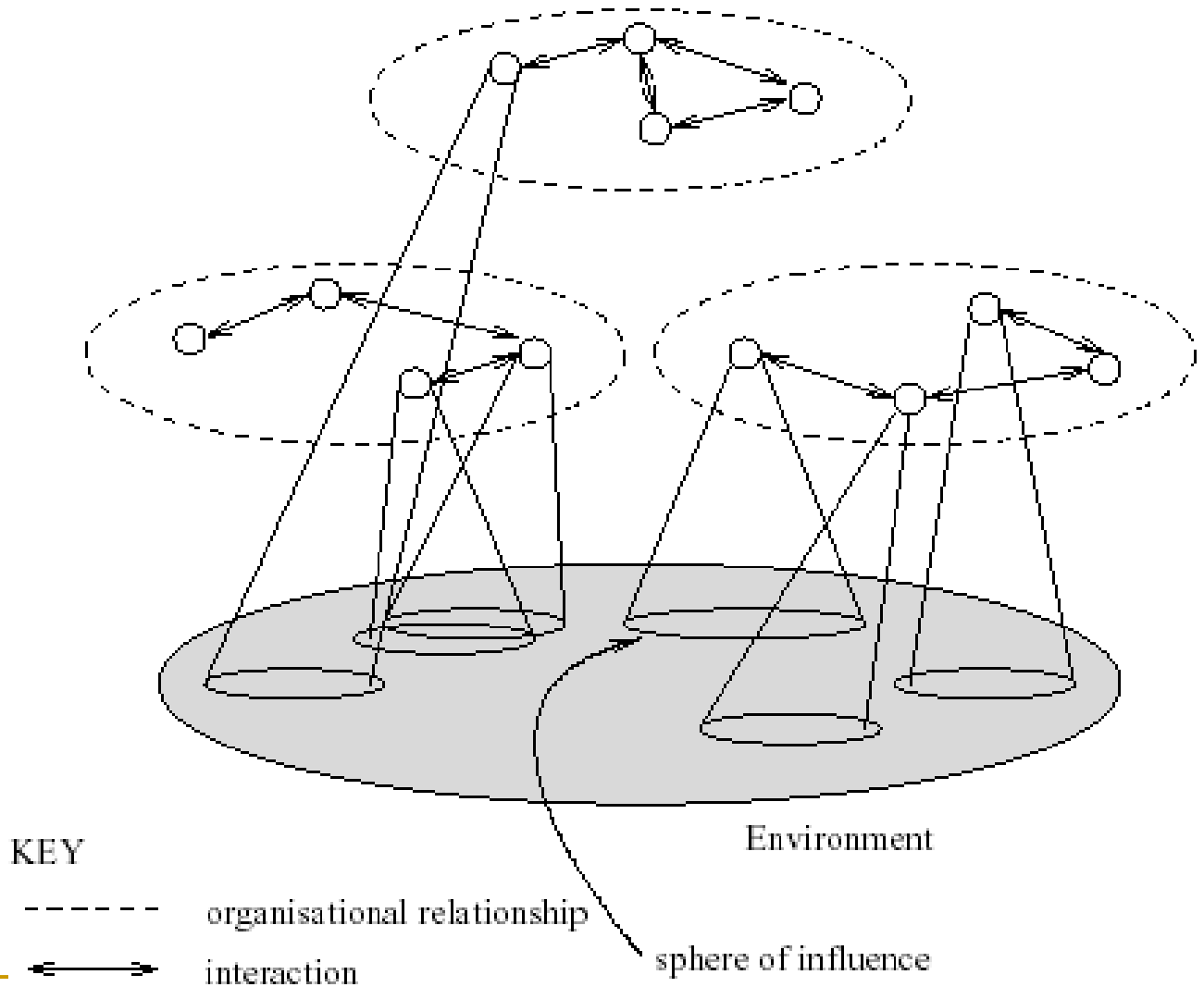
SISTEMAS MULTIAGENTES (SMA)

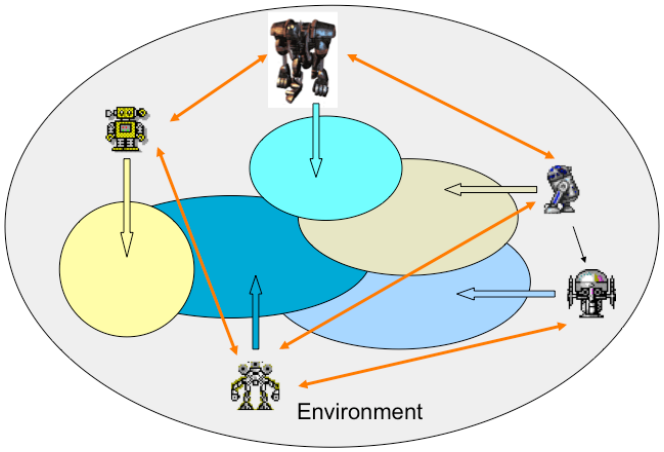
¿Qué son Sistemas *Multiagentes*?



Sistemas MultiAgentes

Luego, un sistema multiagente contiene una cantidad de agentes…

- …que interactúan a través de comunicación…
- …son capaces de actuar en un ambiente…
- …tienen diferentes “esferas de influencia” (que pueden coincidir)…
- …están unidos por otras relaciones (organizacionales)



Trabajo en conjunto

- ¿Por qué y cómo trabajan los agentes en conjunto?

- Es importante distinguir entre:
  - *Agentes benevolentes*
  - *Agentes egoístas (auto-interesados)*

Agentes Benevolentes

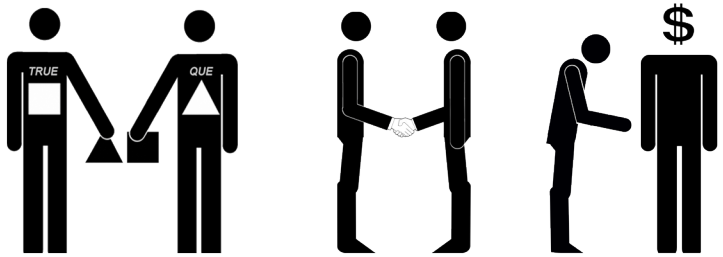
- Si el sistema completo nos “pertenece”, podemos diseñar los agentes para que se ayuden mutuamente cuando se requiera
- En este caso, podemos asumir que los agentes son *benevolentes*: nuestro interés es también su interés
- La resolución de problemas en sistemas benevolentes es *cooperative distributed problem solving* (CDPS)
- *¡La benevolencia simplifica enormemente el diseño del sistema!*

Agentes egoístas

- Si los agentes representan individuos u organizaciones, (el caso más general), entonces no podemos asumir benevolencia
- Se asumirá que los agentes actúan para alcanzar sus propios intereses, posiblemente a costa de los demás
- Existe potencial de *conflicto*
- Puede complicar enormemente el diseño

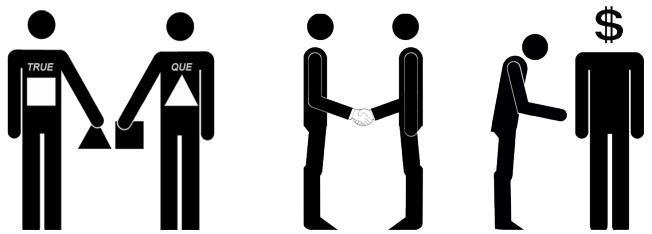
Compartir las Tareas y los Resultados

- Dos modos principales de Cooperative Distributed Problem Solving (CDPS):
  - *Compartir tareas*:
    los componentes de una tarea son distribuidos a los agentes componentes del Sistema
  - *Compartir resultados*:
    la información (resultados parciales, etc.) son distribuidos



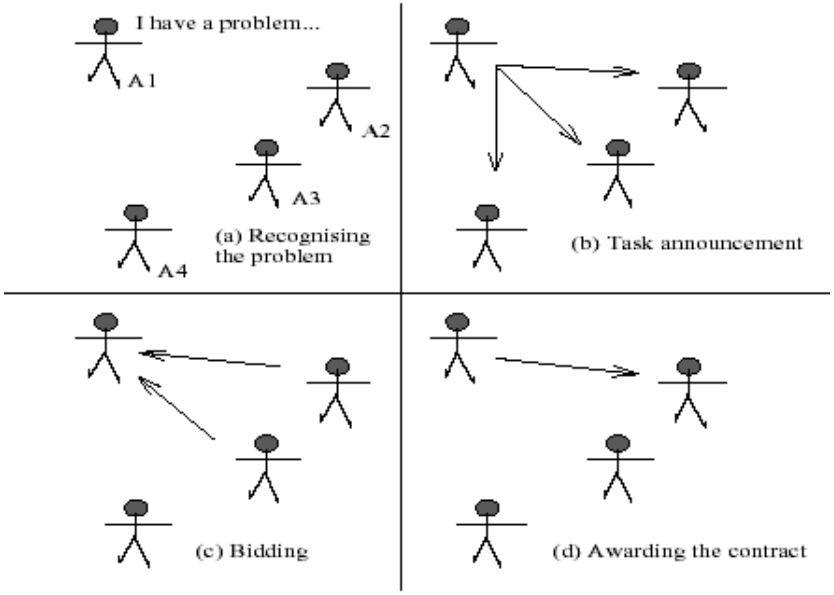
La Contract Net

- Un enfoque de CDPS basado en la distribución de tareas
- La distribución de tareas es vista como un tipo de negociación de un contrato
- Un “Protocolo” especifica el contenido de la comunicación, no solo la forma
- Transferencia en dos sentidos de la información es una extensión natural de los mecanismos de transferencia de control



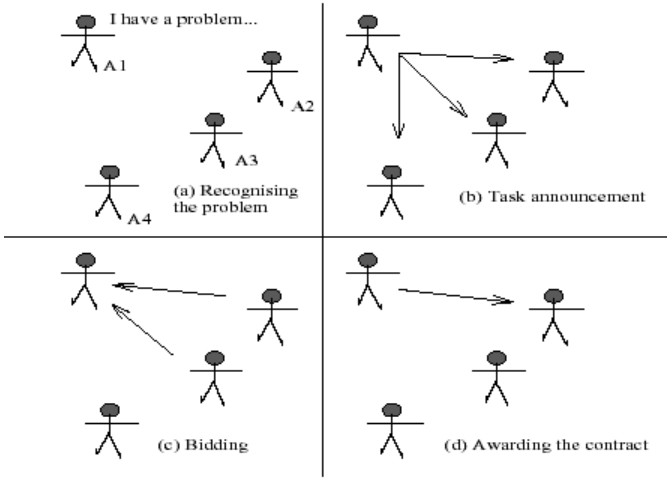
4 fases para solucionar un problema, visto por la Contract Net

1. Descomposición del problema
2. Distribución de subproblemas
3. Solución de subproblemas
4. Síntesis de las respuestas



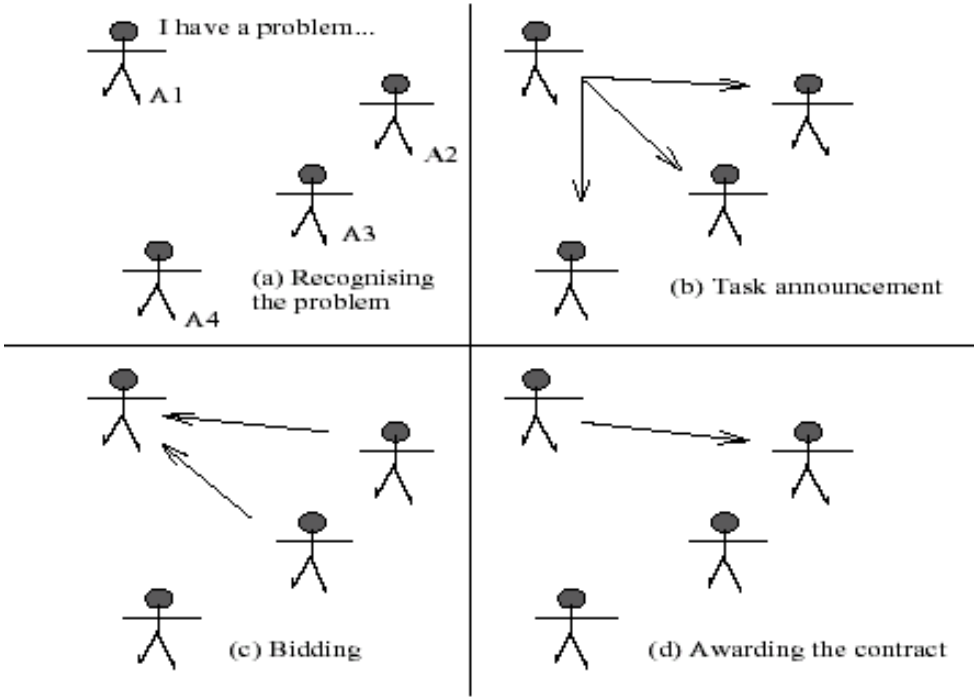
La Contract Net

- La colección de nodos es la “contract net”
- Cada nodo en la red puede, en diferentes tiempos o para diferentes tareas, ser un administrador o contratante
- Cuando un nodo recibe una tarea compuesta (o por cualquier razón no puede resolver la actual tarea):
  - La descompone en subtareas (si es posible) y
  - Las anuncia (actuando como un adm.)
  - Recibe ofertas de potenciales contratantes y
  - Asigna la tarea

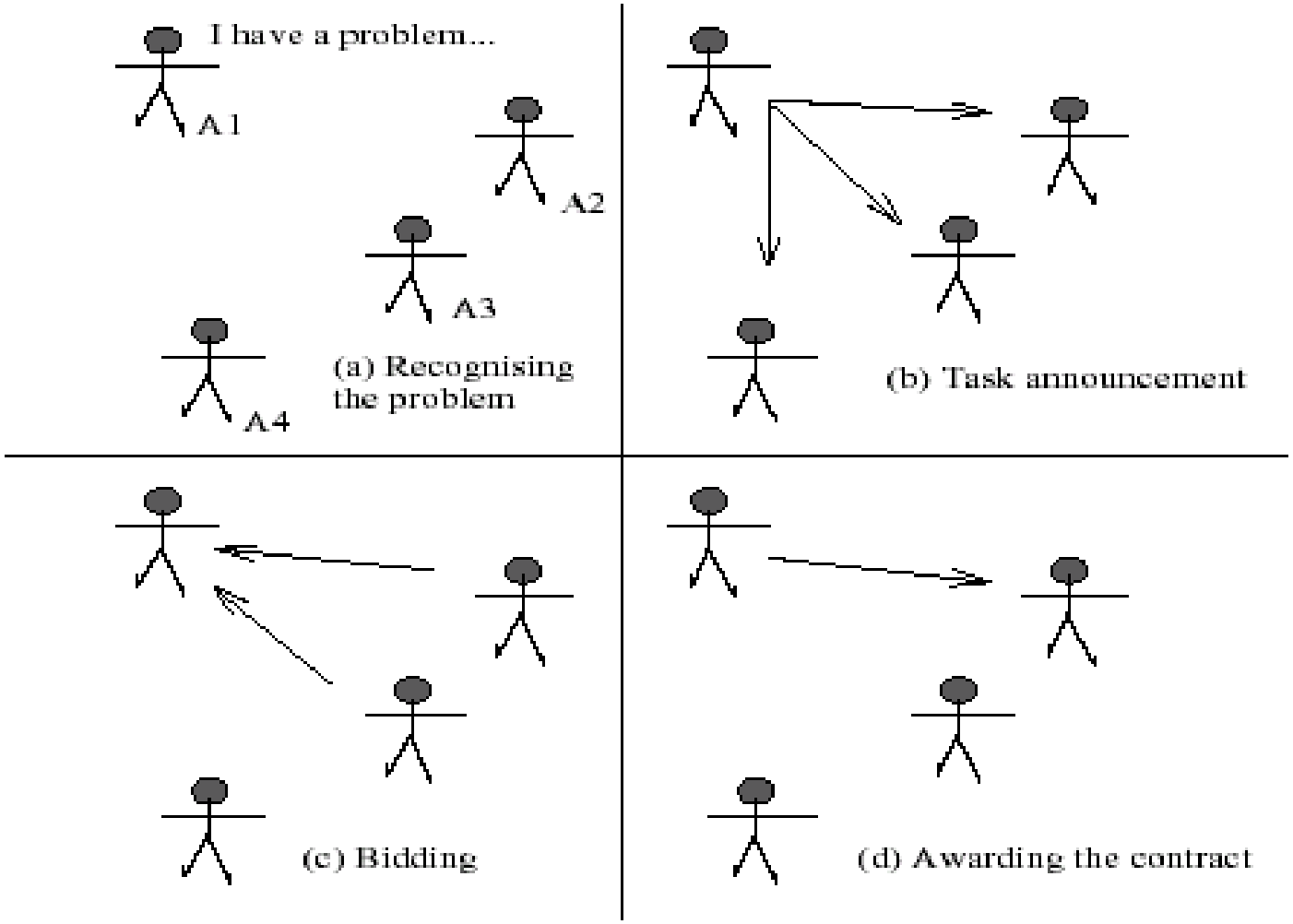


La Contract Net

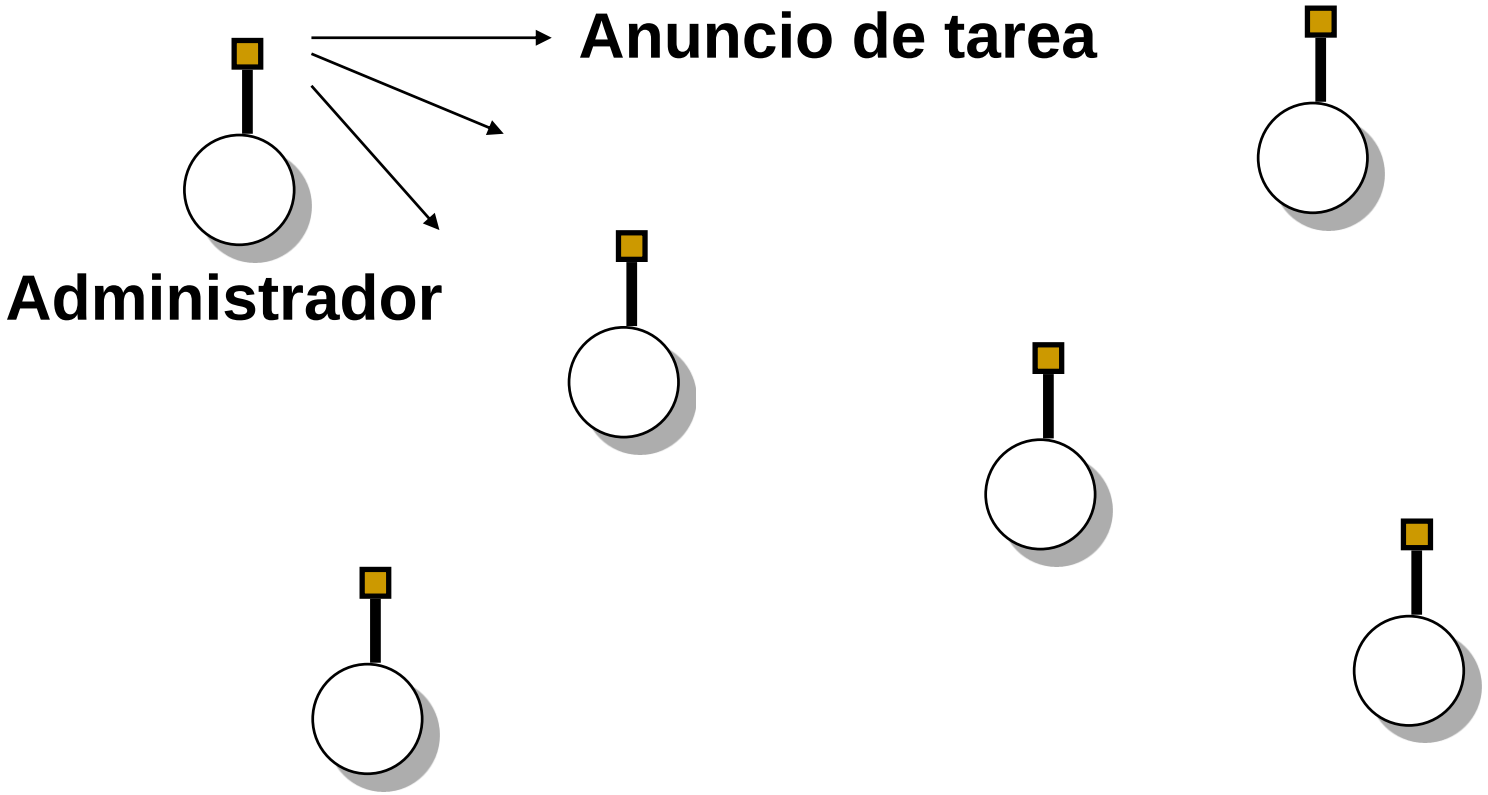
- Protocolo para compartir y asignar tareas en la *Contract Net*:
  1. Reconocimiento
  2. Anuncio
  3. Proceso de ofertas
  4. Asignación
  5. Realizar las tareas



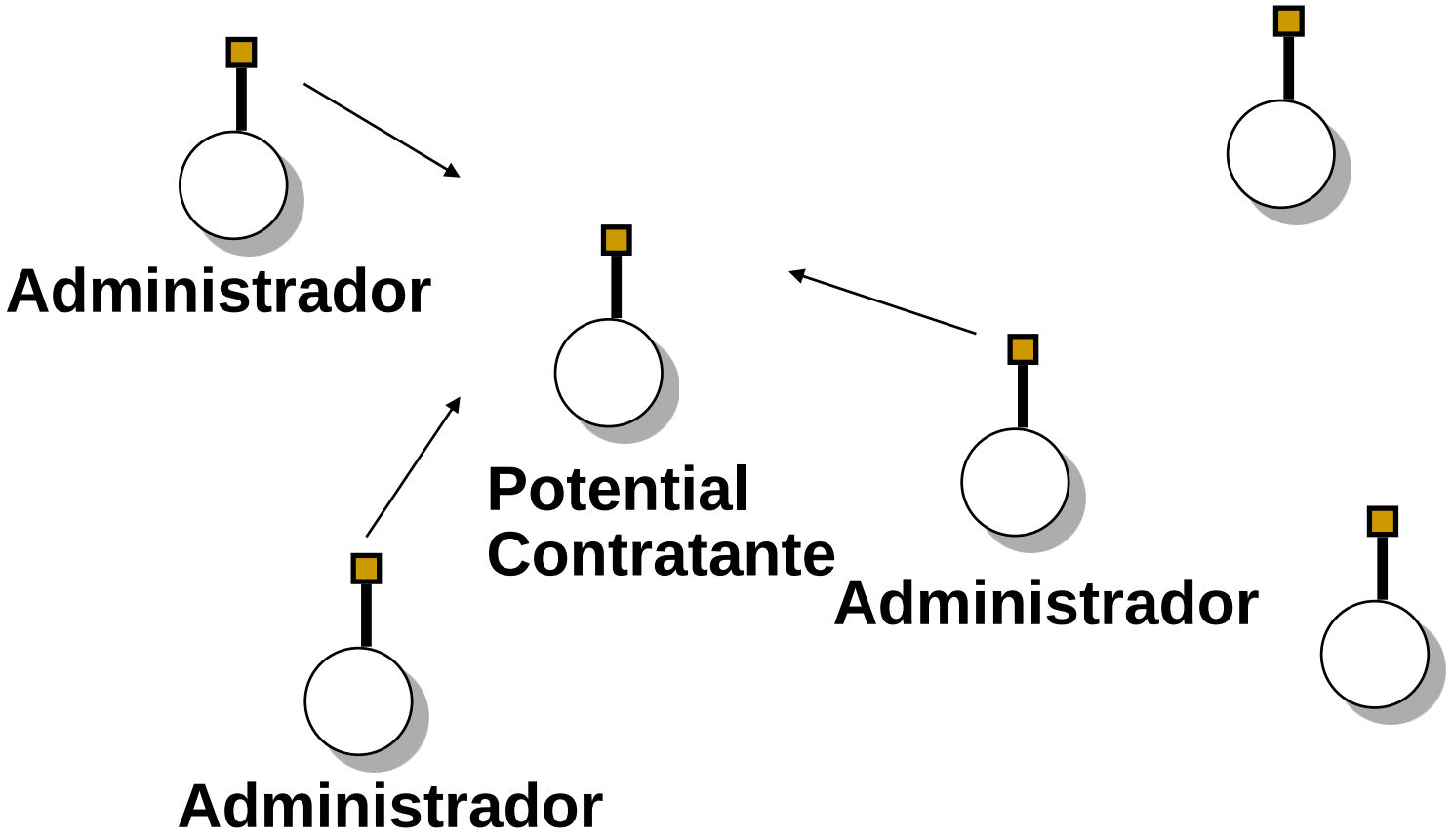
La Contract Net



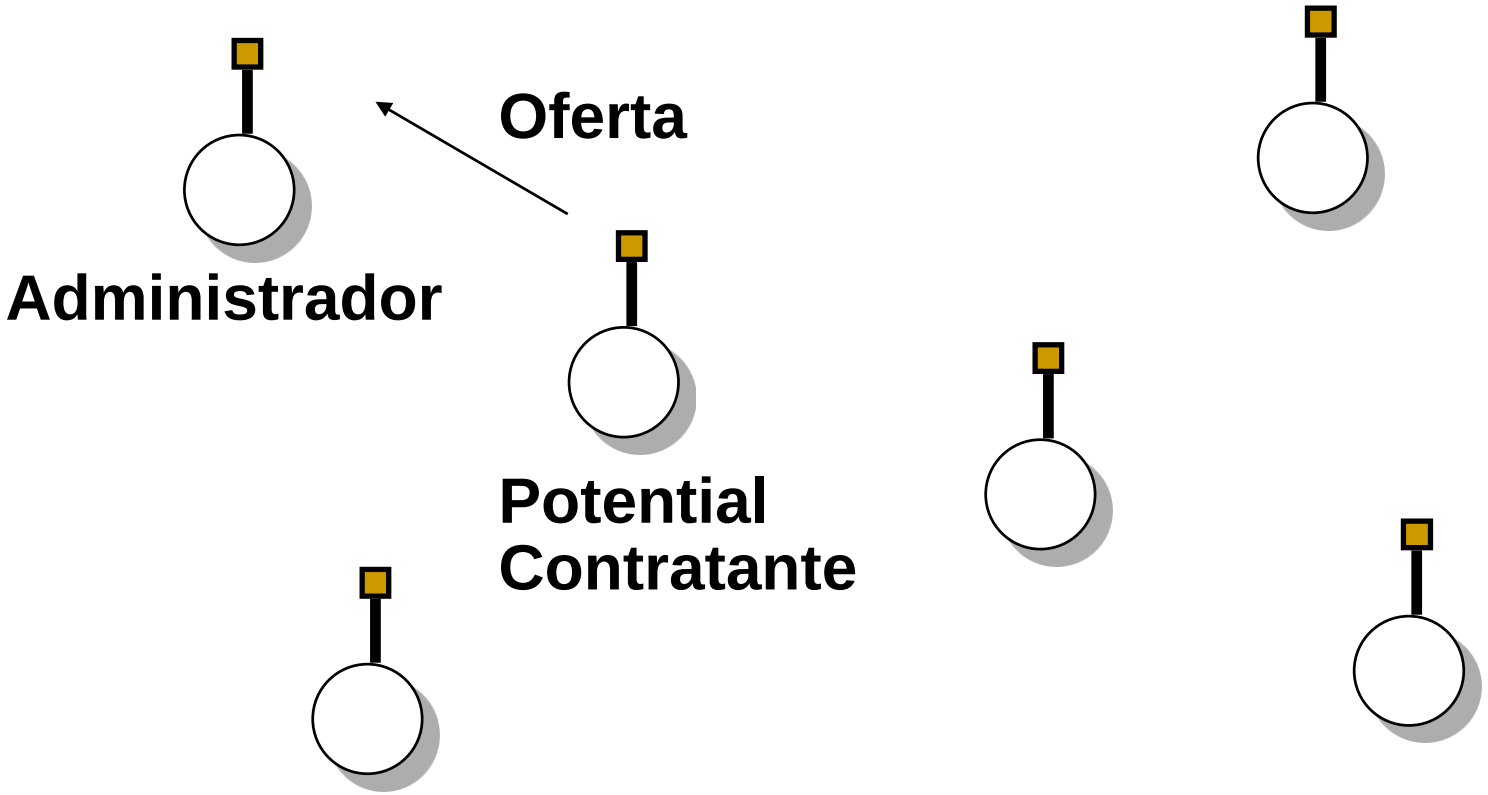
Nodo manda Anuncio de Tarea



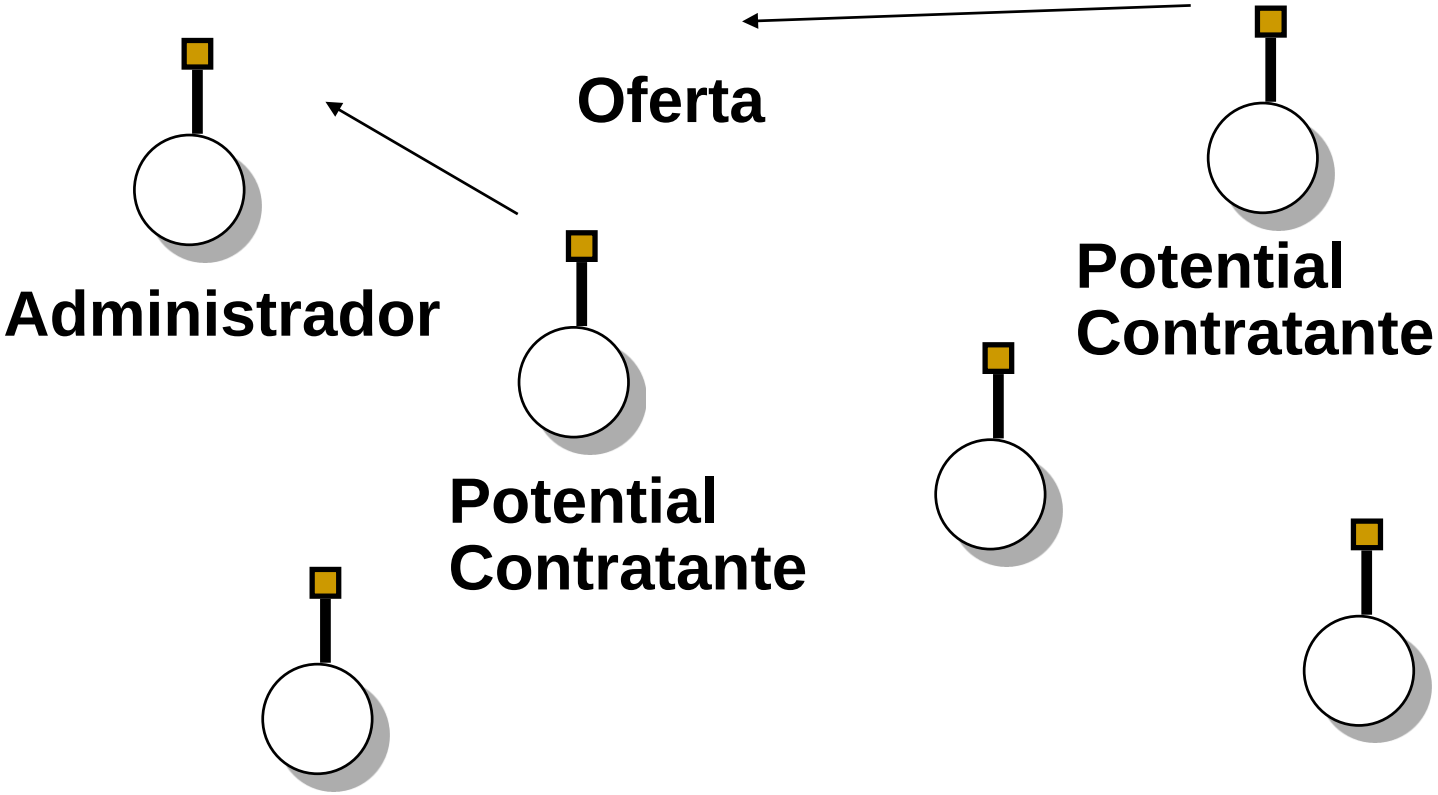
Nodo en reposo escucha el anuncio de tareas



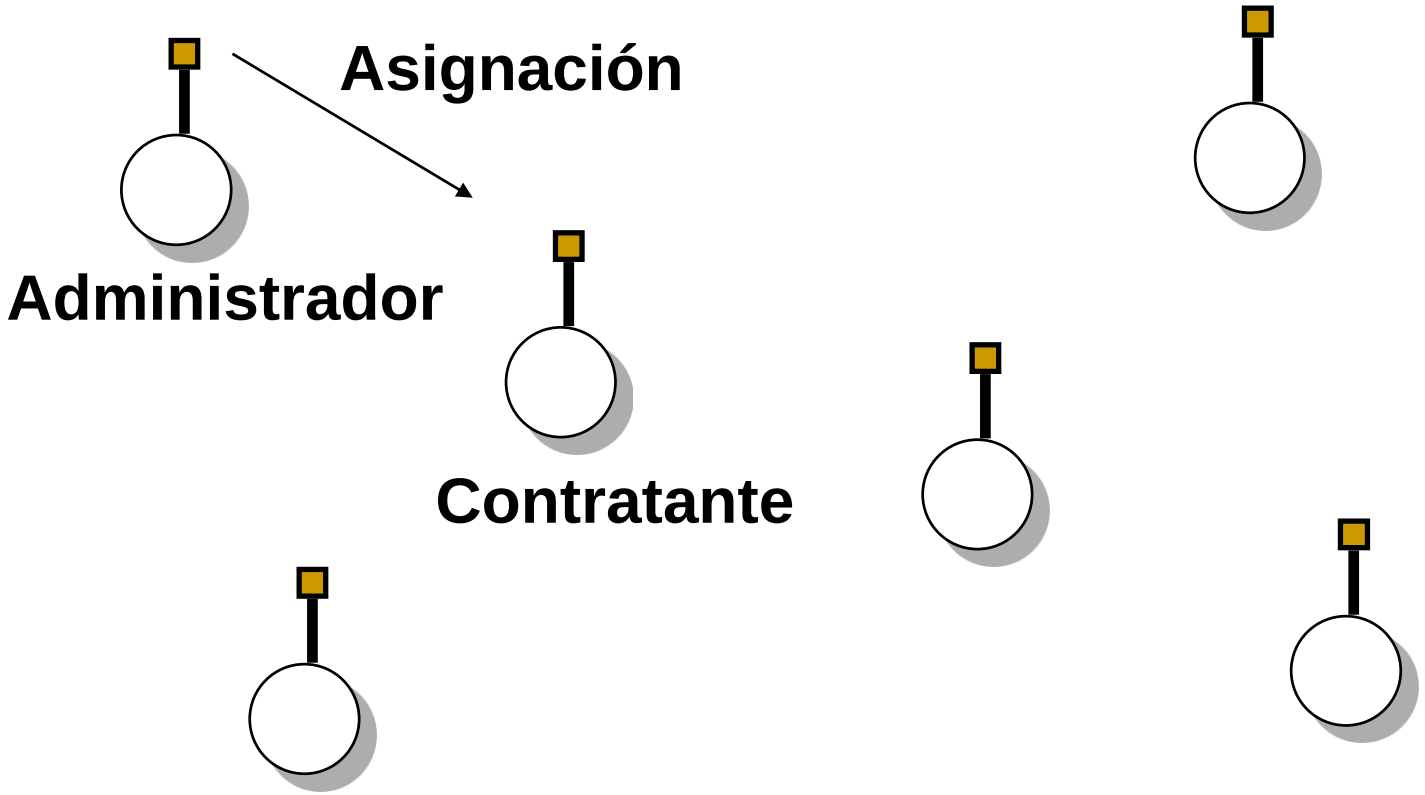
Nodo envía una oferta



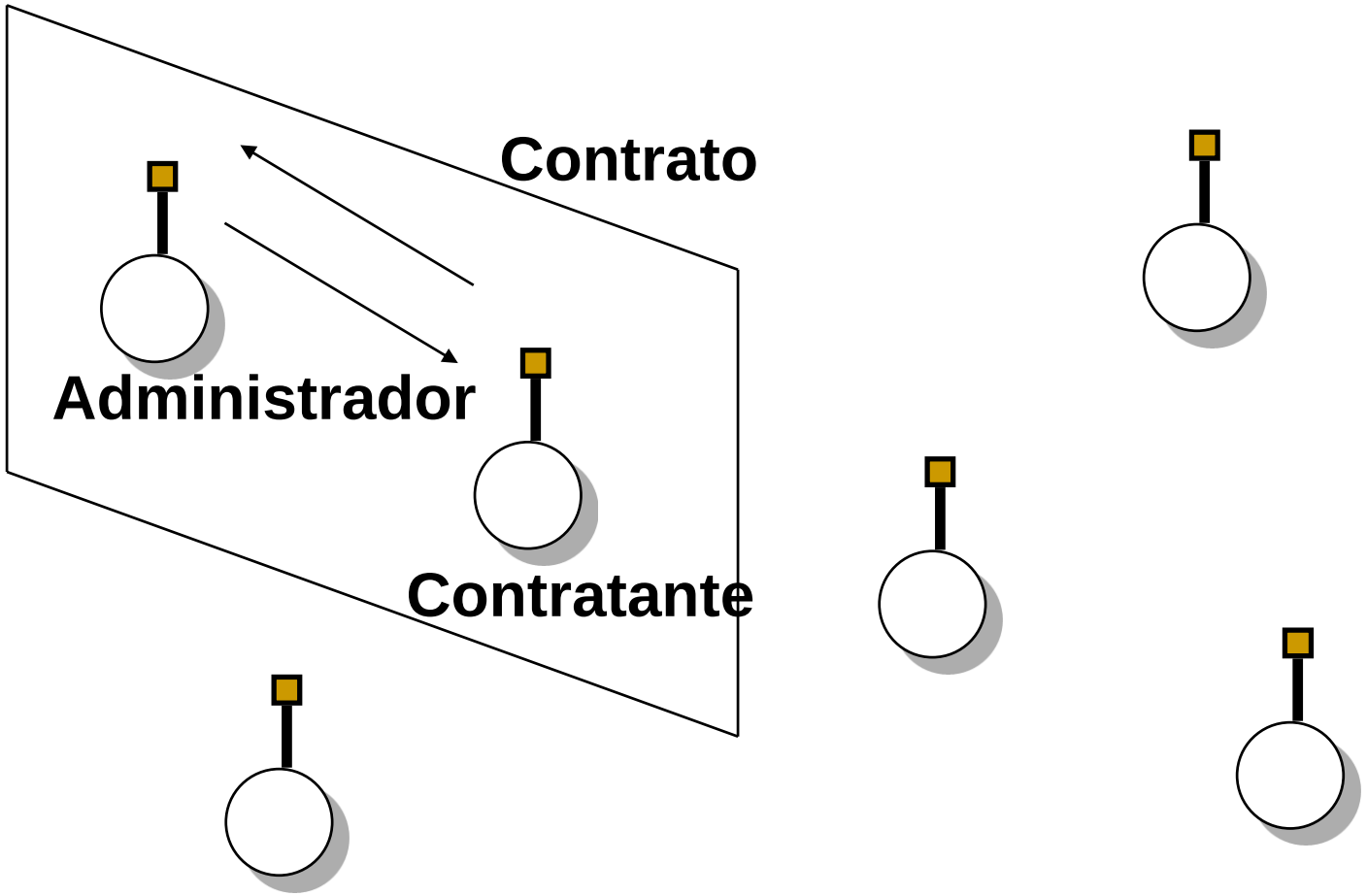
Administrador escucha las ofertas



Administrador asigna tarea



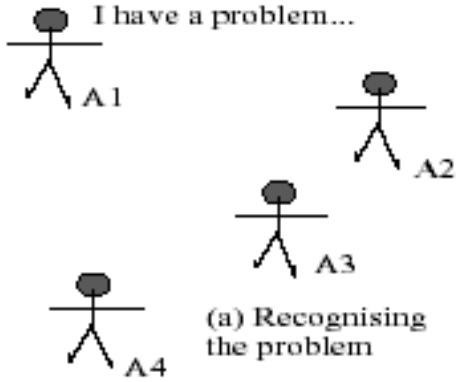
Se establece un Contrato



Reconocimiento

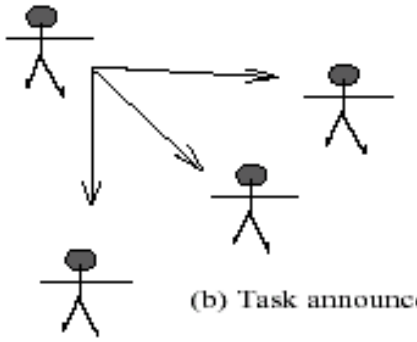
- En esta etapa, un agente reconoce que necesita ayuda para resolver un problema.

- El agente tiene una meta, y …
  - Se da cuenta que no puede lograr la meta solo por si mismo — no tiene la capacidad
  - Se da cuenta que preferiría no lograr la meta solo por si mismo (típicamente porque la solución puede ser de baja calidad, fecha de término no se puede alcanzar, etc.)



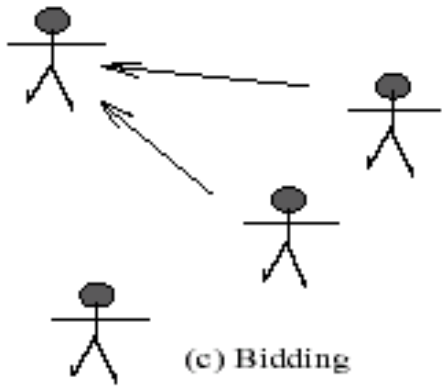
Anuncio

- En esta etapa, el agente con la tarea manda *anuncios* de la tarea incluyendo una *especificación* de la tarea a ser ejecutada
- La especificación debe contener:
  - descripción de la tarea
  - cualquier restricción (fechas límite, calidad)
  - meta-información de la tarea (“ofertas deben ser enviadas el …”)
- El anuncio es difundido (*broadcast*)



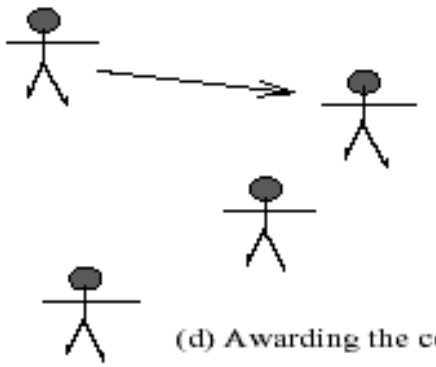
Ofertas

- Los agentes que reciben el anuncio deciden ellos mismos si desean enviar ofertas
- Factores:
  - El agente debe decidir si es capaz de realizar la tarea
  - El agente debe determinar las restricciones de calidad e información de precio (si es relevante)
- Si deciden ofertar, envían una oferta (*tender*)



Asignación y Ejecución

- El agente que envió el anuncio debe elegir entre las ofertas y decidir a quién asignar el contrato
- El resultado de este proceso es comunicado a los agentes que enviaron una oferta
- El contratante exitoso debe realizar la tarea
- Puede involucrar la generación de otros contratos: *subcontractos*



(d) Awarding the contract

Problemas para Implementar la Contract Net

- Cómo…
  - …especificar *tareas* ?
  - … especificar *calidad de servicio*?
  - …seleccionar entre distintas ofertas en competencia?
  - …diferenciar entre ofertas basadas en múltiples criterios?

Arquitectura de Pizarra (Blackboard)

- El 1er esquema para resolución cooperativa de problemas: el sistema *blackboard*
- Resultados son compartidos via una estructura compartida de datos (BB)
- Múltiples agentes pueden escribir y leer al/del BB
- Agentes escriben soluciones parciales al BB
- BB puede ser estructurado en una jerarquía
- Requiere exclusión mutua del BB $\Rightarrow$ cuello de botella
- No hay actividad concurrente

Compartir Resultados en un patrón Subscribir/Notificar

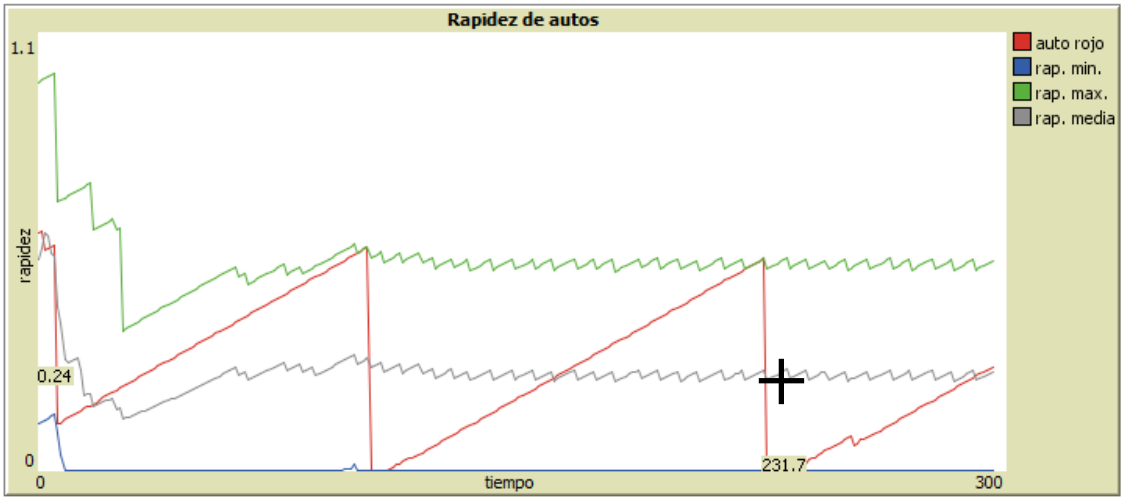
- Patrón común de diseño en OO: *subscribir/notificar*
- Un objeto se *suscribe* a otro objeto, diciéndole “dime cuando el evento e ocurra”
- Cuando el evento e ocurre, objeto original es notificado
- La información es *compartida* proactivamente entre objetos
- Los objetos requieren conocer los *intereses* de otros objetos $\Rightarrow$ informa a los objetos cuando ocurra información relevante

Coherencia y Coordinación

- La actividad de los nodos debe ser coherente con los objetivos de la red
- Los nodos:
  - Deberían evitar duplicación innecesaria de trabajo
  - No deberían estar sin hacer nada mientras otros están sobrecargados de trabajo
  - Deberían transferir información que mejore el rendimiento del sistema (y no aquella que degrade el rendimiento general)
- Dado que los nodos tienen una visión local, su contribución a la coherencia global depende de tener una buena visión de lo que está ocurriendo a nivel local

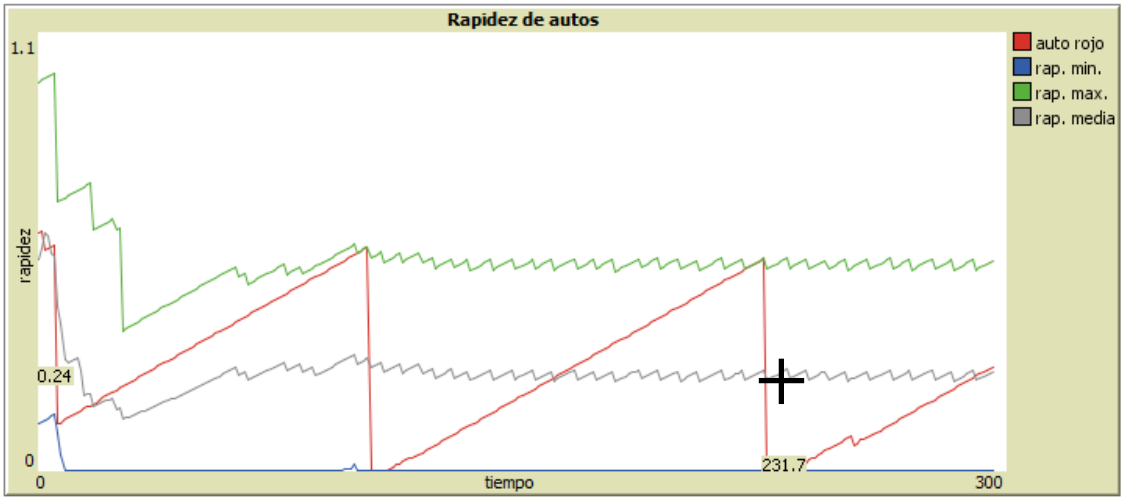
Ejercicio práctico: SMA y coordinación

1. Abra Netlogo v. 5.3.1 y el modelo Trafico_adaptativo_individual
2. Oprima inicializar y simular y vea qué pasa con la rapidez promedio (gráfico Rapidez de autos)
3. “A ojo”, vea qué rapidez promedio se obtiene y anótela
4. Oprima nuevamente simular para parar la simulación
5. Oprima inicializar y simul_adaptativa
6. “A ojo”, vea qué rapidez promedio se obtiene y anótela



Ejercicio práctico: SMA y coordinación

- Abra Netlogo v. 5.3.1 y el modelo Trafico_adaptativo_global
- Oprima inicializar y simular y vea qué pasa con la rapidez promedio (gráfico Rapidez de autos)
- “A ojo”, vea qué rapidez promedio se obtiene y anótela
- Oprima nuevamente simular para parar la simulación
- Oprima inicializar y simul_adaptativa
- “A ojo”, vea qué rapidez promedio se obtiene y anótela



Ejercicio práctico: SMA y coordinación

En general:

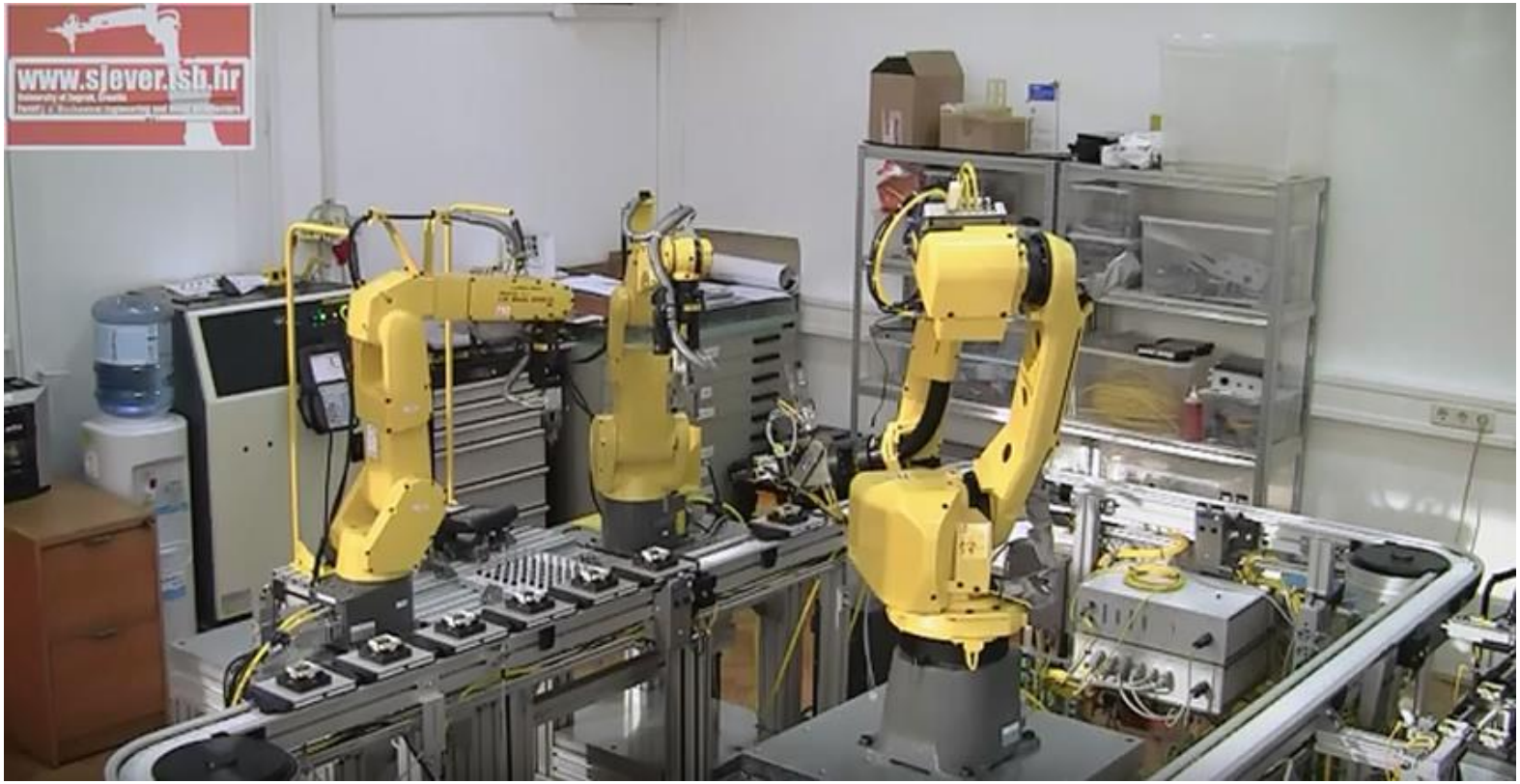
- Sistema NO adaptativo:
  - Cada auto frena cuando está muy cerca del de adelante y acelera cuando está lejos
  - Frena hasta igualar rapidez del auto de adelante

- Sistema adaptativo individual
  - Ídem al sistema no adaptativo, pero:
  - Cada auto lleva un registro individual de su rapidez media y de su aceleración y las cambia para maximizar dicha rapidez

- Sistema adaptativo global
  - Ídem al sistema adaptativo individual, pero:
  - Cada auto lleva un registro individual de su rapidez media y un registro de la aceleración media global y cambia su aceleración para maximizar dicha rapidez

SMA: ejemplos

- Multi Agent Robotic Assembly
  https://www.youtube.com/watch?v=hjK57OezXws



SMA: ejemplos

- Universal Robots Cobots
  https://www.youtube.com/watch?v=PtncirKiBXQ

SMA: ejemplos

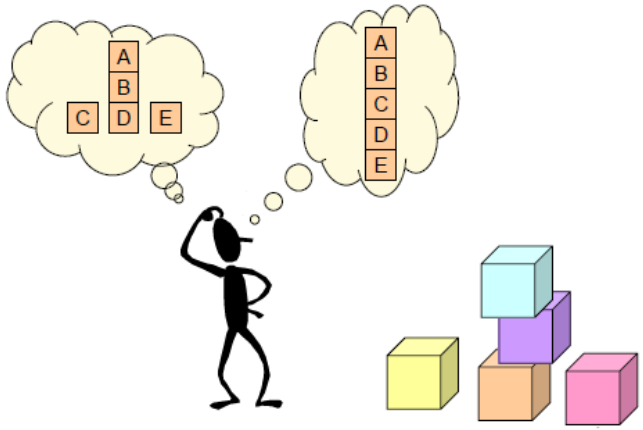
- Robocup
  https://www.youtube.com/watch?v=ZwAjHTI21Dg

Riesgos en el Desarrollo de Agentes

- *Muchos proyectos de* agentes (uno y multi) pero el desarrollo mismo de agentes ha recibido poca atención
- Ahora consideraremos aspectos *pragmáticos* de proyectos de agentes
- Identifica principales riesgos
- Siete categorías:
  - politicos
  - administrativos
  - conceptuales
  - análisis y diseño
  - nivel micro (agente)
  - nivel macro (sociedad)
  - implementación

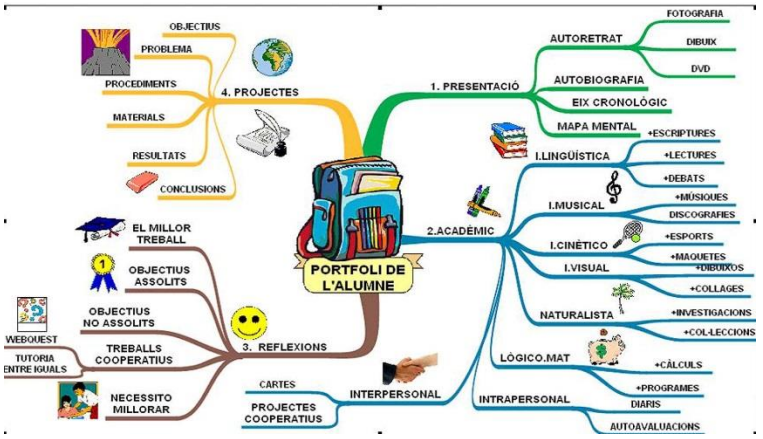
Sobrevender agentes

- ¡Los agentes *no* son mágicos!
- Si no lo puede hacer con software ordinario, probablemente tampoco lo podrá hacer con agentes
- No hay evidencia que un sistema desarrollado usando tecnología de agentes no pudiera haber sido construido usando otra tecnología
- Los agentes *pueden* facilitar la resolución de ciertas clases de problemas … pero no transforman lo imposible en posible
- Los agentes no son IA con otro nombre
- No igualar agentes con IA



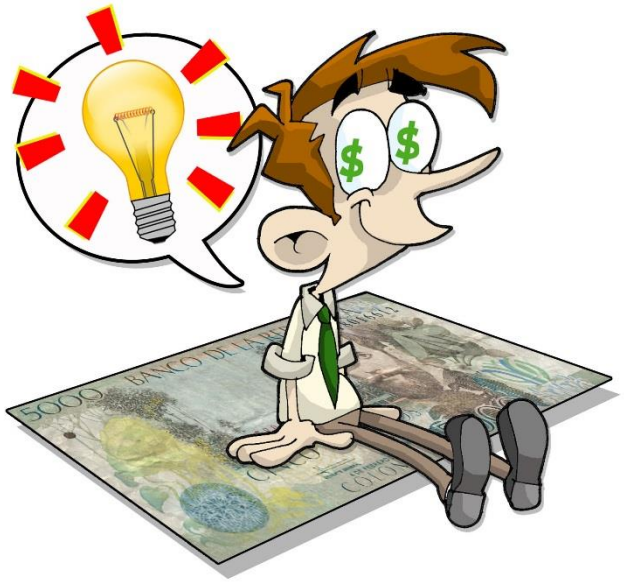
Volverse fanático de los agentes

- Los agentes han sido usados en una amplia gama de aplicaciones, pero no son una solución universal
- Para muchas aplicaciones, los paradigmas convencionales de software (p.ej. OO) son más apropiados
- ¡Dado un problema para el cual parezca igual de bueno emplear un enfoque de agentes o “sin agentes”, prefiera la solución “sin agentes”!
- En resumen: peligro de creer que los agentes son la solución correcta a *todos* los problemas



No saber por qué desea agentes

- Agentes = nueva tecnología = un montón de revuelo!
  "Los agentes generarán US$2.6 MM en ventas hacia el año 2025"
- Reacción de la Gerencia:
  "Podríamos obtener un 10% de eso"
- Los gerentes a veces proponen proyectos con agentes sin tener una clara idea de lo que los agentes les ofrecen
- Ningún *plan de negocios* para el proyecto:
  - ¿Investigación pura?
  - ¿Vendedor de tecnología?
  - ¿Vendedor de soluciones?
  - …



No saber por qué desea agentes

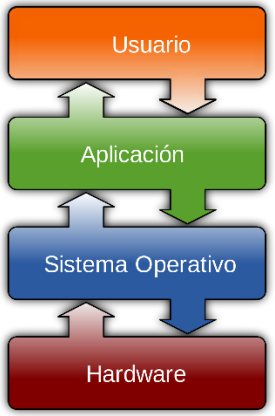
- A menudo, los proyectos *parecen* ir yendo bien. (“¡Tenemos agentes!”) pero no tenemos la visión a donde queremos llegar con ellos.
- Lección: entienda sus razones para realizar un proyecto de desarrollo de agentes y qué espera ganar con ello.

No saber para qué son buenos los agentes

- Habiendo desarrollado una tecnología de agentes, se busca una aplicación para usarla
- ¡Poner la carreta delante de los bueyes!
- Lleva a descalces/insatisfacción
- Lección:
  - Asegúrese de entender cómo y dónde la tecnología puede ser utilizada más fructíferamente.
  - No intenté aplicarla a problemas arbitrarios y resista la tentación de aplicarla a cualquier problema.

Soluciones genéricas para un problema

- El síndrome de “otra plataforma de agentes testeada”
- Crear una arquitectura o plataforma que supuestamente permite la construcción de una gran variedad de sistemas, siendo que se necesita realmente un sistema
- El reuso es difícil de lograr a menos que el desarrollo sea hecho para un tipo de problemas restringido y con caraterísticas similares
- Las soluciones generales son más difíciles y costosas de desarrollar, a menudo necesitan adaptarse a diferentes aplicaciones.



Confundir “Bla bla” y Conceptos

- La idea de agente es extremadamente intuitiva
- Alienta a mucho “bla bla” que desinforma
- Alienta a desarrolladores a creer que entienden los conceptos, pero no es así
- Por ejemplo, algunos llaman “BDI” a páginas Web y scripts de perl

Olvidar que es *Software*

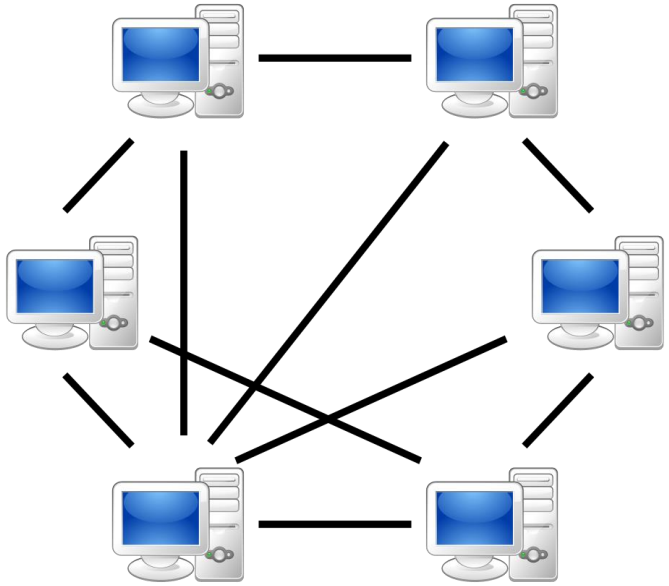
- Al desarrollar agentes se olvida que está desarrollando *software*
- Los planes del proyecto se enfocan en los “bits de los agentes”
- Ingeniería de software mundana (análisis de requerimientos, especificación, diseño, verificación, testeo) es olvidada
- Resultado: el proyecto falla, no porque hayan problemas con los agentes, sino porque se ignoraron conceptos básicos de la Ingeniería de Software
- Justificación frecuente : la Ingeniería de Software para sistemas de agentes no existe
- Pero casi cualquiera técnica de desarrollo de software es mejor que ninguna.

No explotar Tecnología relacionada

- En cualquier sistema de agentes, el porcentaje del sistema que está basado en agentes es comparativamente pequeño
- Por ello es importante explotar tecnologías y técnicas convencionales donde sea posible
- La explotación de tecnología relacionada:
  - Apura el desarrollo
  - Evita reinventar la rueda
  - Concentra el esfuerzo en los componentes de agentes

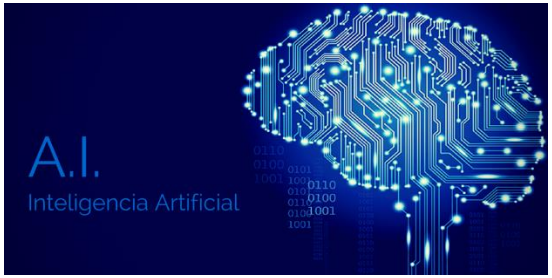
No explotar concurrencia

- Una de las propiedades más obvias de un sistema de agentes diseñado probremente es el poco uso de concurrencia para la resolución del problema o a veces es incluso inexistente
- ¡Procesamiento serial en un sistema distribuido!
- ¿Si no explota concurrencia, para qué desarrollar una solución basada en agentes?



Usar mucha IA

- Tentación de enfocarse en los aspectos específicos de agentes de la aplicación
- Resultado: una infraestructura de agentes demasiado cargada de técnicas experimentales de IA para ser útil
- Debido a la “envidia” de tener agentes que aprendan, planifiquen, hablen, bailen …
- Resista la tentación de creer que esas propiedades son esenciales para el sistema
- Lección: construya agentes con un mínimo de IA; cuando se tenga éxito con eso, progresivamente incluya más propiedades
- Lo que se llama la estrategia de “lo útil primero”



No suficiente IA

- No llame a un interruptor eléctrico un agente
- Otro ejemplo común: llamar a las páginas web que tienen algún tipo de procesamiento como “agentes”
- Problemas:
  - Lleva a que el término “agente” pierda su verdadero significado
  - Aumenta la expectativa de los usuarios
  - Lleva a que los desarrolladores pierdan confianza en los agentes

Ver agentes por todos lados

- Sistema “Puro” A-O = todo es un agente!
  Agentes para sumar, restar,…
- Ingenuamente ver *todo* como un agente es inapropiado
- Más de 10 tipos diferentes de agentes = sistema grande

Demasiados agentes

- Los agentes no deben ser complejos para generar comportamiento complejo
- Gran número de agentes:
  - *Funcionalidad emergente*
  - *Comportamiento caótico*
- Lecciones:
  - Mantenga las interacciones a un mínimo
  - Mantenga los protocolos simples

Muy pocos agentes

- Algunos diseñadores imaginan un agente separado para cada tarea posible
- Otros no reconocen el valor de un enfoque multiagente
- Un agente “superpoderoso”
- El resultado es como un programa OO con una clase
- No pasa la prueba de *coherencia* de la ingeniería de software