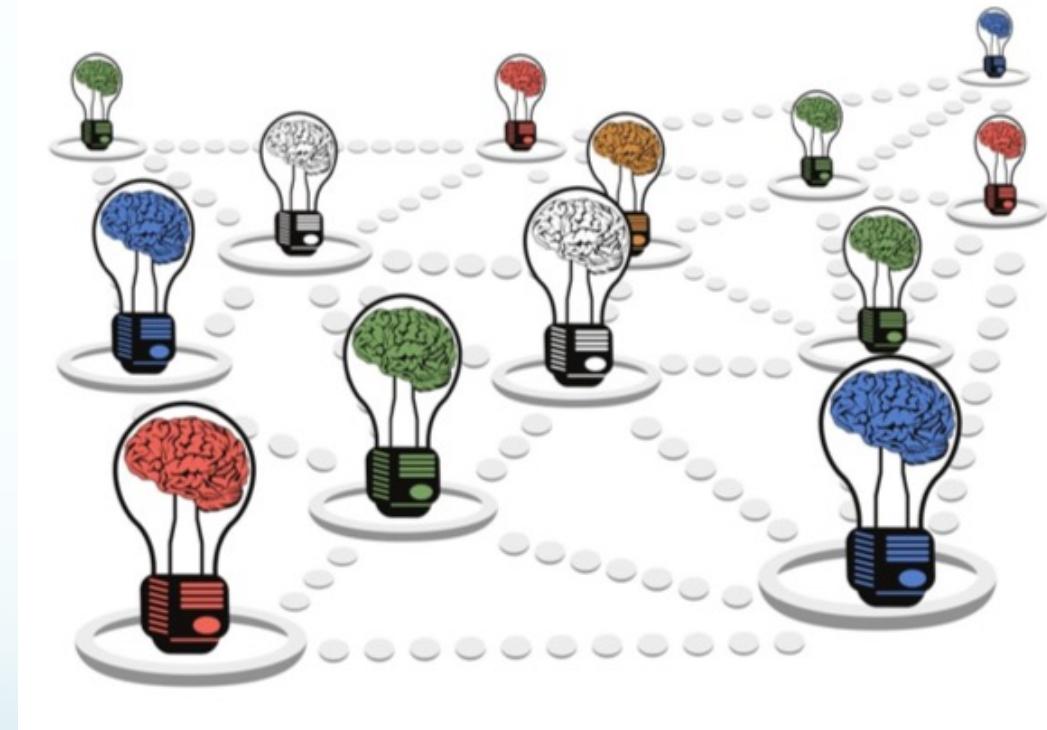
Agentes Software

- Los sistemas compuestos de múltiples agentes, comenzaron a utilizarse en la **Inteligencia Artificial Distribuida** (O'hare et al., 1996), tradicionalmente dividida en dos campos:
 - **Resolución de Problemas distribuidos:**
 - Soluciones basadas en un cjto. de nodos cooperando en dividir y compartir.
 - Cada agente tiene unas tareas y conducta prefijadas.
 - El Sistema se centra en el comportamiento global.
 - **Sistemas Multiagente:**
 - Agentes autónomos trabajan juntos para resolver problemas.
 - Cada agente tiene una información o capacidad incompleta para solucionar el problema.
 - Los agentes pueden decidir dinámicamente que tareas deben realizar y quien realiza cada tarea.
 - No hay un sistema global de control, los datos están descentralizados y la computación es asíncrona.
- La investigación inicial progresó hacia la madurez. Surge la **Programación Orientada a Agentes** y los **Lenguajes de Comunicación de Agentes**
- Posteriormente este concepto se extiende al resto de la Ingeniería del software, planteándose hoy en día la **Ingeniería del Software Orientada a Agente**.

Inteligencia Distribuida



Memoria Común



Comunicación
Inteligente

Definiciones

- Agente (Diccionario RAE):
 - Que obra o tiene virtud de obrar
 - El que realiza una acción
 - Persona o cosa que produce un efecto
 - Persona que obra con poder de otra
 - El que actúa en representación de otro (agente artístico, comercial, inmobiliario, de seguros, de bolsa, etc)
 - Persona que tiene a su cargo una agencia para gestionar asuntos ajenos o prestar determinados servicios
- Agentes software:
 - Aplicaciones informáticas con capacidad para *decidir* cómo deben actuar para alcanzar sus objetivos
- Agentes inteligentes:
 - Agentes software que pueden funcionar fiablemente en un entorno rápidamente cambiante e impredecible

¿Pero qué es un agente?

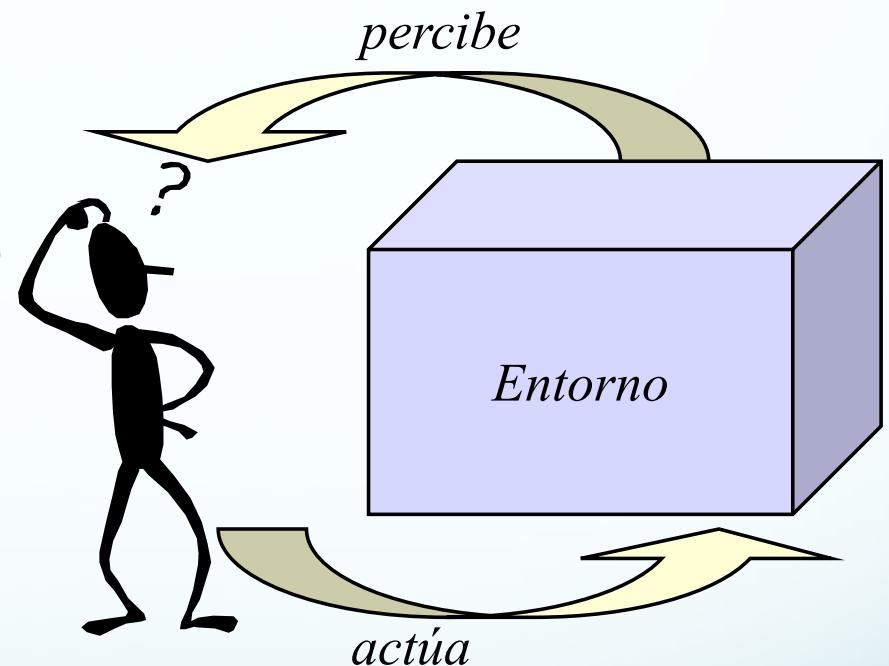
- La principal característica de los agentes es que son **autónomos**: capaces de actuar independientemente.

- Una primera definición (Wooldridge):

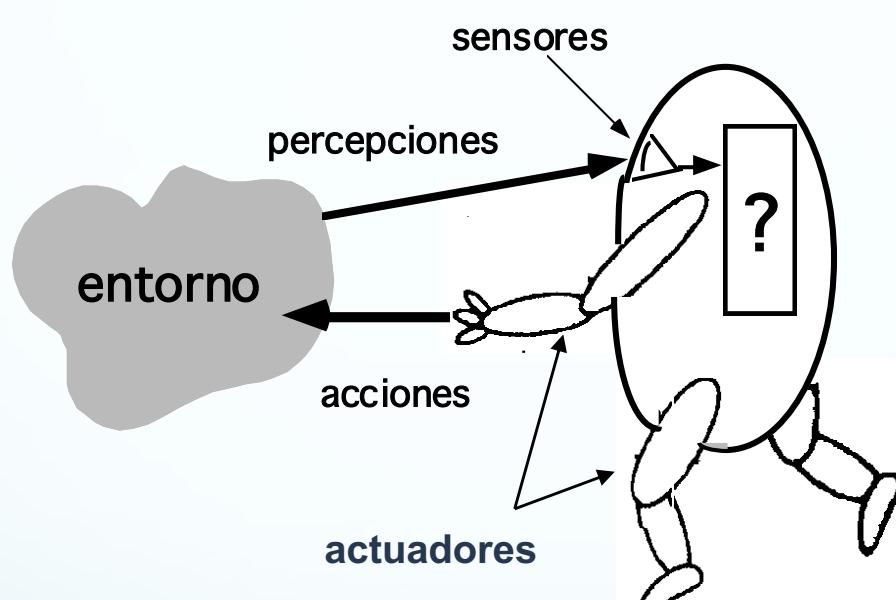
Un agente es un sistema informático capaz de actuar autónomamente en algún entorno con el fin de alcanzar los objetivos que se le han delegado.

- Consideramos que un agente como una entidad que está en continua interacción con su entorno:

percibe – decide – actúa – percibe – decide -



Concepto de Agente Inteligente



**Autónomo
Reactivo
Proactivo**

Razonamiento de alto nivel

COGNICIÓN

conducta
dirigida
por el objetivo

Razonamiento de bajo nivel

ACCIÓN PERCEPCIÓN

actuadores que
cambian el entorno

sensores que
perciben
el entorno

Agentes Simples (sin interés)

- Un termostato
 - el objetivo delegado es mantener la temperatura de una habitación.
 - sus acciones son calor/frio conectar/desconectar.
- Programa biff UNIX
 - el objetivo delegado es monitorizar el correo entrante y etiquetarlo.
 - sus acciones son acciones GUI.
- Estos agentes son triviales porque las decisiones que toman (sus procesos cognitivos) son triviales

Pero ¿qué es un agente?

Houston, we've got a problem!!!

- ¿Proceso de larga vida (permanente)?
- ¿Independencia, autonomía?
- ¿“Inteligencia”?

Pero ¿qué es un agente?



[Franklin&Graesser, 96]

Teoría de Agentes

- Un agente es un sistema informático, situado en algún entorno, que percibe el entorno (entradas sensibles de su entorno) y a partir de tales percepciones determina (mediante técnicas de resolución de problemas) y ejecuta acciones (de forma autónoma y flexible) que le permiten alcanzar sus objetivos y que pueden cambiar el entorno.

Agente: Definición

❖ Wooldridge:

Cualquier proceso computacional dirigido por el objetivo capaz de interaccionar con su entorno de forma **flexible** y **robusta**



Reactividad

- Si está garantizado que el entorno de un programa no cambia (es **estático**), un programa se puede ejecutar a ciegas.
- El mundo real no es así, no es estático, la mayoría de los entorno son **dinámicos**.
- Los programas para entornos dinámicos son difíciles de construir: los programas deben tener en cuenta la posibilidad de fallo – preguntarse a si mismos si conviene continuar su ejecución.
- Un sistema **reactivo** es aquel que mantiene una constante interacción con su entorno y responde (a tiempo para que la respuesta sea útil) a los cambios que ocurren en él.

Proactividad /Iniciativa

- Reaccionar a un entorno es fácil (por ejemplo, reglas estimulo → respuesta).
- Pero nosotros queremos que los agentes **hagan cosas por nosotros**.
- Por ello adoptamos un **comportamiento dirigido por el objetivo**.
- La **proactividad (iniciativa)** = generar e intentar conseguir objetivos, no dirigido solamente por eventos, tomar la iniciativa.
- Reconociendo oportunidades.

Capacidad Social (Sociabilidad)

- El mundo real es un mundo multi-agente: cuando queremos conseguir objetivos tenemos que tener en cuenta al resto de entidades (agentes) del entorno donde nos encontramos.
- En algunos casos para alcanzar un objetivo es necesario interactuar con otros ya que puede que nosotros no tengamos capacidades suficientes para lograrlo.
- Del mismo modo que sucede en entornos de computadores: un testimonio es internet.
- *La sociabilidad (capacidad social)* en agentes es la capacidad de interactuar con otros agentes (y posiblemente con humanos) mediante *cooperación, coordinación y negociación*.

Por lo menos significa la capacidad de comunicarse.

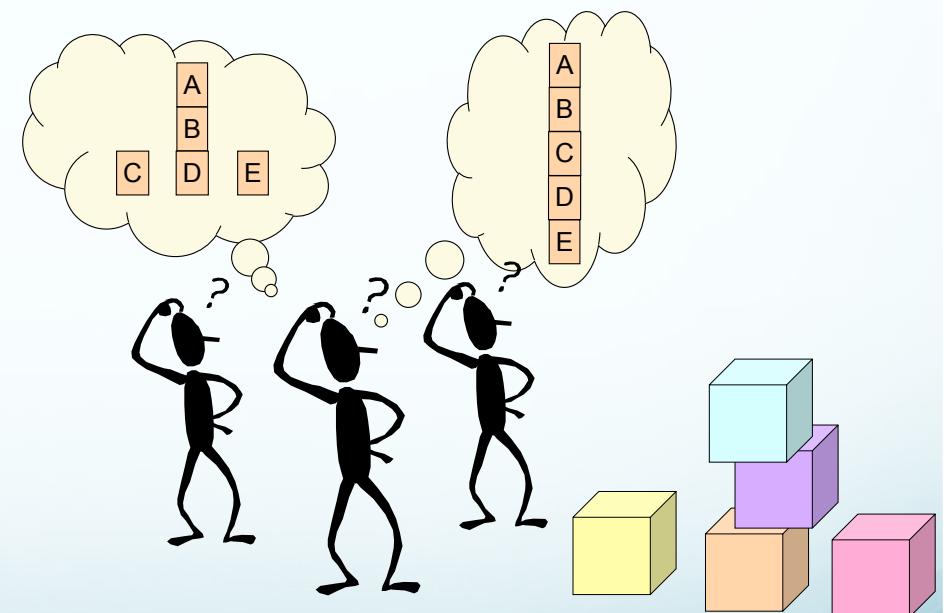
Capacidad Social: Cooperación

- Cooperación es trabajar juntos como un equipo para conseguir un objetivo compartido.
- A menudo se requiere porque ningún agente puede conseguir el objetivo solo, o porque cooperando se obtendrá un resultado mejor (por ejemplo obtener un resultado más rápidamente).



Capacidad Social: Coordinación

- Coordinación es gestionar las interdependencias entre actividades..
- Por ejemplo si hay un recurso no compatible que se desea utilizar por más de una entidad (agente), necesitamos que se coordinen.

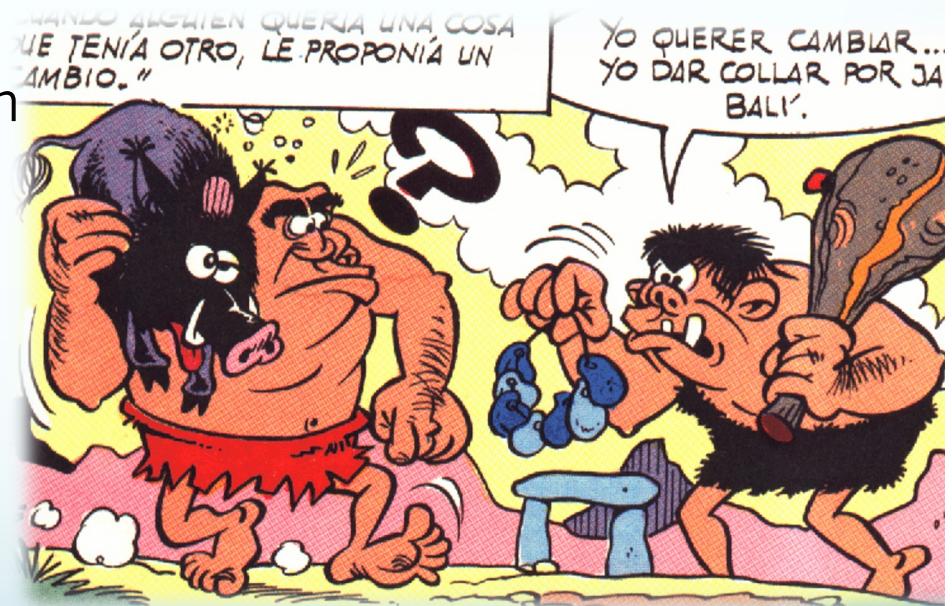


Capacidad Social: Negociación

- La negociación es la capacidad de alcanzar acuerdos sobre temas de interés común.
- Por ejemplo: Estamos en casa delante del televisor, tu quieres ver un partido de futbol y tu pareja una película.

Un posible trato es ver el partido esta noche y mañana una película (este acuerdo puede tener un ‘precio’ que una parte ha de ‘pagar’ a la otra).

- Normalmente la negociación implica ofertas y contraoferta , con compromisos (‘pagos’) asumidos por los participantes.



Otras propiedades

Concepto débil de agente

- Autonomía
- Proactividad
- Reactividad
- Sociabilidad

Concepto fuerte de agente

Concepto débil +

- Movilidad
- Veracidad
- Benevolencia
- Racionalidad
- Aprendizaje / adaptación

Un agente **siempre** tiene las propiedades débiles de agencia y **puede** tener las propiedades fuertes

Otras propiedades

Movilidad: habilidad de trasladarse en una red de comunicación informática.

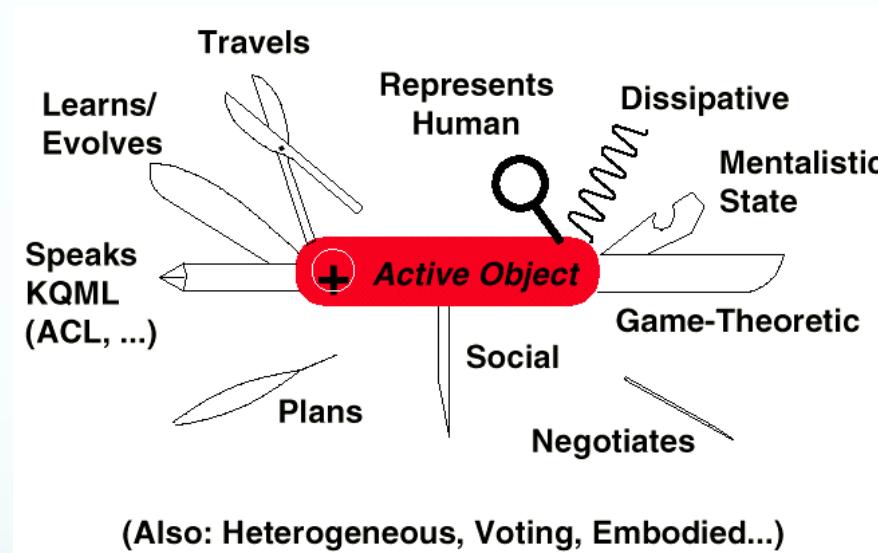
Veracidad: no comunica información falsa intencionadamente.

Benevolencia: no tiene objetivos contradictorios y siempre intenta realizar la tarea que se le solicita.

Racionalidad: tiene unos objetivos específicos y siempre intenta llevarlos a cabo.

Agente: Definición

Van Parunak:



Entornos de agente

- En entornos complejos:
 - Un agente no tiene control completo sobre su entorno, sólo tiene un control parcial.
 - Control parcial significa que el agente puede influir sobre el entorno con sus acciones.
 - Una acción ejecutada por un agente puede fallar o tener el efecto deseado
- Conclusión: los entornos son no deterministas y los agentes deben estar preparados para posibles fallos.
- ¿Qué propiedades tienen los entornos y como influyen las mismas en los agentes?



Entornos de agente: propiedades

- **Accesible (observable) vs inaccesible (parcialmente observable).**

Un entorno observable es aquel en el que el agente puede obtener información completa, exacta y actualizada del estado del entorno (los sensores perciben todos los datos que son relevantes para la toma de decisiones).

Entornos moderadamente complejos (incluyendo por ejemplo, el mundo físico diario e internet) son inaccesibles.

Cuando más accesible es un entorno, más fácil es construir agentes que interactúen con él.

Entorno: propiedades

- **Determinista vs estocástico.**

Un entorno determinista es aquel en el que cualquier acción tiene un único efecto garantizado, no hay incertidumbre sobre el estado resultante de la ejecución de una acción.

El mundo físico puede a todos los efectos ser considerado como no determinista.

Los entornos estocásticos presentan grandes problemas para el diseñador de agentes.

Entorno: propiedades

- **Episódico vs secuencial**

En un entorno episódico el desempeño/actuación de un agente depende de un número discreto de episodios, no existiendo enlaces (relación) entre el desempeño de un agente en escenarios distintos.

Cada episodio consiste en la percepción del agente y la realización de una única acción posterior. Es muy importante saber que el siguiente episodio no depende de las acciones que se realizaron anteriormente y es que en los entornos episódicos la elección de la acción en cada episodio depende sólo del episodio en sí mismo.

Los entornos episódicos son, desde el punto de vista del desarrollador de agentes, más sencillos porque el agente puede decidir que acción ejecutar basándose únicamente en el episodio actual, no necesita razonar sobre las interacciones entre el episodio actual y los episodios futuros.

Entorno: propiedades

- **Estático vs dinámico**

Un entorno estático es aquel en el que se puede asumir que no se producen cambios excepto los provocados por la ejecución de acciones del agente.

Un entorno dinámico es aquel que tiene otros procesos que operan en él, y que por lo tanto se producen cambios que están fuera del control del agente.

El mundo físico es un entorno altamente dinámico.

Entorno: propiedades

- **Discreto vs continuo**

Un entorno es discreto si en él hay un número fijo y finito de acciones y percepciones.

El juego del ajedrez es un ejemplo de entorno discreto, y la conducción de un taxi un ejemplo de entorno continuo (AIMA, Raussell and Norvig).

Entornos: Propiedades

	Solitario	Backgammon	Compra por Internet	Taxi
<u>¿Observable?</u>	Sí	Sí	No	No
<u>¿Determinista?</u>	Sí	No	Parcialmente	No
<u>¿Episódico?</u>	No	No	No	No
<u>¿Estático?</u>	Sí	Semi	Semi	No
<u>¿Discreto?</u>	Sí	Sí	Sí	No
<u>¿Agente individual?</u>	Sí	No	No (excepto las subastas)	No

El tipo de entorno determina el tipo de agente.

El mundo real es (por supuesto) parcialmente observable, estocástico, secuencial, dinámico, continuo y multiagente.

Agentes como Sistemas Intencionales

- Al explicarla actividad humana utilizamos declaraciones como la siguiente:

Lola cogió su paraguas porque creía que estaba lloviendo y no quería mojarse.
- Estas declaraciones hacen uso de una psicología popular, porque el comportamiento humano se predice y explica atribuyendo actitudes como: creer, desear, esperar, temer, ..
- Daniel Dennett acuñó el término **sistema intencional** al describir entidades "cuyo comportamiento puede ser predicho por el método de atribuir creencias, deseos y perspicacia racional".
- Un sistema intencional de primer orden tiene creencias y deseos (etc.), pero no creencias y deseos sobre creencias y deseos.
- Un sistema intencional de segundo orden es más sofisticado, tiene creencias y deseos (y sin duda otros estados intencionales) sobre creencias y deseos (y otros estados intencionales) - tanto los de los otros como de si mismo.

Agentes como Sistemas Intencionales

- ¿Podemos utilizar la actitud intencional en máquinas?

Atribuir creencias, albedrio, intenciones, conciencia, habilidades, o deseos a una máquina es correcto cuando tal atribución expresa la misma información sobre la máquina que expresa sobre una persona.

Las teorías de la creencia, el conocimiento y el deseo se pueden construir para las máquinas de una forma más sencilla que para los seres humanos, y posteriormente aplicarse a los seres humanos.

Atribuir cualidades mentales es más sencillo para máquinas de estructura conocida, tales como termostatos y sistemas operativos, pero es más útil cuando se aplica a entidades cuya estructura no es completamente conocida (John McCarty)

¿Cómo se diseña un agente?

- Considerando el agente como entidad que interactúa con su entorno el diseño de un agente requiere estudiar:
 - Cómo percibir el entorno
 - Cómo representar el entorno
 - Cómo definir los actuadores
- Utilizando la definición de Newell:
 - Cómo representar los objetivos del agente
 - Cómo describir la toma de decisiones del agente
 - Cómo representar el conocimiento
- Y si consideramos un SMA, según Ferber, los mismos elementos de antes y además:
 - Leyes que controlan el entorno
 - Objetos ubicados
 - Coordinación de los agentes
 - Acciones permitidas

La tecnología se asienta

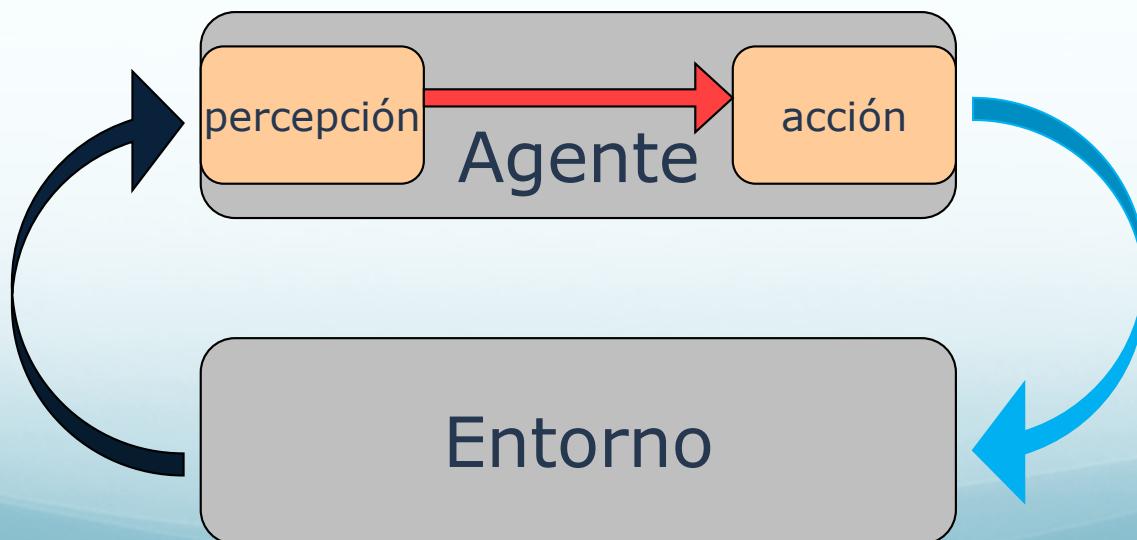
- El estudio de estos elementos ha derivado en
 - Arquitecturas
 - Desde la experimentación en la construcción de sistemas
 - Lenguajes
 - Desde el estudio teórico de los agentes, principalmente con lógicas modales
- El avance en la experimentación de diseño de sistemas ha progresado hacia soluciones más orientadas a la industria
 - Plataformas de desarrollo de agentes
 - Arquitecturas reusables
 - Entornos de desarrollo
 - Metodologías

Arquitecturas de Agente

- Arquitecturas Deliberativas
- Arquitecturas Reactivas
- Arquitecturas Híbridas

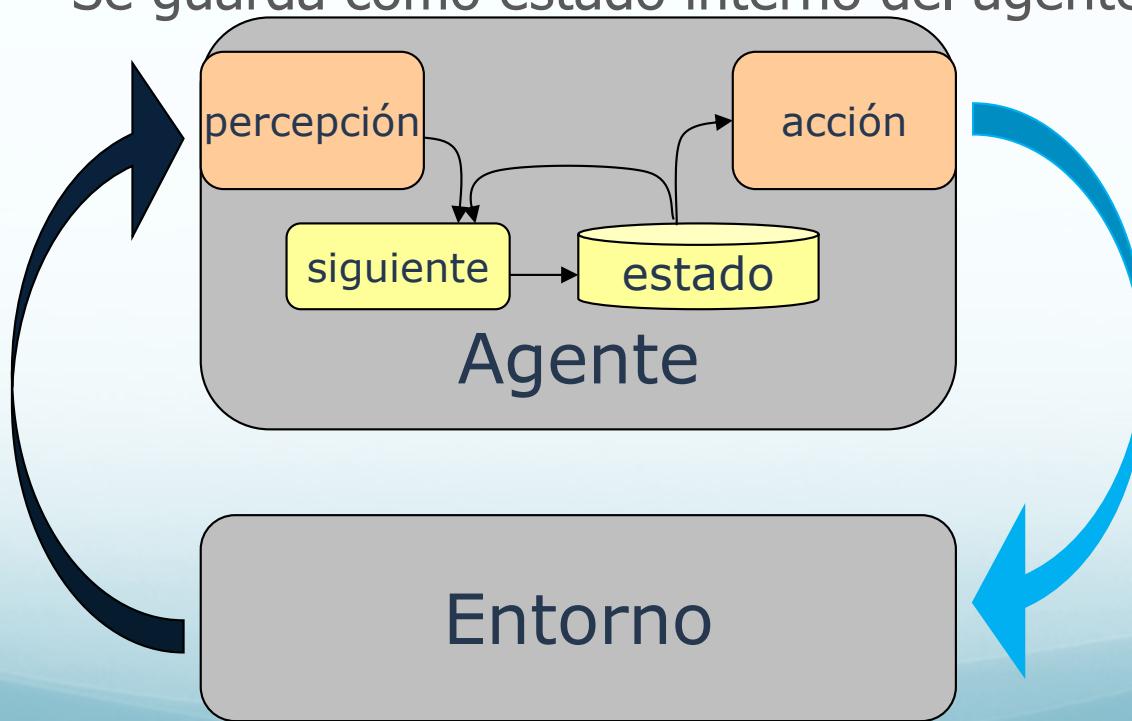
Agentes reactivos

- Modelo de agentes puramente reactivos
 - El proceso del agente es un ciclo percepción-acción (estímulo/respuesta)
 - Reacciona a la evolución del entorno
 - No hay una representación explícita del entorno, de los otros agentes, sus capacidades, etc.
 - Las decisiones no tienen en cuenta ni el pasado (no hay historia) ni el futuro (no hay planificación)



Agentes reactivos

- Agentes reactivos que mantienen su estado interno
 - Deciden la acción a realizar teniendo en cuenta su historia de interacciones con el entorno
 - Secuencia de estados del entorno o secuencia de percepciones
 - Se guarda como estado interno del agente



Arquitecturas Reactivas

- Las arquitecturas **reactivas**, se caracterizan por no tener como elemento central de razonamiento un modelo simbólico y por no utilizar razonamiento simbólico complejo (Brooks, 1991).
- Un ejemplo típico de estas arquitecturas es la propuesta de Roodney Brooks, conocida como **arquitectura de subsunción** (Brooks, 1991). Esta arquitectura se basa en el hecho de que el comportamiento inteligente puede ser generado sin utilizar propuestas del modelo simbólico y en el hecho de que la inteligencia es una propiedad emergente de ciertos sistemas complejos.
- Las arquitecturas de subsunción manejan jerarquías de tareas que definen un comportamiento. Suelen estar organizados en jerarquías de capas, de menor a mayor nivel de abstracción.

Agentes reactivos

- Ejemplo de ciclo de ejecución de un agente reactivo
 - Reglas: situación-acción
 - Conjunto de percepciones

```
while (true) {  
    estado = interpretar_entrada (percepcion);  
    regla = correspondencia (estado, Reglas);  
    ejecutar (regla, accion);  
}
```

Arquitecturas Reactivas

■ Ventajas:

- Respuesta inmediata del agente
- No problema de la representación simbólica

■ Inconvenientes:

- Difícil diseñar agentes puramente reactivos que puedan aprender de la experiencia
- Interacciones difíciles de entender en agentes con muchas conductas

Arquitecturas Reactivas

Sea

- c , un conjunto de percepciones.
- a , una posible acción.
- (c,a) , una conducta.
- R , un conjunto de conductas
- $<_r$, relación binaria de inhibición en $R \times R$

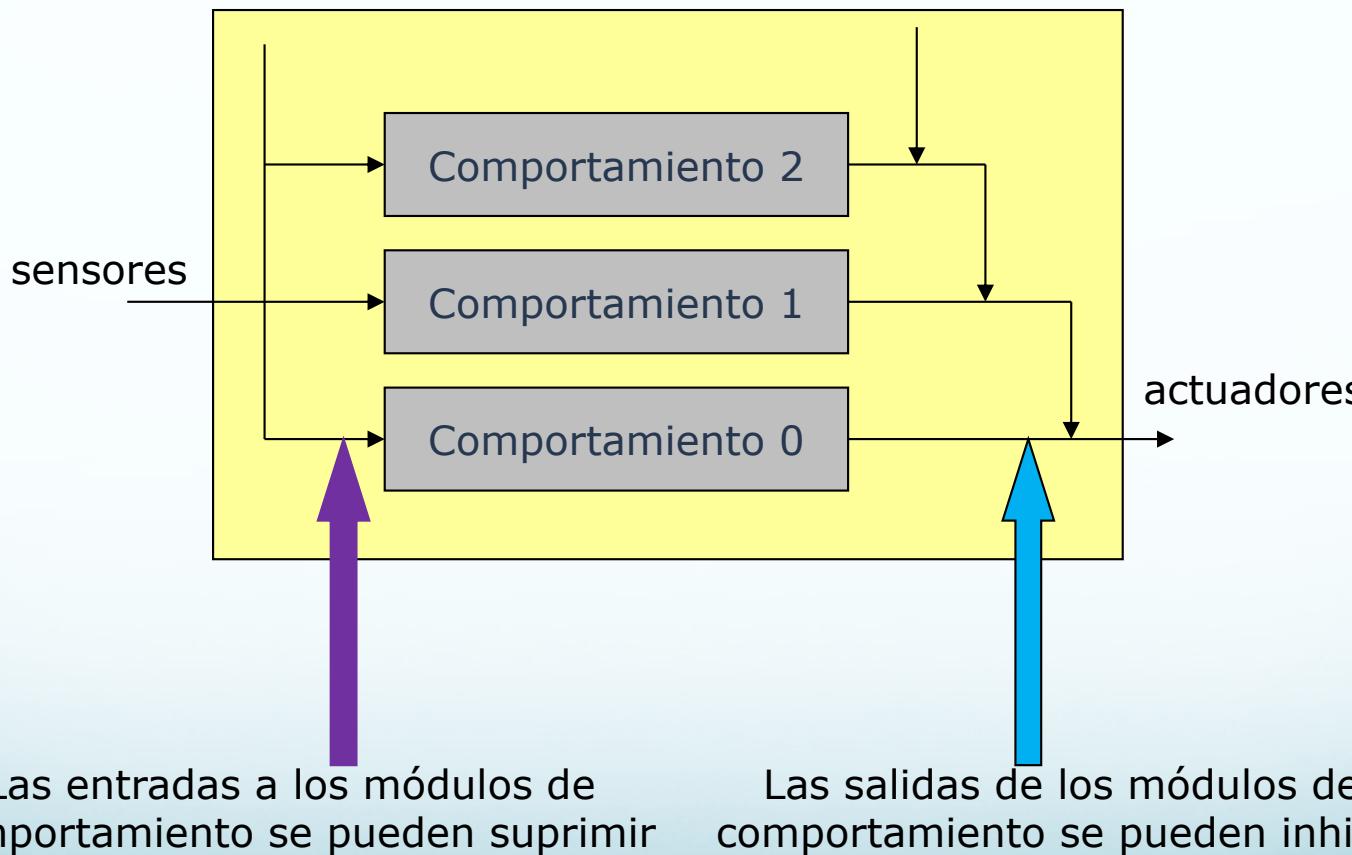
Arquitecturas Reactivas

Función $Acción(p : P) : A$

1. var *conductas*: $\mathcal{R}R$)
2. comienzo
3. $conductas := \{(c,a) \mid (c,a) \in R \text{ y } p \in c\}$
4. para cada $(c,a) \in conductas$ hacer
5. si $\neg(\exists(c^1,a^1) \in conductas \mid (c^1,a^1) <_r (c,a))$ entonces
6. devolver *a*
7. devolver *nulo*
8. fin

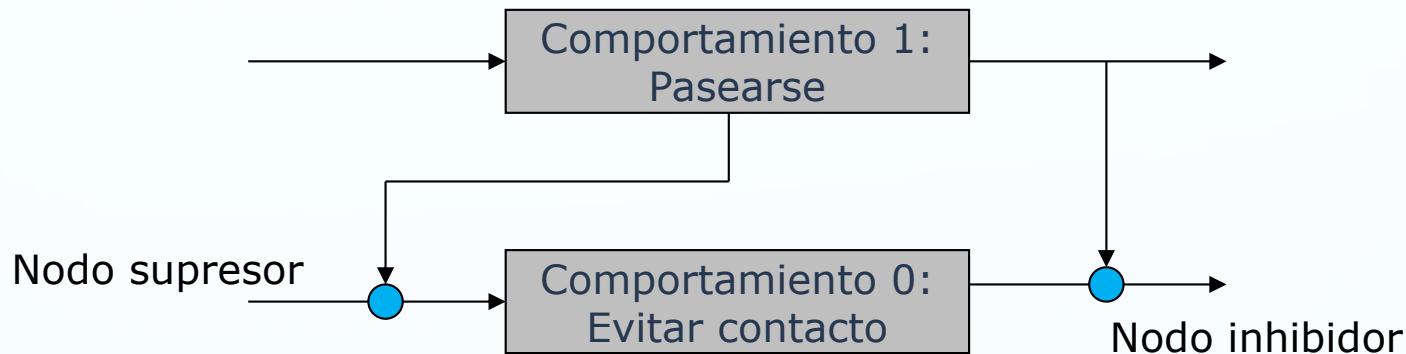
Agentes reactivos (subsunción)

- Arquitectura de subsunción [Brooks 86]



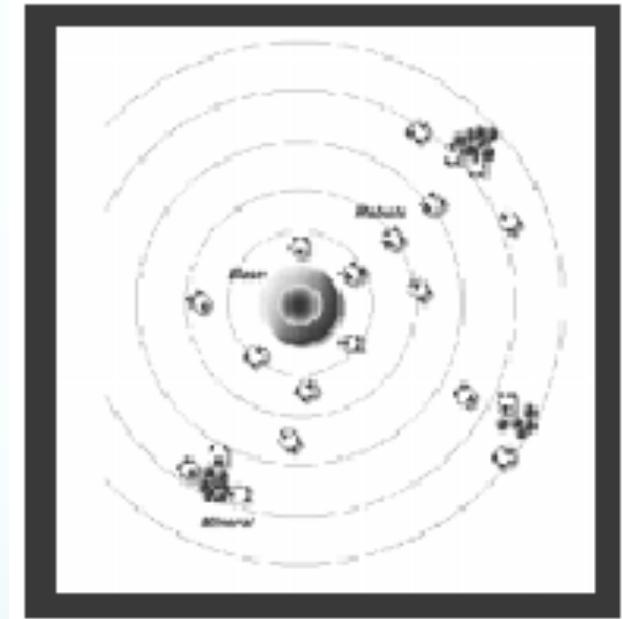
Agentes reactivos (subsunción)

- Nodos supresores e inhibidores



Agentes reactivos (Robots distribuidos)

- Ejemplo: robots distribuidos [Steels 89]
 - Problema
 - Un conjunto de robots tienen que recoger piedras (cuya localización no se conoce de antemano) y llevarlas a una nave nodriza
 - Arquitectura de subsunción cooperativa
 - La cooperación no usa comunicación directa
La comunicación se realiza a través del entorno:
 - Campo gradiente de la señal generada por la nave nodriza
 - Partículas radioactivas que pueden recoger, echar y detectar los robots al pasar

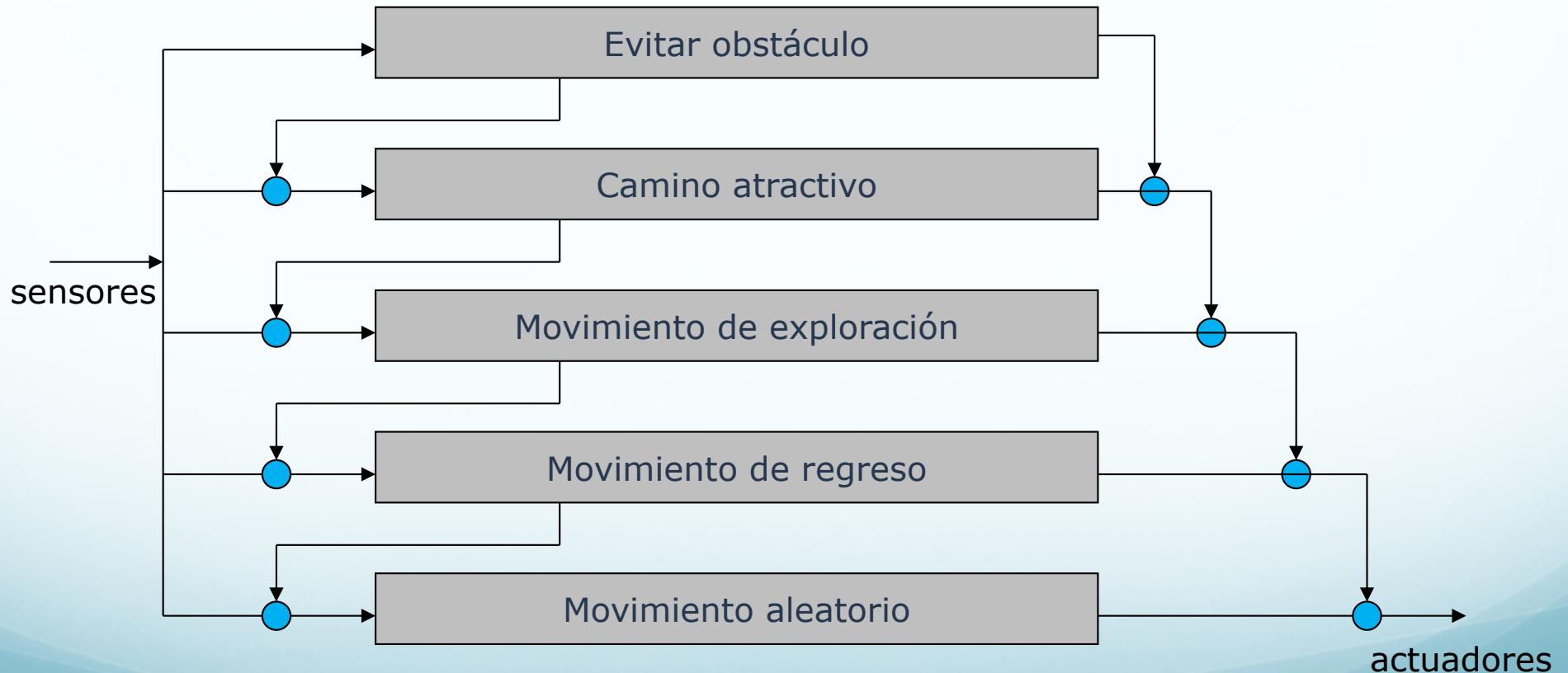


Agentes reactivos (Robots distribuidos)

- Hay dos comportamientos que se ejecutan en paralelo:
 - Comportamientos de manejo de objetos
 - Si detecto una piedra y no llevo ninguna -> recogerla
 - Si detecto la nave nodriza y llevo una piedra -> depositarla
 - Si llevo una piedra -> echar dos partículas
 - Si no llevo ninguna piedra y detecto partículas -> recoger una partícula
 - Comportamientos de movimiento
 - organizados de acuerdo a una jerarquía de subsunción

Agentes reactivos (subsunción)

- Ejemplo de jerarquía de subsunción
 - Sistema de control de un robot móvil



<http://www.youtube.com/watch?v=H68YF9YKKJ8>

Agentes reactivos (MANTA)

- MANTA: *Modeling an ANThill Activity* [Drogoul 93]
 - Simulación de sociedades de hormigas para estudiar la emergencia del reparto de trabajo en el seno de la sociedad
 - Cada hormiga tiene operaciones de percepción, selección y activación que manipulan un conjunto de tareas
 - Cada tarea se define por:
 - Una secuencia de acciones primitivas directamente ejecutables por los actuadores del agente en su entorno
 - Peso (importancia para el agente)
 - Umbral y nivel de activación
 - Selección por competición de tareas
 - A nivel de sistema esta regulación permite la emergencia de especializaciones y de repartición de tareas entre agentes

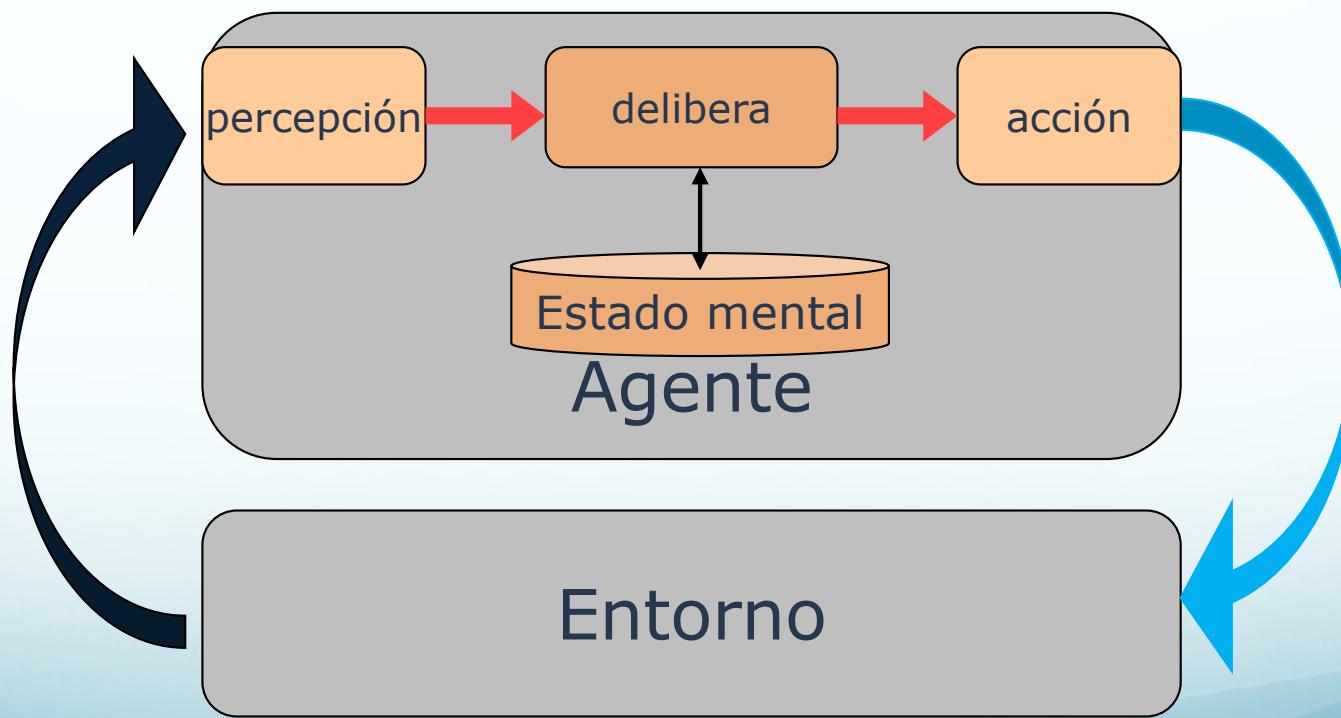
<http://www.youtube.com/watch?v=rsqsZCH36Hk&feature=fvwrel>

Agentes reactivos: conclusiones

- Sistemas constituidos de numerosos agentes homogéneos
 - Sencillos, flexibles, tratables computacionalmente, robustos, tolerantes a fallos
- La inteligencia emerge del SMA
- Problemas:
 - Los agentes necesitan conocer suficiente información sobre su entorno para actuar adecuadamente
 - La visión del agente es a corto plazo ya que está basada únicamente en información local
 - Es difícil el aprendizaje y la mejora de las capacidades de los agentes con el tiempo
 - Es difícil desarrollar agentes con muchas capas de comportamiento
 - La dinámica de las interacciones entre los comportamientos se hace cada vez más compleja
 - No hay metodología para crear este tipo de agentes: prueba y error

Agentes deliberativos

- Extienden arquitecturas cognitivas de la IA
- El proceso del agente introduce una función deliberativa entre la percepción y la ejecución para elegir la acción correcta



Arquitecturas Deliberativas

- Son aquellas arquitectura que utilizan modelos de representación simbólica del conocimiento. Suelen estar basadas en la teoría clásica de planificación, donde existe un estado inicial de partida, un conjunto de planes y un estado objetivo a satisfacer. En estos sistemas parece aceptada la idea de que un agente contenga algún sistema de planificación que sea el encargado de determinar qué paso debe de llevar a cabo para conseguir su objetivo.
- Por tanto un *agente deliberativo* (o con una *arquitectura deliberativa*) es aquel que contiene un modelo simbólico del mundo, explícitamente representado, en donde las decisiones se toman utilizando mecanismos de razonamiento **lógico** basados en la concordancia de patrones y la manipulación simbólica.

Agentes deliberativos

- Ejemplo de ciclo de ejecución de un agente deliberativo

```
EstadoMental s;
```

```
ColaEventos eq;
```

```
...
```

```
s.inicializa();
```

```
while (true) {
```

```
    opciones = generar_opciones (eq, s);
```

```
    seleccionado = delibera (opciones, s);
```

```
    s.actualiza_estado(seleccionado);
```

```
    ejecutar (s);
```

```
    eq.mira_eventos();
```

```
}
```

Agentes deliberativos

- Requieren dos procesos:
 1. Decidir qué objetivos perseguir: deliberación
 2. Decidir cómo alcanzar dichos objetivos: razonamiento basado en medios y fines
- Se basan en el razonamiento práctico (decidir en cada momento la acción a realizar para facilitar la consecución de los objetivos)
- Modelo BDI (Beliefs-Desires-Intentions)
 - Precursor: IRMA [Bratman 88]
 - Arquitectura PRS (Procedural Reasoning System) [Georgeff y Lansky 87]
 - Lógicas BDI [Rao y Georgeff]

Arquitectura BDI

- Arquitectura *B.D.I.* :

Creencias - B (Believes)

Conocimiento del agente sobre el entorno

Deseos - D (Desires)

Metas del agente

Intenciones - I (Intentions)

- Manejan y conducen a acciones dirigidas hacia las metas
- Persisten
- Influyen las creencias

Arquitectura BDI

- Las intenciones del agente juegan un importante papel en el razonamiento práctico:
 - Dirigen el razonamiento basado en medios y fines
 - Restringen las deliberaciones futuras
 - Persisten
 - Influencian las creencias sobre las que se basará el futuro razonamiento práctico
- Cada cierto tiempo el agente deberá replantearse sus intenciones, abandonando aquellas que considera que no va a alcanzar, aquellas que ya ha alcanzado y aquellas cuya justificación ha desaparecido

Arquitectura BDI

- Arquitectura *B.D.I.* :

Sean

- *Bel, Des, Int*, los conjuntos de las posibles creencias, deseos y intenciones
- El estado interno del agente una triple (B,D,I)

Entonces las funciones:

- *Brf* : $\mathcal{P}(Bel) \times P \longrightarrow \mathcal{P}(Bel)$
- *Opciones* : $\mathcal{P}(Bel) \times \mathcal{P}(Int) \longrightarrow \mathcal{P}(Des)$
- *Filtro* : $\mathcal{P}(Bel) \times \mathcal{P}(Des) \times \mathcal{P}(Int) \longrightarrow \mathcal{P}(Int)$
- *Ejecutar* : $\mathcal{P}(Int) \longrightarrow A$

Arquitectura BDI

■ Arquitectura B.D.I. :

Función *Acción* ($p : P$) : A
comienzo

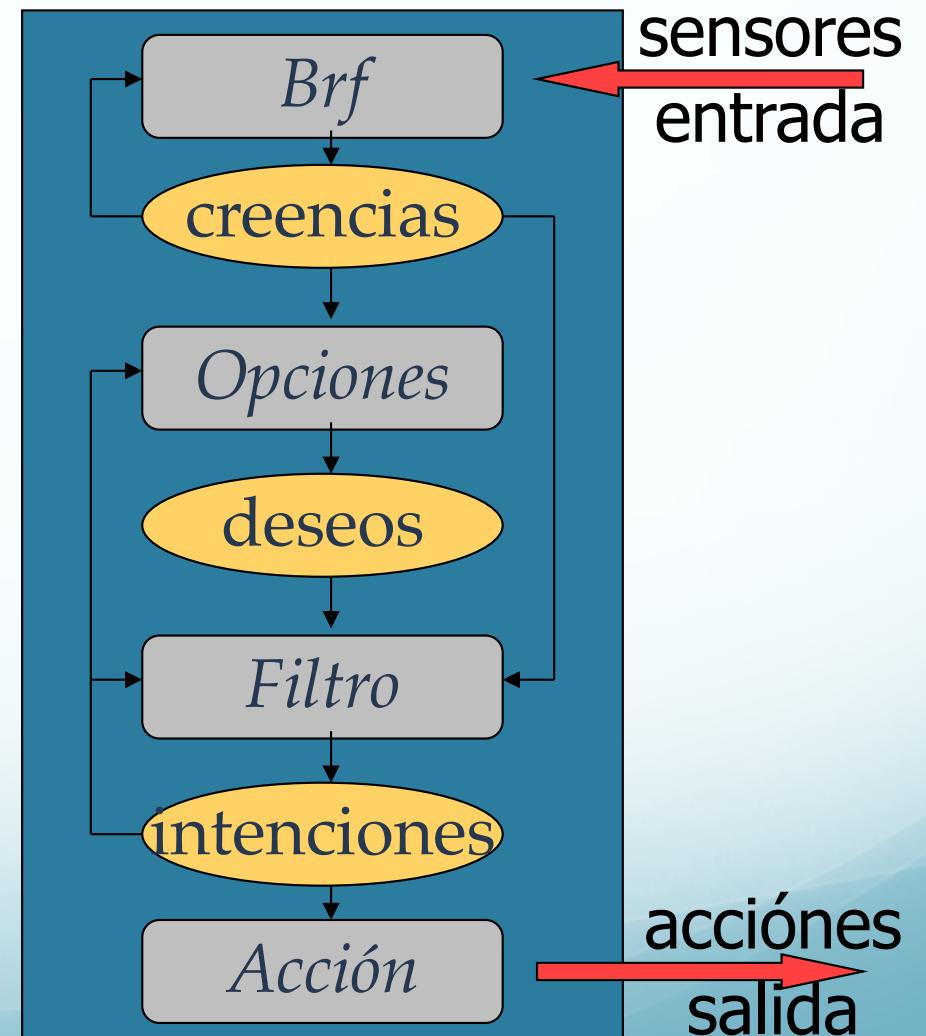
$$B := Brf(B, p)$$

$$D := Opciones(B, D)$$

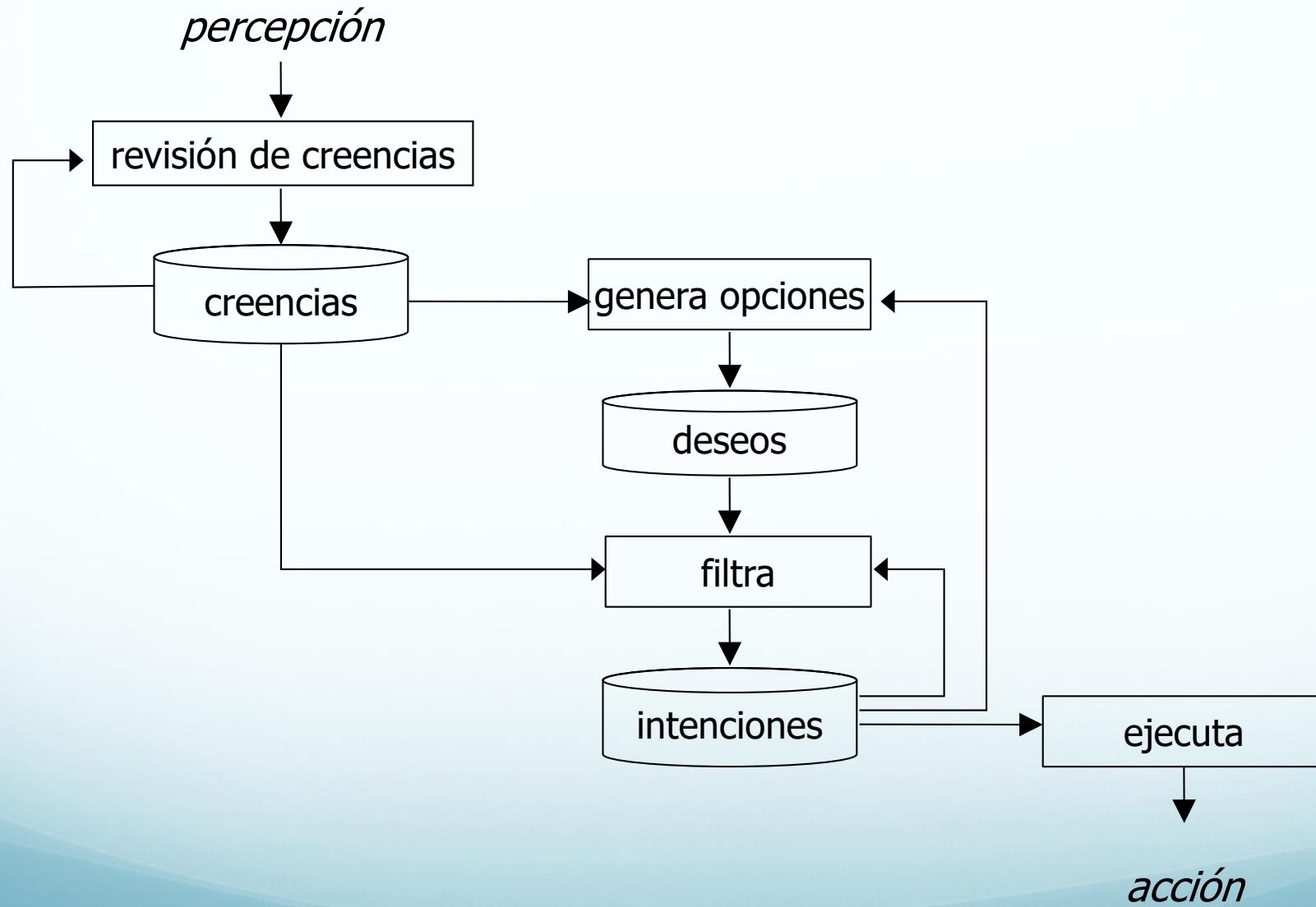
$$I := Filtro(B, D, I)$$

devolver *Ejecutar* (I)

fin



Arquitectura BDI



Arquitectura BDI: Conclusiones

- El proceso de generación de opciones de un agente BDI es realmente un proceso recursivo de generación de una jerarquía de planes
 - En cada paso se generan intenciones más específicas hasta llegar a acciones directamente ejecutables
- El proceso de deliberación de un agente BDI debe ser ajustado en función de la variabilidad del entorno
- En la práctica el problema está en encontrar implementaciones eficientes de esta arquitectura

Arquitecturas Híbridas

- Para la construcción de agentes no es del todo acertado utilizar una arquitectura totalmente deliberativa, o totalmente reactiva, se han propuesto sistemas híbridos que pretenden combinar aspectos de ambos modelos.
- Agente compuesto de dos subsistemas:
 - uno deliberativo, que utilice un modelo simbólico y que genere planes en el sentido expuesto anteriormente, y
 - otro reactivo centrado en reaccionar a los eventos que tengan lugar en el entorno y que no requiera un mecanismo de razonamiento complejo.
- Estructuración por capas:
 - verticalmente, sólo una capa tiene acceso a los sensores y actuadores.
 - horizontalmente, todas las capas tienen acceso a los sensores y a los actuadores.

Arquitecturas Híbridas

- Capas organizadas jerárquicamente. con información sobre el entorno
- Diferentes niveles de abstracción:
 - **Reactivo:**
 - Nivel bajo
 - Se toman decisiones en base a los datos recopilados por el agente.
 - **Conocimiento:**
 - Nivel intermedio
 - Se centra en el conocimiento que posee del medio
 - Normalmente utiliza una representación simbólica del medio.
 - **Social:**
 - Nivel más alto
 - Maneja aspectos sociales del entorno, incluyendo tanto información de otros agentes, como deseos, intenciones,etc.
- Comportamiento global del agente definido por la interacción entre estos niveles.

Arquitecturas de capas

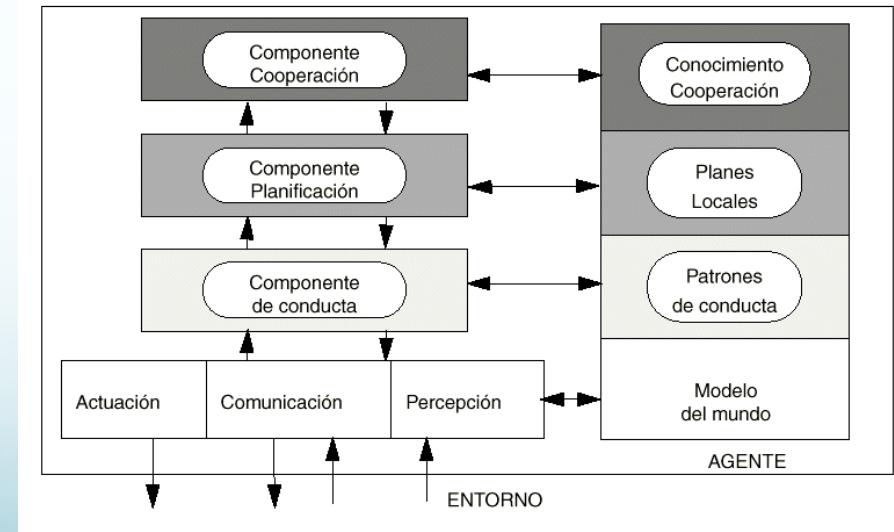
- Arquitectura en Capas:
 - Clase de arquitecturas divididas en subsistemas organizados en una jerarquía de capas que interaccionan
 - Sistema Típico en dos capas:
 - Capa para la conducta reactiva
 - Capa para la iniciativa
 - Tipos de Estructuras:
 - Capas Horizontales
 - Capas Verticales

Arquitecturas Híbridas

- Agente formado por dos o más subsistemas:
 - **Deliberativo:**
 - Modelo del mundo simbólico
 - Determinar acciones a realizar para satisfacer los objetivos locales y cooperativos de los agentes

Arquitectura Híbrida Vertical - Interrap

- **Reactivo:**
 - Procesar los estímulos que no necesitan deliberación.



Agentes híbridos

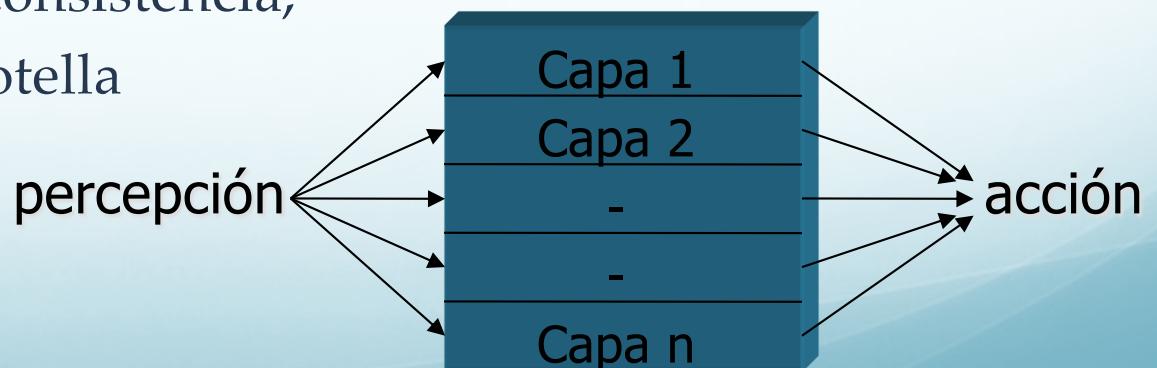
- Arquitecturas basadas en capas
 - Unas capas o subsistemas implementan el comportamiento reactivo y otras el comportamiento deliberativo
 - *Capas horizontales*: todas las capas están conectadas a la entrada y salida del agente
 - *Capas verticales*: la entrada y la salida están conectadas a una única capa del agente
- Inicialmente en robótica
 - AuRA [Arkin 90, *Integrating behavioral, perceptual and world knowledge in reactive navigation*]
 - Sistema reactivo para el control de bajo nivel del robot
 - Planificación para la toma de decisiones
- Otros ejemplos:
 - RAP [Firby 89, *Adaptive execution in dynamic domains*]
 - Interrap [Muller et al. 95, *Modelling reactive behaviour in vertically layered agent architectures*]
 - Touring Machines [Ferguson 92]

Arquitecturas de capas

- Arquitectura en Capas:

- **Capas Horizontales:**

- Cada capa está directamente conectada con los sensores y los actuadores
 - Contribuye con sugerencias a la acción de actuar
 - Función *mediadora*:
 - Decide qué capa tiene el control del agente,
 - Asegura la consistencia,
 - Cuello de Botella



Arquitecturas de capas

- Arquitectura en Capas:

- **Capas Verticales:**

- Los sensores y los actuadores están conectados con una capa
 - No tolerante a fallos
 - Una pasada



Arquitecturas de capas

- Arquitectura en Capas:

- **Ventajas:**

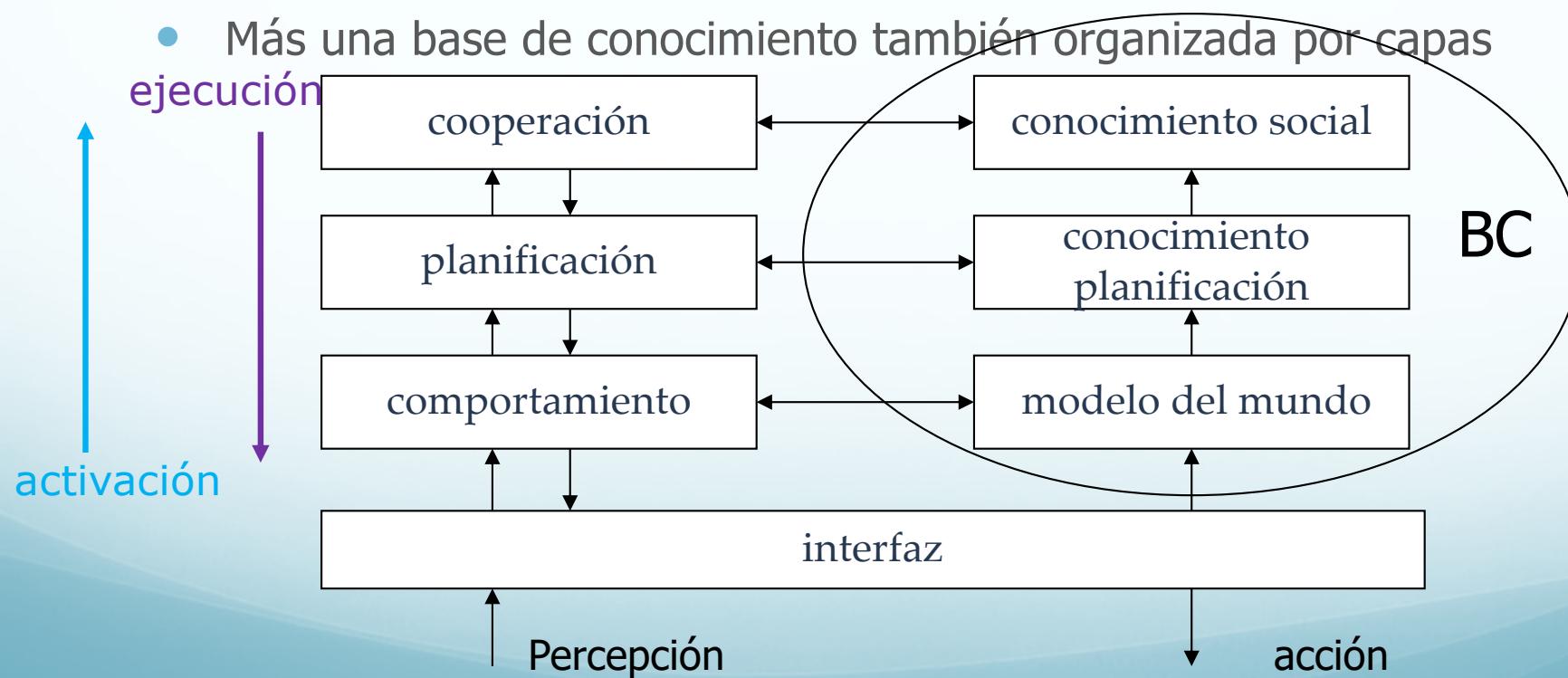
- Optima para equilibrar las diferentes conductas del agente (reactividad, iniciativa)

- **Inconvenientes:**

- Falta de claridad
 - Número elevado de todas las posibles combinaciones de interacción entre las diferentes capas

Arquitecturas híbridas: InterRap

- Arquitectura de 3 capas verticales con 2 pasos:
 - Capa de comportamiento (reactivo): información del entorno
 - Capa de planificación (pro-activo): planes y acciones del agente
 - Capa de cooperación (interacción social): planes y acciones de otros agentes del entorno
- Más una base de conocimiento también organizada por capas



Arquitecturas híbridas: Conclusiones

- Arquitecturas de carácter general
 - Las más utilizadas
 - La propia arquitectura de subsunción de Brooks es otro ejemplo de este tipo de arquitecturas
 - Utilizan una descomposición natural de la funcionalidad del agente
 - No tienen la claridad conceptual y semántica de otras arquitecturas
 - La gestión de interacciones entre capas puede ser compleja

Arquitecturas híbridas: Conclusiones

- Capas horizontales:
 - Ventaja: simplicidad conceptual
 - Si necesitamos que un agente tenga n tipos de comportamientos, implementamos n capas diferentes
 - Inconveniente: el comportamiento global puede no ser coherente
 - Se suele añadir una función de mediación que decide qué capa tiene el control en cada momento → cuello de botella y complejidad de diseño del control de interacciones entre capas
- Capas verticales
 - Arquitecturas de uno o dos pasos
 - Ventaja: La complejidad de las interacciones entre capas se reduce
 - Inconveniente: Para tomar una decisión hay que pasar el control a cada una de las capas → No tolerante a fallos

2.4 Cuestiones básicas para agentes

- Comunicación
- Cooperación
- Coordinación
- Negociación
- Inteligencia

Coordinación de agentes

- Roles, autoridad, jerarquía
- Expectativas, creencias
- Mercados, Contract Net
- Manejo de conflictos, negociación, argumentación.

2.5 Aplicaciones de agentes

- Comercio electrónico
- Monitorización de vehículos, tráfico aéreo
- Fabricación inteligente
- Agentes mediadores de información
- Redes de cooperación
- Ing. De software

Conclusión

- Problemas grandes, distribuidos
- Ambientes abiertos y dinámicos
- Software flexible, interoperable, eficiente, mantenible, confiable, robusto, ...
- Hay un agente en su futuro...

2.6 JASON: Un lenguaje de programación de agentes

- El lenguaje que interpreta es una extensión de AgentSpeak(arquitectura BDI).
- Posibilidad de usar distintas infraestructuras de agentes (Jade, SACI,...)
- Numerosas características configurables por el usuario.
- Disponible en código abierto.

Elementos del Lenguaje

- El alfabeto está compuesto por:
 - variables, constantes, símbolos de función, símbolos de predicado, símbolos de acción, conectivas, cuantificadores y símbolos de puntuación.
- También se utiliza los símbolos:
 - ! : para indicar que se desea alcanzar algo.
 - ? : para indicar que se quiere probar algo.
 - ; : indica secuencia.
 - ← : indica implicación.

Elementos del Lenguaje

- Creencias (Beliefs) Una *creencia* base se indica mediante un símbolo de predicado dentro del cual podemos especificar los términos necesarios, por ejemplo:

adyacente(X, Y)

localizado(robot, X)

localizado(coche , X)

- Objetivos (Goals) Un *objetivo* es un estado del sistema que el agente quiere alcanzar. Tenemos 2 tipos de objetivos:

- Objetivo a realizar: ! g(t1, t2, ... tn)

Ej: ! limpia(b)

- Objetivo a testear: ? g(t1, t2, ... tn)

Ej: ?localizado(coche, b)

- Planes (Plans, Intentions)

Elementos del Lenguaje

- En el lenguaje también se pueden representar *eventos de disparo*, estos pueden ser de adición (+) / borrado (-) de creencias/objetivos.

+ localizado(papel_usado, X)

+ ! limpio(X)

- Para realizar operaciones en el entorno están las *acciones* representadas por un símbolo de acción

coger(papel_usado)

mover(X, Y)

Beliefs

- Beliefs: Colección de literales representados como predicados.
 - tall(jhon).
 - likes(jhon, music)
- Anotaciones: detalles asociados a una creencia.
 - busy(jhon)[expires(autum)]
- Anotación “source”: tiene un significado para el intérprete.
 - Perceptual (colour(box1,blue)[source(percept)])
 - Comunicación (colour(box1,blue)[source(bob)])
 - Notas mentales (colour(box1,blue)[source(Self)])

Beliefs

- StrongNegation: se denota con “~”. Expresa que el agente cree explícitamente que algo es falso.

$\sim \text{colour}(\text{box1}, \text{white})[\text{source}(\text{john})]$

- Reglas: Permiten inferir nueva información a partir de conocimiento que se tiene:

$\text{likely_colour}(C,B) :- \text{colour}(C,B)[\text{source}(S)] \ \&$
 $(S == \text{self} \mid S == \text{percept}).$

$\text{likely_colour}(C,B) :- \text{colour}(C,B)[\text{degOfCert}(D1)] \ \&$
 $\text{not } (\text{colour}(_,B)[\text{degOfCert}(D2)] \ \& D2 > D1) \ \&$
 $\text{not } \sim \text{colour}(C,B).$

Goals

- Achievement goals (operador !): Expresan un estado del mundo que el agente desea conseguir.

`!own(house)`

- Test goals (operador ?): Usados normalmente para recuperar información de la base de creencias.

`?bank_balance(BB)`

Plans

- Un agente dispone de *planes* para desarrollar su labor,
 - Consta de una cabeza y un cuerpo.
 - La cabeza consiste en un evento de disparo y un contexto.
 - El cuerpo de un plan es un conjunto de acciones y objetivos (a alcanzar o a testear).

Ej: +localizado(papel_usado, X) : localizado(robot, X) &
localizado(cubo_basura,Y)

 ← coger(papel_usado);

 ! localizado(robot, Y) ;

 dejar_caer(papel_usado).

Plans



Evento Disparador

Cambios en las creencias u objetivos

+ L Belief addition

- L Belief deletion

+! L Achievement-goal addition

-! L Achievement-goal deletion

+? L Test-goal addition

-! L Test-goal addition

Plan relevante

Literales que deben ser una consecuencia lógica de la base de creencias para que el plan se instancie

L The agent believes L is true
~ L The agent believes L is false
not L The agent does not believe L is true
not ~ L The agent does not believe L is false

Plan aplicable

Plans - Cuerpo

Cuerpo del plan: Secuencia de instrucciones separadas por “;”. Estas instrucciones puedes ser:

- Actions: Acciones externas que se denotan por un predicado. Proporcionan un “feedback”.
`rotate(leftarm, 45)`
- Achievementgoals: subobjetivos que deben ser alcanzados para que el plan continúe su ejecución (si en lugar de emplear “!” se emplea “!!”, el plan no suspenderá su ejecución).
`!!at(home); call(john)` en lugar de `!at(home); call(john)`.
- Test goals: Para recuperar información de la BB o verificar si el agente cree algo.
`?coords(Tarjet,X,Y).`

Plans - Cuerpo

Mental Notes: Anotación source(self). Sirven para añadir, modificar o eliminar nuevas creencias.

+currenttargets(NumTargets); [Se añade]

-+currenttargets(NumTargets); [Se modifica]

InternalActions: Son acciones que no modifican el entorno. Se diferencian de acciones del entorno por el carácter “.”.

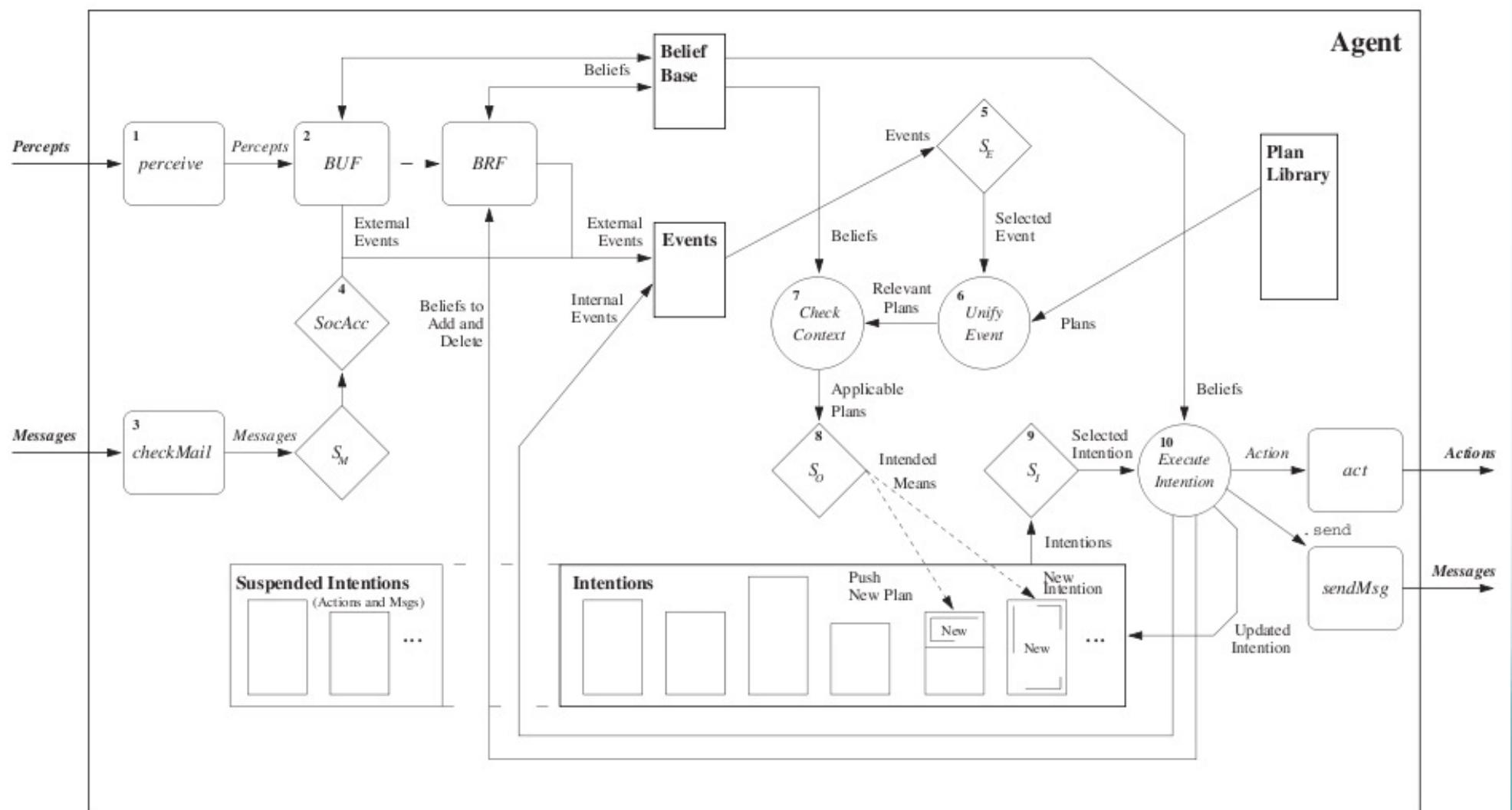
.print(...); .send(...);

Expressions: Su sintaxis es semejante a la de Prolog.

$$X \geq Y^2$$

Plan Labels: Los planes pueden estar etiquetados
@labelte: ctxt<-body.

Arquitectura de Agente



Arquitectura BDI

- Implementación. Bucle intérprete:
 - Observa el mundo y el estado interno.
 - Genera nuevos posibles deseos, encontrando planes que pueden ser disparados.
 - Selecciona de ese conjunto de planes posibles a disparar a uno para su ejecución. (plan instanciado)
 - Dicho plan instanciado puede ser disparado por una intención y se inserta en la pila de dicha intención o bien se crea una nueva intención.
 - Se selecciona la intención a ejecutar del conjunto de intenciones.

Fallo de un Plan

- Un plan puede fallar por tres causas principales:
 - Falta de planes relevantes o aplicables para un ‘achievement goal’
 - Fallo de un ‘test goal’
 - Fallo de una acción
- Cuando un plan falla se genera un evento $-\text{!g(goal deletion event)}$ si se generó por la adición de un ‘achievement goal’ o ‘test goal’.
- El plan que se dispare por el fallo se añade a la pila de intenciones del plan que ha fallado.

Fallo de un Plan

```
!g1. // initialgoal
@p1 +!g1 : true <-!g2(X); .print("end g1",X).
@p2 +!g2(X) : true <-!g3(X); .print("end g2",X).
@p3 +!g3(X) : true <-!g4(X); .print("end g3",X).
@p4 +!g4(X) : true <-!g5(X); .print("end g4",X).
@p5 +!g5(X) : true <-.fail.
@f1 -!g3(failure) : true <-.print("in g3 failure").
```

- [a] saying: in g3 failure
- [a] saying: end g2 failure
- [a] saying: end g1 failure

Ejemplo: Hello World

```
/* Creencias iniciales */
```

inicio.

```
/* Planes */
```

```
+inicio <- .print("Hola Mundo"). //plan que se  
dispara al inicio
```

Ejemplo: Factorial

```
/* Creencias iniciales */
```

```
fact(0,1).
```

```
/* Planes */
```

```
+fact(X,Y) : X < 5
```

```
<- +fact(X+1, (X+1) * Y ).
```

```
+fact(X,Y) : X = 5
```

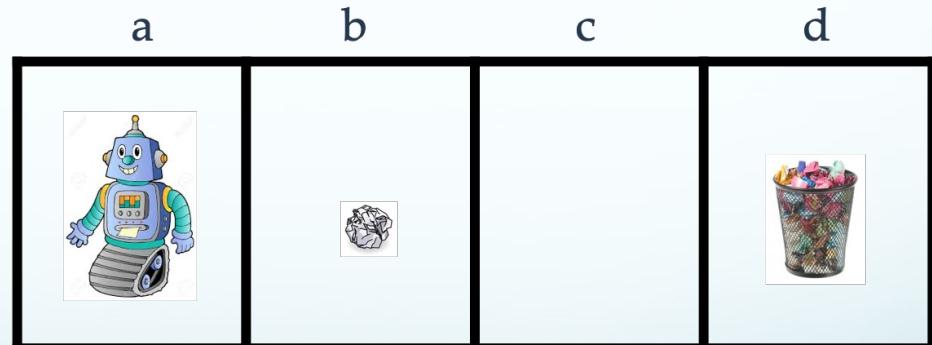
```
<- .print("fact 5 == ", Y ).
```

Ejemplo: Paro y Arranque

```
/* Objetivos iniciales */  
!dots.  
!control.  
/* Planes */  
+!dots  
    <- .print("."); // imprime puntos en un bucle sin fin  
        !!dots.  
    +!control  
        <- .wait(30); // este plan es un bucle que activa y desactiva el otro plan  
            .suspend(dots); // suspende la intención asociada al plan dots  
            .println;  
            .wait(200);  
            .resume(dots); // reactiva la intención asociada al plan dots  
        !!control.
```

Ejemplo: Robot limpiador

- Robot que trata de limpiar basura
 - Movimientos en una sola dimensión
 - La basura hay que cogerla y llevarla a la papelera
- Las creencias iniciales podrían ser:
 - $\text{adyacente}(\text{a}, \text{b}).$
 - $\text{adyacente}(\text{b}, \text{c}).$
 - $\text{adyacente}(\text{c}, \text{d}).$
 - $\text{localizado(robot, a).}$
 - $\text{localizado(papel_usado, b).}$
 - $\text{localizado(papelera, d).}$



Ejemplo: Robot limpiador

Objetivo inicial: !localizado(robot, d).

Planes:

+localizado(robot, X) : localizado(papel_usado, X)

 <- .print("papel cogido en ", X);

 -localizado(papel_usado, X); // simula el coger papel

 +llevando(papel_usado). //simula el llevar papel

+localizado(robot, X) : localizado(papelera, X) & llevando(papel_usado)

 <- -llevando(papel_usado);

 .print("todo el papel tirado a la papelera en ", X). //simula tirar a papelera

+!localizado(robot, X) : localizado(robot, X) <- .print("Ha llegado a su destino").

+!localizado(robot, X) : localizado(robot, Y) & (not (X=Y)) & adyacente(Y,Z)

 <- .print("mover de ", Y, " a ", Z);

 -localizado(robot, Y);

 +localizado(robot, Z); // simula el mover de Y a Z

 !localizado(robot, X).

Alguna anotación sobre listas en Jason

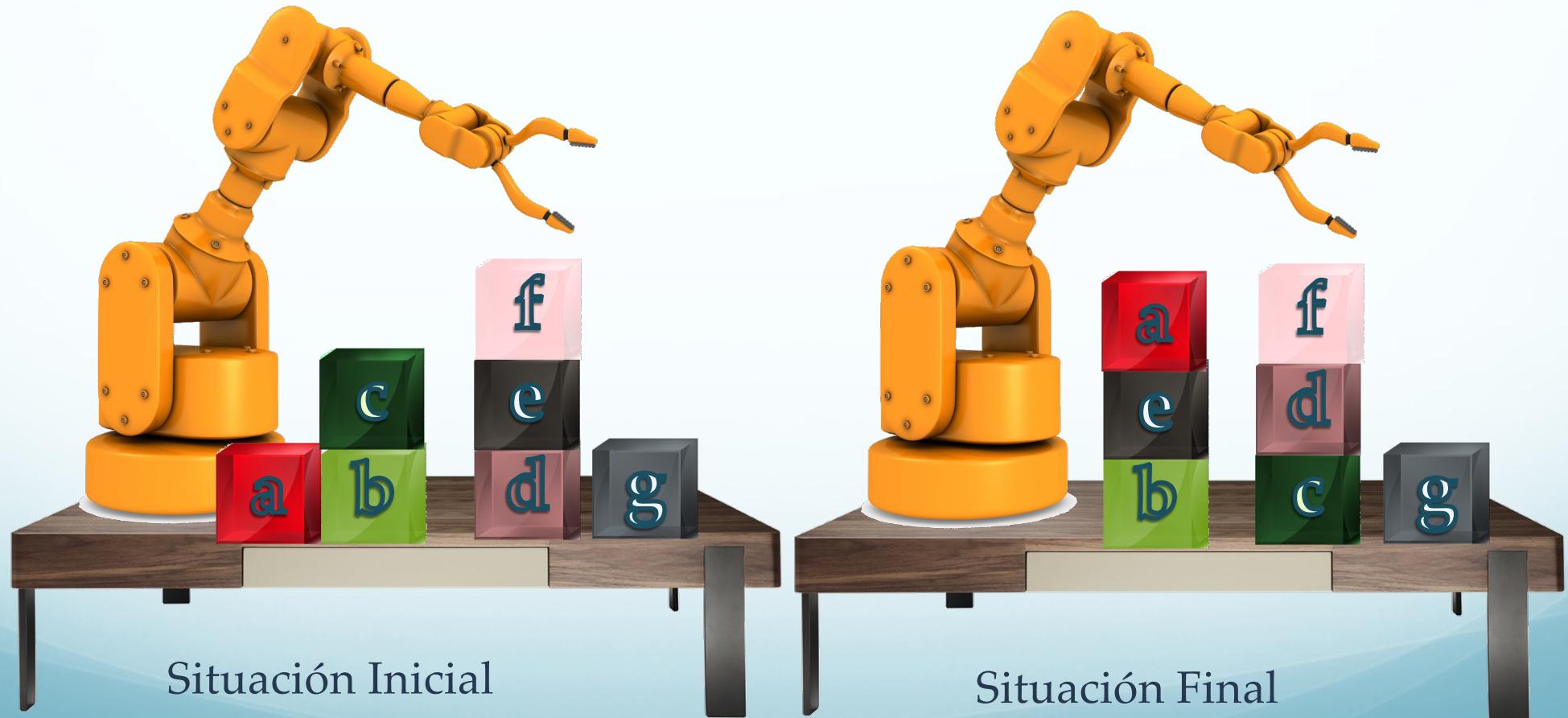
list → "[" [term (", " term) * [" | " (list | <VAR>)]] "] "

- The word term is used to refer to a constant, a variable or a structure.
- A structure is used to represent complex data, for example

```
staff("Jomi Fred Hübner", 145236, lecturer, married, wife(ilze),  
kids([morgan,thales]), salary(133987.56))
```

- list, is very useful in logic programming. There are special operations for lists; in particular ‘|’ can be used to separate the first item in a list from the list of all remaining items in it. So, if we match (more precisely, unify) a list such as [1,2,3] with list [H|T], H (the head) is instantiated with 1 and T (the tail) with the list [2,3].

Mundo de bloques



Mundo de Bloques (I)

// Un agente que soluciona el problema del mundo de bloques

/* Creencias iniciales y reglas */

clear(table).

clear(X) :- not(on(_,X)). // la creencia clear(X) es cierta cuando no tiene nada encima

tower([X]) :- on(X,table). // la torre X es cierta cuando X está sobre la mesa

// X puede ser un bloque o varios

tower([X,Y|T]) :- on(X,Y) & tower([Y|T]).

// la torre va creciendo si un bloque X se pone sobre el bloque Y

/* Objetivos iniciales */

// Marca el estado final a alcanzar

!state([[a,e,b],[f,d,c],[g]]).

Mundo de Bloques (II)

```
/* Planes */
```

```
// Alcanzar una torre  
+!state([])  <- .print("Finalizado!").  
+!state([H|T]) <- !tower(H); !state(T).
```

```
// Alcanzar un estado donde la torre esté construida  
+!tower(T) : tower(T). // La torre deseada ya existe, no hay nada que hacer  
+!tower([T])    <- !on(T,table). // La torre es de un solo elemento  
+!tower([X,Y|T]) <- !tower([Y|T]); !on(X,Y). // divido el problema en  
subproblemas
```

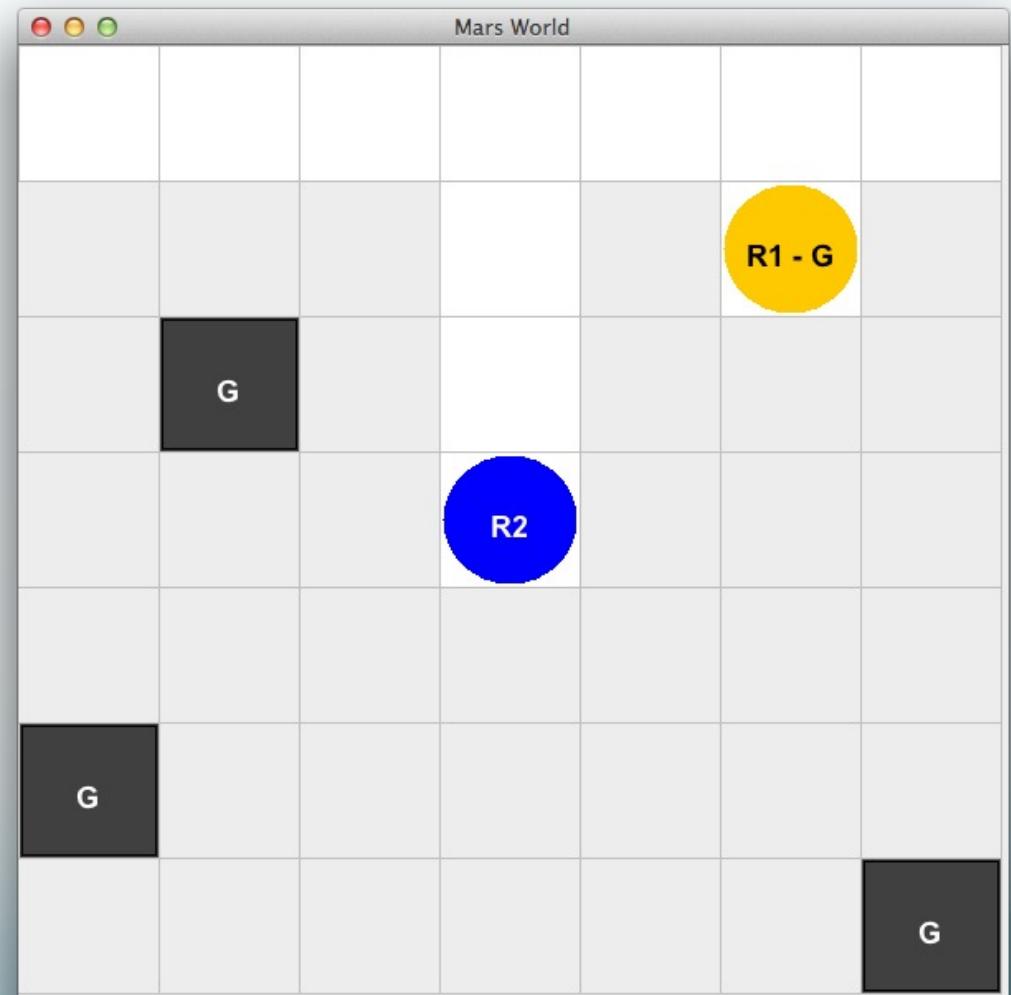
Mundo de Bloques (III)

```
// Planes para alcanzar la creencia de que un bloque X está encima de otro Y
+!on(X,Y) : on(X,Y). // ya conseguido
+!on(X,Y) <- !clear(X); !clear(Y); move(X,Y). // consigue que estén libres y mueve X
sobre Y

// Planes para alcanzar la creencia de que un bloque X está libre
+!clear(X) : clear(X). // ya conseguido
// Quita el bloque que está encima cuando se necesita dejar libre el que está debajo
X
+!clear(X)
: tower([H|T]) & .member(X,T)
<- move(H,table);
!clear(X). // vuelve a lanzar el objetivo hasta que se consiga
```

Ejemplo

- Robots de Limpieza
- 2 agentes: R1 y R2



Ejemplo

Robots de Limpieza

- Creencias

at(P) :- pos(P,X,Y) & pos(r1,X,Y) / garbage(r1)

- Objetivos

!check(slots) / !carry_to(R)

!take(S,L) / !ensure_pick(S)

- Acciones

next(slot) drop(S) pick(garb) move_towards(X,Y) burn(garb).

Ejemplo

Robots de Limpieza (R1)

- **Planes (I)**

```
+!check(slots) : not garbage(r1)
  <- next(slot);
    !!check(slots).

+!carry_to(R)
  <- // remember where to go back
    ?pos(r1,X,Y);
    -+pos(last,X,Y)
    // carry garbage to r2
    !take(garb,R);
    // goes back and continue to check
    !at(last);
    !!check(slots).
```

Ejemplo

Robots de Limpieza (R1)

- **Planes (I)**

```
+!take(S,L) : true
  <- !ensure_pick(S);
    !at(L);
    drop(S).

+!ensure_pick(S) : garbage(r1)
  <- pick(garb);
    !ensure_pick(S).

+!at(L) : at(L).

+!at(L) <- ?pos(L,X,Y);
  move_towards(X,Y);
  !at(L).
```

Ejemplo

Robots de Limpieza (R2)

- **Planes**

```
+garbage(r2) : true <- burn(garb).
```

Ejemplo

Robots de Limpieza : ¿Cómo se ejecuta?

- Fichero de definición del MAS (extensión .mas2j)

```
MAS mars {
```

```
    infrastructure: Centralised
```

```
    environment: MarsEnv
```

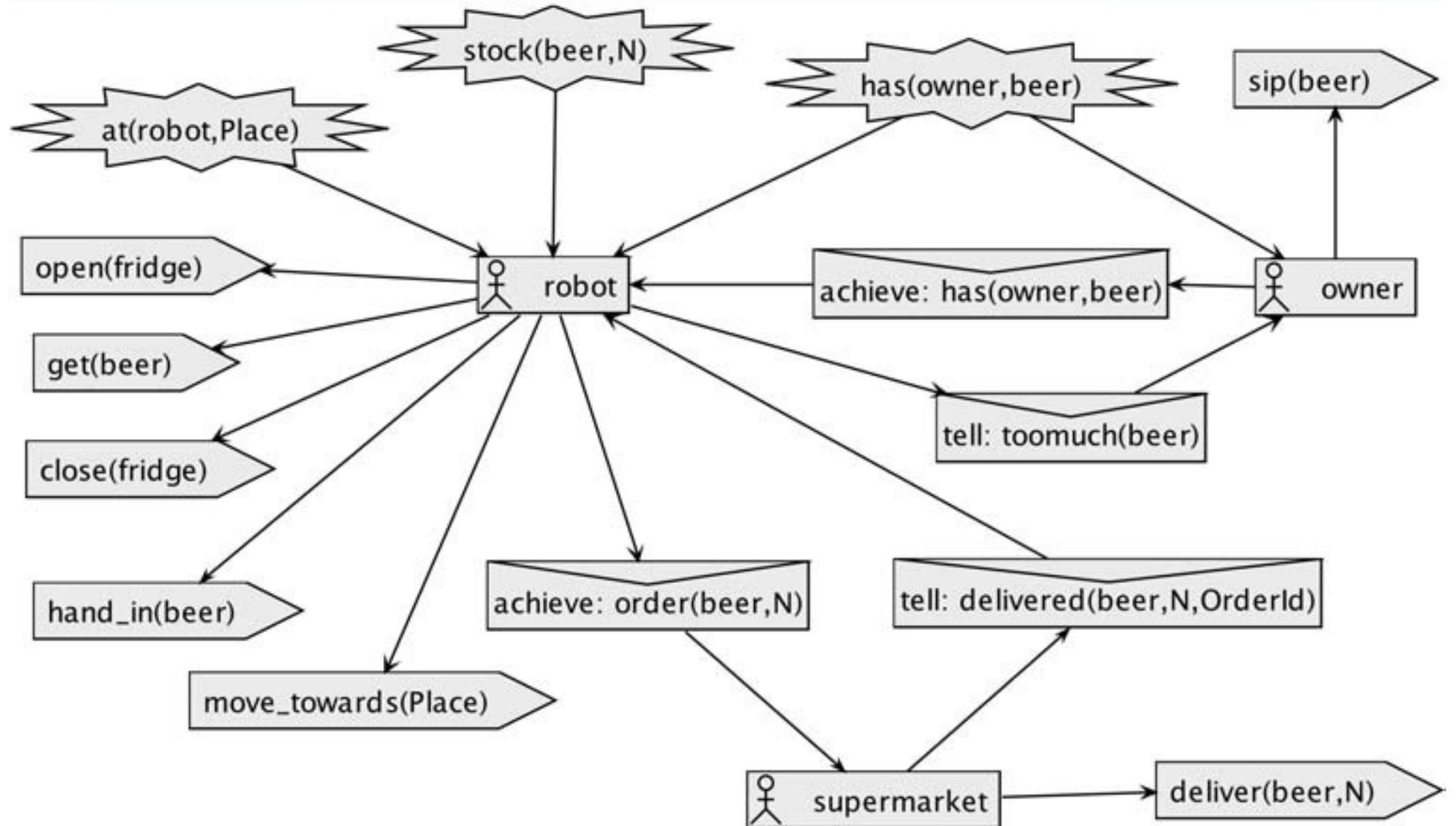
```
    agents: r1; r2;
```

```
}
```

Ejemplo de programa JASON

A domestic robot has the goal of serving beer to its owner. Its mission is quite simple, it just receives some beer requests from the owner, goes to the fridge, takes out a bottle of beer, and brings it back to the owner. However, the robot should also be concerned with the beer stock (and eventually order more beer using the supermarket's home delivery service) and some rules hard-wired into the robot by the Department of Health (in this example this rule defines the limit of daily beer consumption).

Scenario



Perceptions

- `at(robot,Place)`: to simplify the example, only two places are perceived, `fridge` (when the robot is in front of the fridge) and `owner` (when the robot is next to the owner). Thus, depending on its location in the house, the robot will perceive either `at(robot,fridge)` or `at(robot,owner)`, or of course no at percept at all (in case it is in neither of those places);
- `stock(beer,N)`: when the fridge is open, the robot will perceive how many beers are stored in the fridge (the quantity is represented by the variable `N`);
- `has(owner,beer)`: is perceived by the robot and the owner when the owner has a (non-empty) bottle of beer.

Owner Agent

```
!get(beer). // initial goal

/* Plans */

@g
+!get(beer) : true
    <- .send(robot, achieve,
has(owner,beer)). 

@h1
+has(owner,beer) : true
    <- !drink(beer).

@h2
-has(owner,beer) : true
    <- !get(beer).
```

```
// while I have beer, sip

@d1
+!drink(beer) : has(owner,beer)
    <- sip(beer);

        !drink(beer).

@d2
+!drink(beer) : not has(owner,beer)
    <- true.

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

Supermarket Agent

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

Robot Agent

```
/* Initial beliefs */  
// initially, I believe that there are some beers in the fridge  
available(beer,fridge).  
// my owner should not consume more than 10 beers a day :-)  
limit(beer,10).  
  
/* Rules */  
too_much(B) :-  
    .date(YY,MM,DD) &  
    .count(consumed(YY,MM,DD,_,_,_,B),QtdB) &  
    limit(B,Limit) &  
    QtdB > Limit.
```

Robot Agent (2)

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

Robot Agent (3)

@h2

```
+!has(owner,beer)
    : not available(beer,fridge)
    <- .send(supermarket, achieve, order(beer,5));
        !at(robot,fridge). // go to fridge and wait there.
```

@h3

```
+!has(owner,beer)
    : too_much(beer) & limit(beer,L)
    <- .concat("The Department of Health does not allow
me ",
            "to give you more than ", L,
            " beers a day! I am very sorry
about that!",M);
    .send(owner,tell,msg(M)).
```

Robot Agent (4)

- @m1

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

- @m2

+!at(robot,P) : not at(robot,P)

<- move_towards(P);

!at(robot,P).

Robot Agent (5)

// when the supermarket finishes the order, try the
'has' goal again

@a1

+delivered(beer,Qtd,OrderId)[source(supermarket)] :
true

<- +available(beer,fridge);

!has(owner,beer).

Robot Agent (6)

// when the fridge is opened, the beer stock is perceived and thus the available belief is updated

@a2

+stock(beer,0)

: available(beer,fridge)

<- -available(beer,fridge).

@a3

+stock(beer,N)

: N > 0 & not available(beer,fridge)

<- +available(beer,fridge).