

# Desarrollo de Sistemas Multi-Agente con INGENIAS

## Caso de Estudio

Jorge Gómez Sanz  
Juan Pavón Mestras

Dep. Sistemas Informáticos y Programación  
Universidad Complutense Madrid

<http://www.fdi.ucm.es/profesor/jpavon>



## INDICE

- Introducción al caso de estudio
- El proceso de desarrollo

## Juul Moller

- El caso de estudio se puede bajar de la web de doctorado
  - <http://www.fdi.ucm.es/profesor/jpavon/doctorado>
  - justificación de la bondad del caso de estudio:
    - representar modelos de negocio
    - coordinación
    - integración con aplicaciones propietarias
    - autonomía
- Se trata de una librería tradicional que tiene acuerdos con la universidad para la venta de libros en sus asignaturas
  - La universidad provee listas de los libros que se van a utilizar en las asignaturas
  - La librería obtiene los libros y hace un precio especial a la universidad
- Ahora la librería se plantea modernizar su negocio
  - Quiere vender libros electrónicamente
  - Nosotros ampliamos el caso de estudio comprobando también qué hace falta para agilizar la petición de libros

## El proceso de desarrollo

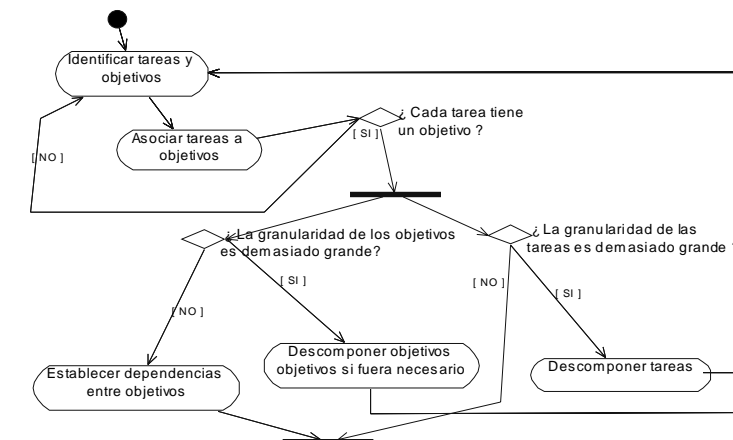
- Para estudiar el problema de Juul aplicamos el proceso de desarrollo de INGENIAS
- El proceso de desarrollo de INGENIAS se basa en el UP
  - Hay iteraciones
  - Hay niveles de refinamiento
- La presentación del proceso de desarrollo hará referencia a distintas actividades identificadas en la tesis "modelado de sistemas multi-agente". Dichas actividades son accesibles en <http://grasia.fdi.ucm.es/ingenias> dentro de la sección de Ciclo de Vida
- Como parte del proceso de desarrollo también explicaremos cómo se genera el prototipo

FASES			
FLUJOS DE TRABAJO FUNDAMENTALES	Análisis	Elaboración	Construcción
	Diseño		
	<ul style="list-style-type: none"> <li>o Generar casos de uso e identificar realizaciones de los casos de uso con modelos de interacciones.</li> <li>o Esbozar la arquitectura con un modelo de organización.</li> <li>o Generar modelos del entorno para trasladar la captura de requisitos a los modelos</li> </ul>	<ul style="list-style-type: none"> <li>o Refinar casos de uso.</li> <li>o Generar modelos de agente para detallar los elementos de la arquitectura.</li> <li>o Continuar con los modelos de organización identificando flujos de trabajo y tareas.</li> <li>o Modelos de tareas y objetivos para generar restricciones de control (objetivos principales, descomposición de objetivos)</li> <li>o Refinar modelo de entorno para incluir nuevos elementos.</li> </ul>	<ul style="list-style-type: none"> <li>o Estudiar resto de casos de uso.</li> </ul>
	<ul style="list-style-type: none"> <li>o Generar un prototipo con herramientas de prototipado rápido, como ZEUS o Agent Tool</li> </ul>	<ul style="list-style-type: none"> <li>o Centrar el modelo de organización en el desarrollo de flujos de trabajo.</li> <li>o Llevar las restricciones identificadas a modelos de tareas y objetivos para dar detalles acerca de las necesidades y resultados de las tareas y su relación con los objetivos del sistema.</li> <li>o Expresar la ejecución de tareas dentro de modelos de interacción.</li> <li>o Generar modelos de agente para detallar <i>patrones de estado mental</i>.</li> </ul>	<ul style="list-style-type: none"> <li>o Generar nuevos modelos de agente o refinar los existentes.</li> <li>o Depurar la organización centrando el desarrollo en las relaciones sociales.</li> </ul>

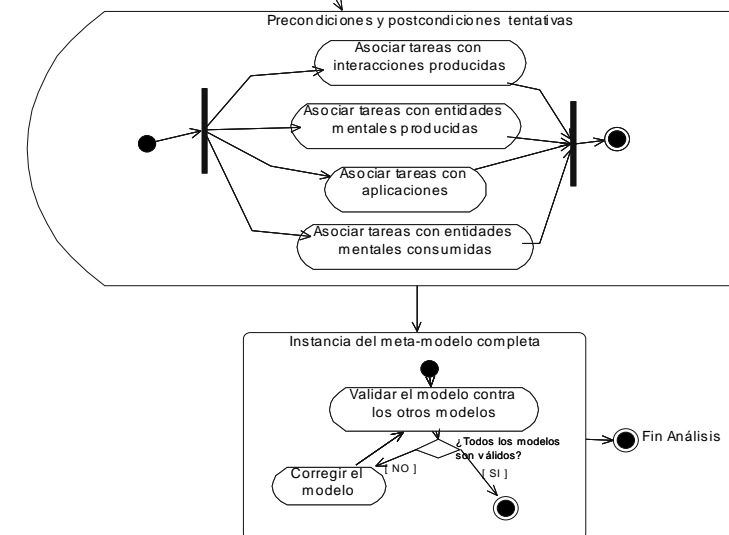
## ¿Y esto cómo se hace?

- Planteamos una serie de actividades expresadas en forma de diagrama de actividades
- Para cada actividad se detalla
  - Qué se espera de la actividad
  - Qué productos hay que generar
  - Las entradas vienen de los diagramas de actividades
- La aplicación de estas actividades se rigen por el UP, que es el que establece el nivel de detalle

## Ejemplo actividades: Generación modelo objetivos y tareas en el análisis

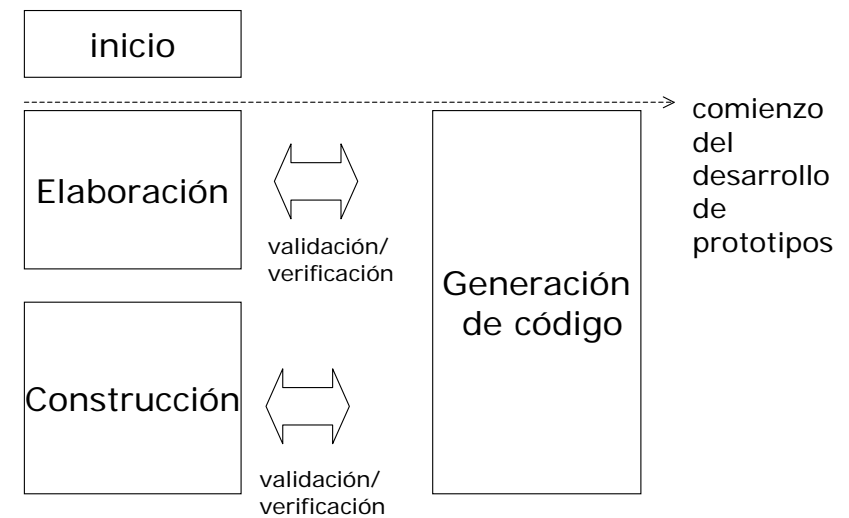


## Ejemplo actividades: Generación modelo objetivos y tareas en el análisis



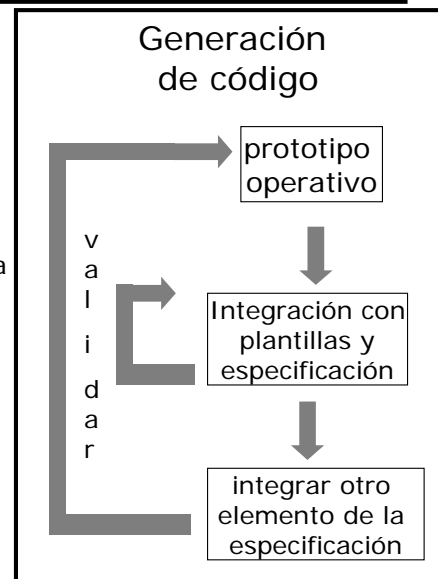
## Y la implementación?

## Papel de las herramientas de generación de código



## Generación de código en el proceso de desarrollo

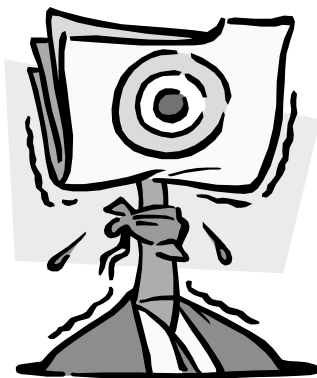
- La generación de código se realiza de forma iterativa en paralelo con la generación de la especificación
- El nexo entre el prototipo y la especificación es el proceso de generación de código
- Este proceso tiene como resultado colateral que verifica la especificación desde el punto de vista de las necesidades del prototipo
- La integración del prototipo con el sistema de plantillas requiere una validación con la especificación existente
  - Puede haber un feedback del generador de código para con el analista



## Desarrollos multi-usuario

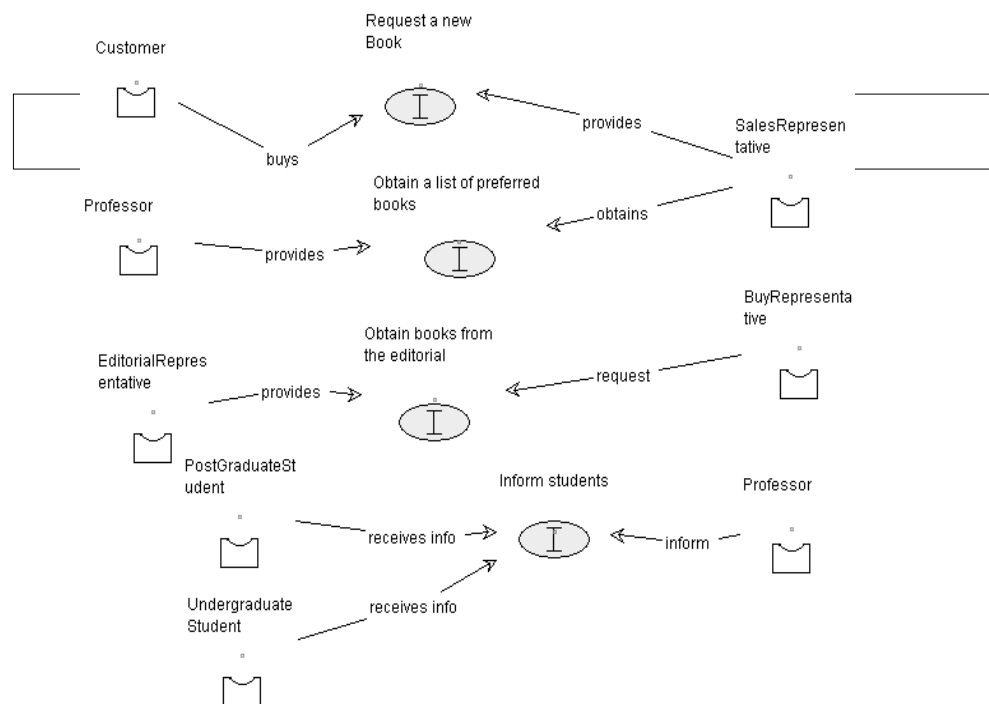
- Actualmente, INGENIAS soporta un proceso de desarrollo con
  - Un analista: es el encargado de generar la especificación
  - Uno/varios desarrolladores
    - Uno/varios desarrolladores del módulo: cvs
    - Un desarrollador de los diferentes aspectos del prototipo: cvs
- En breve planteamos pasar a un proceso
  - Uno/varios analistas
  - Uno/varios desarrolladores
- El cuello de botellas es el editor que no tiene instrumentos para control de versiones de la especificación

## Etapa de inicio



## Productos a conseguir

- Hay que obtener una primera aproximación al problema
- Casos de Uso
- Modelos de entorno
- Modelos de organización



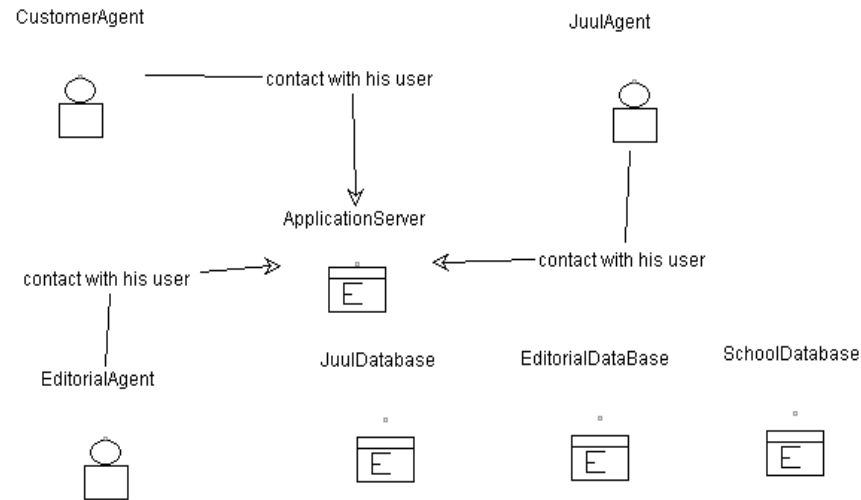
## Actividades Análisis-inicio

### modelo de entorno

Actividad	Tipo de resultado
<i>Identificar aplicaciones del entorno (1.a)</i>	Un conjunto de <i>aplicaciones</i>
<i>Asociar operaciones sobre aplicaciones (2)</i>	Operaciones sobre las <i>aplicaciones</i>
<i>Determinar la percepción de los agentes (6)</i>	Asociaciones entre agentes y <i>aplicaciones</i>



## Actividades 1.a/2/6

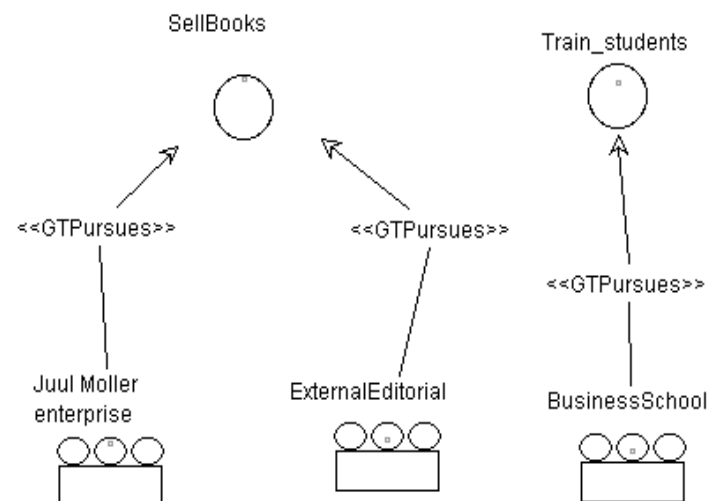


## Actividades Análisis-inicio

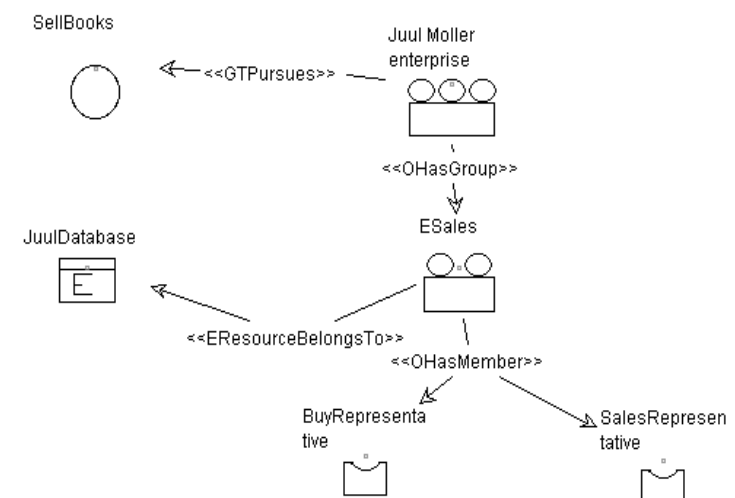
### modelo de organización

Actividad	Tipo de resultado
identificar grupos (1.a)	Un conjunto de <i>grupos</i>
generar miembros (1.b)	Elementos que componen los <i>grupos</i>
identificar objetivos (2.c).	Objetivos asociados a las organizaciones

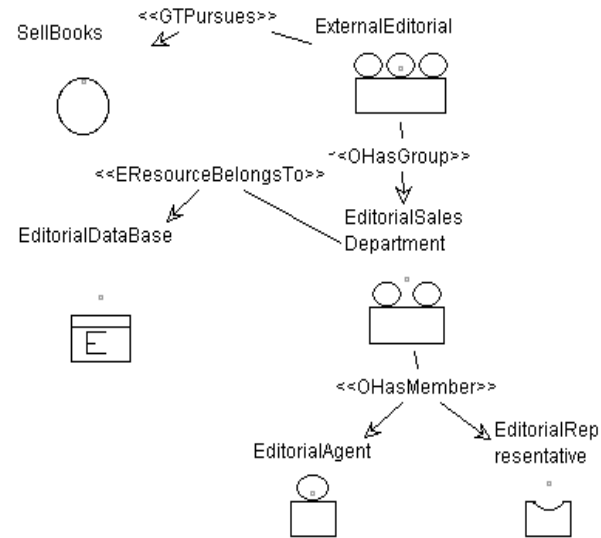
## Actividades 2.c



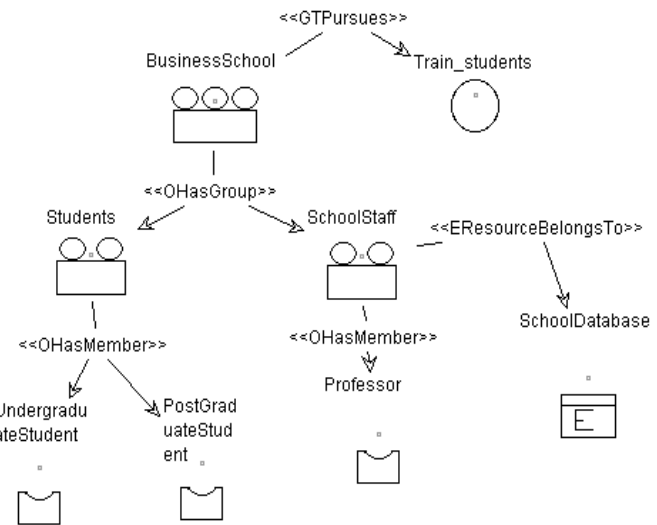
## Actividades 1.a/1.b



## Actividades 1.a/1.b



## Actividades 1.a/1.b



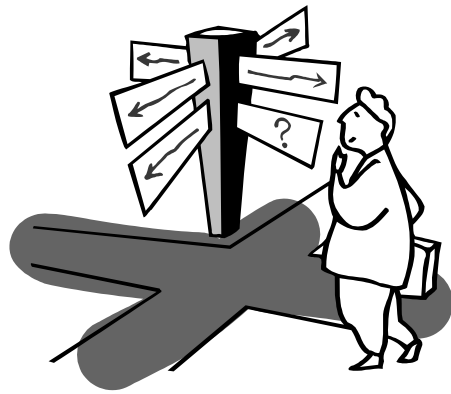
## Prototipo inicial

- Dos agentes JADE
- Detectar necesidades de las infraestructuras de comunicación
  - Cada agente JADE capaz de crear protocolos bajo demanda
  - Seguimiento del protocolo con máquinas de estados
- Definición de un protocolo en JADE

## Resumen

- Tenemos una idea general de qué elementos componen el sistema
- También sabemos qué tiene que hacer
- ¿Es realizable?
  - Poner en común tres organizaciones
    - Se está usando interfaz via web con los usuarios
    - Los agentes se comunican mediante protocolos estándar
  - Integración con aplicaciones de gestión propietarias
    - Accesos a bases de datos
    - ¿ERP? recubrimiento con agentes. Que un agente maneje la interfaz del usuario

## Etapa de Elaboración

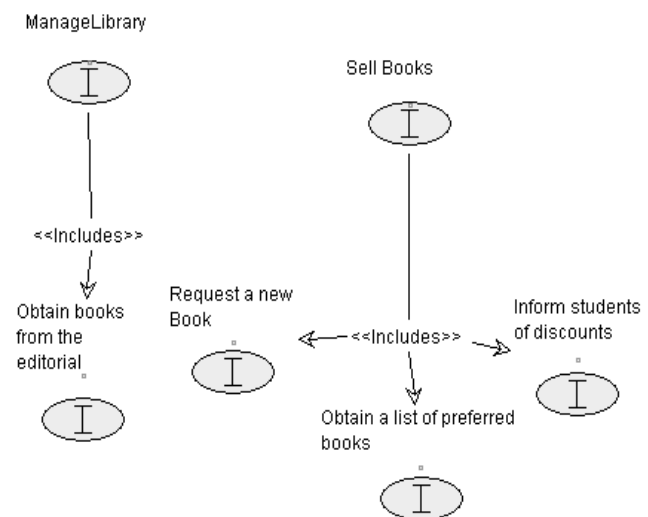


## Problemas generales

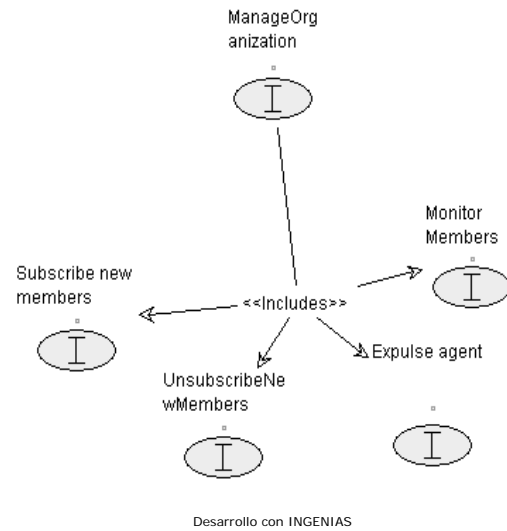
- Se tiene en cuenta el diseño
- Hay muchos más modelos a obtener: de todos los tipos
- Se comienza con el prototipado

## Análisis-elaboración

## Análisis-elaboración

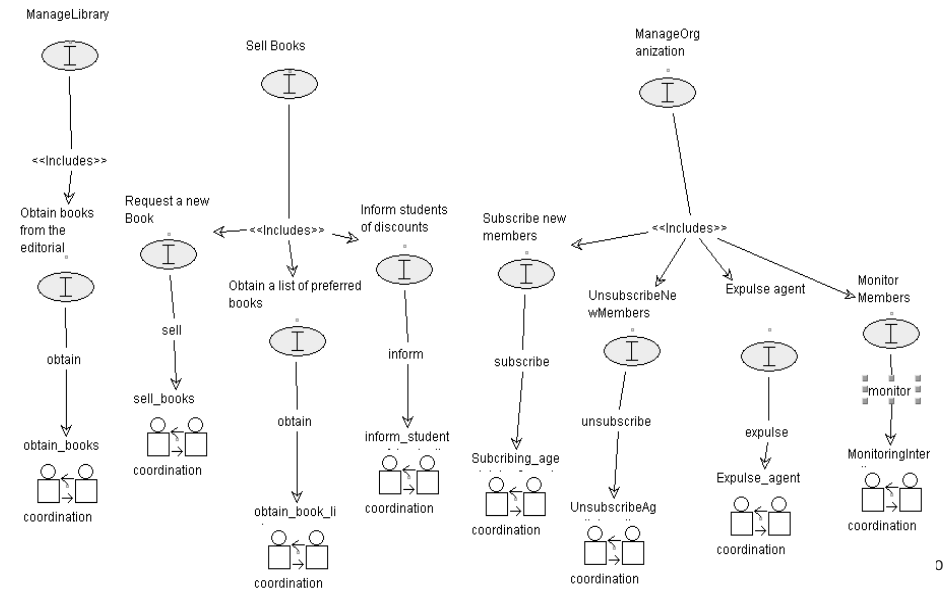


## Análisis-elaboración



29

## Análisis-elaboración



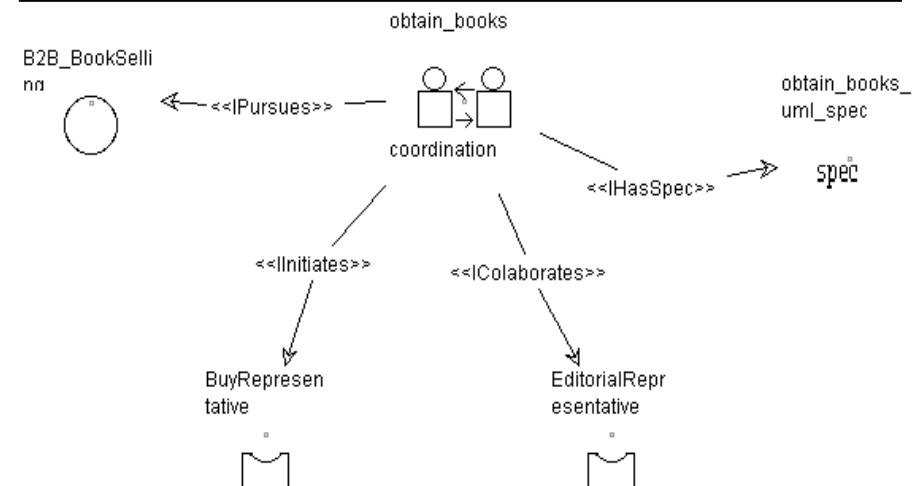
30

## Análisis-elaboración (interacciones)

Actividad	Tipo de resultado
Identificar los objetivos que persigue la interacción (1)	Un conjunto de objetivos
Identificar su naturaleza (3)	Categoría correspondiente a la interacción
Identificar los participantes (2)	Un conjunto de participantes
Generar una primera especificación (4).	Diagramas de colaboración

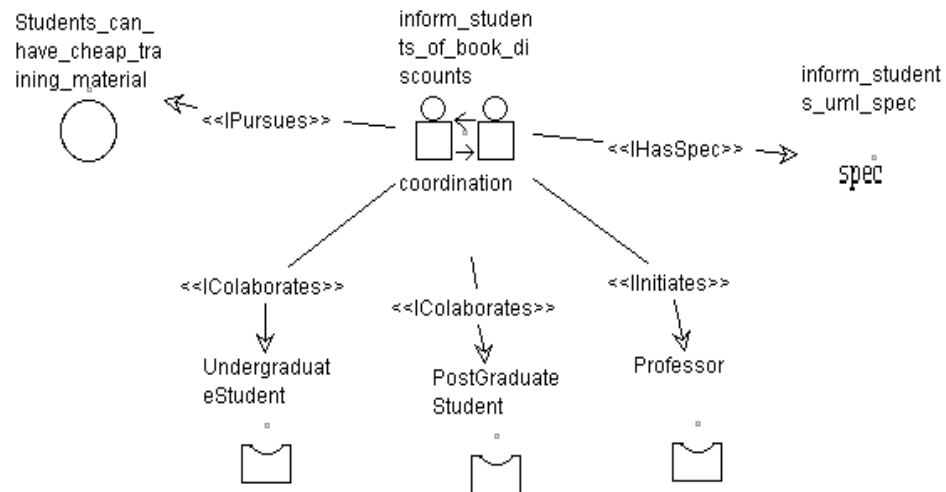
31

## Actividades 1/2/3

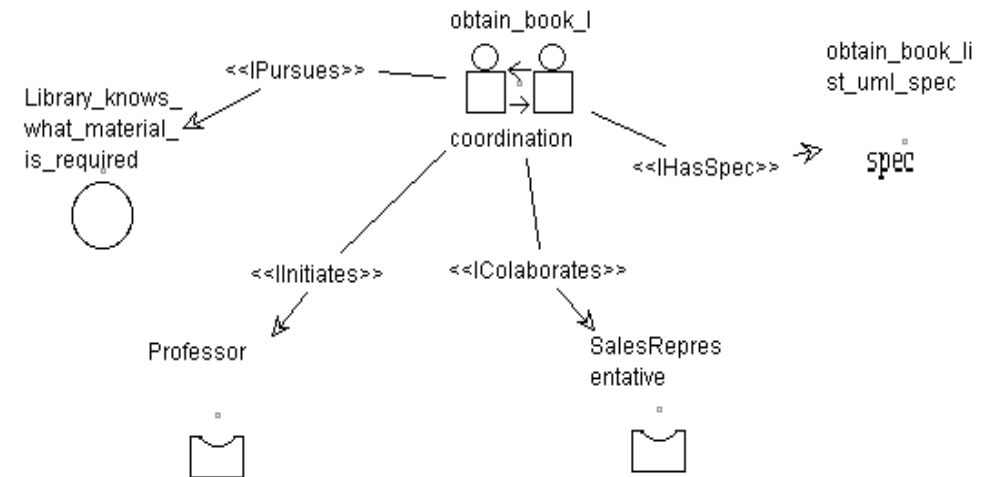


32

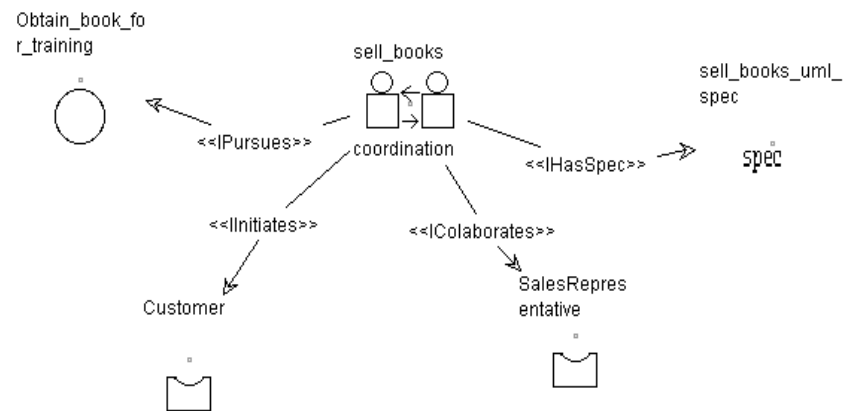
## Actividades 1/2/3



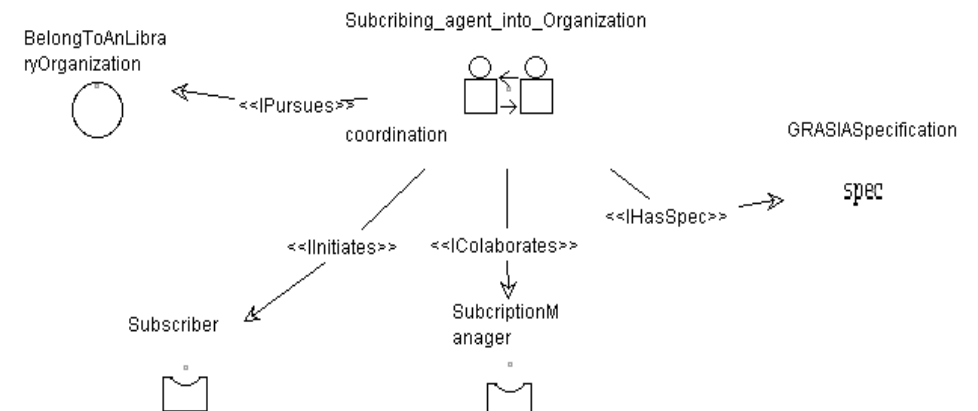
## Actividades 1/2/3



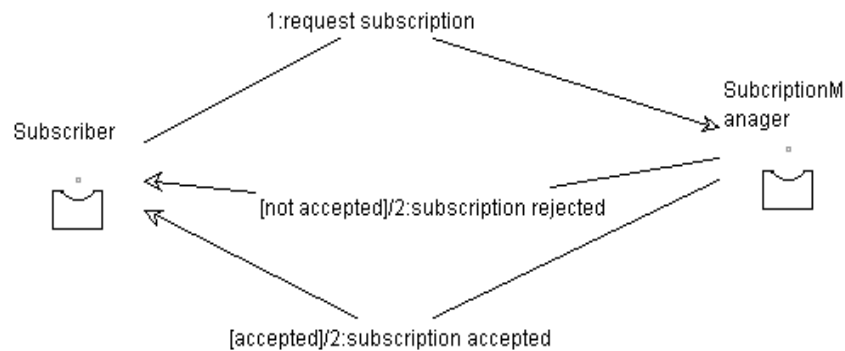
## Actividades 1/2/3



## Actividades 1/2/3



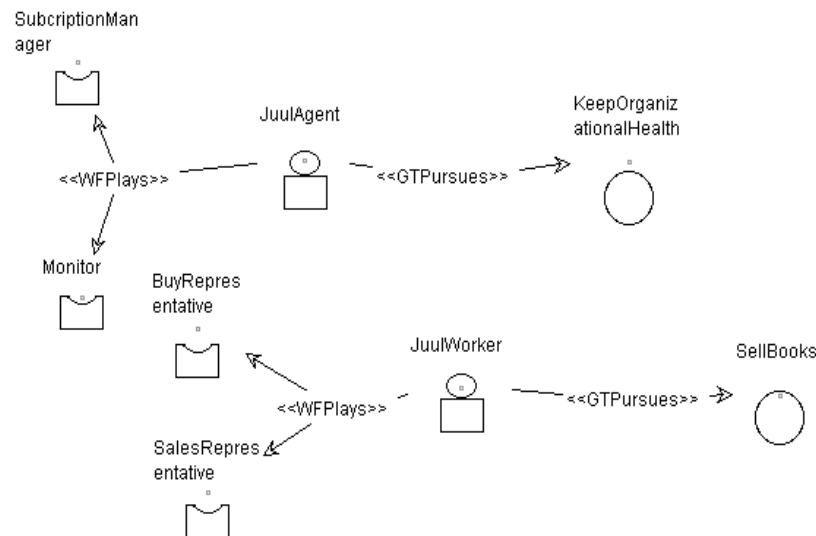
## Actividad 4



## Análisis-elaboración (agentes)

Actividad	Tipo de resultado
Identificar agentes utilizando el principio de racionalidad (1)	Un conjunto de agentes
Identificar los objetivos (2.a)	Un conjunto de objetivos
Identificar funcionalidad (2.b)	Un conjunto de roles y tareas
Asociar funcionalidad con objetivos (2.c)	Asociaciones de roles y tareas con agentes
Identificar aspectos de autonomía e inteligencia (3)	Descripción en lenguaje natural
Determinar cómo será el gestor y procesador de estado mental (4)	Descripción en lenguaje natural

## Actividades 1/2.a/2.b/2.c



## Actividad 3

- Descripción de las capacidades de JuulAgent
- Autonomía: el agente tiene autonomía para decidir qué agentes se admiten en la organización y cuáles de los que ya pertenecen están haciendo sus tareas correctamente. Su objetivo es mantener la *salud organizacional*
- Inteligencia:
  - Se trata de decidir, en base a un conjunto de datos acerca del comportamiento de los agentes, si un agente lo está haciendo bien en función de la opinión que tienen de él los demás
  - Para ello debe tener capacidad de aprender qué datos marcan a un agente como no deseable
    - árboles de decisión: ID3
    - clustering: k-neighbours

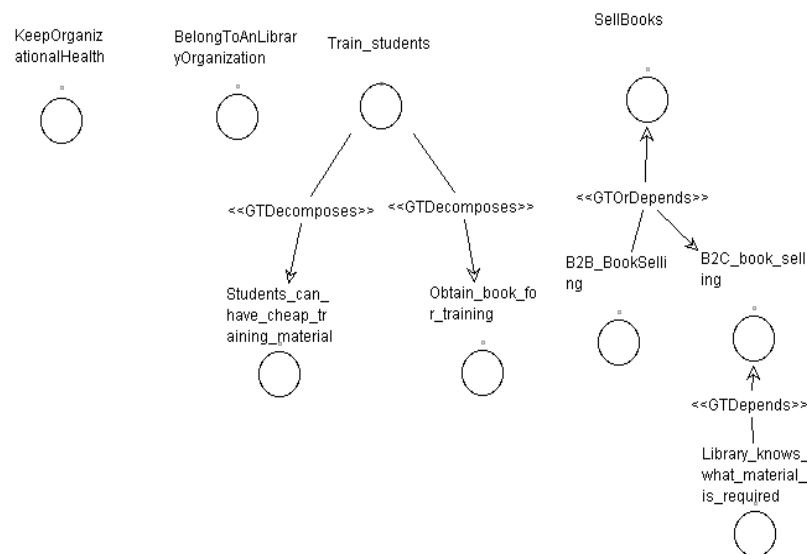
## Actividad 4

- Descripción de las gestión del E.M. de JuulAgent No hay infraestructuras de mantenimiento de la verdad
- Las operaciones básicas serán:
  - añadir entidad mental
  - obtener entidad mental
  - obtener lista de entidades mentales
  - obtener una lista de entidades mentales de un tipo concreto
  - modificar entidades mentales existentes

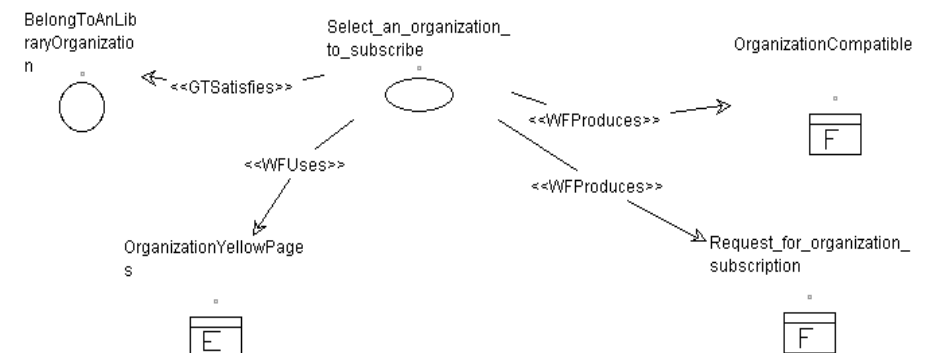
## Análisis-elaboración (tareas y objetivos)

Actividad	Tipo de resultado
Identificar tareas y objetivos (1)	Un conjunto de tareas y objetivos
Descomponer objetivos (4)	Asociaciones de descomposición entre objetivos
Asociar tareas a objetivos (2)	Asociaciones entre objetivos y tareas
Descomponer tareas (3)	Asociaciones de descomposición entre tareas
Establecer dependencias entre objetivos (5).	Dependencias
Asociar tareas con interacciones (6.a)	Instancias de WFProduce asociadas a entidades Interacción
Asociar tareas con entidades producidas (6.c) y consumidas (6.b)	Instancias de WFProduce e instancias de WFConsume
Asociar tareas con aplicaciones (6.d).	Instancias de WFUsa asociando aplicaciones y tareas

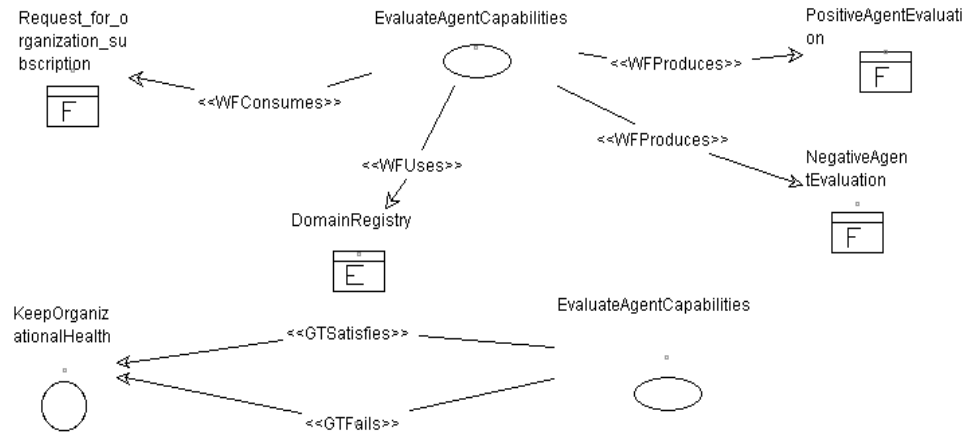
## Actividades 1/4/5



## Actividades 1/2/6.b/6.c/6.d



## Actividades 1/2/6.b/6.c/6.d



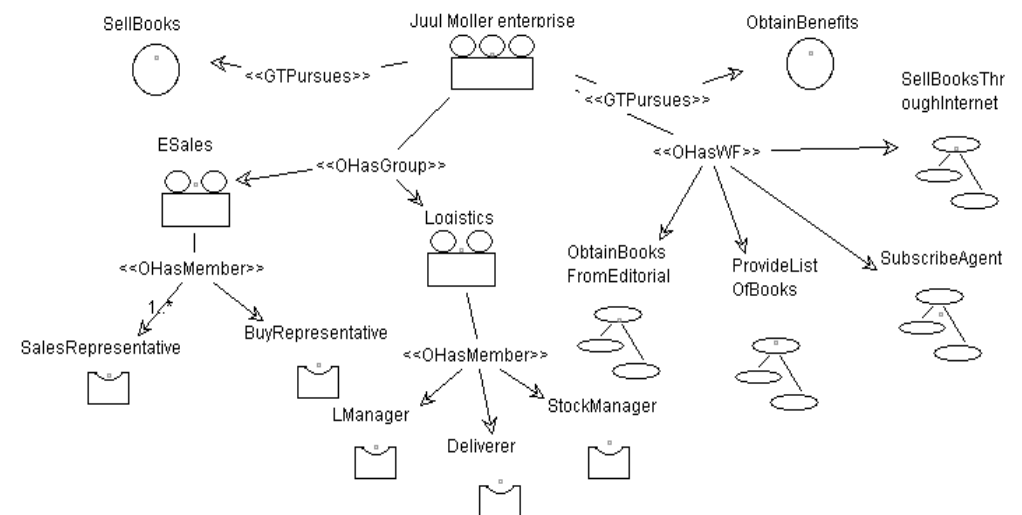
## Análisis-elaboración (entorno)

Actividad	Tipo de resultado
Identificar aplicaciones internas (1.b)	Un conjunto de instancias de aplicación interna.
Asociar operaciones (0)	Operaciones sobre las aplicaciones
Actividades del análisis de las aplicaciones (3)	Diagramas UML para expresar comportamiento
Asociar aplicaciones a grupos	Un conjunto de aplicaciones
Identificar recursos	Un conjunto de recursos

## Análisis-elaboración (organización)

Actividad	Tipo de resultado
Generar miembros (1.b)	Entidades asociadas a instancias de grupo.
Identificar flujos de trabajo (2.a)	Instancias de Flujo de trabajo asociadas a una organización
Aplicar descomposición de flujos (2.b).	Descomposición de los flujos de trabajo

## Actividades 1.b, 2.a., 2.b



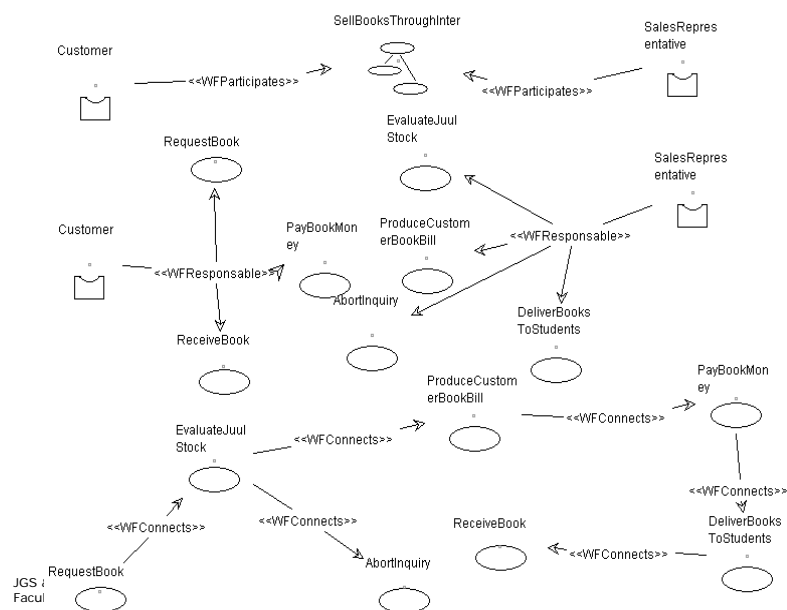


# Diseño-elaboración

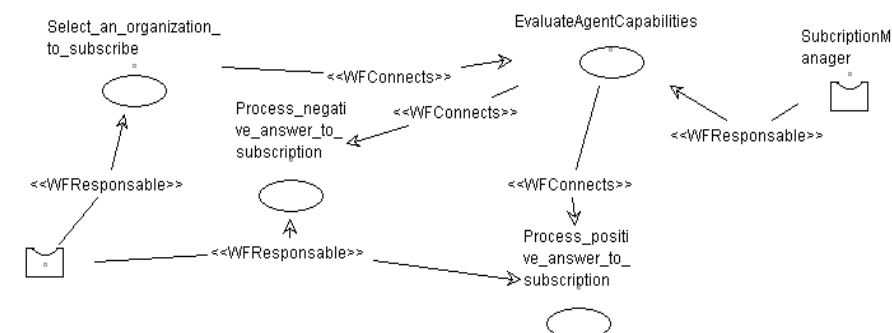
## Diseño-elaboración (organización)

Actividad	Tipo de resultado
identificar tareas (5.a).	Un conjunto de tareas asociadas a un flujo de trabajo
conectar tareas (5.b).	Instancias de <i>WFConecta</i>
identificar tareas no locales (5.c),	Tareas que producen interacciones
identificar responsables (5.d)	Instancias de <i>WFResponsable</i> asociando roles o agentes y tareas
identificar entidades mentales (5.f),	Asociaciones de las tareas con entidades mentales mediante instancias de <i>WFProduce</i> y <i>WFConsume</i> .

## Actividades 5.a / 5.b / 5.d



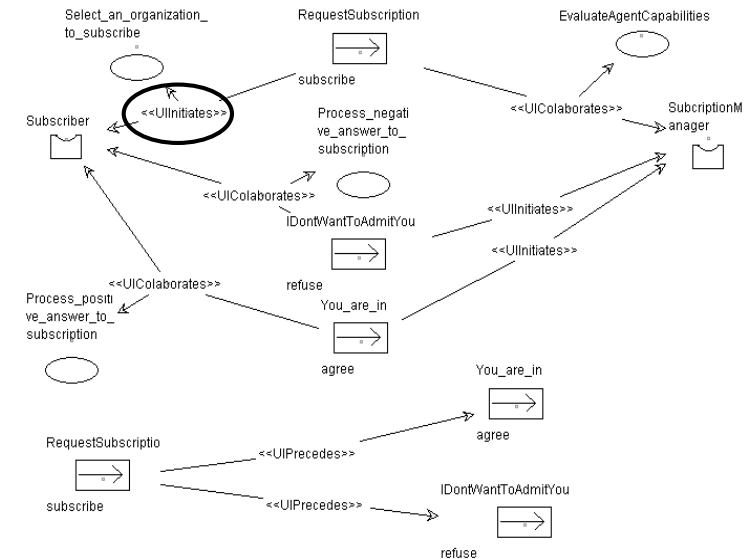
## Actividades 5.a / 5.b / 5.d



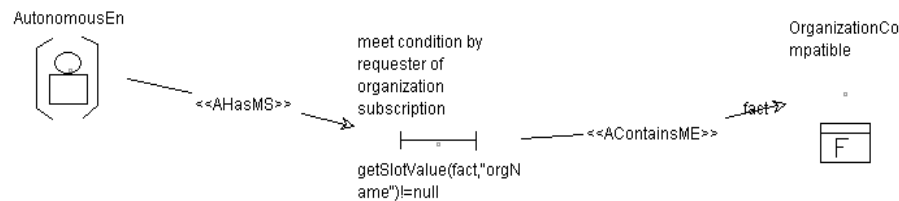
## Diseño-elaboración (interacción)

Actividad	Tipo de resultado
<i>traducir mensajes (5.a)</i>	Diagramas GRASIA donde se asocian roles con unidades de interacción
<i>establecer orden de ejecución (5.d).</i>	Relacionar las unidades de interacción mediante primitivas <i>UIPrecede</i> , <i>UIBifurca</i> , <i>UIItera</i> , <i>UIConcurren</i>
<i>establecer condiciones mentales 5.c</i>	Instancias de patrón de estado mental asociado a unidades de interacción
<i>asociar tareas (5.b)</i>	Asociar tareas a instancias de unidad de interacción

## Actividades 5.a / 5.b /5.d



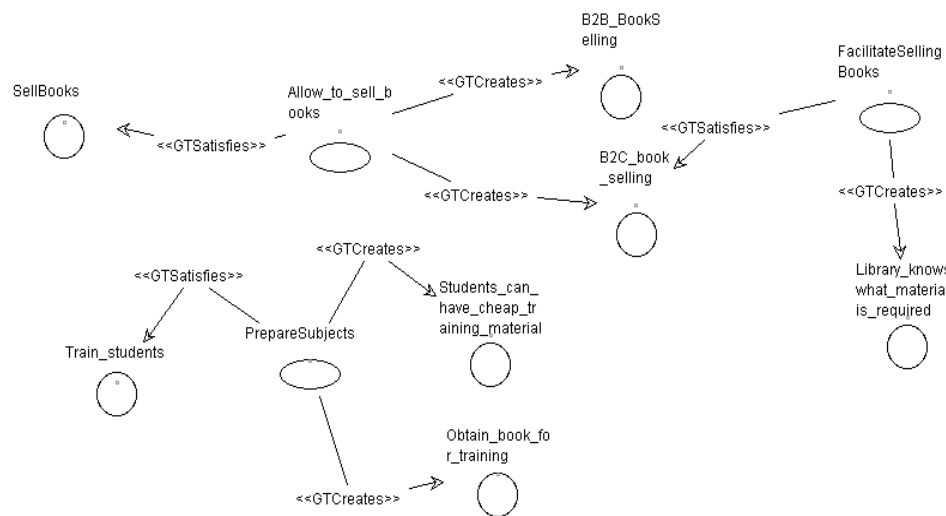
## Actividad 5.c



## Diseño-elaboración (agente)

Actividad	Tipo de resultado
<i>Determinar cómo se pasa de un estado mental a otro (7.a).</i>	Referencias a diagramas GRASIA y a modelos de tareas y objetivos
<i>Detallar los estados intermedios (6)</i>	Instancias de modelos de agente donde se asocia una instancia de agente concreto a instancias de estado mental
<i>Detallar cómo se gestiona el estado mental (7.b)</i>	Instancias de modelos de tareas y objetivos donde las tareas producen, destruyen o modifican objetivos
<i>Asociar los estados mentales a la ejecución de acciones (7.c).</i>	Instancias de modelos de tareas y objetivos donde las tareas producen, destruyen o modifican objetivos

## Actividad 7.b



## Diseño-elaboración (entorno)

Actividad	Tipo de resultado
<i>refinar la percepción de los agentes (9).</i>	Instancias de EPercibeNotifica y EPercibeMuestreo asociando agentes y aplicaciones
<i>definir los atributos propios de cada recurso (11)</i>	Instancias de modelos de agente donde se asocia una instancia de agente concreto a instancias de estado mental
<i>asociar recursos, aplicaciones y tareas (12)</i>	Instancias de modelos de tareas y objetivos donde las tareas producen, destruyen o modifican objetivos

## Diseño-elaboración (tareas y objetivos)

Actividad	Tipo de resultado
<i>determinar recursos a consumir (8.b)</i>	Instancias de WFUsa asociando tareas y recursos
<i>indicar qué recursos se restablecen (7.a)</i>	Instancias de WFProduce asociando tareas y recursos
<i>asociar tareas con entidades mentales (7.b)</i>	Instancias de GTSatisface y GTFalla.
<i>refinar entidades mentales consumidas (8.a)</i>	Instancias de WFConsume
<i>asociar tareas con operaciones de las aplicaciones (7.c)</i>	Instancias de WFUsaLlamada
<i>refinar dependencias de objetivos (10.a)</i>	Determinar dependencias Y/O ente objetivos
<i>refinar satisfacción/fracaso de objetivos (10.b)</i>	Instancias de GTFallaObjetivo o GTSatisfaceObjetivo asociando objetivos y tareas.

## Evaluaciones con los módulos INGENIAS

- Módulo de servlet: flujos de trabajo
- Módulo de documentación

---

## ¿Y la implementación?

---

## Módulo de generación de código

- Se trata de un módulo en desarrollo que traduce tareas, estado mental, mecanismo de selección de tareas, guardas de interacciones
- El objetivo es mostrar cómo funciona un agente y que los estudiantes puedan experimentar
  - El usuario decide cuales de las tareas activas debe de ejecutarse
  - El usuario decide qué información debe ir en cada entidad mental
  - El usuario decide qué interacciones iniciar
- Al dejar el control en manos de los estudiantes, estos pueden comprender mejor la complejidad de los mecanismos internos de un agente
- Además, la propia arquitectura del agente se puede parametrizar con interfaces que permitan al alumno plantear sus propios mecanismos de control

---

## Proceso de desarrollo del prototipo

- El prototipo plantea un sub-proceso de desarrollo que corre paralelo al proceso que genera la especificación
- El desarrollo del prototipo reusa técnicas convencionales
- No está preparado todavía para desarrollo en entornos multi-usuario

---

## Implementación del prototipo

- El prototipo inicial se continua en paralelo a la especificación
  - Generar un módulo
  - Generar plantillas
  - Ir probando con las especificaciones
  - Establecer requisitos de la implementación
    - if-then-else
- Hacer crecer el prototipo para tener en cuenta más elementos del análisis
  - Tareas
  - Estado mental: definición y gestión
  - Condiciones de inicio de interacciones
- Feedback para la especificación: no todo es correcto. El prototipo ayuda a verificar que la especificación tiene todos los elementos necesarios y ninguno de los no deseados
  - No es una verificación en el sentido matemático de la palabra

## Traducción de los protocolos INGENIAS a JADE

---

- Idea general:
  - Cada agente sólo conoce del protocolo las unidades de interacción en las que participa
  - Inicialmente, no se sabe qué agentes van a participar, sólo los roles necesarios
  - Hay que asegurar que la interacción transcurre como debe
  - No se miran aspectos de interbloqueo o viveza
  - El protocolo no será totalmente sincronizado:
    - Existe una señal inicial que informa a todos los agentes participantes quien participa y bajo qué rol
    - Dos agentes, A y B, quieren enviar un mensaje. El destinatario son cualesquiera agentes distintos de A o B, quizá el mismo. Aunque el protocolo de INGENIAS establezca que A debe enviár antes que B, en la práctica, A y B enviarán el mensaje cuando puedan. El mensaje si no es procesado en el momento, quedará guardado en la cola de mensajes de cada agente

## Traducción de los protocolos INGENIAS a JADE

---

- El recorrido de los diagramas plantea varias necesidades
  - Saber qué roles juega cada agente
  - Saber en cuántas interacciones participa un agente: se deducen de los roles desempeñados
  - Saber qué unidades de interacción se definen en cada interacción
    - Establecer un orden de ejecución de las unidades
    - Extraer un orden parcial procedente de averiguar en cuántas de estas unidades de interacción participa el agente
  - Y otros más ....

## Las plantillas

---

- Las plantillas las obtenemos directamente del prototipo inicial
- Estas plantillas irán evolucionando para incorporar nuevos aspectos

## Asociando tareas a las interacciones

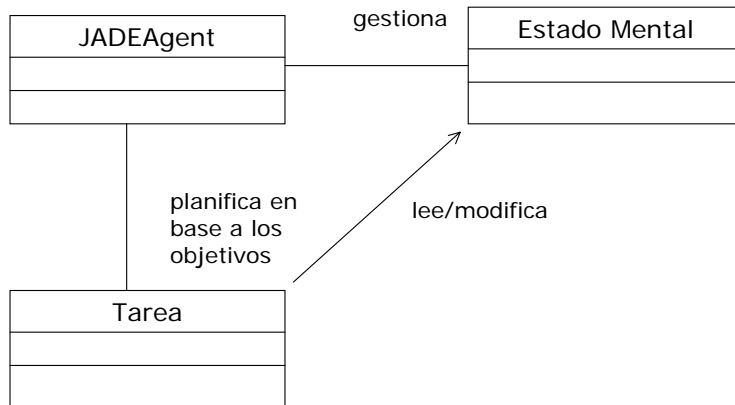
---

- Determinando cómo se ejecutan las tareas en los agentes teniendo en cuenta los objetivos de los agentes, las interacciones y los flujos de trabajo
  - parte del flujo de trabajo se puede desarrollar dentro del mismo agente
  - Las tareas se ejecutan porque forman parte de un flujo de trabajo o porque satisfacen un objetivo
    - Objetivos siempre activos. De esta forma se mantiene la racionalidad del agente y a la vez se facilita la adaptación de flujos de trabajo



## Explicando el tratamiento de protocolos

- Una instancia de *StateBehavior* representa una conversación en curso. El *state behavior* se encarga de recibir/enviar los mensajes que de cada especialización del protocolo se espera.
  - El envío se hace mediante métodos de la clase *SyncSender* y la recepción, con *SyncReceiver*.
  - Estas clases son las que crean comportamientos de envío y recepción de mensajes. Estos comportamientos se añaden a la cola de comportamientos de JADE y se ejecutan de acuerdo a su planificador de tareas.



## Explicando el tratamiento de protocolos

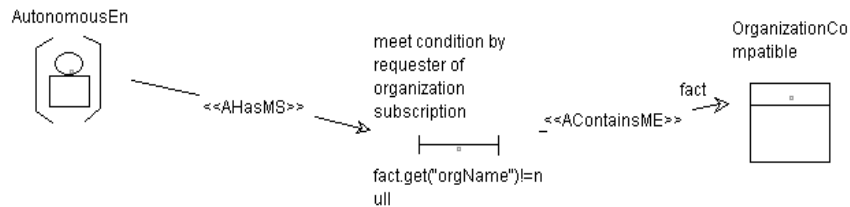
- Basándose en esta estructura, se recorre las unidades de interacción buscando instancias de la meta-relación *UIPrecedes*
  - Para cada unidad de interacción se decide el papel de cada uno de los actores de la interacción: emisores, receptores, o neutro
    - Será emisor si actúa como iniciador de una unidad de interacción
    - Será receptor si actúa como colaborador en una unidad de interacción
    - Será neutro para una unidad de interacción si no interviene ni como iniciador ni como colaborador
- Entonces, generar las máquinas de estados para cada agente consiste en determinar las unidades de interacción en las que participa y construir una sub-máquina con sólo estas entidades
  - Cada agente conoce sólo los estados en los que participa

## Generación de guardas

- Las guardas de las interacciones se interpretan como las condiciones que se deben cumplir para que un agente decida iniciar o colaborar en cualquier punto de la interacción. Estas guardas se expresan utilizando modelos de agente de INGENIAS

## Generación de guardas

- Se expresa que el agente debe tener un hecho llamado *OrganizationCompatible* y que este hecho debe contener el nombre de una organización a la que suscribirse. Este hecho se generaría mediante la tarea *select an organization to subscribe*, que obtendría el dato de un servicio de páginas amarillas para organizaciones



## Plantilla de guarda

- La información del estado mental se extrae con el método *getFact*
- La condición se extrae directamente de la entidad *ConditionalMentalState* del diagrama anterior
- El siguiente estado se extrae de los diagramas de interacción de INGENIAS

```
@repeat id="sendaction"@
if (sequence.equals("@v@iud@/v@"&& options.length>0) {
    int count=0;
    @repeat id="nextstates"@
    if (options[count].equals("@v@possibleiu@/v@")){
        @repeat id="condfacts"@
        Fact @v@label@/v@=getAgent().getFact("@v@type@/v@");
        @/repeat@
        if (@v@condition@/v@){
            sb.setState(options[count]);
        }
        count++;
    @/repeat@
    processed = true;
}
@/repeat@
```

## Plantilla rellena

- Después del proceso de instanciación con la información del diagrama visto anteriormente, la plantilla quedaría

```
if (options[count].equals("IDontWantToAdmitYou")){
    Fact fact=getAgent().getFact("OrganizationCompatible");
    if (fact.get("orgName")!=null){
        sb.setState(options[count]);
    }
}
count++;
processed=true
```

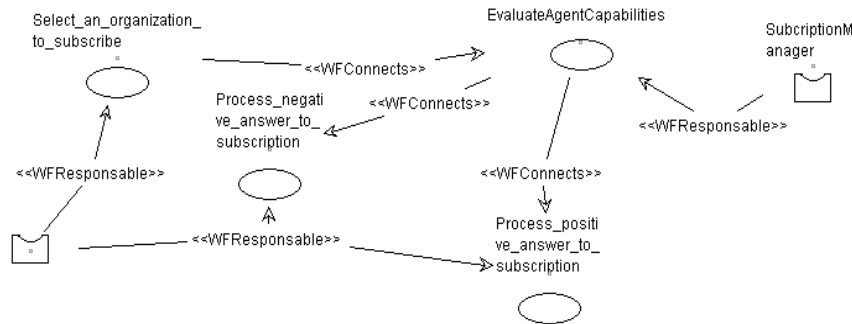
## Limitaciones de este código

- En el estado mental del agente no puede haber dos hechos con el mismo nombre
- No pueden tenerse dos etiquetas con el mismo nombre asociando dos hechos distintos en el mismo diagrama
- La condición incluida en el estado mental ha de ser una expresión booleana JAVA válida.



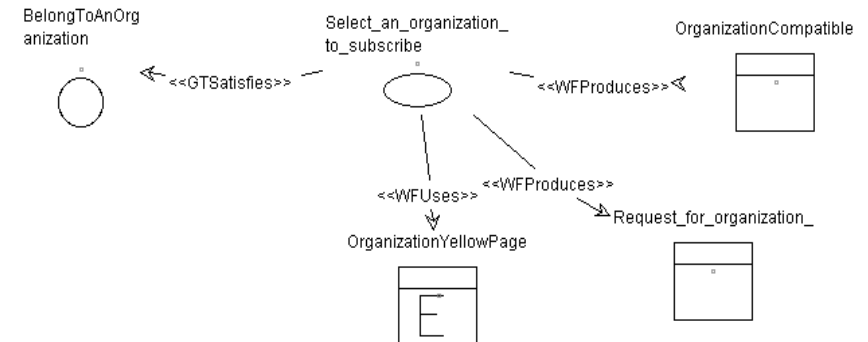
Y el mensaje a enviar

- Se obtiene al examinar el flujo de trabajo correspondiente
- Y las entidades que produce la tarea en cuestión, en este caso `Select_an_organization_to_subscribe`



La tarea a generar

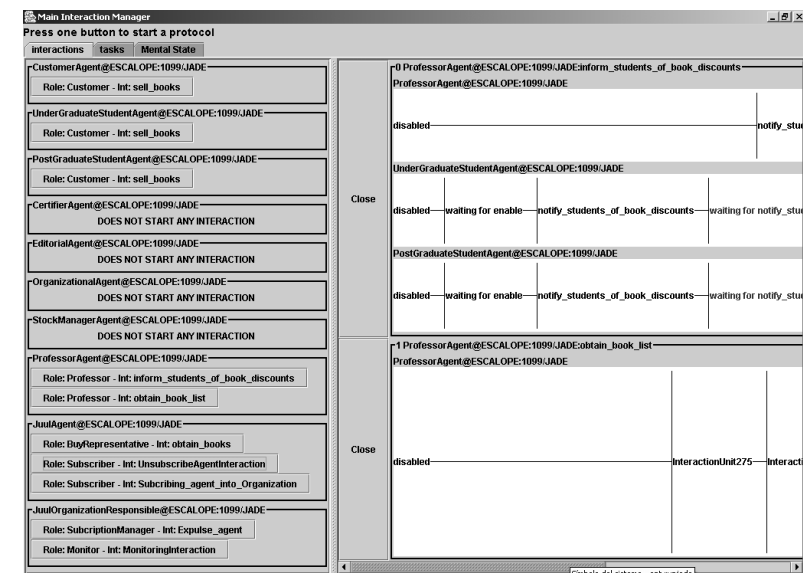
- La tarea que se está estudiando se ejecuta porque satisface el objetivo *Pertenecer a una Organización*. Esta tarea produce dos hechos: la organización a la que hay que subscribirse y la petición de subscripción



## Evitando errores mediante código redundante

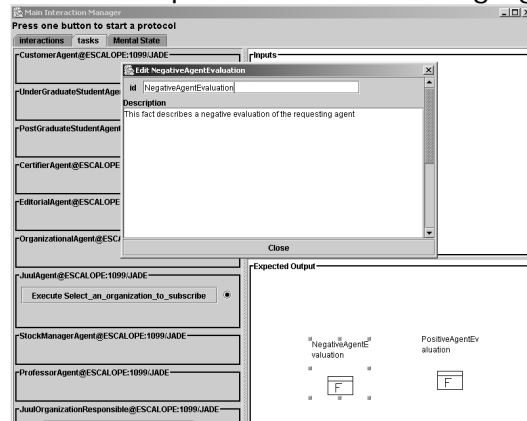
- En la versión actual, no se realiza ninguna verificación formal de que el protocolo no lleva a interbloqueos ni a inaniciones. Lo que sí se hace es introducir código redundante para asegurar que la implementación de los protocolos sigue lo establecido.
- En este sentido, los estados definidos así como las transiciones están replicados en varios lugares:
  - Cada máquina de estados está asociada con una clase que proporciona representación gráfica a los estados y las transiciones, la *StateMachineFrame*. Esta clase representa en tiempo de ejecución los diferentes estados por los que pasa la interacción y lanza excepciones si en algún momento se intenta hacer una transición no permitida.
  - La operación de cambio de estado está implementada como método *final* en la clase *State Behavior*, de la que todas las implementaciones de máquinas de estados heredan. Esta clase es la que tiene asociada una instancia de un *State Machine Frame*. Al realizar una cambio de estado, automáticamente, *State Behavior* invoca una transición de estado en *State Machine Frame* para detectar si es correcta o no.
  - La decisión de realizar una transición se toma en especializaciones de *Default Comm Control*. Esta clase contiene if-then-else que codifican, para cada estado, qué estados vienen a continuación y qué condiciones han de cumplirse para que la transición tenga lugar.
- Además, la información con la que se generan los estados y transiciones en las clases implicadas es la misma. De esta forma, la implementación hace menos probable que se cometan errores en la implementación de la especificación.

## Monitor de interacciones de los agentes



## Monitor de tareas activas

- Muestra las tareas activas de cada agente en función de los objetivos de los que dispone. Para cada tarea muestra las entradas de que dispone y las salidas que se esperan
- Estas entidades pueden modificarse/agregar/eliminarse



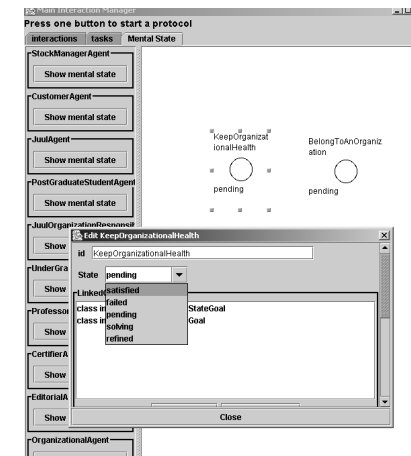
JGS & JPM  
Facultad de Informática UCM, 2004

Desarrollo con INGENIAS

85

## Monitor del estado mental de los agentes

- Muestra los objetivos, hechos, creencias y demás entidades mentales de los agentes. Permite modificarlas directamente para comprobar su efecto



JGS & JPM  
Facultad de Informática UCM, 2004

Desarrollo con INGENIAS

86

## Resumen

- El sistema ya está bastante especificado
  - Se han recorrido las etapas de inicio y de elaboración
- Se ha visto cómo el módulo afecta a la especificación
- ¿Y ahora qué?
  - Continuar con las actividades del proceso de desarrollo. Faltan diagramas
  - Continuar con el desarrollo del módulo: incorporar más aspectos
  - Seguir con la etapa de construcción e identificar dependencias sociales

JGS & JPM  
Facultad de Informática UCM, 2004

Desarrollo con INGENIAS

87

## Conclusiones de INGENIAS

- Líneas de trabajo
  - Más módulos
  - Reutilizar especificaciones
  - Identificar patrones
    - de comunicación: protocolos FIPA
    - de organización: modelos de empresa
    - de flujo de trabajo: reparto de responsabilidades

JGS & JPM  
Facultad de Informática UCM, 2004

Desarrollo con INGENIAS

88

## Filtrado colaborativo de información

---

- Se trata del caso de estudio desarrollado en la tesis original de INGENIAS

[http://grasia.fdi.ucm.es/ingenias/ejemplos/ejemplo\\_actividades/index.htm](http://grasia.fdi.ucm.es/ingenias/ejemplos/ejemplo_actividades/index.htm)

- Proporciona una descripción detallada de actividades involucradas

- Las actividades genéricas están en

<http://grasia.fdi.ucm.es/ingenias/integracion/index.htm>