

Tema 4: Capacidad Social

Authors: Vicent Botti, Carlos Carrascosa, Vicente Julián

Tema 4- Índice

4.1 Introducción

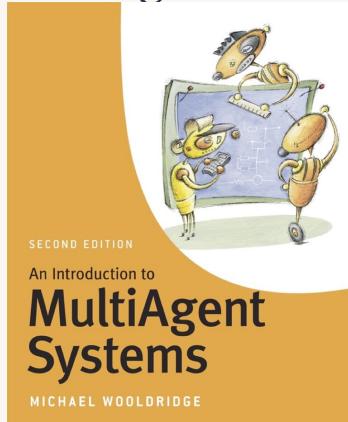
4.2 Comunicación

4.3 Lenguaje de Comunicación

4.4 Ejemplo: Mundo de Bloques – 2 robots

Bibliografía

Bibliografía básica:



Tema 6, 7, 8

An Introduction to MultiAgent Systems – Second Edition
by Michael Wooldridge

Published May 2009 by John Wiley & Sons

ISBN-10: 0470519460

ISBN-13: 978-0470519462

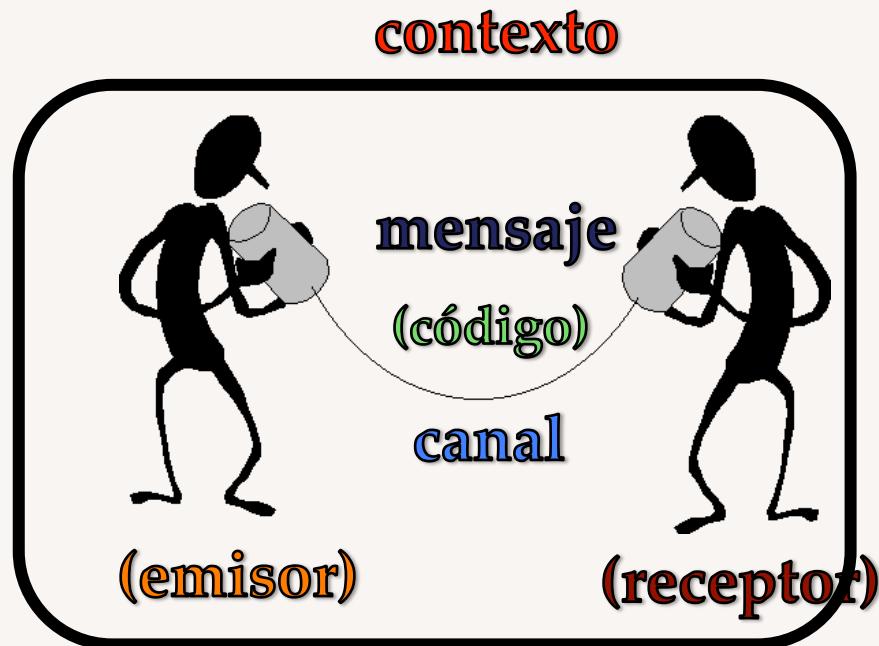
4.1 Introducción

¡Recordemos!

- 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.
- *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.

4.2 Comunicación



La comunicación es el acto o proceso por el que una o varias personas (emisores) transmiten un mensaje con información a otra u otras (receptores).

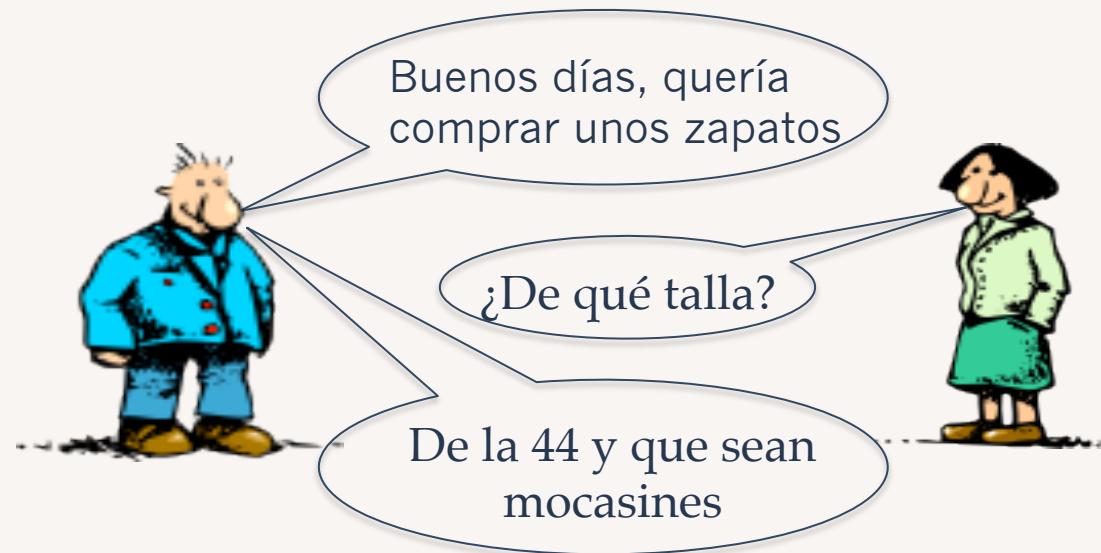
Código: es el conjunto de signos y reglas con que el emisor ha formado su mensaje.

Canal: es el medio físico o el soporte tecnológico que hace que el mensaje llegue del emisor al receptor.

Mensaje: es la información que el emisor le transmite al receptor.

Contexto: es el lugar y el momento en el que se produce la comunicación.

Comunicación



Si dos agentes tienen que comunicarse en un dominio o contexto determinado es necesario que comparten la terminología que utilizan para describir el dominio.

¿Qué significan los términos zapatos, talla, mocasines, comprar,?

Estos términos tienen que tener el mismo significado para ambos agentes.

Una **ontología** es una especificación de un conjunto de términos que tiene por objeto proporcionar una base común de comprensión sobre algún dominio.

Knowledge Sharing Effort (KSE)

Iniciado por ARPA (Asociación para Proyectos de investigación Avanzados del Dpto. Defensa de EEUU) hacia 1990, y apoyado por organismos norteamericanos de investigación (ASOFR, NSF, NRI)

Propósito:

Desarrollo de técnicas, metodologías y herramientas software para la compartición y reutilización del conocimiento entre sistemas

... a lo largo de las etapas del ciclo de vida del software:

diseño

implementación

Ejecución

Componentes fundamentales para la interacción eficaz de agentes sw

1. Un lenguaje común
2. Una comprensión común del conocimiento intercambiado
3. Una habilidad para intercambiar todo lo relativo a (1) y a (2)

Knowledge Sharing Effort (KSE)

Compartir conocimiento entre agentes requiere la capacidad de comunicarse, la capacidad de comunicarse requiere un lenguaje de comunicación común:

Sintaxis: KIF (Knowledge Interchange Format). Lenguaje para el intercambio de conocimiento <http://www.cs.umbc.edu/kse/kif/>

Semántica: Ontolingua. Lenguaje para la definición de Ontologías <http://www.cs.umbc.edu/kse/ontology/>

Pragmática: KQML (Knowledge Query and Manipulation Language). Lenguaje para la comunicación entre agentes <http://www.cs.umbc.edu/kqml>. Proporciona interoperabilidad de alto nivel (interacción) entre agentes en un entorno distribuido

4.3 Lenguaje de comunicación (código)

- ❖ El lenguaje humano ha sido la clave para el desarrollo de la inteligencia y de la sociedad.

¡Recordemos!

- ❖ La computación como interacción
 - La computación ocurre mediante y a través de la comunicación entre entidades computacionales.
 - La computación es una actividad inherentemente social, en lugar de solitaria, llevando a nuevas formas de concebir, diseñar, desarrollar y manejar sistemas computacionales.
- ❖ Sistemas multiagente
- ❖ La comunicación entre agentes es la clave para obtener todo el potencial del paradigma de agentes.
- ❖ Los agentes requieren un lenguaje de comunicación (ACL – Agent Communication Language) para comunicar información y conocimiento.

La comunicación en los agentes

La comunicación es la base para las interacciones y la organización social de los agentes

Interacciones

Hay interacciones cuando la dinámica de un agente está perturbada por las influencias de otros [O. Boissier, 2001]

Las interacciones son el motor de los SMA

Distintas formas de interaccionar (modelos de comunicación)

Acciones sobre el entorno (ejemplo ya visto)

Pizarra compartida

Paso de mensajes

Comunicación

Las capacidades de comunicación son los “ladrillos” básicos con los que construir mecanismos de coordinación, cooperación y negociación entre agentes

Modelos de comunicación (canal)

Sistemas de pizarra

Paso de mensajes

Lenguajes de comunicación de agentes

KQML

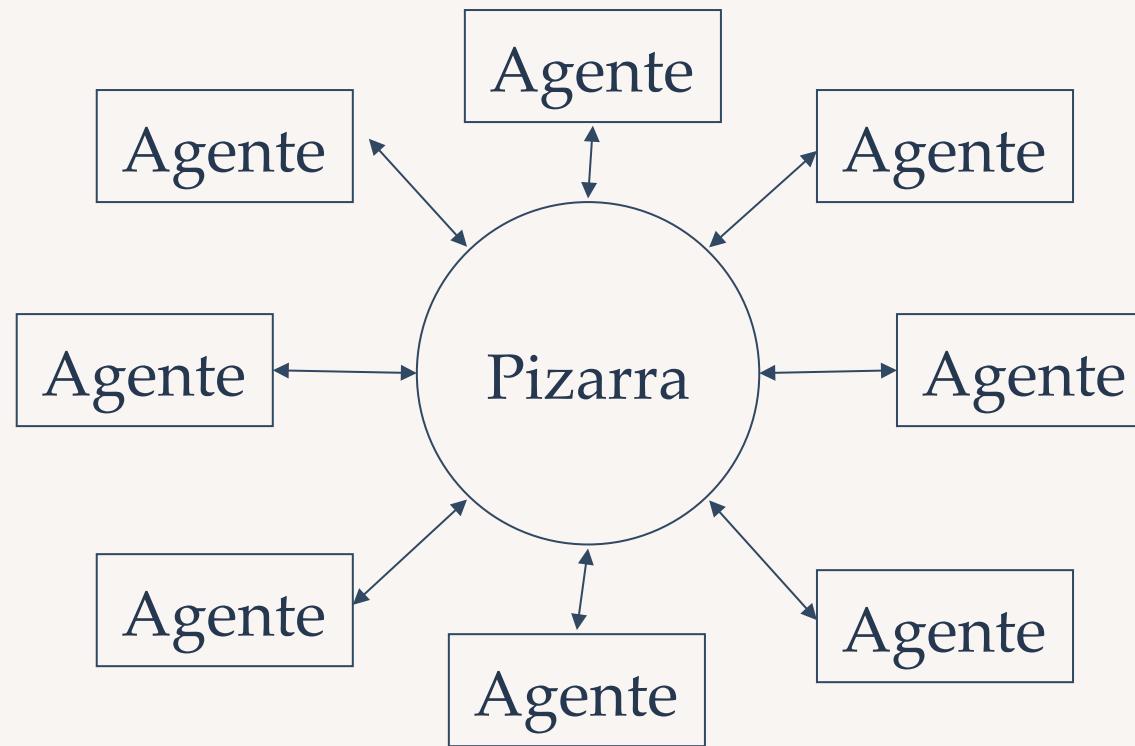
FIPA ACL

Soporte físico para la comunicación entre agentes software

Memoria Común (Blackboard)

Paso de Mensajes

Sistemas de pizarra (1)



La pizarra es una zona de trabajo común que permite a los agentes compartir todo tipo de información. Un sistema multiagente puede tener varias pizarras con distintos agentes registrados en cada una. No hay comunicación directa entre agentes

Sistemas de pizarra (2)

- Los sistemas más avanzados incorporan nuevos conceptos:
 - Moderador. Agente especializado con conocimiento de control y de evaluación que publica en la pizarra los subproblemas a resolver y decide a qué agentes se asignan de entre aquellos que se han ofrecido a resolverlos
 - Despachador. Agente que avisa a los agentes afectados por algún cambio producido en la pizarra
- Los sistemas de pizarra constituyen un método muy flexible de comunicación para la resolución distribuida de problemas. Son independientes de la estrategia de cooperación que se vaya a utilizar y no afectan a la arquitectura de los agentes individuales
- Sin embargo, la estructura central de la pizarra representa cada vez más un inconveniente ya que todos los agentes distribuidos por una red se ven obligados a acceder al dispositivo central donde se encuentra la pizarra

Paso de mensajes

- Método de comunicación muy flexible que se establece directamente entre dos agentes (remitente y receptor del mensaje)
- Los mensajes pueden ser de muy distinto tipo, permitiendo así la implementación de muy diversas estrategias de interacción entre agentes
- Es necesario establecer **protocolos de comunicación** que especifiquen el proceso de comunicación, el **formato de los mensajes** y el **lenguaje de comunicación**. Todos los agentes deben conocer la semántica del lenguaje de comunicación
- Realmente el significado de un mensaje depende del estado mental de los agentes involucrados (remitente y receptor) y de la relación entre ambos
- Un mensaje producirá cambios inicialmente en el estado mental del receptor y eventualmente en el resto del entorno

Actos de habla (*speech acts*)

Language as Action

J.L. Austin (1962), *How to do things with words*, Clarendon Press

- La lingüística tradicional intentaba entender el significado de las frases indicando cómo es posible usar una combinación de palabras para hacer una declaración con significado
 - Interés en la función denotativa del lenguaje: determinar la verdad o falsedad de una frase
- Los actos del habla hacen referencia a la función conativa
 - Un acto del habla designa las acciones intencionales en el curso de una conversación

Actos de habla (*speech acts*)

- *Language as Action*
- J.L. Austin (1962), *How to do things with words*, Clarendon Press
- Quien habla no declara solamente sentencias ciertas o falsas
- Quien habla realiza actos de habla:
peticiones, sugerencias, promesas, amenazas, etc.
- Cada declaración es un acto de habla
- Identificar el acto del habla es imprescindible para una correcta comunicación

Tipos de actos de habla

J.R. Searle (1969), *Speech Acts*, Cambridge University Press

Actos asertivos: dan información sobre el mundo

Estoy de acuerdo *2 y 2 son 4* *Estamos en clase*

Actos directivos: para solicitar algo al destinatario

Siéntate *¿Cuántas pesetas son un euro?*

Actos de promesa: comprometen al locutor a realizar ciertas acciones en el futuro

Mañana vuelvo a las 8 *Te enviaré las fotos*

Actos expresivos: dan indicaciones del estado mental del locutor

Soy feliz *Gracias* *Siento lo de tu perro*

Actos declarativos: el mero hecho de la declaración es la realización de un acto

Estás contratado *Empezamos la clase*

Componentes de los actos de habla

Locución: modo de producción de frases utilizando una gramática y un léxico

Ilocución: acto realizado por el locutor sobre el destinatario mediante la declaración (*utterance*)

Fuerza ilocutoria (F): afirmación, pregunta, petición, promesa, orden => PERFORMATIVA

Contenido proposicional (P), objeto de la fuerza ilocutoria

Se puede representar como F(P) (o performativa(contenido))

aserta(está nevando) responde(está nevando)

Perlocución: efectos que pueden tener los actos ilocutorios en el estado del destinatario y en sus acciones, creencias y juicios

Ejemplos: convencer, inspirar, persuadir, atemorizar

Componentes de los actos de habla

Ejemplo:

Cierra la puerta

locución: declaración física con contexto y referencia:
quién habla y quién escucha, qué puerta, etc.

ilocución: acto de llevar intenciones: el que habla quiere que el que escucha cierre la puerta

perlocución: acciones que ocurren como resultado de la ilocución: el que escucha cierra la puerta

Hipótesis F(P)

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

F(P), performativa(contenido)

Ilocución (performativa)

Información

Pregunta

Orden

Petición

Promesa

Oferta

Sentencia (contenido)

La ventana está abierta

¿Está abierta la ventana?

Abre la ventana

Podrías abrir la ventana

Te prometo que abriré la ventana

Abriré la ventana si tu quieres

Éxito de actos de habla

Una declaración no es verdadera o falsa:

tiene éxito o fracasa

Un acto de habla puede fallar

- en su enunciación, porque no llegue el mensaje o llegue corrompido o el destinatario no lo entiende
- en su interpretación, por el destinatario
- en su consecución final, por ejemplo porque el destinatario no sea capaz de llevar a cabo la acción solicitada

KQML (Knowledge Query Management Language)

En la comunicación intervienen los siguientes elementos:

- El protocolo de interacción
 - Estrategia de alto nivel seguida por el agente software para controlar la interacción con otros agentes
 - Desde esquemas de negociación hasta esquemas tan simples como: “cada vez que desconozcas algo de tu interés busca a algún agente que lo sepa y pregúntale”
- El lenguaje de comunicación
 - Es el medio a través del que se intercambian los actos de la comunicación
 - Indica si el contenido de la comunicación es una información, una respuesta o algún tipo de consulta
- El protocolo de transporte
 - Mecanismo de transporte utilizado en la comunicación
 - TCP, SMTP, HTTP...

KQML

- Asume un modelo de agentes:
 - entidades de alto nivel con capacidades cognitivas (representación simbólica, base de conocimientos, ...)
 - tienen una descripción de nivel intencional: su estado es un conjunto de componentes mentales como creencias, capacidades, elecciones, compromisos, etc.
 - los agentes residen en el **nivel del conocimiento**
- Los mensajes KQML comunican una actitud sobre el contenido que llevan (aserto, solicitud, pregunta)
 - Las primitivas del lenguaje se llaman **performativas**
 - Cada mensaje KQML representa un acto de habla

KQML

El lenguaje KQML está dividido en 3 capas:

- La capa de contenido
 - Relacionada con el contenido del mensaje
 - Puede expresarse en cualquier lenguaje
 - KQML sólo tiene interés en identificar el inicio y el final del contenido
- La capa de mensaje
 - Es el núcleo del lenguaje KQML
 - Determina los tipos de interacción que puede realizar un agente
 - Identifica el protocolo, la ontología y el acto del habla (performative)
- La capa de comunicación
 - Identifica las características del mensaje que describen los parámetros de bajo nivel de la comunicación (emisor, receptor e identificador de mensaje)

KQML

❖ Mensajes KQML

- Representa un acto de habla o performativa (intención de realizar alguna acción)
- Consta de una lista de pares atributo-valor (el primer elemento es el identificador del acto del habla, el resto son pares atributo-valor)



KQML

La respuesta al anterior:

```
(tell
  :sender servidor-bolsa
  :content (PRECIO TELEFONICA 19)
  :receiver pepe
  :in-reply-to accion-telefonica
  :language LPROLOG
  :ontology IBEX
)
```

KQML (palabras reservadas)

Atributo	Significado
:content	La información
:sender	Emisor del mensaje
:receiver	Receptor del mensaje
: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

KQML (performativas)

Nombre	Significado
tell	S comunica una información que se encuentra en su BC
evaluate	S quiere que R evalúe la expresión
reply	S comunica a R una respuesta esperada
ask-if	S quiere saber si la expresión se encuentra en la BC de R
ask-about	S quiere conocer todo el conocimiento de R relacionado con la expresión
ask-one	S quiere que R conteste a una pregunta
stream-about	Versión en respuesta múltiple de ask-about
achieve	S quiere que R convierta en verdadera la expresión en su BC
standby	S quiere que R esté preparado para responder a una <i>performative</i>
ready	S está preparado para responder a R
next	S quiere que R le devuelva la siguiente respuesta (de una consulta múltiple)
advertise	S está preparado para responder a una determinada <i>performative</i>
register	S puede responder a <i>performatives</i> de un determinado agente
broadcast	S quiere que R envíe la <i>performative</i> a todos los agentes conectados
recommend-one	S quiere el nombre de un agente que pueda contestar a una <i>performative</i>
recruit-one	S quiere que R le proporcione el nombre de otro agente que conteste a la <i>performative</i>

KQML: ejemplo

```
(ask-one
  :sender A
  :content (interested '(sobre, bloqueA, ?x)
  :receiver B
  :reply-with q1
  :language KIF
  :ontology bloques)
```

```
(reply
  :sender B
  :content (sobre bloqueB bloqueA)

  :receiver A
  :in-reply-to q1
  :language KIF
  :ontology bloques)
```

KQML: ejemplo

```
(stream-about  
  :sender A  
  :content bloque A  
  :receiver B  
  :reply-with q1  
  :language KIF  
  :ontology bloques)
```

```
(tell  
  :sender B  
  :content (sobre bloqueB bloqueA)  
  :receiver A  
  :in-reply-to q1  
  :language KIF  
  :ontology bloques)
```

```
(tell  
  :sender B  
  :content (sobre mesa bloqueA)  
  :receiver A  
  :in-reply-to q1  
  :language KIF  
  :ontology bloques)
```

```
(sorry  
  :sender B  
  :receiver A  
  :in-reply-to q1)
```

KQML: ejemplo

Después de completada la tarea

```
(achieve  
  :sender A  
  :content (sobre mesa bloque )  
  :receiver B  
  :reply-with q1  
  :language KIF  
  :ontology bloques)
```

```
(tell  
  :sender B  
  :content (sobre mesa bloqueB)  
  :receiver A  
  :in-reply-to q1  
  :language KIF  
  :ontology bloques)
```

KQML: ejemplo

```
(standby
  :sender A
  :content (stream-about
    :language KIF
    :ontology bloques
    :reply-with q3
    :content bloqueA)
  :receiver B
  :reply-with q1
  :language KQML
  :ontology K10)
```

```
(next
  :sender A
  :receiver B
  :in-reply-to XA77)
```

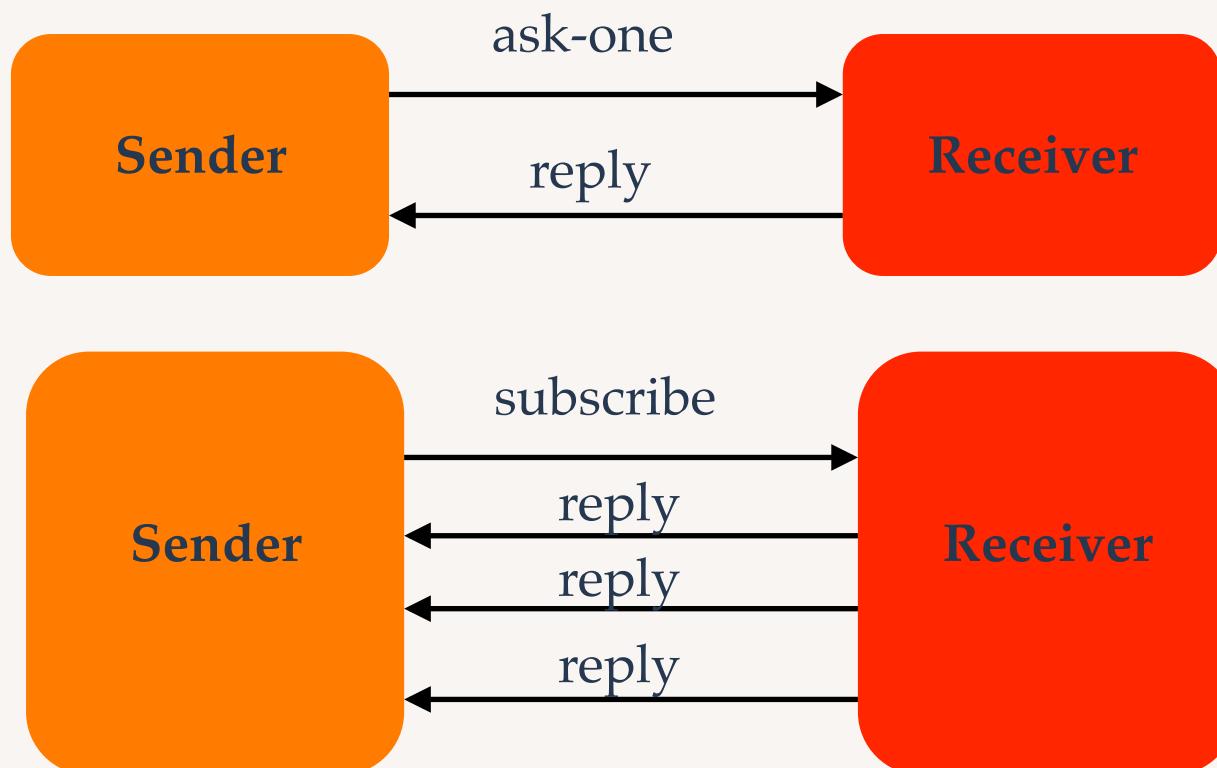
```
(discard
  :sender A
  :receiver B
  :in-reply-to XA77)
```

```
(ready
  :sender B
  :content (sobre mesa bloqueB)
  :receiver A
  :in-reply-to XA77
  :language KQML
  :ontology K10)
```

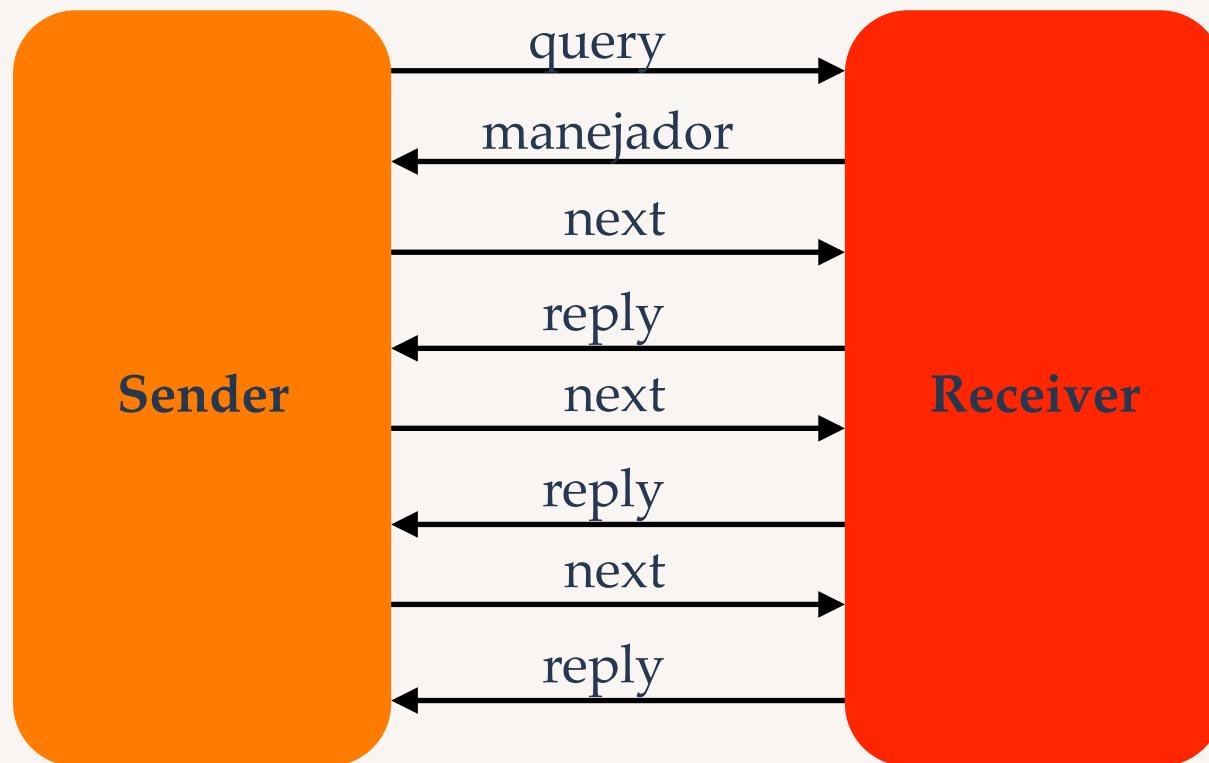
```
(tell
  :sender B
  :content (sobre bloqueB bloqueA)
  :receiver A
  :in-reply-to q3
  :language KIF
  :ontology bloques)
```

Hasta discard

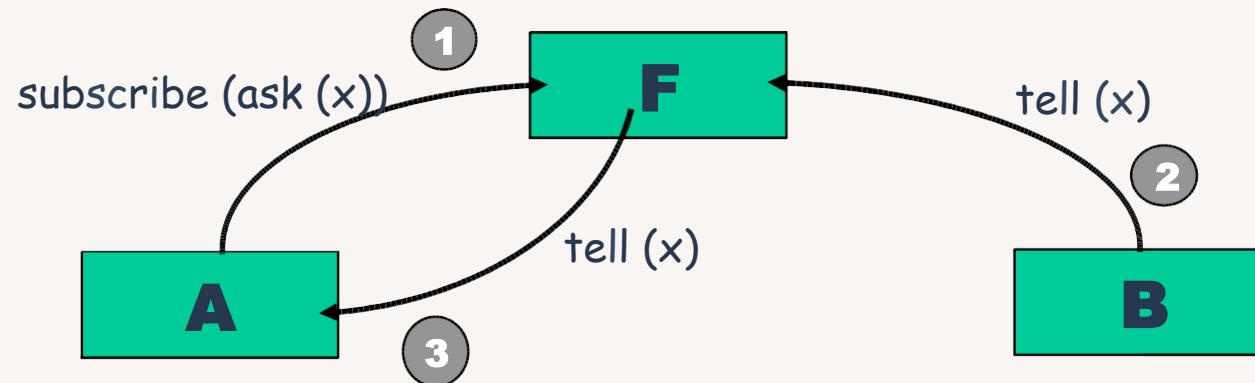
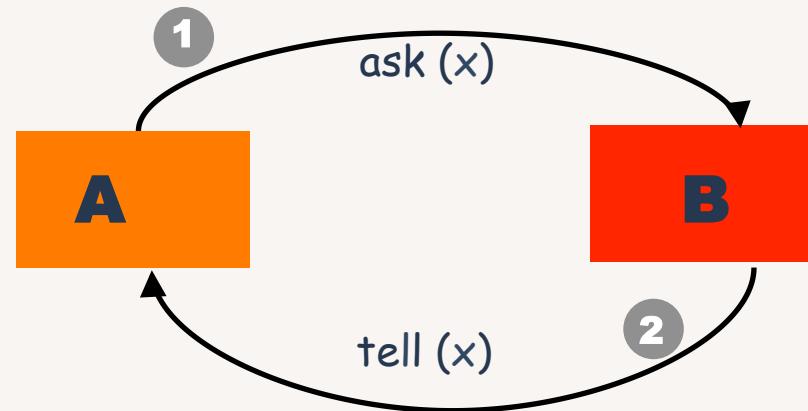
KQML: protocolos



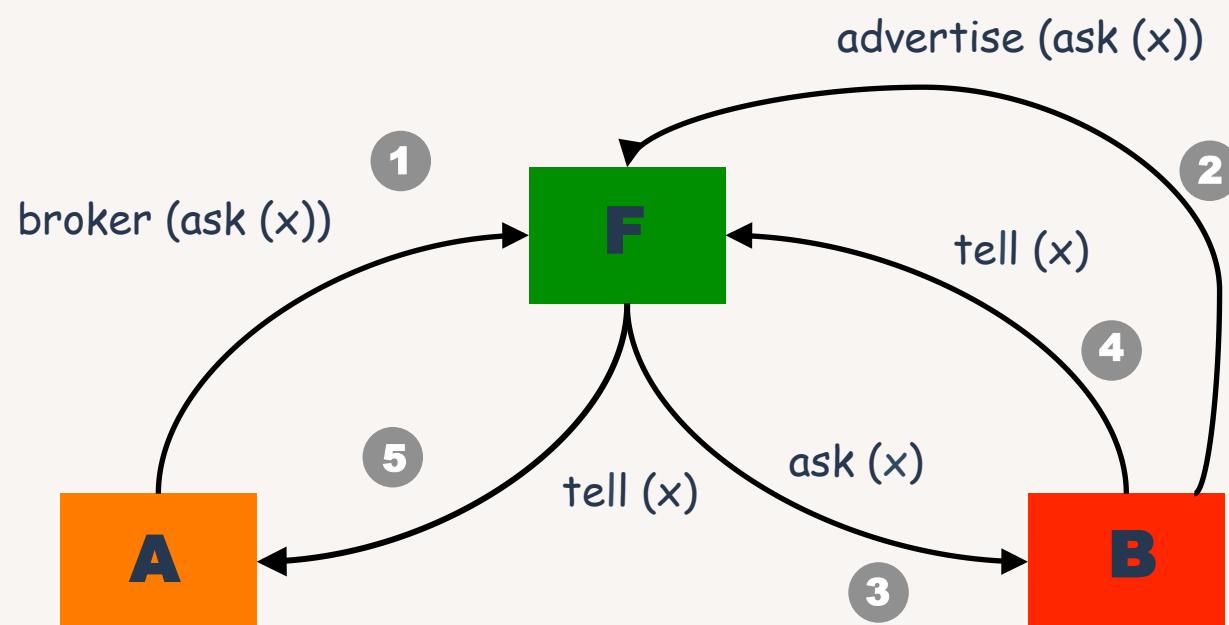
KQML: protocolos



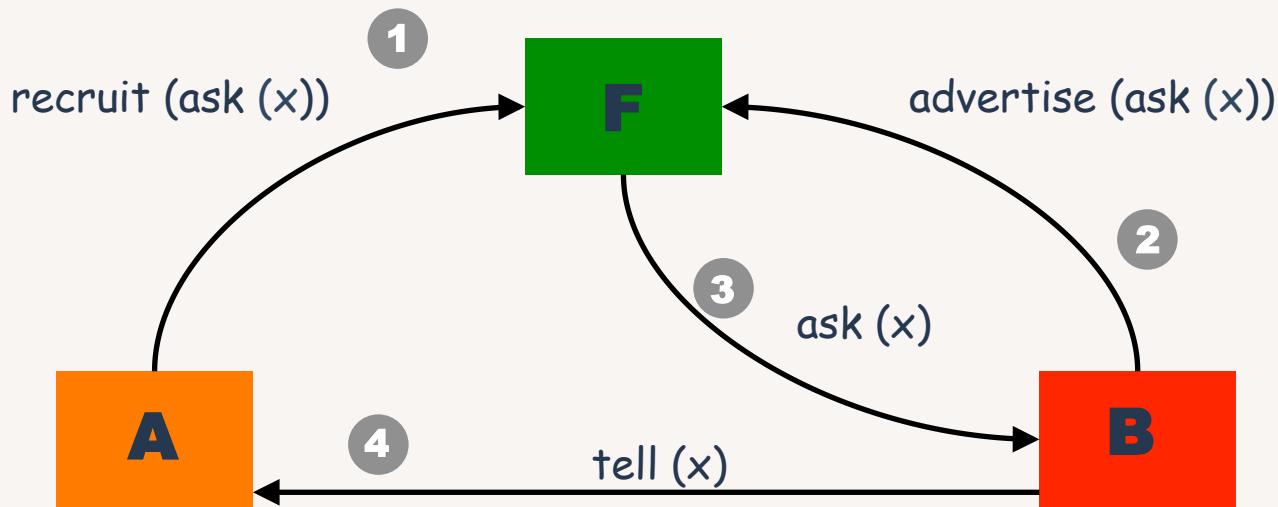
KQML: facilitadores



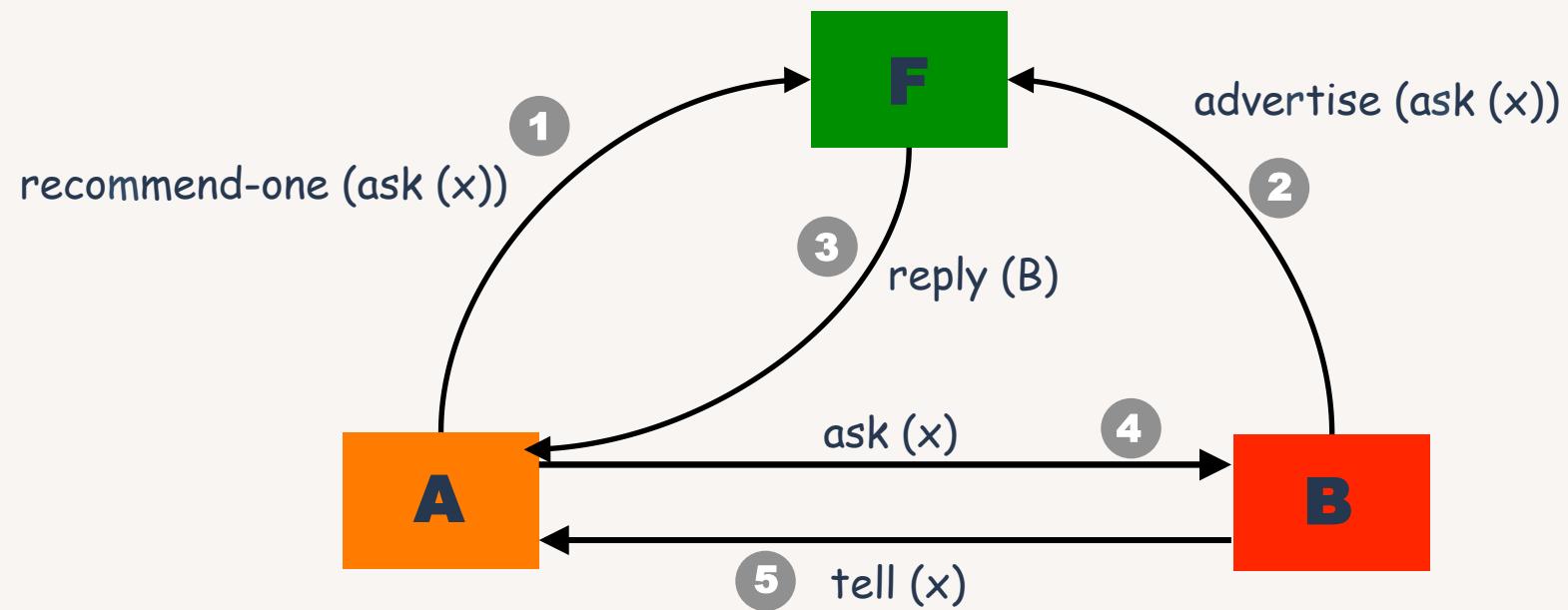
KQML: facilitadores



KQML: facilitadores



KQML: facilitadores



Comunicación en JASON

Los mensajes recibidos por un agente Jason tienen la siguiente estructura:

`<emisor, fuerza_ilocutoria, contenido>`

La forma general de la acción interna predefinida para la comunicación es:

`.send(receptor, fuerza_ilocutoria, contenido_proposicional)`

también se puede hacer:

`.broadcast(fuerza_ilocutoria, contenido_proposicional)`

Comunicación en JASON: performativas definidas

El agente s ejecuta $.send(r, \text{performativa}, \text{contenido})$

- **tell:** s pretende que r crea (lo que s cree) que el literal del contenido es verdadero
- **untell:** s pretende que r no crea (lo que s cree) que el literal del contenido es verdadero
- **achieve:** s pide a r que intente alcanzar un estado en el que el literal del contenido del mensaje sea verdadero (s delega un objetivo en r)
- **unachieve:** s pide a r que abandone el objetivo de alcanzar un estado donde el contenido del mensaje sea cierto
- **askone:** s quiere saber si el contenido del mensaje es verdadero para r (es decir, si hay una respuesta que hace que el contenido del mensaje sea una consecuencia lógica de la base creencias de r , mediante la sustitución apropiada de variables)
- **askAll:** s quiere todas las respuestas de r a una pregunta
- **tellHow:** s informa a r de un plan (que s conoce)
- **untellHow:** s pide a r que ignore un determinado plan (es decir, que elimine ese plan de su biblioteca de planes)
- **askHow:** s desea obtener todos los planes de r que son relevantes para el evento de disparo especificado en el contenido del mensaje.

Comunicación en JASON: intercambio de información

Programming Multi-Agent Systems in AgentSpeak using Jason

Rafael H. Bordini, Jomi Fred Hübner, Michael Wooldridge

John Wiley & Sons Ltd

`.send(r, tell, open(left_door))`

the belief `open(left_door)` [source(s)] will be added to r's belief base.

`.send([r1,r2], tell, open(left_door))`

the belief `open(left_door)`[source(s)] will be added to the belief base of both r1 and r2.

Comunicación en JASON: intercambio de información

.send(r, tell, [open(left_door),open(right_door)])

the belief `open(left_door)[source(s)]` and the belief `open(right_door)[source(s)]` will both be added to r's belief base.

.send(r,untell,open(left_door))

the belief `open(left_door)[source(s)]` will be removed from r's belief base.

Note that if agent r currently believes `open(left_door)[source(t), source(s), source(percept)]`, only the fact that s wanted r to believe `open(left_door)` is removed from the belief base.

Comunicación en JASON: delegación de objetivos

.send(r, achieve, open(left_door))

the event `+!open(left_door)[source(s)]`, is created and added to r's set of events.

Jason adds an annotation that the goal was delegated by agent s, so this can be used in the plans to decided whether to actually take any action or not, and to decide what is the most appropriate course of action as well (note that this is on top of the message being socially Acceptable)

.send(r, unachieve, open(left_door))

the internal action `.drop_desire(open(left_door))` is executed. This internal action causes current and potential intentions on instances of a goal `open(left_door)` to be dropped, without generating a plan failure event for the plans that required that goal to be achieved.

Comunicación en JASON: búsqueda de información

.send(r, askOne, open(Door), Reply)

s's intention that executed this internal action will be suspended until a reply from **r** is received. On **r**'s side, the message **s, askOne, open(Door)** is received and (if accepted) the Jason predefined plan to interpret **askOne** messages itself sends the appropriate reply, a reply with content **open(left_door)** if that was in **r**'s belief base at the time. This is done with a test goal, so if the relevant information is not in the belief base the agent can also try to execute a plan to obtain the information, if such plan exists in the plan library. When the reply is received by **s**, the content in that message is instantiated to the **Reply** variable in the internal action executed by **s**. The intention is then resumed, hence the formula after the **.send** internal action is only executed after **s** has received a reply from **r**.

Comunicación en JASON: búsqueda de información

.send(r, askOne, open(Door))

as above but asynchronous, in the sense that the askOne message is sent to r but the intention is not suspended to wait for a reply; this means that the formula in the plan body immediately after the .send internal action could in principle execute in the next reasoning cycle (if selected for execution).

.send(r, askAll, open(Door), Reply)

as for askOne (the synchronous version) but Reply is a list with all possible answers that r has for the content. For example, if at the time of processing the askAll message agent r believed both open(left_door) and open(right_door), variable Reply in the internal action would be instantiated with the list '[open(left_door),open(right_door)]'. An empty list as reply denotes r did not believe anything that unifies with open(Door).

Comunicación en JASON: planes

```
.send(r, tellHow, "@pOD +!open(Door): not locked(Door)<- turn_handle(Door);  
push(Door); ?open(Door).")
```

the string in the message content will be parsed into an AgentSpeak plan and added to r's plan library. Note that a string representation for a plan in the agent's plan library can be obtained with the **.plan_label** internal action given the label that identifies the required plan

```
.send(r, tellHow, ["+!open(Door) : locked(Door)<- unlock(Door); !!open(Door).", "+!  
open(Door) : not locked(Door) <- turn_handle(Door); push(Door);?open(Door)."])
```

each member of the list must be a string that can be parsed into a plan; all those plans are added to r's plan library

Comunicación en JASON: planes

`.send(r, untellHow, "@pOD")`

the string in the message content will be parsed into an AgentSpeak plan label; if a plan with that label is currently in `r`'s plan library, it is removed from there. Make sure a plan label is explicitly added in a plan sent with `tellHow` if there is a chance the agent might later need to tell other agents to withdraw that plan from their belief base.

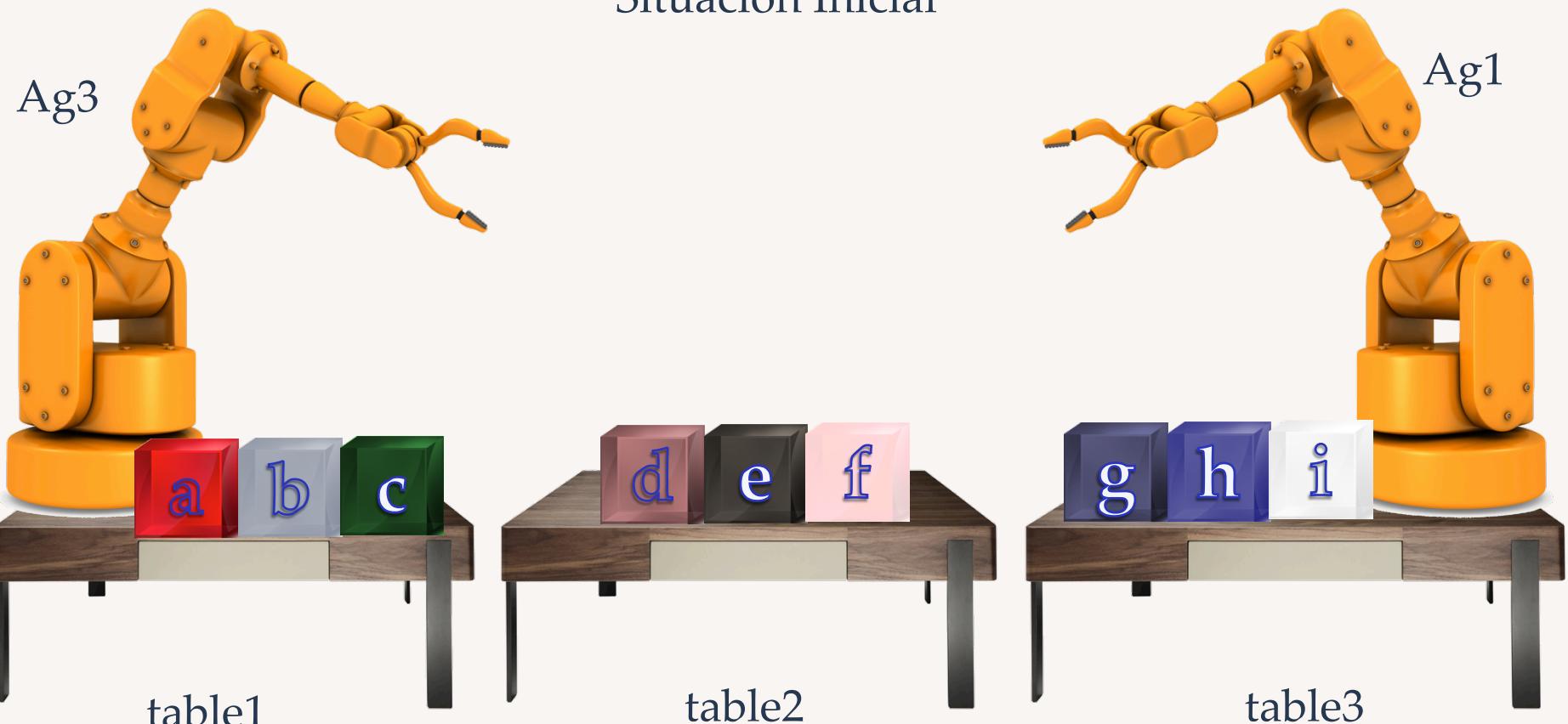
`.send(r, askHow, "+!open(Door)")`

the string in the message content must be such that it can be parsed into a triggering event. Agent `r` will reply with a list of all relevant plans it currently has for that particular triggering event. In this example, `s` wants `r` to pass on his know-how on how to achieve a state of affairs where something is opened (presumably doors, in this example). Note that there is not a fourth argument with a 'reply' in this case, differently from the other 'ask' messages. The plans sent by `r` (if any) will be automatically added to `s`'s plan library.

Ejemplo: Mundo de Bloques

– 2 robots

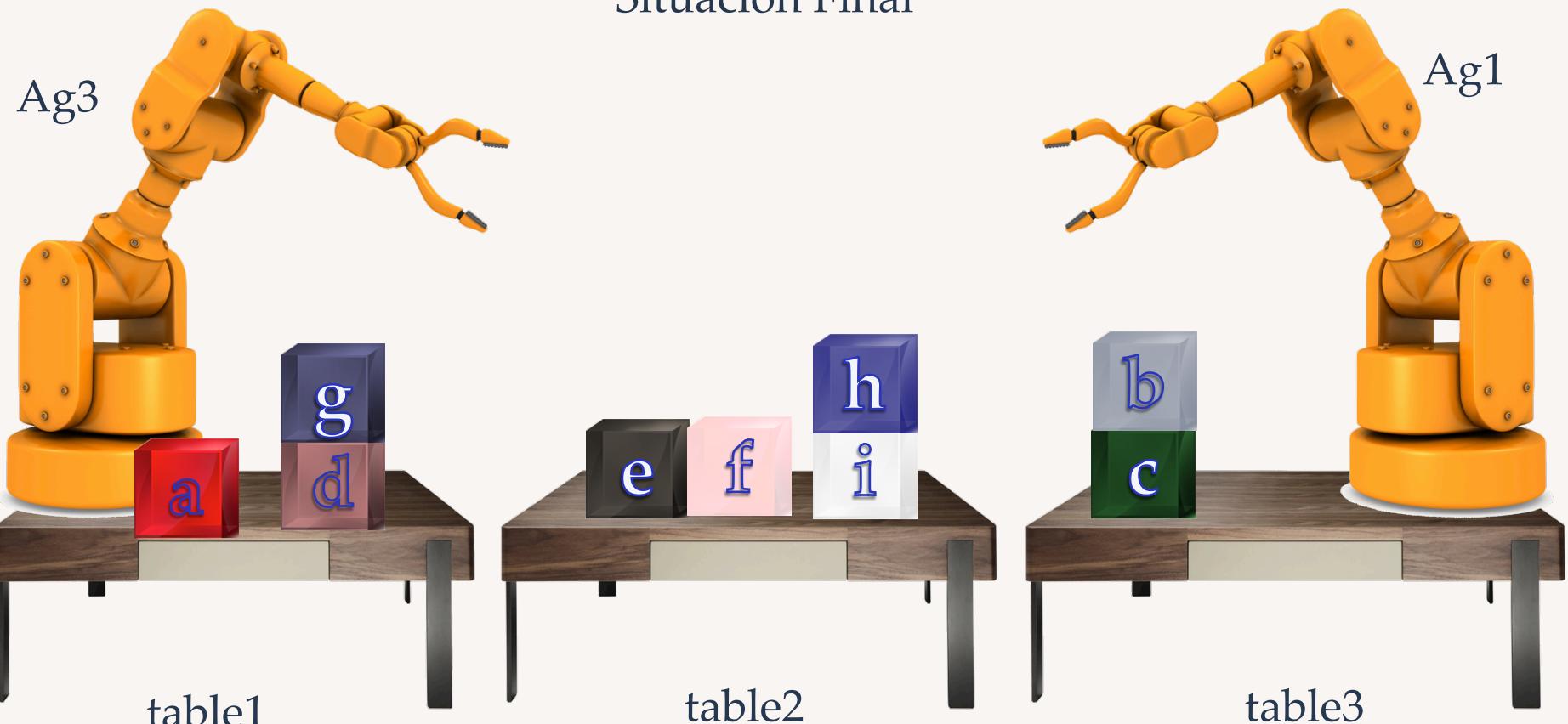
Situación Inicial



Ejemplo: Mundo de Bloques

– 2 robots

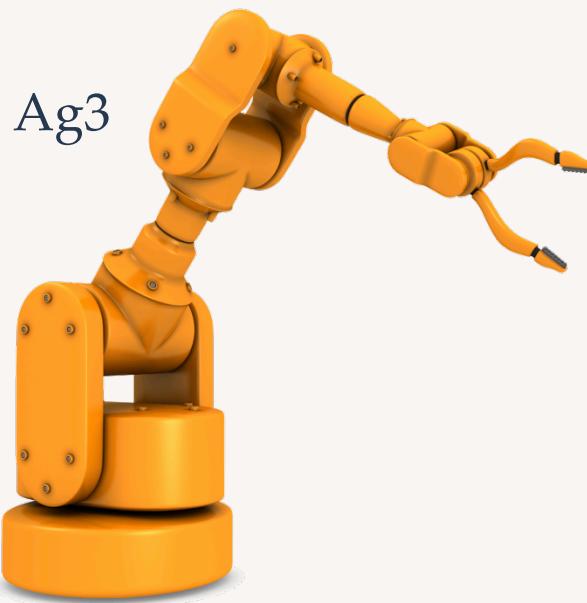
Situación Final



Ejemplo: Mundo de Bloques

– 2 robots

Creencias



```
clear(table1).  
clear(table2).  
clear(X) :- not(on(_,X)) & on(X,_).  
tower([X, table1]) :- on(X,table1).  
tower([X, table2]) :- on(X,table2).
```

// Situación inicial de los bloques y la mesas

```
on(a, table1).  
on(b, table1).  
on(c, table1).  
on(d, table2).  
on(e, table2).  
on(f, table2).
```

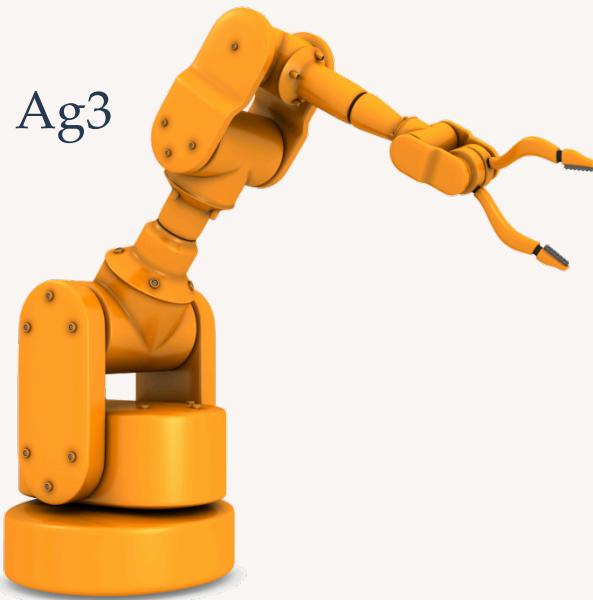
tower([X,Y|T]) :- on(X,Y) & tower([Y|T]).

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

Ejemplo: Mundo de Bloques

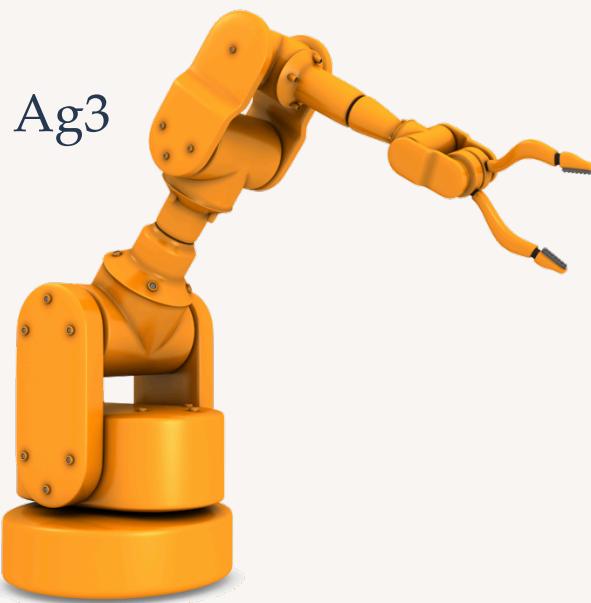
– 2 robots

Objetivos



`!state([g,d,table1]).`

Ejemplo: Mundo de Bloques – 2 robots



Planes

```
// Alcanzar una torre
+!state([]) <- .print("Finalizado!").

+!state([H,table1]) <- !on(H,table1).

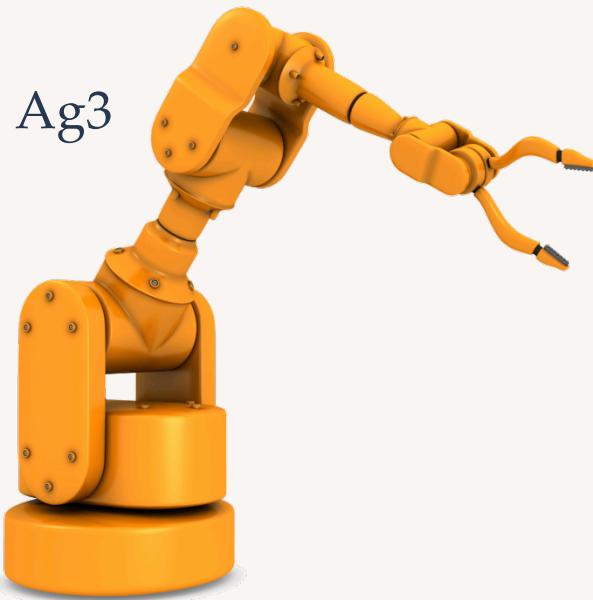
+!state([H,table2]) <- !on(H,table2).

+!state([H|T]) : .list(H) <- !tower(H); !state(T).

+!state([H|T]) : not(.list(H)) & .list(T) & .member(table1,T)
    <- !on(H,table1);
    !state(T);
    .nth(0, T, A);
    !on(H,A).

+!state([H|T]) : not(.list(H)) & .list(T) & .member(table2,T)
    <- !on(H,table2);
    !state(T);
    .nth(0, T, A);
    !on(H,A).
```

Ejemplo: Mundo de Bloques – 2 robots



Planes

```
// Alcanzar un estado donde la torre está construida
+!tower(T) : tower(T). // La torre deseada ya existe

+!tower([T,table1]) <- !on(T,table1). // torre de un solo elemento

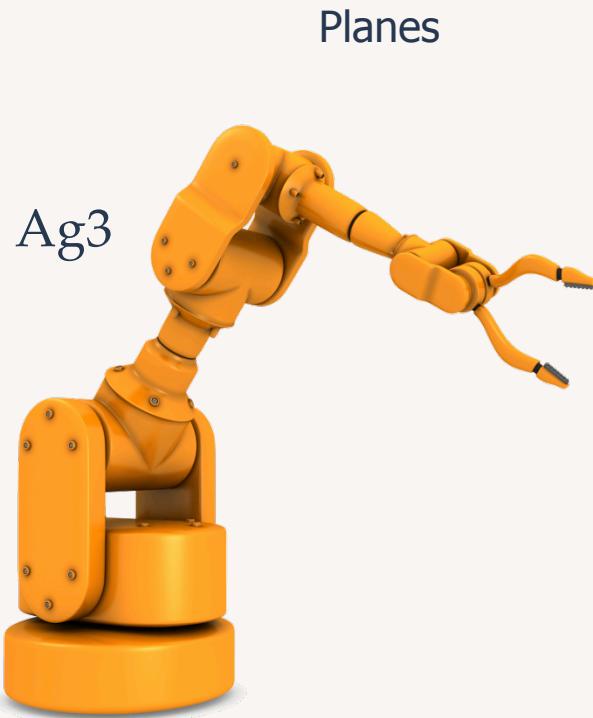
+!tower([T,table2]) <- !on(T,table2).

+!tower([X,Y|T]) <- !tower([Y|T]); !on(X,Y).

// Planes para alcanzar 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).
```

Ejemplo: Mundo de Bloques – 2 robots



Planes

```
// Planes para alcanzar la creencia de que un bloque X esté libre
+!clear(X) : clear(X). // ya conseguido

// Quita el bloque de encima cuando se necesita libre X
+!clear(X) : tower([H|T]) & .member(X,T)
  <- !move(H,table1);
    !clear(X).

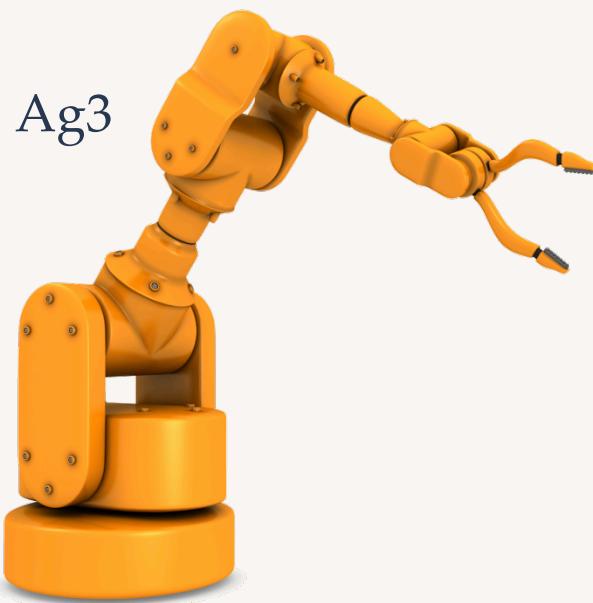
+!clear(X) : not(tower([H])) & .member(X,H))
  <- [redacted]

+!move(X, Y) : clear(X) & clear(Y)
  <- -on(X, _);
    +on(X, Y);
    .print(X, " movido sobre ", Y). /// Falta acción externa

+!move(X,Y)
  <- !clear(X); !clear(Y); !move(X,Y).
```

Ejemplo: Mundo de Bloques

– 2 robots



Planes

+!on(X,Y) : not(clear(X)) & not(on(X, _))

<-

+on(X,table2) <-