Programación de SMAs

Juan A. Botía

Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

5º Curso, Ing. Superior en Informática

1. Introducción a JADE
   - Comportamientos en JADE
   - Comunicación entre agentes con JADE
   - Protocolos de interaccíon FIPA en JADE
   - Manejo de Ontologías en JADE

2. INGENIAS IDK
   - La metodología INGENIAS
   - Modelado de conceptos típicos de agentes

3. INGENIAS
   - Meta-modelo de Tareas y Objetivos
   - Meta-modelo de Interacciones
   - Meta-modelo de agente
   - Análisis y diseño en INGENIAS

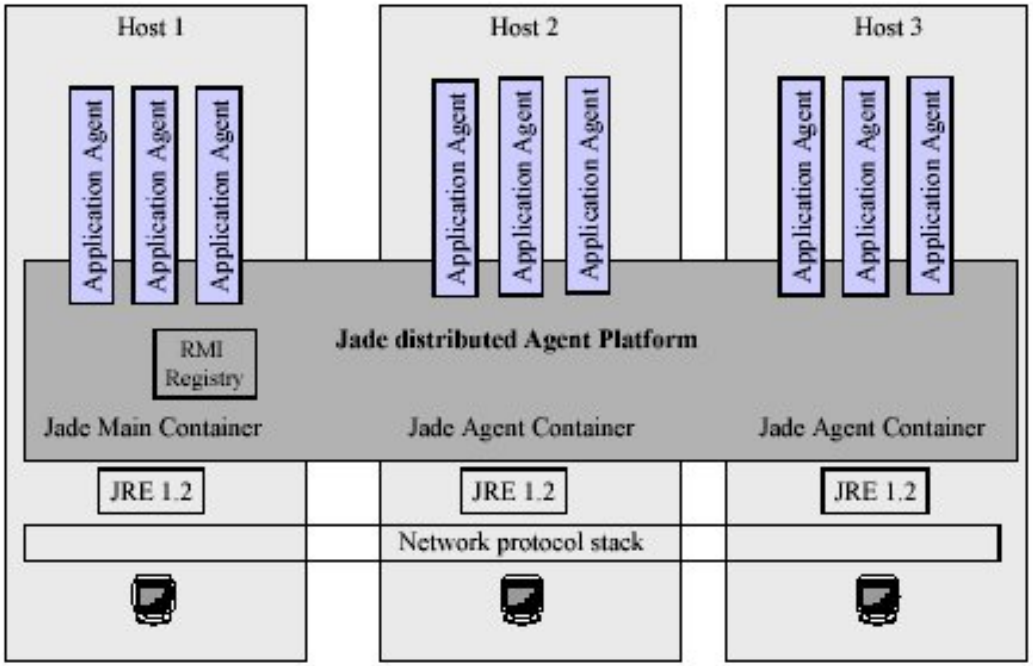
La plataforma JADE

- JADE está compuesta de
  - Una plataforma FIPA para la ejecución de agentes
  - Un conjunto de paquetes para la programación de agentes FIPA
- Es 100% Java (con el JDK 1.4 o superiores)
- Incluye
  - Creación básica de agentes
  - Programación del comportamiento de los agentes en base a *behaviors*
  - ACL FIPA para envío y recepción de mensajes
  - Clases útiles para programación de protocolos FIPA (y no FIPA)
  - Distintos codecs (SL, RDF, etc)
  - Manejo de información usando ontologías

La plataforma JADE

- Plataforma FIPA (AMS, Facilitador de directorio y MTS)
- Puede ejecutarse en una o varias JVM
- Cada JVM es vista como un entorno en donde los agentes pueden ejecutarse concurrentemente e intercambiarse mensajes
- Organizada en contenedores
  - 1 principal: AMS, DF y el registro rmi están localizados ahí
  - *n* containers no princpales y conectados al principal

La plataforma JADE (y II)



Servicios básicos: directorio

El directorio, como en FIPA, es un servicio básico accesible a través de `jade.domain.DFService` (en realidad es un acceso al agente de páginas amarillas desde un interface) para los servicios

- `register`
- `deregister`
- `modify`
- `search`

La clase `Agent`

Programar un agente en JADE consiste en definir una clase Java que representa al agente y:

- Determinar y codificar los comportamientos que va a *manifestar*
- Hacer que herede de la clase `jade.core.Agent`
- Programar sus métodos `setup()`, `takeDown`

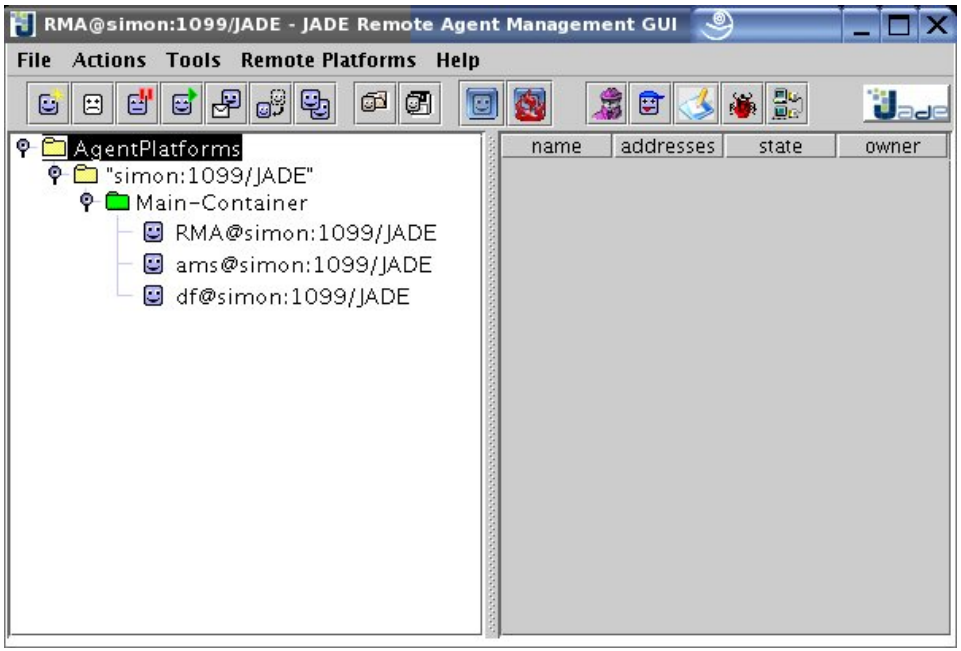
Para ejecutar el agente, podemos hacerlo desde el GUI de JADE o desde cualquier otro programa JAVA explícitamente

Lanzar un agente desde el GUI

Desde una consola:

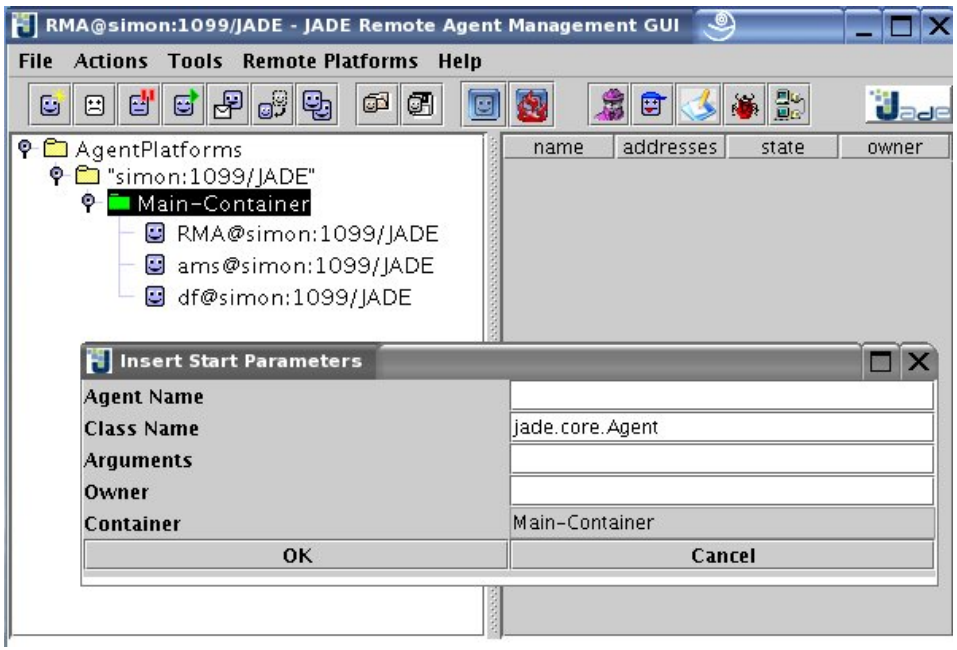
```
export CLASSPATH=/ruta/jade/lib/jade.jar:/ruta/jade/lib/iiop.jar:/ruta/jade/lib/http.jar
java jade.Boot -gui
```

y tenemos



Lanzar un agente desde el GUI (y II)

Seleccionando el botón de `New Agent`, posicionados en el contenedor principal



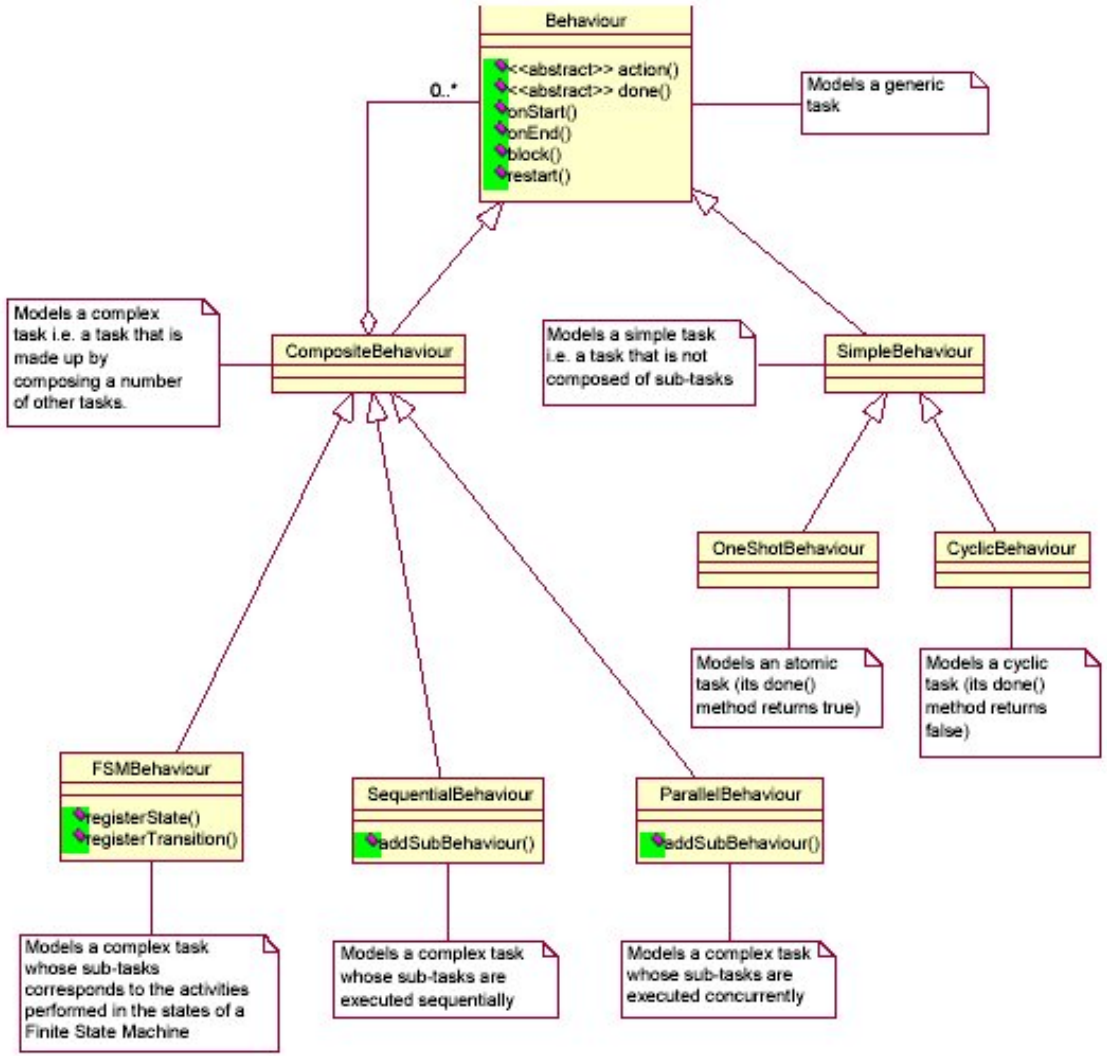
Los Comportamientos de un agente JADE

- Los agentes deben poder ejecutar diferentes tareas al mismo tiempo
- JADE propone un modelo de agente *single threaded* y añade un nivel de scheduling sobre la única thread a nivel de comportamientos
- Programación basada en comportamientos:
  1. determinar qué debe ser capaz de hacer el agente
  2. asociar cada funcionalidad con un comportamiento
  3. escoger el tipo de comportamiento
  4. dejar a JADE la tarea del scheduling (un solo comportamiento se está ejecutando en cada instante)

El scheduling de comportamientos

- Cada agente tiene para sí una cola de comportamientos activos
- El cuerpo de acciones de un comportamiento se programa redefiniendo el método action()
- Cuando el método anterior finaliza, y dependiendo del tipo de comportamiento, el scheduler lo saca de la cola o lo vuelve a colocar al final
- Un comportamiento puede bloquearse (`block()`) hasta que lleguen más mensajes al agente; el bloqueo significa que, cuando `action()` termina, se le colocal en una cola de bloqueados
- Cuando llega un nuevo mensaje, se le saca de esa cola y se coloca al final de la de comportamientos activos

Los Comportamientos de un agente JADE (y II)



El ACL estándar de FIPA en JADE

Ideas principales:

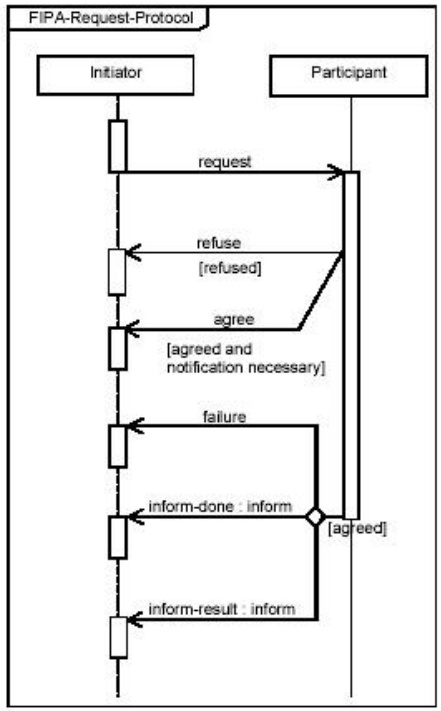
- La clase `jade.lang.acl.ACLMessage` es la base para composición de mensajes (métodos set y get para todos los parámetros de un mensaje FIPA)
- Los métodos `Agent.send(...)`, `Agent.receive(...)` y `Agent.blockingReceive()` para envío y recepción
- La clase `jade.lang.acl.MessageTemplate` es útil para hacer matching de mensajes

Protocolos de Interacción

- FIPA define, como vimos ayer, protocolos de interacción estándares
- Filosofía de JADE: en lugar de programar el flujo del protocolo (i.e. la secuencia de intercambio de mensajes) programar qué hacer en cada situación (manejadores)
- Basado en las clases `jade.proto.AchieveREInitiator` y `jade.proto.AchieveREResponder` (suficientes para implementar FIPA-Request, FIPA-query, FIPA-Request-When, FIPA-recruiting, FIPA-brokering y FIPA-subscribe)
- Para el FIPA-contract-net disponemos de `jade.proto.ContractNetInitiator` y `jade.proto.ContractNetResponder`

Progamación de una interacción

Si queremos programar una interacción simple (i.e. 1 a 1), como por ejemplo `fipa-request`, echamos mano del estándar para el protocolo de interacción



Para programar el iniciador, nos fijamos en los mensajes entrantes y análogamente para el *responder*

Progamación de una interacción (y II)

Nos vamos al `javadoc` de `jade.proto.SimpleAchieveREInitiator`

Method Summary

void	**action**() Runs the behaviour.
boolean	**done**() Check if this behaviour is done.
protected void	**handleAgree**(ACLMessage msg) This method is called every time an agree message is received, which is not out-of-sequence according to the protocol rules.
protected void	**handleAllResponses**(java.util.Vector msgs) This method is called when all the responses have been collected or when the timeout is expired.
protected void	**handleAllResultNotifications**(java.util.Vector msgs) This method is called when all the result notification messages have been collected.
protected void	**handleFailure**(ACLMessage msg) This method is called every time a failure message is received, which is not out-of-sequence according to the protocol rules.
protected void	**handleInform**(ACLMessage msg) This method is called every time a inform message is received, which is not out-of-sequence according to the protocol rules.
protected void	**handleNotUnderstood**(ACLMessage msg) This method is called every time a not-understood message is received, which is not out-of-sequence according to the protocol rules.
protected void	**handleOutOfSequence**(ACLMessage msg) This method is called every time a message is received, which is out-of-sequence according to the protocol rules.
protected void	**handleRefuse**(ACLMessage msg) This method is called every time a refuse message is received, which is not out-of-sequence according to the protocol rules.
void	**onStart**() This method is just an empty placeholders for subclasses.
protected ACLMessage	**prepareRequest**(ACLMessage msg) This method must return the ACLMessage to be sent.
void	**reset**() This method resets this behaviour so that it restarts from the initial state of the protocol with a null message.
void	**reset**(ACLMessage msg) This method resets this behaviour so that it restarts the protocol with another request message.

Progamación de una interacción (y III)

Nos vamos al `javadoc` de `jade.proto.SimpleAchieveREResponder`

Method Summary

static MessageTemplate	**createMessageTemplate**(java.lang.String iprotocol) This static method can be used to set the proper message Template (based on the interaction protocol and the performative) into the constructor of this behaviour.
boolean	**done**() This method checks whether this behaviour has finished or not.
protected ACLMessage	**prepareResponse**(ACLMessage request) This method is called when the initiator's message is received that matches the message template passed in the constructor.
protected ACLMessage	**prepareResultNotification**(ACLMessage request, ACLMessage response) This method is called after the response has been sent and only when one of the folliwing two cases arise: the response was an agree message OR no response message was sent.
void	**reset**() Reset this behaviour using the same MessageTemplate.
void	**reset**(MessageTemplate mt) This method allows to change the MessageTemplate that defines what messages this FIPARequestResponder will react to and reset the protocol.

Ontologías básicas en JADE

- JADE permite el manejo de ontologías para
  - Representar el dominio de aplicación mediante conceptos, predicados, acciones, agentes, etc.
  - Intercambiar elementos de la ontología entre agentes (i.e. en el cuerpo del mensaje FIPA)
- + Conceptualmente sencillo
- - Muy engorroso de manejar (definición tediosa de conceptos)
- Solución: podemos utilizar un sistema gestor de ontologías como Protégé2000 para el modelado y generación de código Java-JADE

Manejo de ontologías desde Protégé2000

Los pasos a seguir en la programación de un sistema JADE sencillo, usando Protégé podrían ser los siguientes:

1. Conceptualización del problema
   - Definición de los elementos a participar en la ontología
   - Definición de los agentes
   - Definición de las interacciones que tendrán lugar entre los agentes (comprobando que la ontología sea adecuada para todos los casos)
2. Diseño de la ontología con Protégé
3. Programación de los protocolos de interacción con JADE (i.e. en forma de comportamientos), integrando el código generado por Protégé
4. Programación de los agentes
5. Y ya está

Conclusiones

Para dominar JADE, tenemos que conocer

- Agentes y behaviours (mecanismos básicos de scheduling)
- Mensajes y plantillas para el matching
- Protocolos de interacción y su programación mediante clases Initiator y Responder
- Manejo de Ontologías con algún programa de apoyo (e.g. Protégé)

Manos a la obra...

Conclusiones iniciales

Hasta ahora, hemos visto como construir sistemas de agentes

- artesanalmente
  - Poca productividad
  - Curva de aprendizaje muy elevada
- ¿Podemos poner el énfasis en la productividad manteniendo la metáfora?
  - Podemos hacer uso de metodologías
  - Podemos diseñar con meta-modelos
  - Hasta disponemos de IDEs con esos meta-modelos integrados (IDK)

Construyendo sistemas multi-agente

Una metodología de desarrollo de software está compuesta de

- Un lenguaje de modelado para realizar el diseño
- Un proceso software que define las actividades de desarrollo y sus interrelaciones

¿Qué podemos encontrar para desarrollo de SMA?

- Metodologías de análisis y diseño (e.g. GAIA)
- Metodologías de análisis, diseño e implementación (Tropos, MASE, INGENIAS, MAS-CommonKADS)
- Lenguajes de modelado per se (AUML, UML 2.0)

INGENIAS

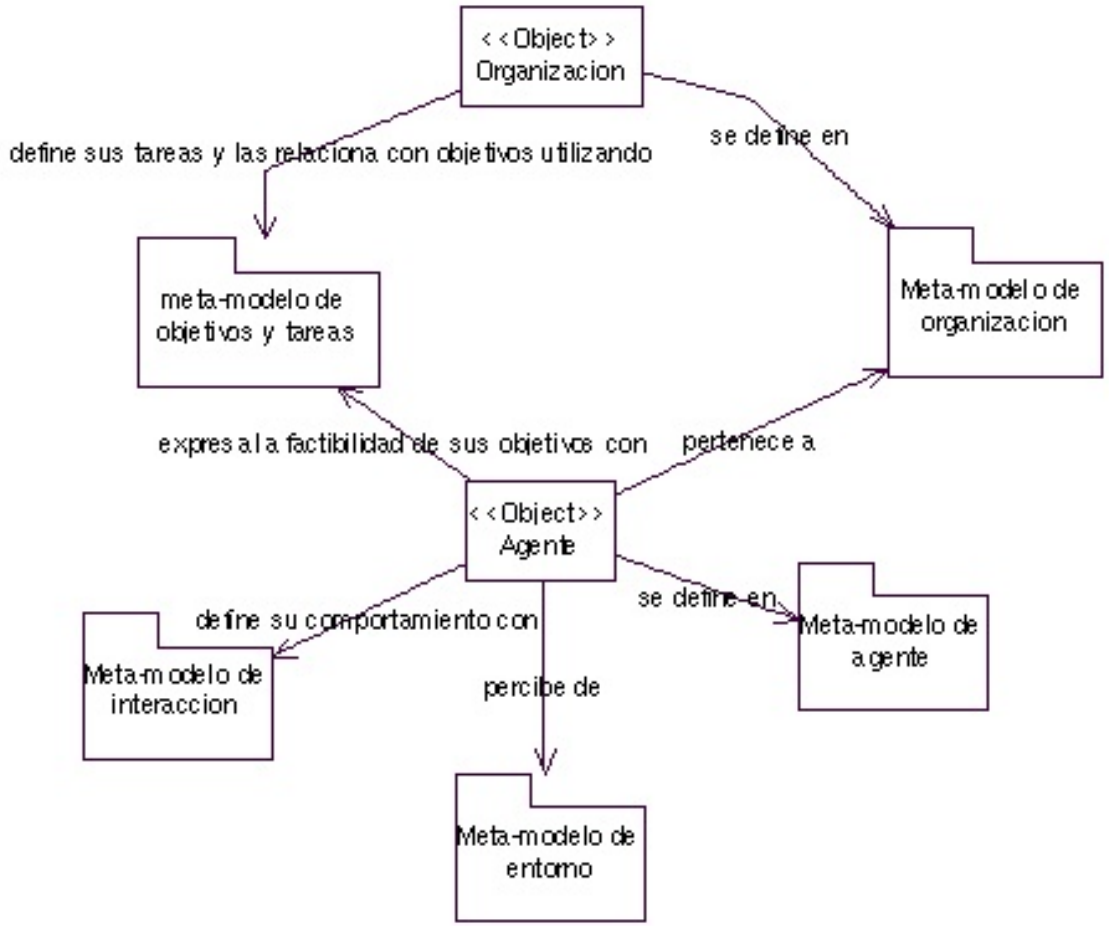
INGENIAS extiende la ingeniería OO con conceptos del área de los agentes software

- Diseño basado en la especificación de models
  1. Agente
  2. Organización
  3. Dominio
  4. Tareas & objetivos
  5. Interacciones
- Método de desarrollo: RUP (*Rational Unified Process*)

INGENIAS, metamodelado

En INGENIAS,

$$SMA = M_{agente} + M_{interaccion} + M_{entorno} + M_{objetivos} + M_{organizacion}$$



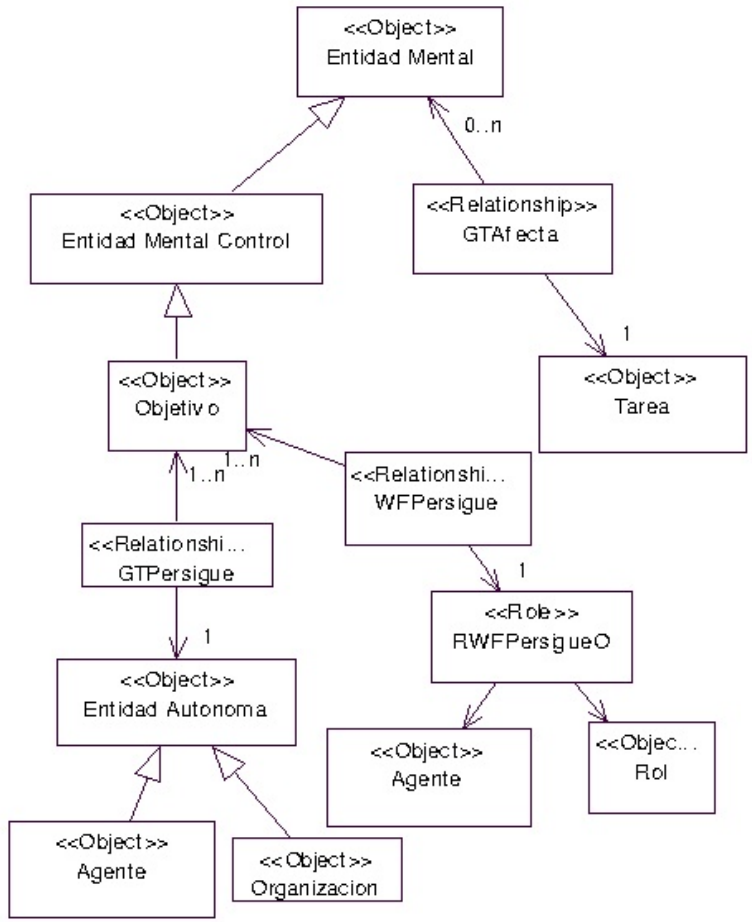
Meta-modelo de Tareas y Objetivos[1]

Este modelo expresa la **motivación** tras las tareas y qué alternativas de actuación tiene cada agente

Las organizaciones y los agentes, como entidades autónomas, persiguen objetivos

Los roles también, aunque mediante otro tipo de relación derivada de los flujos de trabajo (WF)

Las tareas afectan a entidades mentales (del agente que ejecuta la tarea) luego pueden hacer ciertos a los objetivos

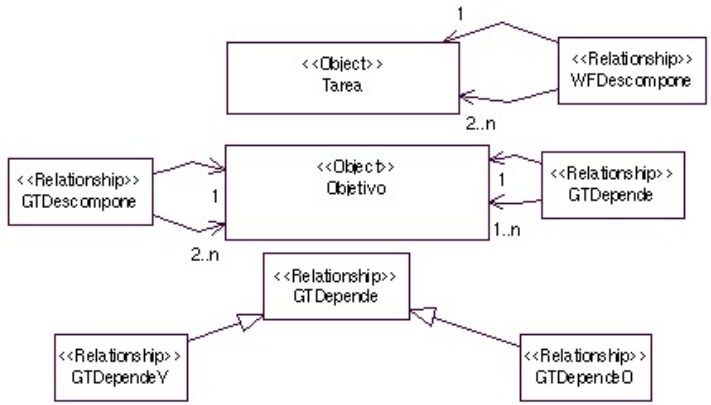


[1] http://grasia.fdi.ucm.es/ingenias/Spain/lenguaje/mtareas.htm

Las tareas y los objetivos se descomponen

Tanto los objetivos como las tareas se descomponen en subobjetivos y subtareas, respectivamente

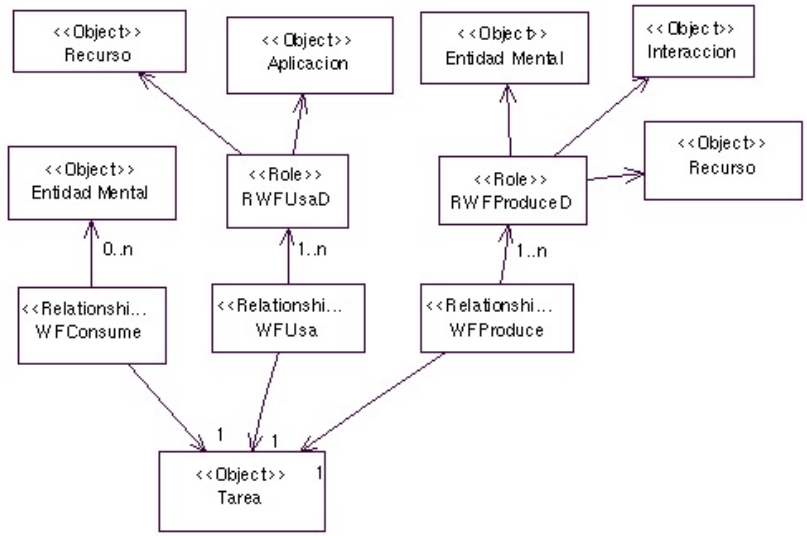
Las relaciones de dependencia entre objetivos forman árboles Y/O



Descripción de tareas

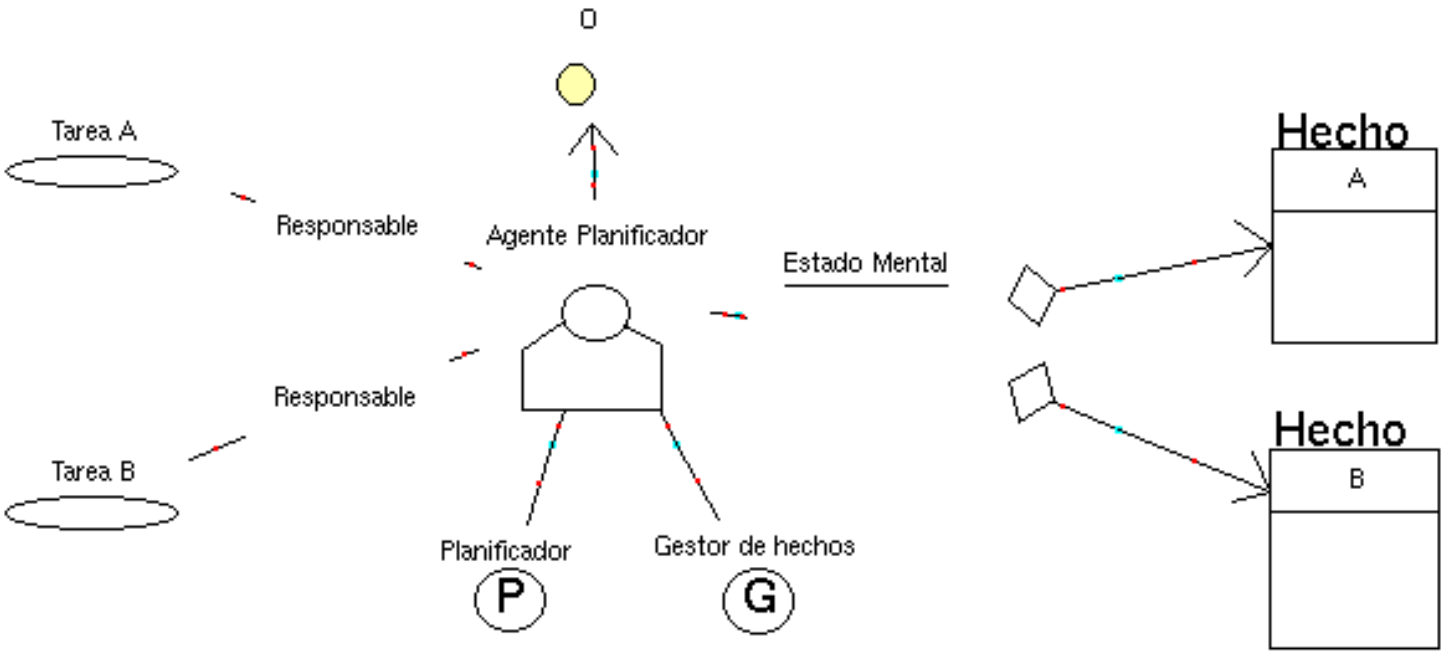
Las tareas se describen mediante precondiciones (i.e. WFConsume, WFUsa, GTAfecta) y postcondiciones (i.e. WFProduce, GTAfecta)

No se ejecutarán aquellas tareas que no satisfagan las precondiciones



Un ejemplo

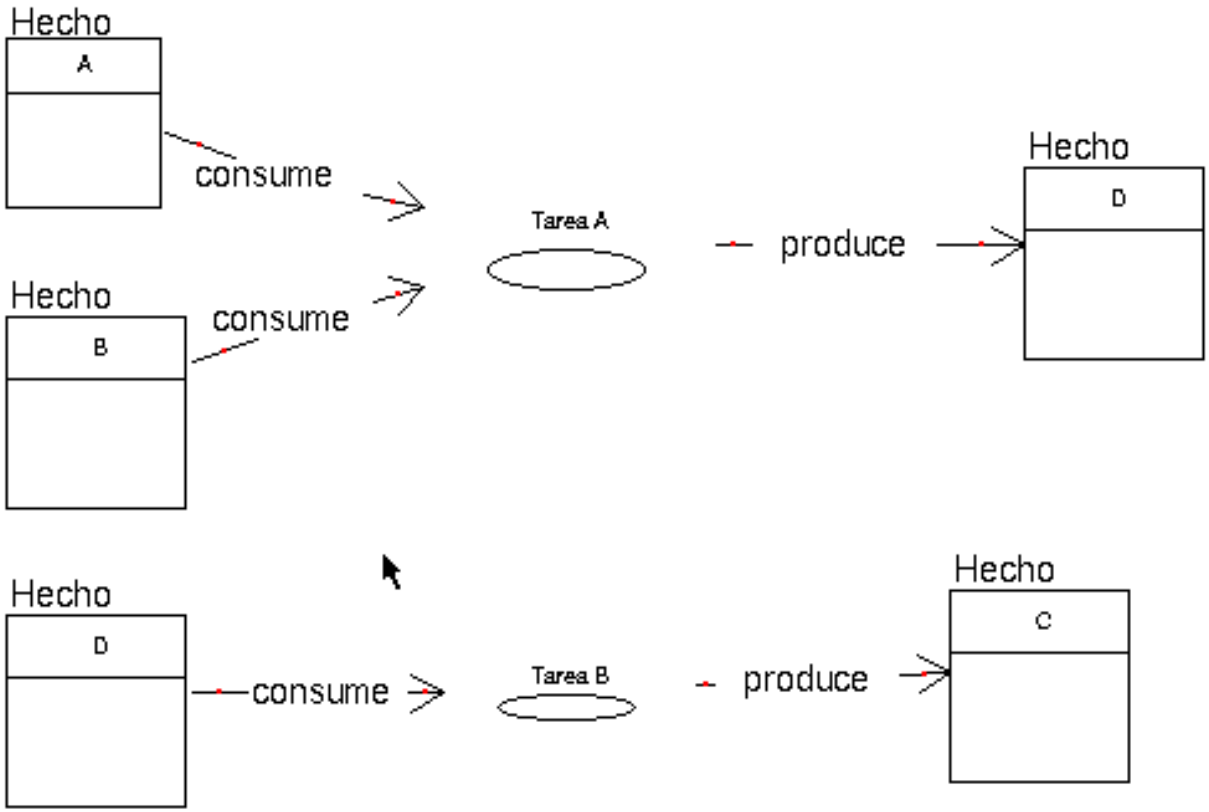
El ejemplo utilizado es el modelado de un agente planificador de tareas. El agente sabe ejecutar dos tipos de tareas: tarea A y tarea B. De la utilización de éstas depende que se alcance el objetivo O



Modelo de agente

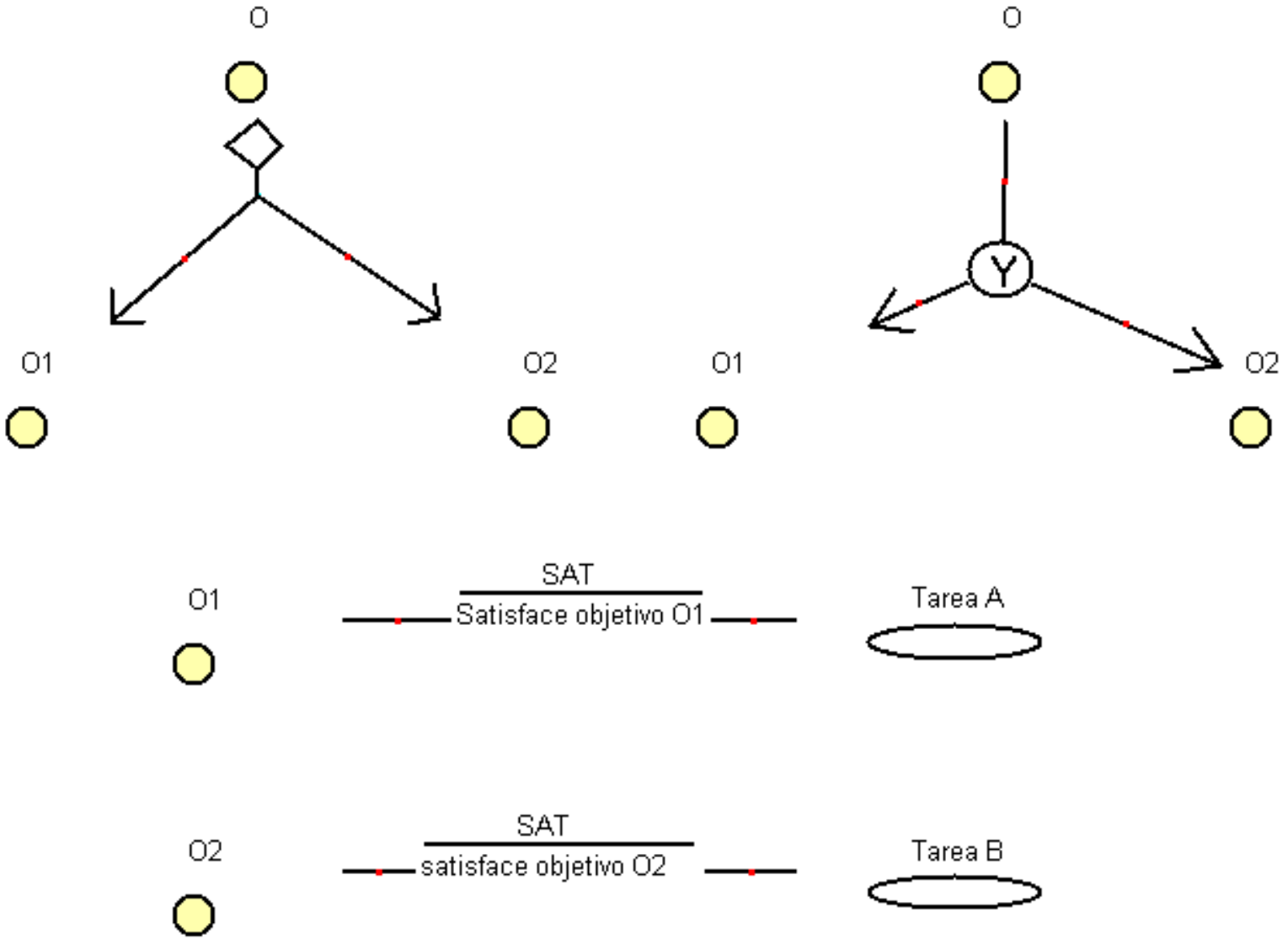
Un ejemplo (y II)

Las tareas simplemente generan hechos nuevos



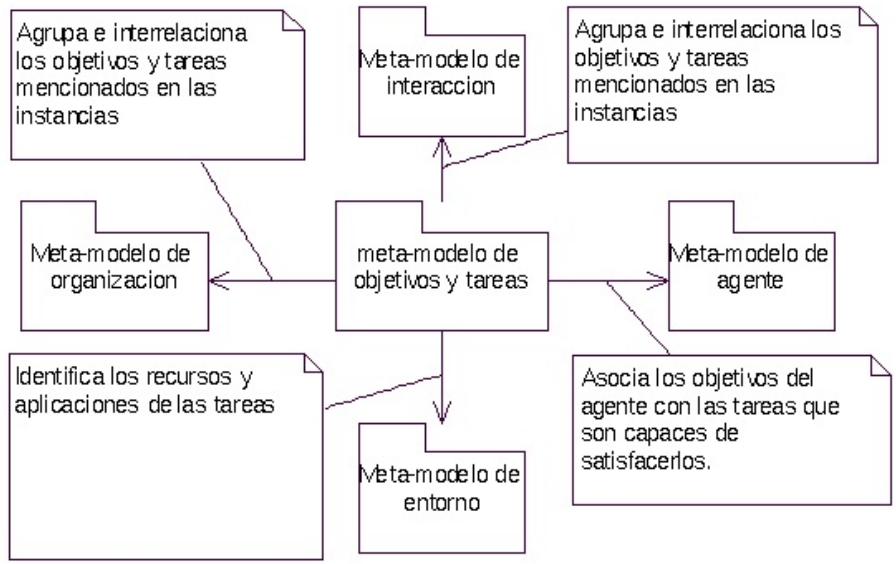
Modelo de tareas

Para alcanzar el objetivo O, se ejecutan A y B



Interdependencias con otros modelos

- Todas las tareas que aparecen en un modelo de tareas y objetivos también deben aparecer en algún modelo de agente o en algún modelo de organización.
- Todo objetivo que aparezca en un modelo de tareas y objetivos debe aparecer en un modelo de agente o en un modelo de organización.
- Si una tarea produce una interacción, debe existir un modelo de organización donde se enmarque esta tarea dentro de un flujo de trabajo
- Cuando los resultados de una tarea se necesiten en otra, se entiende que se tiene un flujo de trabajo. Por lo tanto, debe crearse una nueva entidad flujo de trabajo en un modelo de organización y especificar allí cómo se conectan las tareas.
- Todo recurso que aparece en este modelo debe aparecer en un modelo de entorno.
- Las entidades mentales consumidas, producidas, modificadas o destruidas deben pertenecer al estado mental del agente ejecutor



Las Interacciones en INGENIAS

¿Qué es una interacción?

Es una especificación sobre conversaciones que pueden mantener un grupo de agentes o un agente con un elemento del entorno, destinada a la realización de una tarea concreta.

A destacar sobre las interacciones

- No se tratan de la misma forma en todas las metodologías (GAIA, MAS-CommonKADS, etc)
- Independientemente de la metodología, disponemos de estándares tipo KQML o ACL-FIPA
- INGENIAS modela las interacciones a más alto nivel, pudiéndose instanciar posteriormente a ACLs concretos

La especificación de una interacción debe cubrir

las siguientes informaciones

- Los actores que participan (mostrar por qué está participando en la interacción, i.e. agente racional)
- La definición de unidades de interacción (e.g. puede ser tan simple como un paso de mensaje o tan compleja como un mensaje activo)
- Un orden sobre las unidades de interacción (i.e. un protocolo estándar)
- Acciones ejecutadas en la interacción mediante
  - Criterios para decidir cuándo ejecutar una tarea (no es suficiente con que alguien lo solicite)
  - Consecuencias de la ejecución de una tarea (se esperan cambios en el estado mental del agente)

Materialización de interacciones dependiendo de la tecnología subyacente

Al definir las unidades de interacción

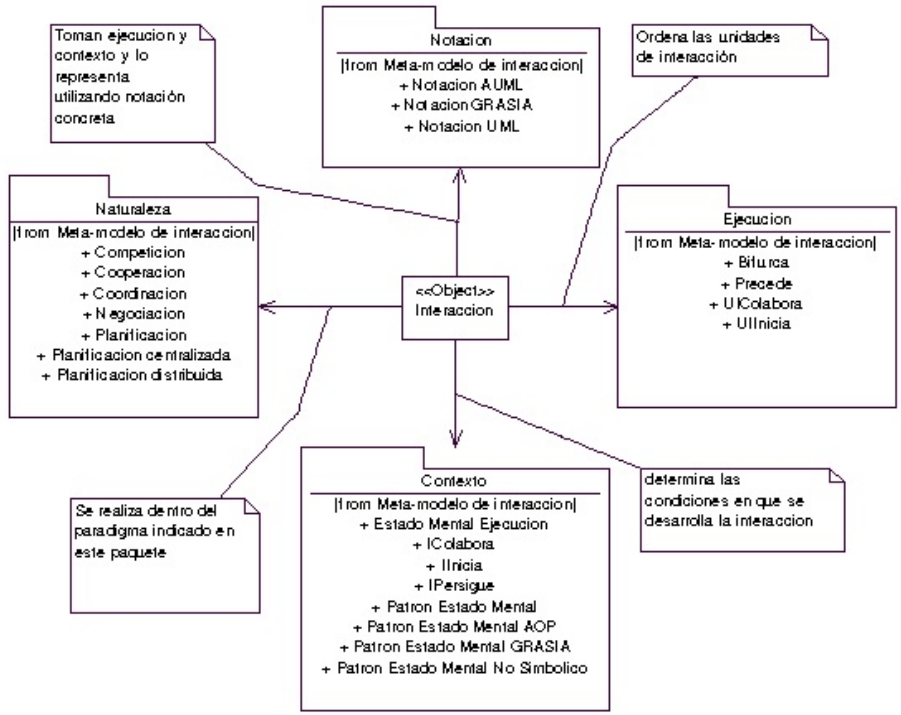
- Si usamos mensajes, se definen el número y contenido de sus parámetros
- Si usamos objetos, definimos tipo y valor de los argumentos de la llamada
- Si tenemos un espacio de tuplas (e.g. Linda), qué información se debe dejar en el espacio compartido y cómo esta información debe ser leída por los colaboradores

Al construir los protocolos

- Arquitectura de pizarra
- En la práctica se usa más el modelo FIPA (paso asíncrono de mensajes ACL)
- Paradigmas cliente servidor (CORBA, DCOM o RMI)

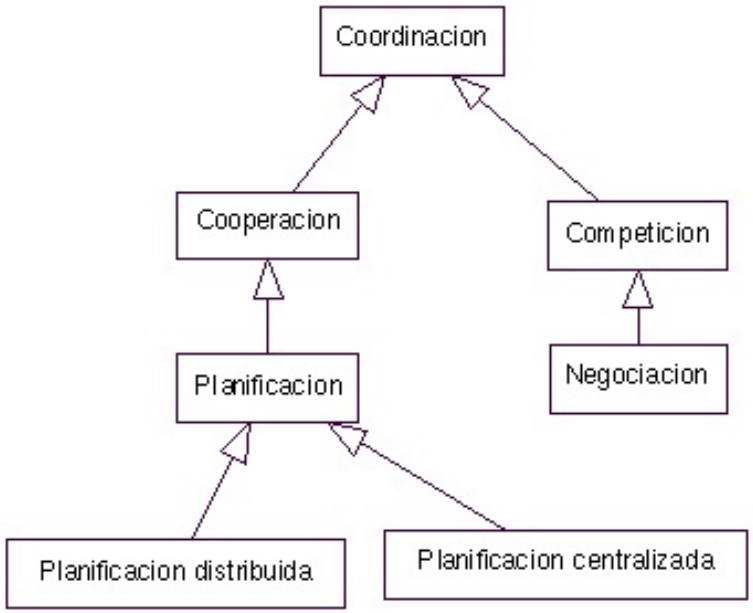
Panorama general

- Se construye sobre agentes, roles, objetivos, interacciones y unidades de interacción
- Los agentes y roles son los actores de las interacciones
- En las interacciones se ejecutan unidades de interacción (pasos de mensaje, lectura y escritura en un espacio de tuplas)
- Hay un iniciador y colaboradores
- La participación de los actores en la interacción y la existencia de la interacción en sí se justifica mediante objetivos



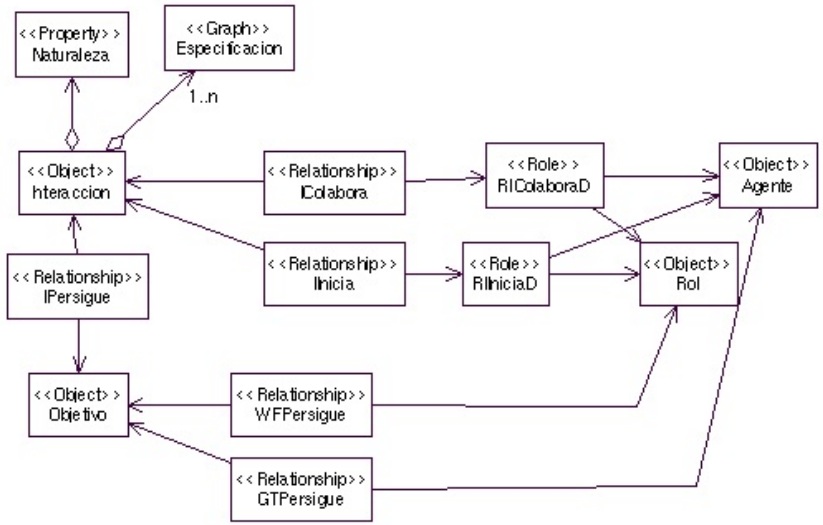
Naturaleza de las interacciones

- La naturaleza de la interacción define qué proceso coordinador se está llevando a cabo entre los agentes participantes
- Según su naturaleza, deberán aparecer unos elementos u otros en el modelo (e.g. si se está negociando, el bien sobre el que se negocia)



Meta-modelo de interacción

- La relación `IPersigue` representa la motivación de la interacción
- Los actores se indican mediante `WFPersigue` y `GTPersigue`
- La naturaleza se incluye como una propiedad y la especificación es un conjunto de relaciones, objetos y roles (i.e. tipo `Graph`)



Análisis en INGENIAS

Un ejemplo ilustrativo basado en un sistema recomendador de documentos

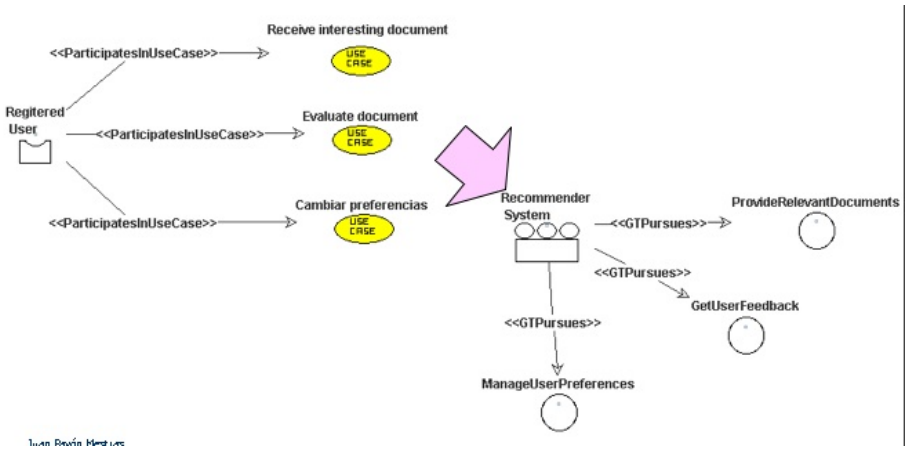
- Los documentos se evalúan según los gustos de los usuarios
- Los documentos llegan a la comunidad de usuarios desde fuera y se evalúan

En una primera tarea de análisis identificaremos requisitos mediante casos de uso y luego pasamos al diseño de

- Objetivos, identificados por los requisitos
- Tareas, que son procedimientos para satisfacer objetivos
- Roles, que definen servicios determinados por las tareas y responsabilidades
- Asignamos objetivos de la organización a los roles definidos
- Definimos workflows: relaciones entre tareas, roles y recursos
- Interacciones, para modelar cómo se comunican los roles
- Agentes, que desempeñan unos roles determinados

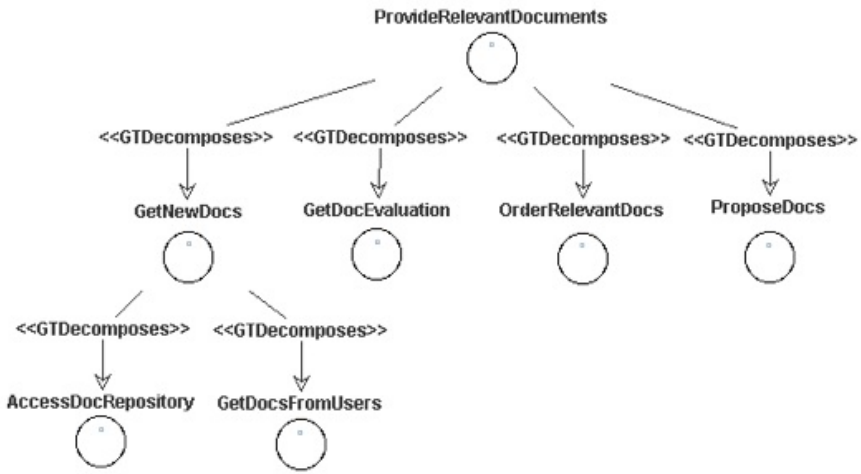
De los casos de uso a los objetivos en la organización

- Los casos de uso se emplean como usualmente, los actores serán roles posteriormente
- El sistema multi-agente es una organización
- Por cada caso de uso se organizan objetivos a cumplir en la organización que es el SMA



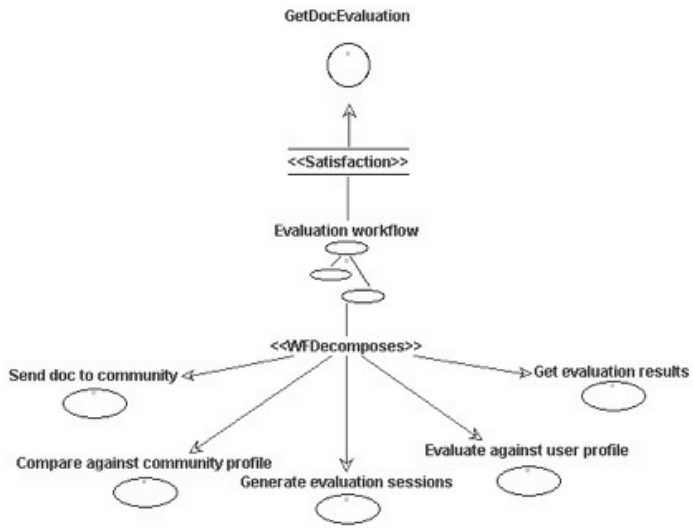
Modelado de objetivos jerárquicamente

- Los objetivos se descomponen hasta que son realizables mediante una tarea
- Así, cada objetivo más pequeño tendrá una tarea asociada dentro del modelo (o un flujo de tareas como en este caso)



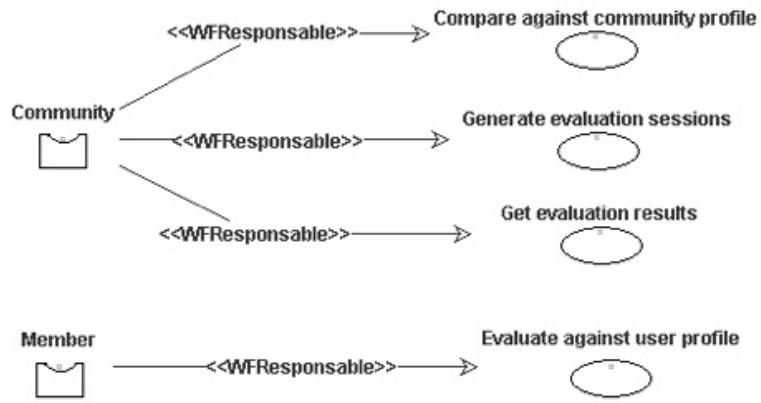
Modelo de objetivos para satisfacer tareas

- Un objetivo se puede satisfacer mediante un flujo de tareas complejo, o bien directamente por una tarea
- Un workflow no es más que la descomposición de una tarea en tareas más simples
- En el workflow se deberá indicar cómo se van modificando los datos de tarea en tarea



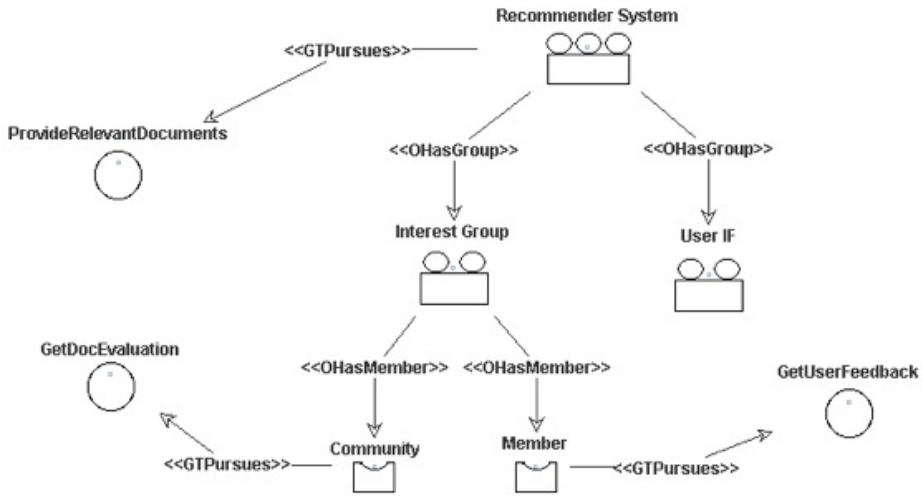
Asignación de roles a las tareas

- Un rol puede verse como un conjunto de servicios relacionados, junto con un conjunto de responsibilidades
- Indicamos los servicios de esta forma



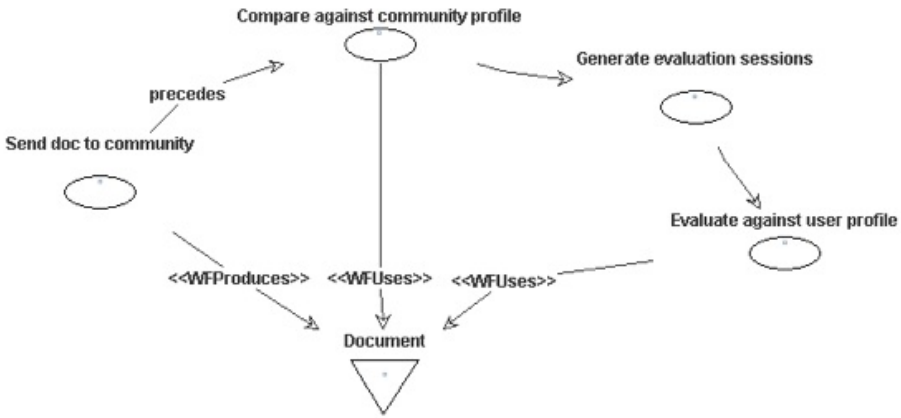
Asignación de roles a objetivos, dentro de la organización

- En el modelado puede resultar útil crear grupos dentro de la organización
- Luego los grupos tendrán sus respectivos roles
- También los roles perseguirán objetivos, mediante el desempeño de tareas



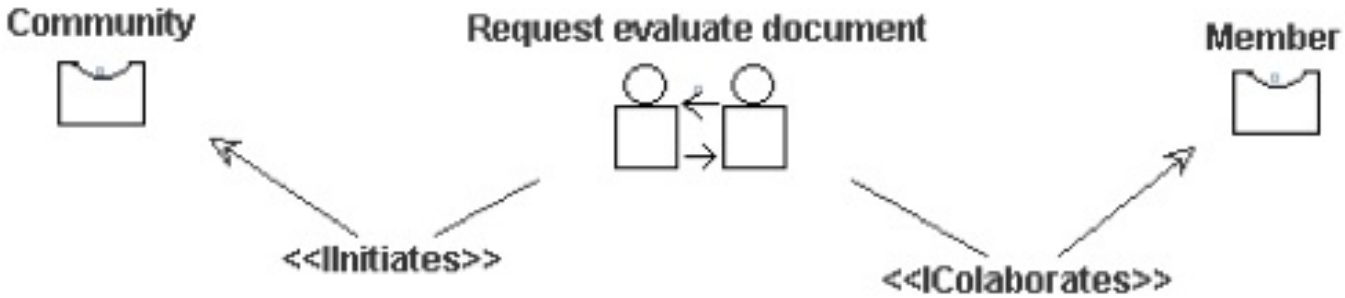
Workflows

- Definen relaciones entre tareas, roles y recursos
- En este ejemplo tenemos un recursos y varias tareas que se disponen a modo de flujo
- También podemos indicar cómo evolucionan los datos de una tarea a otra, qué genera y consume cada tarea



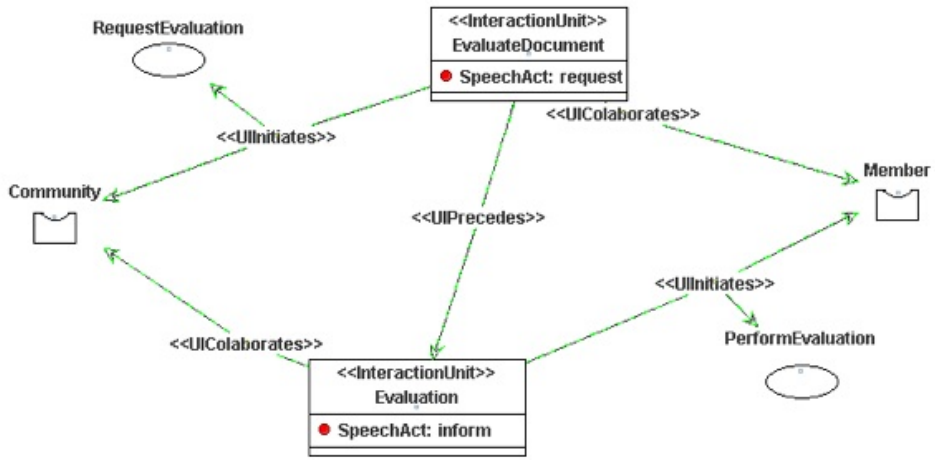
Las interacciones en el ejemplo

- En un modelo INGENIAS, las interacciones definen el procedimiento mediante el cual los roles interaccionan (i.e. negocian, cooperan, etc.)
- Uno de los roles inicia la conversación (`IInitiates`) y el otro u otros responden (*IColaborates*) y siguen interaccionando (si procede)
- La naturaleza se incluye como una propiedad y la especificación es un conjunto de relaciones, objetos y roles (i.e. tipo `Graph`)



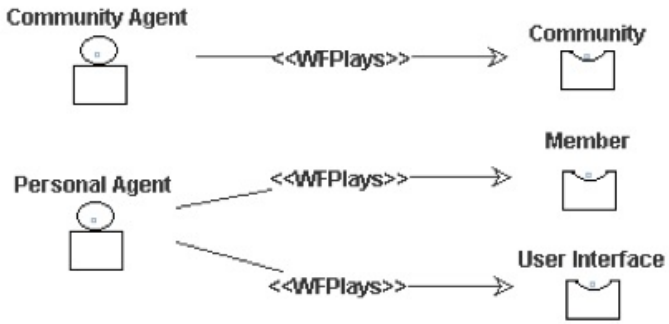
Luego hay que definir cada interaccion por separado

- Una unidad de interacción es una comunicación puntual dentro del diálogo (i.e. en FIPA es un acto comunicativo)
- Cada unidad de interacción es generada por un rol, recibida por otro y quizás, motivada por una tarea
- Además, existe un orden (`UIPrecedes`) entre las unidades de interacción
- Podemos usar modelado AUML



Un modelo de agente simple

- Los agentes aglutinan roles que desempeñan concurrentemente
- Con el IDK veremos que hay varias maneras de implementar agentes
- Cada uno de ellos puede tener un conjunto de creencias inicial



En este caso, el diseño era dirigido por objetivos

Hay otras posibilidades, dependiendo del problema

- Centrarse y comenzar con los workflows
  - Cuando la organización está orientada a procesos
- Centrarse en la coordinación y las interacciones
  - Cuando el problema es la definición de un algoritmo distribuido
  - Sistemas cooperativos
- Centrarse en el entorno
  - Sistemas empotrados
  - Robótica
- Centrarse en los actores
  - Simulación social