MODELADO DE SISTEMAS MULTI-AGENTE

Memoria que presenta para optar al grado de Doctor

Jorge J. Gómez Sanz

Dirigida por los profesores

Francisco Garijo Mazario
Juan Pavón Mestras

Departamento de Sistemas Informáticos y Programación
Facultad de Informática
Universidad Complutense de Madrid

Junio 2002

Agradecimientos

Antes de nada, quiero agradecer a mis padres, a los que dedico esta tesis, la paciencia que han tenido conmigo (que han sido muchos años), los consejos que me han dado convenciéndome de que hiciera la carrera de informática (yo quería ser matemático o físico), y por supuesto el apoyo (moral y económico) que he recibido de ellos y sigo recibiendo (más moral que económico). También me gustaría agradecerle a Inma las tardes de salir juntos que me ha perdonado por tener que trabajar en esta tesis, proyectos, y otras cosas que han ido saliendo sin parar, y que aún no se como detener. Cualquier otra hubiera decidido buscarse alguién más atento y dedicado.

Hay que mencionar, por supuesto, a Jose Luis y Antonio, con los que he mantenido agradables charlas en diversos lugares algunos de los cuales ya no me atrevo a visitar. Han sido buenos amigos estos años de penalidades doctoriles y me han hecho pensar mucho en cómo debiera ser mi tesis.

A mis co-directores de tesis, Juan Pavón y Francisco Garijo. Comenzé a trabajar con Juan Pavón por una de esas casualidades de la vida y he seguido trabajando con él desde hace ya cuatro años en muchos proyectos. Espero que la racha se prolongue. Por cierto, que hay que tener paciencia para leerse esta tesis las veces que se las ha leído él. Conocí a Francisco Garijo al mismo tiempo que empecé a trabajar con Juan. Aunque al principio lo pasé mal, he aprendido con él más de lo él se imagina y de lo que en un principio quise admitir. Hay que reconocer que si he sido capaz de hablar de ingeniería del software en esta tesis, es gracias a él.

También quiero agradecer a Carmen Fernández su poder de convicción y su buen juicio. Si no es por estos dones, hace tiempo que habría emigrado a la empresa. También quiero mencionar a Antonio Vaquero, con el que redescubrí que existen cosas interesantes anteriores a los 90. Gracias a él, di con Newell y Simons y otros grandes de la historia de la informática.

Por supuesto, por haber aguantado mi presentación de tesis, a Balta, Alfredo, Pedro, Mercedes, Luis, Antonio Sarasa, que no los habia mencionado, y a Carmen, Jose Luis, Antonio Navarro y Antonio Vaquero, otra vez. Creo que no seré el único en alegrarse cuando se admita esta tesis.

Para terminar, querría recordar a mi abuelo. Estaba muy orgulloso de su nieto el ingeniero, y, por desgracia, ya no podrá presumir de su nieto el doctor ingeniero.

Índice

Prólogo

En la última década ha cobrado fuerza la hipótesis de que si los programas que componen los sistemas distribuidos fueran inteligentes, si pudiesen reorganizar sus funciones para corregir o tolerar los fallos y si su coordinación pudiese estructurarse en términos intuitivos, el diseño y el mantenimiento de dichos sistemas sería más fácil de elaborar, más adaptable y más fiable.

En la construcción de estos programas se está aplicando una tecnología íntimamente relacionada con la inteligencia artificial. Se trata de los *agentes* que, de momento, se entenderán como programas autónomos e inteligentes. Bajo el punto de vista de esta tecnología, los sistemas distribuidos pasan a ser *sistemas multi-agente* (SMA). Los SMA se desarrollan sobre *middleware* y proporcionan un nuevo nivel de abstracción más intuitivo. El diseño de SMA, generalmente, se aborda pensando en los agentes como entes con motivación. En lugar de modelar un sistema con componentes que ejecutan métodos, el desarrollador tiene que pensar en los objetivos que deben alcanzar los componentes y en las tareas necesarias para resolver sus objetivos. Al desarrollar los componentes así, se espera que el proceso sea más intuitivo ya que esta forma de modelar y de razonar de acuerdo con planteamientos cognitivos se halla más cerca del pensamiento humano que los paradigmas de programación tradicionales.

La construcción de SMA integra tecnologías de distintas áreas de conocimiento: técnicas de *ingeniería del software* para estructurar el proceso de desarrollo; técnicas de *inteligencia artificial* para dotar a los programas de capacidad para tratar situaciones imprevistas y tomar decisiones, y *programación concurrente y distribuida* para tratar la coordinación de tareas ejecutadas en diferentes máquinas bajo diferentes políticas de planificación. Debido a esta combinación de tecnologías, el desarrollo de SMA se complica. Existen plataformas de desarrollo que dan soluciones parciales al modelado de comportamiento y a la coordinación de agentes. El rango de estas soluciones va desde proporcionar servicios básicos (gestión de agentes, librerías de algoritmos, localización de agentes o movilidad), como JADE [Bellifemine, Poggi y Rimassa 01], Grasshopper [Baümer et al. 00] o ABLE [IBM 02], hasta entornos de desarrollo donde se parametrizan armazones (*framework*) software, como ZEUS [Nwana et al. 99] o AgentTool [Wood y DeLoach 00].

Aunque facilitan el proceso, las plataformas de desarrollo quedan incompletas sin un proceso de desarrollo de software especializado para agentes que haga similar la creación de SMA a la producción de software convencional. Las técnicas convencionales de ingeniería (RUP [Jacobson, Booch y Rumbaugh 00], CommonKADS [Tansley y Hayball 93]) no tienen en cuenta las necesidades de especificación de los SMA, como la especificación de planificación de tareas, intercambio de información con lenguajes de comunicación orientados a agentes, movilidad del código o motivación de los componentes del sistema. Por ello, se plantean nuevas metodologías basadas en agentes (BDI [Kinny, Georgeff y Rao 97], *Vowel Engineering* [Ricordel 01], MAS-CommonKADS [Iglesias 98], GAIA [Wooldridge, Jennings y Kinny 00]). Estas metodologías parten de un modelo (informal) de cómo debe ser un SMA y dan guías para su construcción. En las primeras metodologías, las guías consistían en una lista breve de pasos a seguir. Las más modernas, aunque han progresado en la integración con la ingeniería del software clásica, aún no muestran un grado de adaptación adecuado. El motivo principal es que siguen faltando herramientas de soporte y un lenguaje para la especificación del SMA que permitan trabajar de forma similar a como se trabaja en Rational Rose, TogetherJ o Paradigm +.

El objetivo de esta tesis es hacer que el desarrollo de SMA pueda seguir un proceso bien definido de la misma manera que ocurre con otro tipo de software. Para ello se propone un

lenguaje de especificación de SMA y su integración en el proceso de desarrollo basándose en el trabajo que he realizado dentro de MESSAGE [Caire et al. 02] [Caire et al. 01]. La metodología MESSAGE tiene como característica más relevante la utilización de *meta-modelos* para definir cómo debe ser la especificación del sistema y herramientas de soporte para facilitar el proceso de generación de especificaciones, concretamente METAEDIT+ [Lyytinen y Rossi 99]. Los *meta-modelos* son especificaciones para la construcción de *modelos* generadas con *lenguajes de meta-modelado*. Un *meta-modelo* define las primitivas y las propiedades sintácticas y semánticas de un *modelo*.

MESSAGE aportó mejoras sobre las metodologías existentes, pero también dejó numerosos aspectos abiertos, en concreto, la cohesión de los meta-modelos, el papel de los meta-modelos en el ciclo de desarrollo, la cantidad de información que debe contener un modelo y cómo modelar el entorno. Respecto de la cohesión de los meta-modelos, los definidos en MESSAGE fueron desarrollados de forma independiente e integrados sin tener en cuenta lo que aportaba cada uno a la especificación del sistema y cómo debían combinarse. En cuanto al uso de los meta-modelos en el ciclo de desarrollo, MESSAGE pretendía abordar todas las fases del ciclo de desarrollo de los SMA desde la definición de requisitos hasta la implementación. Sin embargo, el uso de los meta-modelos se centró con prioridad en el análisis. El resto de las etapas de desarrollo no fue estudiado de la misma forma. Por ejemplo, la aplicación de los meta-modelos en diseño no llegó a detallarse más allá de una traducción de entidades de análisis a elementos computacionales tales como máquinas de estado o bases de datos. Acerca del nivel de detalle que soporta un modelo, los modelos reflejaban su orientación a la etapa de análisis del sistema. El meta-modelo de interacción, por ejemplo, se limita a reflejar pasos de mensaje entre agentes del sistema. No se tienen en cuenta el estado mental de los agentes durante las interacciones, ni la posibilidad de trabajar con protocolos en lugar de mensajes, ni la asociación de tareas durante la ejecución de la interacción. En cuanto a la representación del entorno, se trata de un aspecto que no se solucionó completamente. En un principio, el meta-modelo del dominio era el responsable de modelar aspectos del entorno, pero tal y como estaba definido era complicado. No había forma de representar cómo se llevaba a cabo la percepción de los agentes o cómo actuaban los agentes sobre los elementos del entorno.

El trabajo de investigación descrito en la presente memoria parte de los resultados de MESSAGE mejorándolos en los aspectos antes mencionados. Para ello, se han revisado los modelos iniciales, completándolos con resultados de investigación relacionados, y se han definido nuevos modelos teniendo en cuenta las necesidades del proceso de desarrollo. Como resultado, se tienen cinco meta-modelos: el meta-modelo de agente, de organización, de interacción, de entorno y tareas y objetivos.

Como herramienta de soporte para el lenguaje de especificación se ha utilizado la herramienta utilizada en MESSAGE personalizándola con los meta-modelos descritos en este trabajo. Mediante esta herramienta, se han desarrollado varios casos prácticos para validar los meta-modelos. Estos casos prácticos han servido también para probar la integración de los meta-modelos en las actividades del ciclo de ingeniería de software aportando soluciones para su utilización práctica por los ingenieros.

El método de trabajo para conseguir los resultados precedentes ha sido el siguiente:

- Estudiar el estado del arte para determinar qué metodologías existen y qué aspectos hay sin tratar o que necesitan mayor detalle. También se ha estudiado las exigencias metodológicas por parte de SMAs existentes. Para cada uno de los meta-modelos identificados, se han estudiado las áreas de investigación

relacionadas para determinar su aportación. Cada resultado de investigación incluido ha sido probado dentro de diversos casos de uso que serán mostrados a lo largo de este trabajo.

- Definir los meta-modelos y probar su viabilidad. Las construcciones de los meta-modelos deben adaptarse al uso que se va a hacer de ellos y no al revés. El uso en este caso está dado por el sistema real que haya que modelar.

- Determinar las actividades necesarias para generar los -modelos a partir de los meta-modelos. El descubrimiento de estas actividades surge de la aplicación de los meta-modelos a los casos de estudio de este trabajo.

- Estudiar cómo traducir los modelos resultantes a entidades computacionales. El paso de diseño a implementación, se ve como la parametrización de un armazón software con los diferentes modelos obtenidos en el diseño. A nivel metodológico, se proporciona un medio de parametrización genérico que no depende del lenguaje de implementación y que se basa en la aplicación de lenguajes de marcas.

Los resultados obtenidos se describen en los cinco apartados de la presente memoria.

En el capítulo primero, se presentan diferentes enfoques seguidos en el desarrollo de agentes y SMA. En este capítulo, se repasan brevemente desarrollos basados en lenguajes de agentes y plataformas de agentes. La revisión de estos enfoques tiene como fin mostrar que hacen falta metodologías que guíen el proceso de desarrollo. El capítulo se completa introduciendo las metodologías más relevantes en el estudio de SMA desde el punto de vista de este trabajo.

El capítulo segundo, describe los meta-modelos usados en la especificación del sistema. Cada meta-modelo se centra en una vista o aspecto de definición de un SMA (agentes, entorno, interacción, organización, objetivos y tareas). Al estudiar estos aspectos se hace referencia a investigación relacionada como teoría de coordinación, organización de los agentes o modelado de estado mental. Así, cada meta-modelo se correlaciona con resultados de investigación en agentes o áreas relacionadas. Además, la presentación se complementa con ejemplos de aplicación y con estudios de su influencia en los otros cuatro meta-modelos.

El capítulo tercero se centra en la aplicación de los meta-modelos en un proceso de desarrollo específico como el RUP(Rational Unified Process). Para ello, se determinan las actividades de generación de los modelos que definen el sistema dentro de las fases de análisis y diseño, siguiendo las etapas marcadas por RUP. El capítulo incluye como paso de implementación la generación automática de código a partir de la especificación de los meta-modelos. Se propone una generación de código basada en XML. Con ello se consigue que sea independiente de la plataforma de agentes o del lenguaje de agentes utilizado.

El cuarto capítulo presenta el desarrollo de un sistema para la personalización de información textual en Internet siguiendo la metodología propuesta y según las etapas especificadas dentro del RUP. Este caso de estudio corresponde a un desarrollo alternativo del trabajo realizado dentro del proyecto PSI3 [PSI3 01].

Las conclusiones y el trabajo futuro se presentan en el quinto capítulo. La metodología presentada permite desarrollos estructurados según el RUP utilizando un lenguaje de definición de SMA (los meta-modelos) de un nivel de abstracción similar al UML y soportado por herramientas software. Se proponen también diferentes líneas de investigación de entre las que destacan la integración de técnicas para el desarrollo de sistemas en tiempo real y sistemas móviles.

Capítulo 1. INTRODUCCIÓN

Los *agentes* nacieron de la investigación en Inteligencia Artificial (IA) y más concretamente de la Inteligencia Artificial Distribuida (DIA). Al principio, la IA hablaba de los agentes como programas especiales cuya naturaleza y construcción no se llegaba a detallar. Recientemente se ha escrito mucho acerca de este tema explicando qué se puede entender por un agente y qué no [Franklin y Graesser 96] [Wooldridge M. y Jennings N.R. 98] [Gómez-Sanz 00], en general atendiendo a características que debe tener un agente: autonomía, reactividad, iniciativa, habilidad social, etc.

En esta tesis se toma la definición de Newell [Newell 82], una definición de agente influenciada por la IA. Newell asume que existen diferentes niveles de abstracción en el diseño de sistemas, entre ellos el *nivel de símbolos* y el *nivel de conocimiento*. Del mismo modo que un programa convencional maneja símbolos (e.g. expresiones, variables) en el *nivel de símbolos*, un agente, que equivale a un programa en el *nivel del conocimiento*, maneja únicamente conocimiento y se comporta de acuerdo con el *principio de racionalidad*. Este principio estipula que un agente emprende acciones porque está buscando satisfacer un objetivo.

Se ha tomado esta definición porque establece que el concepto de agente es simplemente una forma de ver un sistema, como también indica Shoham [Shoham 93], que luego puede reflejarse *a nivel de simbolos*, con un programa tradicional. Esta concepción es compatible con el ciclo de vida del software. De forma general, se pensaría en el agente como un ente que se comporta de acuerdo con el *principio de racionalidad*. Cuando hubiese que desarrollar físicamente el agente, se atendería a su arquitectura y componentes, lo cual entraría dentro del *nivel simbólico*.

El diseño de agentes aislados ha sido estudiado en profundidad en la IA. Como resultado, se tienen las arquitecturas de subsunción (Brooks [Brooks 91b]), arquitecturas de pizarra (Hearsay II [Carver y Lesser 92]), arquitecturas BDI (IRMA [Bratman, Israel y Pollack 88]) y arquitecturas para la resolución genérica de problemas (SOAR [Laird, Newell y Rosenbloom 87] [Laird, Bates Congdom y Coulter 99]).

Teniendo en cuenta el carácter distribuido de los entornos de aplicación de agentes (especialmente tratamiento de información en internet) es normal concebir la colaboración de varios agentes. En tales casos, la atención se centra en cómo construir Sistema Multi-Agentes (SMA) que pueden entenderse como grupos de agentes que interaccionan entre sí para conseguir objetivos comunes. La definición de un SMA se puede encontrar en [Ferber 99], donde un SMA es un sistema que reúne los elementos de la Ilustración 1

1. Un entorno
2. Un conjunto de objetos. Estos objetos se encuentran integrados con el entorno, i.e. es posible en un momento dado asociar uno de estos objetos con un lugar en el entorno. Estos objetos son pasivos, pueden ser percibidos, creados, destruidos y modificados por agentes.
3. Un conjunto de agentes que se consideran como objetos especiales que representan las entidades activas del sistema.
4. Un conjunto de relaciones que unen objetos, y, por lo tanto, agentes.
5. Un conjunto de operaciones, que hacen posible que los agentes perciban, produzcan, consuman, transformen y manipulen objetos.
6. Operadores que representan la aplicación de operaciones sobre el mundo y la reacción de éste al ser alterado. Estos operadores se pueden entender como *las leyes del universo*.

Ilustración 1. Elementos requeridos en un SMA.

Según esta definición, la influencia de unos agentes en otros viene dada no solo por la comunicación explícita sino también por la actuación sobre el entorno. Este hecho aumenta la complejidad del desarrollo de SMA enormemente, ya que obliga a estudiar el entorno con detalle para detectar qué acciones realizadas por un agente pueden afectar a otro agente.

En el desarrollo de SMA existen dos filosofías distinguidas por la carga de componente práctica. La primera ve el SMA como el resultado de utilizar un lenguaje de especificación de agentes. Para generar SMA de esta manera, se parte de principios basados en modelos operacionales y modelos formales de SMA. La segunda estudia el SMA como un sistema software que hay que construir. El desarrollo no parte de cero, sino que utiliza plataformas de desarrollo de agentes que proporcionan servicios básicos de comunicación, gestión de agentes y una arquitectura de agente. En cualquiera de los dos casos, y sobre todo cuando el sistema a desarrollar es grande, se necesitan metodologías que estructuren el desarrollo de acuerdo con las prácticas de ingeniería del software. Para ilustrar ambas filosofías y entender el lugar de esta tesis dentro de la investigación en Ingeniería del Software Orientada a Agentes, se ha dividido este capítulo en varias secciones. El objetivo de las dos primeras es mostrar que disponer de plataformas de desarrollo o lenguajes de agentes no es suficiente para generar SMA.

La primera sección estudia la relación de este trabajo con los lenguajes de descripción de agentes. Se presta atención a lenguajes de agentes basados en teorías de agentes y lenguajes basados en especificaciones operaciones de cómo son sus intérpretes. Como representativos de los primeros, se introduce brevemente ConGOLOG [Lespérance et al. 96] [De Giacomo, Lespérance y Levesque 00], basado en demostradores automáticos, y como representante de los segundos, Agent0 [Shoham 93] el lenguaje de agentes más referenciado en la literatura.

La segunda muestra plataformas de desarrollo relacionadas con este trabajo. Aunque existen muchos desarrollos de este tipo, la sección se centra en plataformas que implementan estándares de desarrollo de SMA (JADE [Bellifemine, Poggi y Rimassa 01] y Grasshopper [Baümer et al. 00]) y plataformas para el desarrollo rápido de SMA (ABLE [IBM 02] y ZEUS [Nwana et al. 99]).

El resto del capítulo se centra en el aspecto de más interés para esta tesis: metodologías para el desarrollo de SMA. Se ha seleccionado un conjunto de metodologías para determinar qué ofrecen frente a este trabajo. En la elección de las metodologías han sido determinantes dos aspectos: utilización de diferentes vistas para la especificación del sistema y la voluntad de integración de técnicas de ingeniería y teoría de agentes. De acuerdo con estos criterios, se han identificado siete metodologías relevantes. La primera es la ingeniería de vocales (*vowel engineering*) [Demazeau 95] que fue una de las primeras en considerar diferentes aspectos (agentes, entorno, interacciones y organización) en el desarrollo de SMA. La segunda es MAS-CommonKADS [Iglesias et al. 98] que, debido a su origen (CommonKADS [Tansley y Hayball 93]), se trata de una metodología orientada al desarrollo utilizando experiencia práctica en sistemas expertos. A continuación el diseño basado en BDI [Kinny y Georgeff 97] que ha influido notablemente en la forma de concebir el control de los agentes. Después, se estudian dos metodologías que son las únicas que están soportadas por herramientas: ZEUS [Nwana et al. 99] y MaSE [DeLoach 01]. Seguidamente, se estudia GAIA [Wooldridge, Jennings y Kinny 00], de gran influencia en el mundo anglo-sajón, que estudia la definición de vistas en una metodología y trata de integrarse en un ciclo de vida de software tipo cascada. Por último, MESSAGE [Caire et al. 02], en la que los directores de este trabajo y el autor participaron activamente, que define los elementos que se tienen que considerar para desarrollar un SMA, mediante la especificación de meta-modelos, y propone la adopción de un proceso de desarrollo estándar, el RUP.

1.1 Lenguajes de agentes y el diseño de Sistemas Multi-Agente

Los lenguajes de agentes parten, en su mayoría, de modelos operacionales que definen la semántica de sus instrucciones. Estos modelos se suelen describir informalmente, aunque también existen trabajos que primero parten de un modelo operacional expresado como arquitectura software que luego es formalizado, como es el caso de *AgentSpeak(L)* [Rao 96].

El trabajo más referenciado en lenguajes de agentes es Agent0 [Shoham 93] que acuñó el término *programación orientada a agentes*. Agent0 propone un nuevo paradigma de programación en el que la entidad principal es el agente. En Agent0, un agente es una entidad cuyo estado se ve como un conjunto de componentes mentales tales como *creencias*, *habilidades*, *elecciones* y *compromisos*. Con estas entidades y un conjunto de primitivas, como enviar mensaje, comprometerse o solicitar la ejecución de una tarea, se elabora un lenguaje de descripción de agentes. Exiten otros lenguajes que siguen el modelo de Shoham como CASA [Flake y Geiger 99] o PLACA. Estos lenguajes añaden principalmente la capacidad de planificar acciones de los agentes en el SMA.

Como alternativa a Agent0, se tienen enfoques orientados a teorías de agentes, donde se enuncian definiciones formales con las que se construye el SMA para a continuación demostrar propiedades del mismo. Ejemplos de tales propiedades son la viveza del sistema, pensando en el SMA como un sistema concurrente, o que los agentes no necesiten considerar como logros intencionados los efectos secundarios de sus acciones, desde el punto de vista de la teoría de la intención [Wooldridge y Jennings 95]. El interés de las teorías de agentes en esta tesis, no consiste en las propiedades que se enuncian, sino en su implementación ya que dan lugar a lenguajes de agentes. El problema de este otro tipo de

lenguajes de agentes, es que dependen fuertemente de métodos de demostración automática [Davila Quintero 97]. Para dar idea de cómo son estos lenguajes se ha elegido ConGOLOG [Lespérance et al. 96] [De Giacomo, Lespérance y Levesque 00]. ConGOLOG se centra en modelar la ejecución de tareas asignadas a varios agentes y su efecto en el entorno. Habla de entidades de conocimiento modificables por las tareas, a las que denomina *fluents*, define axiomas de ejecución de tareas, axiomas de precondición de tareas y axiomas de marco para especificar a qué *fluents* afecta la ejecución de tareas. La ejecución del SMA consiste en descubrir qué acciones se pueden ejecutar en el SMA que no creen inconsistencias en el conjunto de axiomas acumulados, que son el conjunto de precondiciones asociadas a las acciones posibles de los agentes, los efectos de las tareas que hayan ejecutado más las precondiciones asociadas al estado inicial. Si se descubren estas acciones, se puede hablar de un estado siguiente al actual añadiendo el efecto de las acciones seleccionadas a los axiomas acumulados.

Existen más lenguajes de alto nivel con los que se pueden construir agentes, sin embargo, no se trata de lenguajes de agentes en el sentido de Shoham o ConGOLOG, sino de lenguajes donde se proporcionan medios para tratar la concurrencia, como April [McCabe y Clark 95] o Concurrent Metamem [Fisher 91] [Fisher 95].

De los dos enfoques mostrados, los representados por Agent0 y los basados en teorías de agentes, en este trabajo se ha dado más relevancia los primeros. El motivo es que esta tesis se centra en el desarrollo de SMA partiendo de modelos cercanos a las prácticas de ingeniería. Los desarrollos basados en teorías de agentes son más cercanos a la lógica matemática y requieren esfuerzos en su aplicación que todavía no son aceptables. Por ejemplo, para aplicar ConGOLOG, hay que predecir el efecto de las acciones sobre el entorno, lo cual no siempre es posible. Sin embargo, los lenguajes como Agent0 no requieren grandes conocimientos de lógica y, además, se basan en modelos operacionales de los agentes. La descripción de estos modelos va desde algoritmos descritos utilizando pseudo-código, como en AgentSpeak(L), hasta la enumeración de los componentes del agente, su función y cómo fluye la información de un componente a otro, como en Agent0 y CASA.

Desde un punto de vista metodológico, el principal problema de estos lenguajes es su aplicación a desarrollos de complejidad media. Los lenguajes de agentes pueden verse como lenguajes de implementación de más alto nivel que otros más convencionales como C++ o JAVA, pero con una particularidad: son muy pobres en mecanismos de abstracción y encapsulación. Los programas o especificaciones generados con ellos tienen como unidad principal de encapsulación el agente y de abstracción procedimental la tarea, lo cual limita su aplicación. Así pues, a los problemas de desarrollar un sistema directamente con un lenguaje de implementación se une la falta de medios para realizar esta tarea incrementalmente.

Entonces, la necesidad de metodologías es doblemente justificable. Por un lado, para cubrir las deficiencias de los lenguajes de agentes en cuanto a mecanismos de encapsulación y abstracción. En este sentido, las metodologías actuales definen elementos que les sirven para agrupar las distintas funcionalidades asociadas a un agente o a un grupo de agentes. Así aparecen conceptos como *rol* o *servicio*. Y por otro lado, para facilitar la comprensión de sistemas complejos tal y como ocurre con los lenguajes convencionales de implementación. Este aspecto, como se verá en las secciones posteriores, no se haya resuelto completamente en las metodologías existentes, ya que la idea de desarrollo incremental no aparece de forma explícita.

1.2 Plataformas para diseño de Sistemas Multi-Agente

El desarrollo de SMA hoy en día es más proclive a la utilización de plataformas de desarrollo que a la aplicación de lenguajes de agentes. Esto se debe en gran parte al nivel de conocimientos necesarios que generalmente implica programar con un lenguaje de agentes. Por ello, han proliferado por un lado armazones software de SMA adaptables a diferentes dominios de aplicación y por otro plataformas de desarrollo de ámbito genérico que son implementaciones de estándares de agentes. Aunque el desarrollo con los armazones es más sencillo, hoy en día predominan los segundos.

Las plataformas de desarrollo más extendidas son JADE [Bellifemine, Poggi y Rimassa 01] y Grasshopper [Breugst y Magedanz 98]. JADE es la implementación oficial del estándar FIPA [FIPA 95], y soporta todos los servicios básicos de infraestructura especificados en FIPA (comunicaciones, movilidad, gestión de agentes y localización de agentes), a los que añade algunas utilidades gráficas para facilitar la administración de las plataformas y la depuración de los mensajes intercambiados por agentes en tiempo de ejecución. Grasshopper es la implementación del estándar MASIF [Baümer et al. 00], que soporta la movilidad de agentes en un entorno distribuido utilizando comunicación y servicios CORBA [OMG 00a]. En JADE y Grasshopper existe una arquitectura básica de agente que hay que utilizar para acceder a los servicios de la plataforma correspondiente.

El diseño de agentes con estas plataformas significa atenerse a unos estándares de comunicación y de gestión de agentes. El resto, como la especificación del control del agente, su inteligencia o las relaciones entre las tareas del sistema, se deja al criterio del desarrollador. La aportación de una metodología a desarrollos basados en este tipo de plataformas consistiría en organizar el proceso de generación del SMA y en proporcionar elementos para el diseñador pueda describir estos aspectos teniendo en cuenta las restricciones de la plataforma destino.

En cuanto a los armazones software, la aportación de las metodologías varía según el caso. ZEUS [Nwana et al. 99], por ejemplo, presenta un entorno de desarrollo con el que se programa visualmente el SMA. Las posibilidades de configuración del SMA a través de este entorno son muy variadas. Entre otros, hay que suministrar una ontología, reglas de comportamiento, planes de ejecución de tareas y mensajes a enviar a otros agentes.

Debido a esta versatilidad en su configuración, el diseño con ZEUS se hace tan complejo como programar con lenguajes de agentes. Aunque se facilita el desarrollo de aspectos como la coordinación, quedan pendientes decisiones como qué hay que coordinar y para qué. Por ello, la aportación de una metodología en este caso consistiría en proporcionar un ciclo de desarrollo, actividades y elementos conceptuales que ayudasen al descubrimiento incremental de los elementos requeridos por ZEUS. Un ejemplo, aunque limitado, de metodología para ZEUS se tiene de mano de sus propios desarrolladores [Collis y Ndumu 99], donde se aplica la idea de rol para guiar el descubrimiento de funciones del sistema.

ABLE (*Agent Building and Learning Environment*) [IBM 02] aunque también permite el prototipado rápido de SMA, no llega al nivel de ZEUS. ABLE es una herramienta de IBM para la construcción de sistemas de agentes inteligentes donde todos sus elementos, incluso los agentes, se construyen por composición de *AbleBeans*, una extensión de los *JavaBeans*[1]. Son de interés un conjunto de AbleBeans especializados que implementan sistemas de

[1] Un JavaBean es un componente portable e independiente de la plataforma escrito en Java. Y como componente que es, una JavaBean permite ser inspeccionado, personalizado y maneja una serie de eventos que le permiten comunicarse con el exterior o con otros JavaBeans

aprendizaje estadísticos (mapas autoorganizativos, redes neuronales, mapas de conexión) y control simbólico (razonamiento por encadenamiento hacia delante y lógica de predicados). ABLE incorpora también un entorno visual de desarrollo donde se interconectan *AbleBeans*. El interés de esta plataforma es que soluciona visualmente la construcción y comunicación de los agentes.

ABLE no llega a detallar cómo se obtiene la funcionalidad de los agentes y cómo deben actuar en cada situación. La aportación de una metodología a ABLE sería más amplia que en el caso de ZEUS. Se trataría de dar medios para detallar aspectos de control del agente teniendo en cuenta los elementos de control que vienen predefinidos en ABLE. Sería útil, por ejemplo, la idea de casos de uso [Jacobson, Booch y Rumbaugh 00] para identificar conjuntos de funciones a proporcionar por el sistema y escenarios para describir cómo se espera que se realice el proceso de aprendizaje de los agentes.

1.3 Vowel Engineering

Se trata de la metodología seguida en el grupo MAGMA [MAGMA 02]. El término *vowel engineering* viene de que el sistema final depende de la ordenación y agrupamiento de cuatro vocales A (por agentes), E(por entorno), I (por interacciones) y O (por organización). Los modelos de agente abarcan desde simples autómatas hasta complejos sistemas basados en conocimiento. La forma de ver las interacciones van desde modelos físicos (propagación de onda en el medio físico) hasta los actos del habla (*speech acts*). Las organizaciones van desde aquellas inspiradas en modelos biológicos hasta las gobernadas por leyes sociales basadas en modelos sociológicos.

El propósito de *Vowel engineering* es lograr librerías de componentes que den soluciones al diseño de cada uno de estos aspectos, para que posteriormente, el diseñador seleccione un modelo de agente, un modelo de entorno, un modelo de interacciones y modelos de organización a instanciar.

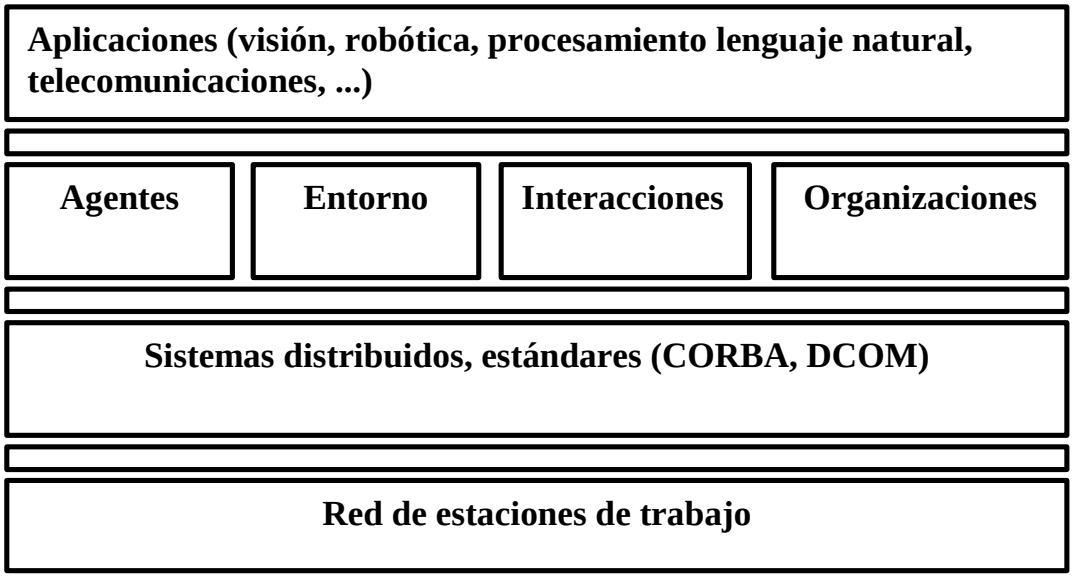


Ilustración 2. Arquitectura de capas del Sistema Multi-Agente propuesto

Como ejemplo, [Demazeau 95] propone para aspectos de interacción un lenguaje para la descripción de protocolos de interacción basados en procesos de comunicación síncronos o

asíncronos donde la semántica es muy similar a la de los actos del habla. La representación en sí se hace mediante redes de transición en las que los arcos se corresponden con los mensajes intercambiados y los estados reflejan la situación global (i.e. no hay posibilidad de que un estado se refiera al estado de un agente concreto).

Una de las más recientes publicaciones [Ricordel 01] de *Vowel Engineering* propone la implementación mediante la plataforma *Volcano*. La plataforma utiliza el ensamblaje de componentes utilizando lenguajes de composición arquitecturas, concretamente UniCon [Carnegie Mellon 02]. El desarrollo consiste en ensamblar componentes que pueden ser desarrolladas *ad-hoc* o proceder de una librería. Cada componente pertenece a una categoría concreta de las cuatro consideradas.

Pese a que en la literatura se indica que existe un entorno de desarrollo basado en estas ideas, este no es público. De todas formas, el trabajo desarrollado dentro de MAGMA con esta metodología es reseñable debido a su variedad en los dominios de aplicación (Sistemas de información geográfica, Robocup, simulaciones de mercados o agentes en tiempo real).

Vowel Engineering ha sido una de las primeras metodologías en modelar sistemas utilizando diferentes vistas. Aunque es prometedor, el trabajo en *Vowel Engineering* está incompleto ya que no termina de estabilizarse con herramientas de soporte. Además, no existen instrucciones acerca de cómo describir cada uno de los aspectos considerados.

A favor de esta metodología está la visión del modelado de sistemas como composición de elementos. Esta composición se define con un lenguaje apropiado para el problema como es un lenguaje de descripción de arquitecturas, Unicon en este caso. Como consecuencia de esta forma de desarrollo, hay que señalar que facilita la reutilización de código. Metodológicamente es mejorable. Se ha incluido aquí por ser un pionero en la definición de sistemas mediante vistas, pero necesita de más trabajo sobre todo en el estudio de cómo especificar el sistema y cómo esta especificación evoluciona a lo largo del desarrollo.

1.4 MAS-CommonKADS

Esta metodología extiende CommonKADS [Tansley y Hayball 93] aplicando ideas de metodologías orientadas a objetos y metodologías de diseño de protocolos para su aplicación a la producción de SMAs [Iglesias 98;Iglesias et al. 98]. La metodología CommonKADS gira en torno del modelo de experiencia (explicado más tarde) y está pensada para desarrollar sistemas expertos que interactúan con el usuario. De hecho considera sólo dos agentes básicos: el usuario y el sistema. Este hecho influye en el modelo de comunicación que, consecuentemente, trata de interacciones hombre-máquina.

Esta metodología ha sido la primera en hacer un planteamiento de SMA integrado con un ciclo de vida de software, concretamente el espiral dirigido por riesgos [Pressman 82]. Propone siete modelos para la definición del sistema: agente, tareas, experiencia, coordinación, comunicación, organización y diseño. Cada modelo presenta una reseña a la teoría sobre la que se basa. El modelo en sí parte de una descripción gráfica que luego se complementa con explicaciones en lenguaje natural de cada elemento. Existe por cada modelo una descripción de las dependencias respecto de otros modelos y de las actividades involucradas. Estos modelos se hayan descritos ampliamente en [Iglesias 98] en lenguaje natural, complementándose con otras notaciones como SDL (*Specification and Description Language*) [International Telecommunication Union 99 A.D.b] o MSC (*Message Sequence Chart*) [International Telecommunication Union 99 A.D.a] para describir el comportamiento de los agentes cuando interaccionan.

MAS-CommonKADS es la metodología más cercana a las líneas principales de esta tesis. Incorpora la idea de *proceso de ingeniería* en el sentido de Pressman [Pressman 82] y describe con bastante detalle cómo se debe definir el sistema teniendo en cuenta las dependencias entre los modelos. En contra hay que decir que no tiene herramientas de soporte específicas y que presenta problemas a la hora de razonar sobre la especificación de forma automática. Por otro lado, se aprecia en este trabajo el esfuerzo de cubrir todo el ciclo de vida de la aplicación, llegando hasta la implementación del sistema, lo cual no es frecuente.

La especificación de SMA que proporciona MAS-CommonKADS detalla la mayoría de aspectos en lenguaje natural. Esta particularidad dificulta el análisis automático de la especificación generada y supone una gran desventaja frente a semi-formalismos como UML, soportado por muchas herramientas y con la posibilidad de hacer chequeos para verificar el desarrollo (¿Existen elementos no utilizados? ¿Se ha asociado especificaciones de comportamiento a los casos de uso?). Para lograr lo mismo en MAS-CommonKADS habría que restringir el uso de lenguaje natural o bien incluir formalismos que logren una definición más precisa y menos ambigua del SMA.

En esta tesis, el problema se salva utilizando meta-modelos como mecanismo de especificación. El desarrollo de meta-modelos está soportado por herramientas que permite el procesamiento automático de los modelos generados. Los meta-modelos en algunos casos profundizan más en el detalle que MAS-CommonKADS. Tal es el caso del meta-modelo de organización, el de tareas y objetivos, o el de agentes.

1.5 BDI

Las arquitecturas BDI se inspiran en un modelo cognitivo del ser humano [Bratman 87]. Los agentes utilizan un modelo del mundo, una representación de cómo se les muestra el entorno. El agente recibe estímulos a través de sensores ubicados en el mundo. Estos estímulos modifican el modelo del mundo que tiene el agente (representado por un conjunto de creencias). Para guiar sus acciones, el agente tiene *Deseos*. Un *deseo* es un estado que el agente quiere alcanzar a través de *intenciones*. Éstas son acciones especiales que pueden abortarse debido a cambios en el modelo del mundo. Aunque la formulación inicial es de Bratman, fueron Georgeff, Rao y Kinny [Kinny y Georgeff 97] quienes formalizaron este modelo y le dieron visos de metodología, de hecho es su trabajo lo que aquí se comenta.

Para especificar el sistema de agentes, se emplean un conjunto de modelos que operan a dos niveles de abstracción: externo e interno. Primero, desde un punto de vista externo, un sistema se modela como una jerarquía de herencia de clases de agentes, de la que los agentes individuales son instancias. Las clases de agente se caracterizan por su propósito, sus responsabilidades, los servicios que ejecutan, la información acerca del mundo que necesitan y las interacciones externas. Segundo, desde un punto de vista interno, se emplean un conjunto de modelos (modelos internos) que permiten imponer una estructura sobre el estado de información y motivación de los agentes y las estructuras de control que determinan su comportamiento (creencias, objetivos y planes en este caso).

En esta metodología, la integración con el ciclo de vida de software es muy reducida. Los autores proponen una serie muy reducida de pasos para generar los modelos. Estos pasos se repiten haciendo que los modelos, que capturan los resultados del análisis, sean progresivamente elaborados, revisados y refinados. Además, el refinamiento de los modelos internos conlleva la realimentación de los modelos externos. Por ejemplo, el construir los

planes y creencias de una clase de agente clarifica qué nivel de detalle se requiere en la representación del mundo que posee el agente.

Además, sobre la arquitectura que soporte estos modelos se imponen varias restricciones: que asegure que los eventos se responden en su momento, que las creencias se mantengan consistentemente, y que la selección de planes y ejecución se desarrolle de manera que refleje ciertas nociones de racionalidad.

El modelo BDI ha sido importante para este trabajo, como aplicación del *principio de racionalidad*. Proporciona una definición, basada en elementos intuitivos, de cómo actúan los agentes. Esta definición es lo suficientemente genérica como para permitir múltiples implementaciones sobre diferentes plataformas.

Metodológicamente, la propuesta de Georgeff, Kinny y Rao es consecuente con la dificultad de generar los modelos que proponen. Como ellos admiten, existen dependencias entre los diferentes modelos que dificultan el proceso de generación de los modelos que describen el SMA. Como solución proponen un proceso iterativo e incremental (como el Proceso Racional Unificado [Jacobson, Booch y Rumbaugh 00]) con realimentaciones. Sin embargo, la forma en que tienen lugar estas realimentaciones no queda completamente definida.

En esta tesis, los principios del BDI se hallan presentes de diferentes formas. Los planes, entendidos como secuenciación de tareas, aparecen como flujos de trabajo. Los objetivos han sido introducidos explícitamente, incluyendo su refinamiento en subobjetivos, el establecimiento de dependencias entre objetivos (inicialmente árboles Y/O [Rich y Knight 90]) y dependencias con tareas. Su uso se ha extendido a otros modelos como el de organización y el de interacciones, vertebrando la ejecución de tareas e iniciación de interacciones a lo largo de toda la metodología. Por lo tanto, el grado de integración del modelo BDI en la metodología es superior al logrado en la propuesta de [Kinny y Georgeff 97]. Y todo ello dentro de un ciclo de vida de software complejo, como es el RUP. Además, se ha logrado la independencia total de implementación dotando a la metodología de un procedimiento para parametrizar con los modelos generados armazones software escritos en cualquier lenguaje.

1.6 MaSE

MaSE (Multi-agent systems Software Engineering) [DeLoach 01] se concibe como una abstracción del paradigma orientado a objetos donde los agentes son especializaciones de objetos. En lugar de simples objetos, con métodos que pueden invocarse desde otros objetos, los agentes se coordinan unos con otros vía conversaciones y actúan proactivamente para alcanzar metas individuales y del sistema.

En MaSE los agentes son simplemente una abstracción conveniente, que puede o no poseer inteligencia. En este sentido, los componentes inteligentes y no inteligentes se gestionan igualmente dentro del mismo armazón. Dado el enfoque inicial, los agentes se ven como especializaciones de objetos. De hecho, el sistema se construye sobre técnología orientada a objetos y su aplicación a la especificación y diseño de sistemas multi-agente.

El análisis en MaSE consta de tres pasos: capturar los objetivos, aplicar los casos de uso y refinar roles. El diseño consta de cuatro pasos: crear clases de agentes, construir conversaciones, ensamblar clases de agentes y diseño del sistema. La mayoría de estos pasos se ejecutan dentro de la herramienta que soporta MaSE, AgentTool [DeLoach 01]. Como productos de estas etapas, MaSE espera: diagramas de secuencia para especificar interacciones, diagramas de estados para representar procesos internos a las tareas y

modelar interacciones, descomposición del sistema (agente) en subsistemas (componentes del agente) e interconexión de los mismos (definición de la arquitectura del agente). Estos elementos son característicos del UML, de hecho su uso recuerda mucho a SDL [International Telecommunication Union 99 A.D.b]. Está por ver si las especificaciones resultantes, como las mostradas en [MultiAgent and Cooperative Robotics Lab 02], son capaces de expresar elementos característicos como razonamiento de los agentes, organización de los agentes o caracterización de su estado mental.

Además, la integración de estos diagramas en el proceso de desarrollo parece demasiado simple. Al final, la metodología podría traducirse como *tome la herramienta de soporte y rellene los diferentes apartados*. Esto supone ignorar que, como en el modelo BDI, se tienen dependencias entre los diagramas propuestos (como entre los diagramas de secuencia y las conversaciones) y que no es tan sencillo el saber qué máquinas de estados definen la ejecución de una tarea en el contexto de una interacción.

La herramienta de soporte permite generar código automáticamente a partir de la especificación. La generación de código es independiente del lenguaje de programación utilizado, ya que se realiza recorriendo las estructuras de datos generadas en la especificación y generando texto como salida. No obstante, el proceso de generación de código es mejorable, ya que el código de los componentes a generar está entremezclado con el código que lee la especificación.

En esta tesis, se persigue un proceso de desarrollo similar al seguido mediante otras herramientas, como Rational Rose, TogetherJ o Paradigm+. En estas herramientas, el usuario tiene libertad para elaborar diagramas incompletos o generar sus propias vistas del sistema, tomando elementos de diferentes modelos e incorporándolos a un nuevo diagrama. También se intenta separar de UML en el sentido de no repetir las mismas soluciones. La forma en que se tratan las interacciones de esta tesis es un buen ejemplo de ello. Las interacciones buscan generalizar diferentes alternativas existentes, como los MSC [International Telecommunication Union 99 A.D.a], diagramas de secuencia o colaboración [Jacobson, Booch y Rumbaugh 00], y diagramas de protocolo (AUML [Bauer, Müller y Odell 01]). Aparte de la generalización, las interacciones se integran completamente dentro de otros modelos como un elemento más. De hecho, la iniciación de interacciones se representa como el producto de ejecutar una tarea.

Otro aspecto tenido en cuenta en esta tesis es el proceso de generación de los modelos, esto es, qué actividades están involucradas en su producción. El resultado es un conjunto estructurado de actividades detalladas que guiarán a futuros usuarios de la metodología en su utilización. Las actividades se enmarcan en diferentes flujos de trabajo para cada modelo, con lo que se facilita una futura integración de la metodología en entornos automatizados de gestión de proyectos software como el Entorno de Creación de Servicios [Eurescom P815 99] de Telefónica I+D.

1.7 ZEUS

ZEUS consta de una herramienta y una metodología [Collis y Ndumu 99], de forma similar a AgentTool y MaSE. Desde su aparición, ZEUS se ha convertido en referencia de cómo debe ser una herramienta para el desarrollo de SMA. Sobre todo, por la forma en que combinan los distintos resultados de investigación en agentes (planificación, ontologías, asignación de responsabilidades, relaciones sociales entre agentes) en un sistema completamente funcional. De hecho, la aplicación genera incluso programas para arrancar el sistema especificado e incluye herramientas de monitorización como el *Visor de Sociedad*

que muestra los agentes existentes y sus relaciones, la *Herramienta de Control* para ver o modificar remotamente el estado de los agentes y los *Generadores de Informes* para obtener estadísticas de funcionamiento e informes de actuación de la sociedad de agentes.

La metodología ZEUS propone un desarrollo en cuatro etapas [Collis y Ndumu 99]: el análisis del dominio, el diseño de los agentes, la realización de los agentes y el soporte en tiempo de ejecución. Las etapas soportadas por la herramienta son la de *realización de los agentes* y la de *soporte en tiempo de ejecución*. Las etapas anteriores se basan en el uso de roles para analizar el dominio y en su asignación a agentes.

Ante la similitud de enfoques, se impone una breve comparativa entre ZEUS y MaSE. Conceptualmente, ZEUS es superior a MaSE. Si bien la primera está más orientada a la aplicación de tecnología de agentes (planificación, definición de ontologías, secuenciación de tareas), la segunda se orienta más a las prácticas de ingeniería convencional. Metodológicamente, ZEUS es más pobre que MaSE. El modelado de roles, propuesto en [Collis y Ndumu 99], no profundiza en la aplicación de la herramienta dentro del proceso de desarrollo. El ámbito de la metodología se limita a estudiar cómo agrupar la funcionalidad del sistema dentro de cada rol, dejando aparte consideraciones acerca de cómo organizar las tareas, definir las ontologías y las dependencias sociales, aspectos que son modelables dentro de la herramienta.

La cuestión que se plantea al estudiar ZEUS y MaSE es si hay que tener una buena herramienta o una buena metodología. La postura en esta tesis es que las herramientas de soporte no tienen que condicionar la metodología y que de hecho han de ser independientes. La independencia en este trabajo se consigue usando meta-modelos como elemento de construcción. Ello facilita el portar la metodología a diferentes herramientas, ya que cualquier herramienta que soporte meta-modelado podría servir como herramienta soporte de desarrollo.

Otra ventaja de los meta-modelos es que facilitan la evolución de la metodología. Tanto ZEUS como MaSE tendrían que cambiar en gran medida sus herramientas asociadas, para, por ejemplo, incluir el meta-modelo de organización de este trabajo. Sin embargo, el paso inverso, incluir elementos de MaSE o ZEUS en meta-modelos, no supone un gran esfuerzo.

Este trabajo y ZEUS tienen objetivos similares. ZEUS demuestra la aplicación de diferentes técnicas en la producción de SMA, mientras que en este trabajo se busca integrar diferentes técnicas para la especificación de SMA. Estas técnicas se hacen explícitas en este trabajo, mientras que en ZEUS eran implícitas en la construcción de la aplicación. En cuanto a la integración de la especificación en el ciclo de vida de software, ya se ha comentado en la sección anterior las aportaciones en este sentido.

1.8 GAIA

GAIA [Wooldridge, Jennings y Kinny 00] [Zambonelly, Wooldridge M. y Jennings N.R. 00] es una metodología para el diseño de sistemas basados en agentes cuyo objetivo es obtener un sistema que maximice alguna medida de calidad global (no se llega a detallar cual). GAIA pretende ayudar al analista a ir sistemáticamente desde unos requisitos iniciales a un diseño que, según los autores, esté lo suficientemente detallado como para ser implementado directamente.

En GAIA se entiende que el objetivo del análisis es conseguir comprender el sistema y su estructura sin referenciar ningún aspecto de implementación. Esto se consigue a través de la idea de *organización*. Una organización en GAIA es una colección de roles, los cuales mantienen ciertas relaciones con otros y toman parte en patrones institucionalizados de

interacción con otros roles. Los roles agrupan cuatro aspectos: responsabilidades del agente, los recursos que se le permite utilizar, las tareas asociadas e interacciones.

GAIA propone trabajar inicialmente con un análisis a alto nivel. En este análisis se usan dos modelos, *el modelo de roles* para identificar los roles clave en el sistema junto con sus propiedades definitorias y el *modelo de interacciones* que define las interacciones mediante una referencia a un modelo institucionalizado de intercambio de mensajes, como el FIPA-Request [FIPA 01]. Tras esta etapa, se entraría en lo que GAIA considera diseño a alto nivel. El objetivo de este diseño es generar tres modelos: el *modelo de agentes* que define los tipos de agente que existen, cuántas instancias de cada tipo y qué papeles juega cada agente, el m*odelo de servicios* que identifica los *servicios* (funciones del agente) asociados a cada rol, y un *Modelo de conocidos*, que define los enlaces de comunicaciones que existen entre los agentes.

A partir de aquí se aplicarían técnicas clásicas de diseño orientado a objetos. Sin embargo, esto queda fuera del ámbito de GAIA. Esta metodología sólo buscar especificar cómo una sociedad de agentes colabora para alcanzar los objetivos del sistema, y qué se requiere de cada uno para lograr esto último.

La principal crítica que se puede hacer a GAIA es que se queda a un nivel de abstracción demasiado alto. Según los autores, con ello se consigue desacoplarse de las distintas soluciones de implementación de agentes. Sin embargo, es cuestionable la utilidad de una metodología que genera especificaciones cuya implementación no se llega siquiera a considerar cuando en principio se pretendía llegar a un nivel de detalle fácilmente implementable.

Como solución genérica, en esta tesis, se proporciona un procedimiento con el que se facilita el salto a la implementación del sistema. La implementación se plantea como la parametrización de un armazón software utilizando los modelos que componen la especificación.

Otro aspecto discutible es el uso combinado de fórmulas lógicas con fichas de documentación clásicas en ingeniería del software [Pressman 82]. En GAIA parece asumirse que poniendo fórmulas lógicas se consigue una mejor comprensión del problema y una mayor exactitud. El problema de estas fórmulas, aparte de su comprensión, es su definición. ¿De qué sirve establecer las precondiciones de una tarea con un conjunto de predicados cuya semántica y existencia no está definida en ningún sitio?

Para salvar este problema, en este trabajo, aparte de dejar libertad al que quiera utilizar fórmulas lógicas, se ofrecen representaciones donde los términos que componen la representación son los propios elementos del meta-modelo. Por ejemplo, se puede hablar de las cualidades de un agente con el que se quiere interactuar utilizando un modelo de agente que represente los roles que debe tener, los objetivos asociados y estado mental en el que se supone que debe estar el agente colaborador.

Como en MaSE, además, se comete el error de obviar las distintas dependencias entre los modelos propuestos, lo cual es fundamental a la hora de proponer un proceso que dé cómo salida la especificación del sistema.

En esta tesis las dependencias entre los distintos meta-modelos se revisan independientemente. La mayoría se refiere a que al introducir ciertas entidades, hay que definir aspectos adicionales en otros modelos. Por ejemplo, al crear un objetivo, siempre hay que asociar una tarea o tareas que permiten alcanzarlo. Si el objetivo se identifica dentro de un modelo de agente, entonces, se necesita que en un modelo de objetivos y tareas se indique qué tarea o tareas deben ejecutarse.

Para terminar, el modelo de organización en GAIA es superficial ya que no se tienen en cuenta las relaciones estructurales. En la extensión de GAIA comentada en [Zambonelly,

Wooldridge M. y Jennings N.R. 00] y supuestamente dedicada a cubrir este hueco, se habla más de restricciones sociales respecto a uso de roles que de la organización en sí.

En esta tesis el meta-modelo de organización se ve respecto del SMA como el equivalente a la arquitectura del sistema de un sistema convencional. Sirve para definir a alto nivel cómo se organizan los elementos del sistema para hacer posible los objetivos comunes a los agentes que participan en la organización.

1.9 MESSAGE

MESSAGE [Caire et al. 02] es la metodología más reciente de las estudiadas y por tanto trata de integrar resultados de las anteriores. Propone el análisis y diseño del SMA desde cinco puntos de vista para capturar los diferentes aspectos de un SMA: el de Organización, que captura la estructura global del sistema; el de Tareas/Objetivos, que determina qué hace el SMA y sus agentes constituyentes en términos de los objetivos que persiguen y las tareas implicadas en el proceso; el de Agente, que contiene una descripción detallada y extensa de cada agente y rol dentro del SMA; el de Dominio que actúa como repositorio de información (para entidades y relaciones) concernientes al dominio del problema; y el de Interacción, que trata las interacciones a distintos niveles de abstracción.

Estos elementos están presentes en los dos modelos fundamentales que propone MESSAGE: el modelo de análisis y el modelo de diseño. El modelo de análisis se limita a generar modelos a partir de los meta-modelos. El modelo de diseño no llegó a concretarse completamente. Se decidió que el propósito del diseño sería producir entidades computacionales que representen el SMA descrito en el análisis. Por ello, cada artefacto producido en el análisis debería transformarse en una entidad computacional o varias cuyo comportamiento fuera el que se esperaba en el análisis. Esto significa que las entidades del análisis se deberían traducir a subsistemas, interfaces, clases, signaturas de operaciones, algoritmos, objetos, diagramas de objetos y otros.

MESSAGE aporta mejoras en cuanto a conceptos de ingeniería respecto de las alternativas existentes, entre ellas el desarrollo dentro de un paradigma de ingeniería del software (el Proceso Racional Unificado), aportación de métodos para la traducción de entidades de análisis a entidades de diseño y guías para la generación de los modelos.

Sin embargo, los objetivos de MESSAGE no se completaron totalmente. La integración con el Proceso Racional Unificado no fue total, ya que las actividades definidas no se adecuaban a las necesidades reales y no se indicó cómo encajaban dentro de este proceso. Además, faltó trabajo en el estudio de las interdependencias entre los distintos modelos propuestos.

A favor de MESSAGE hay que destacar que ha sido la primera metodología en utilizar una herramienta para soporte del proceso de especificación de SMA de forma visual, como en UML [OMG 00d]. En cuanto a la implementación, MESSAGE provee guías en cuanto a posibles arquitecturas y componentes a utilizar en esta etapa. Basándose en estas guías y los modelos de análisis y diseño, se realizó manualmente la implementación, lo cual hizo que se detectaran incorrecciones en las definiciones iniciales de los modelos. Esta experiencia es la base de la crítica realizada con anterioridad a ZEUS.

En esta tesis, se plantea la evolución a lo largo del ciclo de vida del software de los modelos generados. El paso de una etapa a otra está marcado por el nivel de detalle alcanzado en cada modelo. Así, las interacciones inicialmente pueden detallarse con diagramas de colaboración para luego concretarse en el diseño con otros tipos de diagramas

que alcancen más detalle en aspectos como la motivación de la interacción o actos del habla [Singh 91] empleados durante el proceso. El paso a implementación, como se ha comentado antes, se ha generalizado en forma de proceso de parametrización de armazones software. Esta forma de implementación es una evolución del trabajo de MESSAGE, donde se proponían arquitecturas y componentes adecuados para esta tarea.

Los meta-modelos en general se han modificado para integrar resultados de investigación tales como planificación de tareas, el modelo BDI, la estructuración de elementos de la comunidad o el uso de tareas. De forma similar a MAS-CommonKADS se ha estudiado el dominio de aplicación de cada meta-modelo para que se puedan aplicar los resultados correspondientes.

También se ha incluido un nuevo meta-modelo, el de entorno. MESSAGE no tenía en cuenta lo que rodeaba la aplicación, por lo cual la inclusión de elementos como servicios del sistema, recursos o aplicaciones que no fueran agentes, eran difíciles de tratar. En esta tesis, el meta-modelo de entorno permite incluir este tipo de elementos de forma coherente. De hecho, la percepción de los agentes se expresa en función de estos elementos. Así, se puede representar que un agente de interfaz se conecte a una aplicación existente.

1.10 Conclusiones

Partiendo de la definición de agente de Newell, se ha hecho un breve recorrido por arquitecturas de agentes, definición de SMAs con lenguajes de agentes y plataformas de desarrollo para la implementación de SMA. Durante este recorrido, se ha mostrado la necesidad de aplicar metodologías para estructurar el desarrollo de SMAs. Saber diseñar agentes y encajarlos en un sistema no es suficiente. Un sistema debe satisfacer las necesidades del cliente que lo solicitó. Hay que tomar decisiones fundamentales como elegir las cualidades que se quieren presentes en los agentes (utilizando las diferentes arquitecturas), decidir qué entidades del sistema van a ser agentes o no (utilizando la definición de Newell) y organizarlo todo según un armazón software o utilizando una plataforma de desarrollo. Debido a que este proceso no es trivial, se necesitan metodologías que guíen al ingeniero a lo largo del camino.

Se ha dedicado la mayoría de las secciones a estudiar qué metodologías son más relevantes en relación con la línea de trabajo seguida en esta tesis y qué mejoras se aportan respecto de lo que hay hecho. Es importante reseñar que el trabajo de esta tesis parte del trabajo realizado en MESSAGE. Esta es la primera metodología en utilizar herramientas de meta-modelado para reflejar los resultados del análisis. Este hecho permite trabajar directamente con los conceptos que intervienen en el desarrollo de SMA y probarlos *in situ* en casos reales. Otra ventaja de la utilización de este tipo de herramientas es que permiten asegurar que se están siguiendo los modelos indicados en la metodología en la forma prevista.

La experiencia de MESSAGE, ZEUS y MaSE ha demostrado la importancia de trabajar con herramientas que soporten el proceso de desarrollo. Metodologías como GAIA no pueden afirmar su efectividad a la hora de resolver problemas porque no hay nada que asegure que los productos de la metodología se adecúan a lo que está especificado en la misma. Por ello, muchas de las metodologías existentes precisan de sensibles mejoras antes de poder ser aplicadas a problemas reales.

Otro elemento importante a la hora de desarrollar una metodología es la idea de proceso. La generación de un SMA sigue una serie de pasos predefinidos y, generalmente, incrementales. Metodologías que siguen esta idea son MAS-CommonKADS, BDI, GAIA y, especialmente, MESSAGE. MESSAGE es más avanzada que GAIA en el sentido de que no se queda en el nivel conceptual cuando desarrolla el SMA, como hace GAIA. MESSAGE plantea transformaciones de entidades conceptuales en entidades computacionales para completar la generación del SMA.

Relacionada con la idea de proceso está la posibilidad de realizar el desarrollo de forma incremental. Las especificaciones generadas dentro del ciclo de vida del software no se desechan, sino que se reutilizan para ir ahondando en el detalle según aumenta la comprensión del sistema. De las metodologías vistas, las que mejor soportan este tipo de desarrollo son MESSAGE y MaSE. En MESSAGE, al utilizar meta-modelos, la adición de nuevos elementos y relaciones resulta algo natural. En el caso de MaSE, los diagramas utilizados para describir el sistema han sido diseñados para soportar este tipo de uso. En ambos, al disponer de una herramienta de soporte, se puede aprovechar de forma muy sencilla las especificaciones generadas en otras actividades. En las otras metodologías, esto no es tan sencillo porque no existen herramientas de soporte o bien la especificación se hace utilizando documentos de difícil reaprovechamiento, como los propuestos por GAIA o MAS-COMMONKADS.

Subyacente a todas las metodologías revisadas existe una teoría de cómo es el sistema multi-agente. Existen modelos puramente formales, como el que subyace a ConGOLOG y otros que utilizan modelos operacionales, como Agent0. De cualquier forma, en base a esta teoría es posible razonar acerca del comportamiento del sistema. Este tipo de razonamientos es más complicado con los modelos semi-formales, como los generados con UML. Sin embargo, los modelos semi-formales posibilitan que el ingeniero pueda jugar con el sistema y trabajar con problemas más complejos. Para compensar esta semi-formalidad, existen procedimientos de verificación de la especificación. Un ingeniero establece, dentro de los modelos semi-formales planteados, restricciones acerca de qué está permitido y qué no. La expresión de tales restricciones se realiza mediante lenguajes específicos, como el lenguaje *Object Constraint Language* (OCL) de UML.

Después de ver el trabajo de ZEUS y MaSE se puede argumentar que ya existen metodologías junto con herramientas capaces de desarrollar SMAs. Sin embargo, esto no es del todo cierto, lo que han demostrado es que se pueden desarrollar SMAs de una forma muy concreta. Estas metodologías están fuertemente restringidas por la herramienta en la que se basan ya que la herramienta se encarga al fin y al cabo de obtener parámetros de configuración para un armazón de SMA que ya existe. Además, están orientadas al desarrollo rápido con decisiones de diseño fijas más que a un desarrollo personalizado. AgentTool, por ejemplo, tiene fijas restricciones de diseño fundamentales, como el asimilar cada objetivo con requisitos y transformar estos requisitos en roles, o establecer protocolos de comunicación entre tareas internas del agente.

La aportación de esta tesis es una metodología que extiende MESSAGE y que ofrece mejoras frente a metodologías existentes. Como ya se ha mostrado en las secciones anteriores, las propuestas existentes dejan abiertos aspectos referentes a la forma en que se modela el SMA y a la integración del desarrollo del SMA con las prácticas habituales de ingeniería.

En esta tesis, la línea que se sigue es la de una ingeniería del software orientada a agentes según las ideas de [Pressman 82]. Pressman concibe tres elementos en una ingeniería del software: herramientas, métodos y procedimientos. En este trabajo, se

proporcionan meta-modelos para definir el SMA, herramientas de soporte para generarlos y hacerlos progresar en las distintas etapas del ciclo de vida (análisis, diseño e implementación) y un conjunto de actividades que estructuran el desarrollo del SMA.

Capítulo 2. META-MODELOS DEL SISTEMA MULTI AGENTE

El método de desarrollo de SMA propuesto en esta tesis concibe el SMA como la representación computacional de un conjunto de modelos. Cada uno de estos modelos muestra una visión parcial del SMA: los agentes que lo componen, las interacciones que existen entre ellos, cómo se organizan para proporcionar la funcionalidad del sistema, qué información es relevante en el dominio y cómo es el entorno en el que se ubica el sistema a desarrollar. Para facilitar la representación computacional de los modelos, en este trabajo se asume que la implementación final se realiza parametrizando un armazón software.

Para especificar cómo tienen que ser estos modelos se definen meta-modelos. Un meta-modelo es una representación de los tipos de entidades que pueden existir en un modelo, sus relaciones y restricciones de aplicación. Los meta-modelos que se describen en este capítulo son una evolución del trabajo realizado en MESSAGE [Caire et al. 02]. En MESSAGE se propusieron meta-modelos para representar agentes, organizaciones, el dominio, interacciones, tareas y objetivos. Como se señaló en la sección anterior, el trabajo de MESSAGE es mejorable en cuatro aspectos: la integración de los meta-modelos con las prácticas de ingeniería, un mayor nivel de detalle en los meta-modelos, una mayor cohesión entre los meta-modelos y representación del entorno del sistema.

Para lograr estas mejoras, ha sido necesario rediseñar todos los meta-modelos para resaltar aspectos comunes entre los diferentes aspectos modelados, incluir el nivel de detalle requerido en la etapa de diseño del sistema y probar los resultados en casos de estudio para determinar cómo debía ser la integración con un proceso de desarrollo. Como ya se mencionó en el prólogo, se ha eliminado el meta-modelo del dominio en favor de un meta-modelo de entorno. El resultado son cinco meta-modelos que giran alrededor de dos entidades la *organización* y el *agente* (Ilustración 3). El motivo de este cambio es que, según

la experiencia de MESSAGE, la información proporcionada por la vista del meta-modelo del dominio podía ser expresada dentro de otros sin perjuicio aparente. Sin embargo, no era posible expresar cuál era la naturaleza del entorno ni cómo se interactuaba con él.

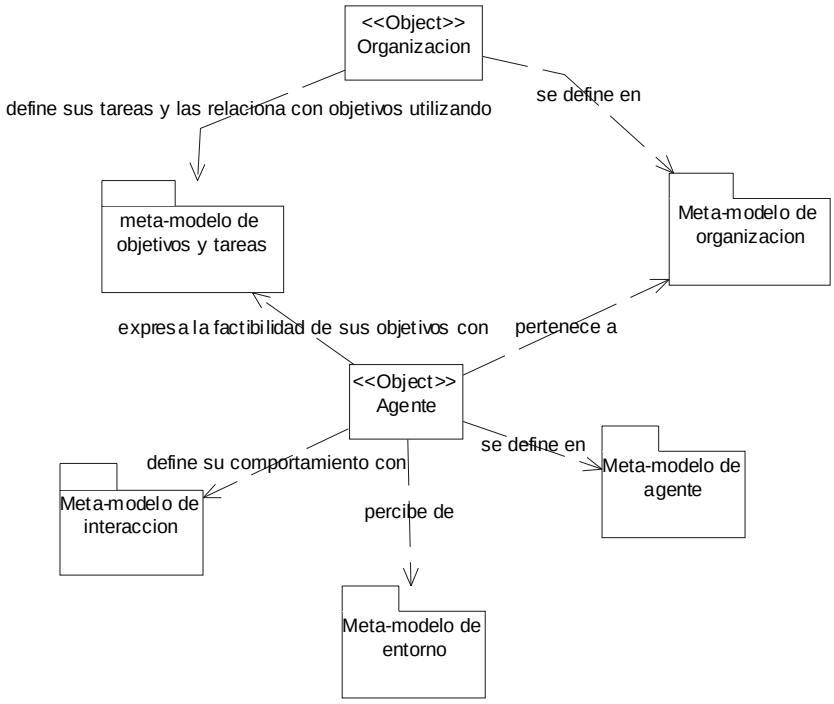


Ilustración 3. Relaciones entre los diferentes meta-modelos y las dos entidades principales, la organización y el agente.

El meta-modelo de agente describe agentes particulares y los estados mentales en que se encontrarán a lo largo de su vida. El meta-modelo de interacción se ocupa de detallar cómo se coordinan y comunican los agentes. El meta-modelo de tareas y objetivos se usa para asociar el estado mental del agente con las tareas que ejecuta. El meta-modelo de organización define cómo se agrupan los agentes, la funcionalidad del sistema y qué restricciones hay que imponer sobre el comportamiento de los agentes. Por último, el modelo del entorno define qué existe alrededor del nuevo sistema.

Para comprender los meta-modelos, es necesario entender cómo se aplican en este trabajo. Por ello, se ha incluido una sección donde se explican los fundamentos del meta-modelado. Tras esta sección, se procede a explicar con detalle cada uno de los meta-modelos propuestos, en qué se basan, cómo se utilizan y cómo asegurar la consistencia entre los distintos meta-modelos.

2.1 Meta-modelado

Un *meta-modelo* define las primitivas y las propiedades sintácticas y semánticas de un *modelo*. A diferencia de otros enfoques más formales, como Z [Spivey 92], los *meta-modelos* están orientados a la generación de representaciones visuales de aspectos concretos del sistema de forma incremental y flexible. Los *modelos* crecen incorporando más detalle gracias a que no es necesario que se instancien absolutamente todos los elementos del *meta-modelo* para tener un modelo. Como ha demostrado UML [OMG 00d], construido también con *meta-modelos*, este tipo de notación facilita enormemente el desarrollo de sistemas. Otra ventaja de utilizar *meta-modelos* es que las especificaciones generadas de SMA son lo suficientemente estructuradas para ser procesadas de forma automática. Así, se puede plantear la verificación automática de la construcción de los modelos para ver si cumplen restricciones identificadas por el desarrollador, generar documentación del sistema en diferentes formatos de diferentes partes de los modelos, e incluso establecer la generación automática de código desde la información recogida en los *modelos*.

El meta-modelado se comprende a partir de una estructura de cuatro niveles (Ilustración 4). En esta estructura, el nivel M1 establece las *primitivas de meta-modelado*, esto es, el lenguaje de meta-modelado. En el nivel M2, se definen los *meta-modelos* en sí con las primitivas de M1. En M3 se aplican los meta-modelos para generar instancias, que se denominan *modelos*. Por último, la instanciación de los modelos, lleva al nivel donde se tiene la información que se manejará en el sistema.

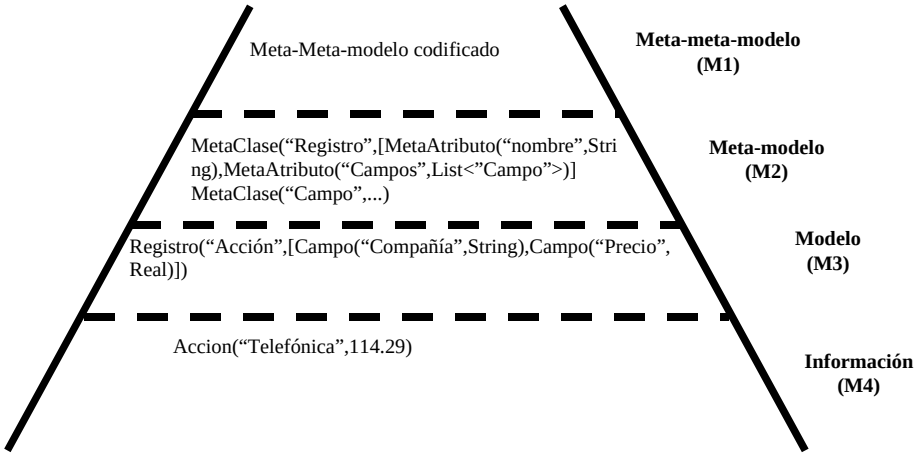


Ilustración 4. Estructura de cuatro niveles utilizada en el meta-modelado [OMG 00b]

Los lenguajes de meta-modelado (nivel M1) considerados son dos: *Meta-Object Facilities* (MOF) [OMG 00b] y GOPRR (*Graph, Object, Property, Relationship, and Role*) [Lyytinen y Rossi 99] . Ambos son similares y de hecho se pueden hacer traducciones de uno a otro. En esta tesis la definición e implementación de los meta-modelos se hace con GOPRR. El motivo es que actualmente no hay herramientas que soporten meta-modelado con MOF, pero sí con GOPRR.

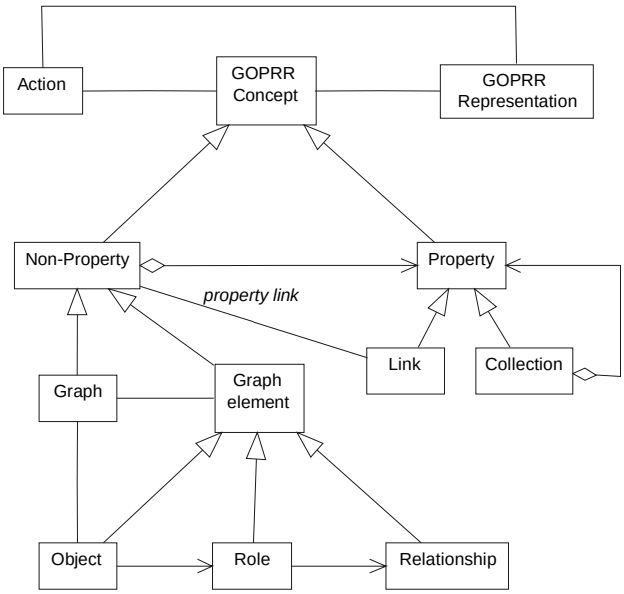


Ilustración 5. Primitivas de GOPRR

Las primitivas de GOPRR son reducidas (Ilustración 5). Estas primitivas permiten definir la documentación del sistema, que se concibe como grafos (entidad *Graph*), en términos de asociaciones (*Relationship)* cuyos extremos se definen con roles (*Role*) y que conectan entidades (*Object*). Todos estos elementos contienen propiedades (*Property*) que pueden ser otros objetos, grafos, roles o relaciones. Existen más elementos en GOPRR como las restricciones en gráficos (número de elementos, presencia o ausencia de elementos) o la cardinalidad de las relaciones. Con estos elementos es sencillo definir una gramática para la construcción de UML. De hecho, la herramienta que implementa este lenguaje (METAEDIT+ [Lyytinen y Rossi 99]) incluye entre sus demostraciones la especificación GOPRR de UML.

2.1.1 Validación de los modelos

Cuando se describe un meta-modelo, también es necesario indicar qué construcciones tienen sentido según el diseñador del meta-modelo. Un modelo puede ser correcto de acuerdo con el meta-modelo pero no corresponder a nada ejecutable. Haciendo un símil con los lenguajes de programación, el que un programa satisfaga la gramática del lenguaje no significa que sea correcto respecto de la semántica asociada al lenguaje.

La semántica asociada a los meta-modelos de este trabajo se expresa en lenguaje natural dentro de cada meta-modelo. No pretende ser una especificación exhaustiva, ya que las construcciones *correctas* aquí dependen de la implementación que se vaya a realizar. Así, la semántica inicial dada en la metodología se completa con la semántica asociada a las plataformas del desarrollo. Esta situación es análoga a la del programador en un proyecto de ingeniería del software, donde es el programador quien dota de un significado preciso a las especificaciones de diseño utilizando lenguajes de programación.

Existen medios para restringir la libertad en la generación de modelos a partir de meta-modelos. En MOF, se utiliza un lenguaje especial para definir cuáles son las construcciones *correctas,* el OCL (*Object Constraint Language*) [OMG 00d]. Sin embargo, su aplicación está bastante restringida. Aunque existen algunas librerías experimentales para la creación y verificación de expresiones OCL, su desarrollo aún no está completo[2]. El caso de GOPRR es más complicado ya que no existe ningún lenguaje específico para la expresión de restricciones. Si bien la herramienta METAEDIT permite detallar algunas restricciones (básicamente, restringir la aparición de objetos como extremos de relaciones), estas no son suficientes para desarrollos metodológicos.

Debido a los problemas que suponía la aplicación de OCL y a las carencias de GOPRR, se ha optado en este trabajo por realizar la comprobación de la validez de las instancias del meta-modelo utilizando herramientas externas, en este caso un conjunto de reglas implementadas en PROLOG. La aplicación de estas reglas ha sido únicamente en el ámbito de generación automática de código descrito en el capítulo cuarto.

2.1.2 Aplicación de GOPRR a la metodología

A lo largo de la tesis, la definición de los meta-modelos se hace utilizando notación UML siguiendo las restricciones indicadas en GOPRR (Ilustración 5). El tipo de entidad (grafo, objeto, role, relación o propiedad) se indica con estereotipos [OMG 00d], tal y como se hace en el ejemplo de la Ilustración 6. En este ejemplo aparecen relaciones de herencia y asociación cuyo uso requiere una explicación.

En GOPRR existe la primitiva de generalización, que permite extender tanto *object*, como *relationship* o *role*. Sin embargo, esta primitiva no se extiende al nivel M3, esto es, si se quiere utilizar la relación de herencia en el nivel M3, hay que definirla en el nivel M2, tal y como plantea el ejemplo. Respecto de la relación de asociación, no existe como tal en GOPRR. Lo que aparece como una asociación en estos diagramas, se corresponden con parámetros requeridos por GOPRR. Por ejemplo, un *property* a veces hace referencia a una entidad del meta-modelo en desarrollo, un *relationship* se debe asociar con uno o más objetos de tipo *role* y estos, a su vez, con elementos de tipo *object*. Para no tener que distinguir entre todas estas situaciones, se admitirá la existencia de la asociación como primitiva del lenguaje de meta-modelado.

---

[2] De hecho, el único entorno de diseño con OCL que se ha encontrado es ArgoUML (http://argouml.tigris.org), donde se pueden incluir restricciones OCL a diagramas UML.

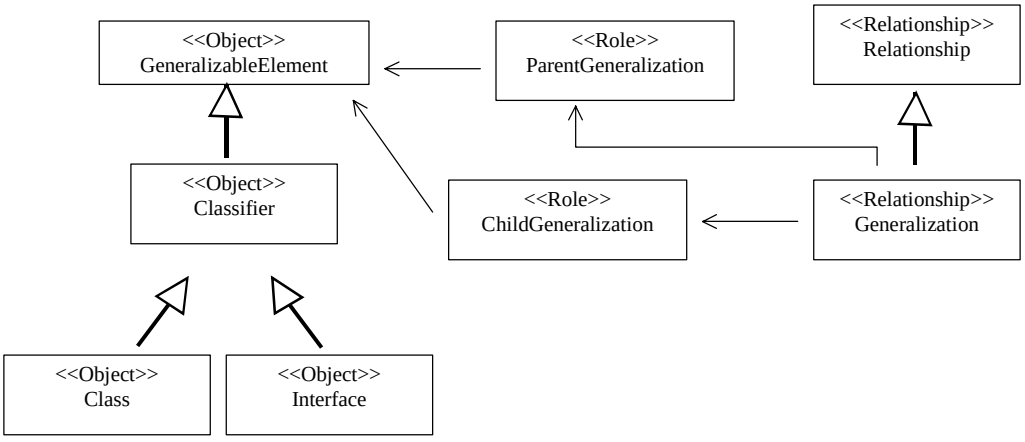


Ilustración 6. Representación en UML de un meta-modelo para especificar las relación de herencia de clases e interfaces (nivel M2). Extráído de [OMG 00d]

Este ejemplo hace referencia a la definición de la herencia entre clases. Partiendo de una relación genérica (*relationship*), se define la relación de herencia (*generalization*) como una especialización. Para definir una relación, hay que establecer cómo serán los extremos de la relación, esto es, su dominio. Los extremos, según GOPRR, deben de ser del tipo *role*. Por ello, *generalization* se asocia con *childgeneralization* y *parentgeneralization*. Un *role* ha de ser desempeñado por uno o varios *object*. Así, cada role se asocia con un único *object*, generalizable *element*.

El que una propiedad haga referencia a un objeto de la especificación se expresa con una asociación simple. En la Ilustración 7 se muestra que un *classifier*, generalización de *class* según la Ilustración 6, tiene un atributo *feature*. Según la especificación de UML [OMG 00d], este atributo denota una propiedad, como una operación o atributo que está encapsulado en un *classifier*.

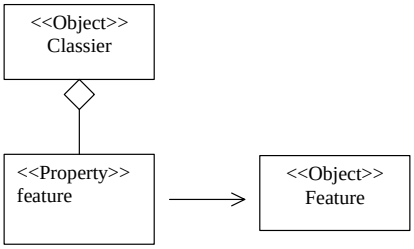


Ilustración 7. Asociación entre un *classifier* y uno de sus atributos. Extráído de [OMG 00d]

Los ejemplos vistos (Ilustración 7 e Ilustración 6) denotan meta-modelos que definen modelos como los que se pueden visualizar con Rational Rose o TogetherJ. Como se puede intuir, la lectura de meta-modelos requiere cierto entrenamiento. Para facilitar esta labor, se han aplicado dos simplificaciones. Una ya se ha visto, se trata de la utilización de asociaciones UML entre entidades. La otra se refiere al uso de los *roles*. Como indicar en cada caso la existencia de *roles* hace ilegibles los diagramas, se ha simplificado la notación GOPRR inicial omitiendo instancias de *Roles* en asociaciones n-arias con extremos asociados a una única entidad del meta-modelo (Ilustración 8). En los casos donde sea

necesario indicarlo y los extremos de la relación sean de un único tipo, pueden asociarse etiquetas.

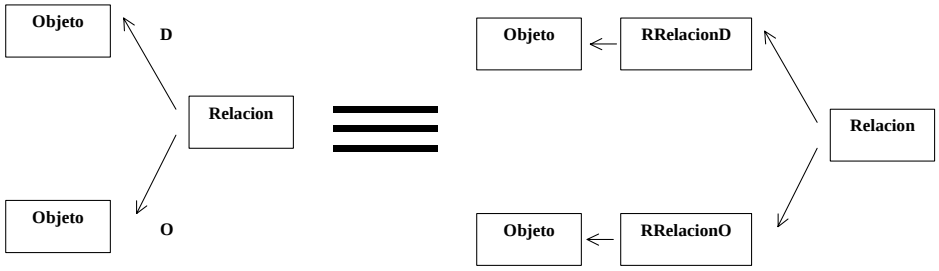


Ilustración 8. Omisión del realizaciones de *Role* en relaciones binarias. El origen y el destino de las relaciones aparece indicado con etiquetas en la representación de la izquierda.

2.1.3 Nomenclatura

Durante el desarrollo de la metodología se comprobó que el número de instancias de *relationship* era mayor que las de otros tipos básicos de GOPRR. Además, a diferencia de instancias de otros tipos GOPRR que aparecían indistintamente en los diferentes meta-modelos, el uso de relaciones se enmarcaba siempre en meta-modelos muy concretos. Por ello, se decidió que la nomenclatura de relaciones debía seguir unas reglas nemotécnicas que ayudasen a identificar de qué meta-modelo o parte específica del meta-modelo habían surgido.

El nombre asociado a las relaciones obedece a unas reglas sencillas mostradas en la Ilustración 9. Se trata de hacer que el nombre de la relación sea precedido por un conjunto de letras que denote su procedencia, como el flujo de trabajo (*WF*), meta-modelo de agente (A), interacción (I), unidad de interacción (UI), modelos de tareas y objetivos (GT), relaciones sociales (AGO), organización (O) o el entorno (E).

IdentificadorRelacion::= RelacionFlujoTrabajo | RelacionAgente |
RelacionInteraccion | RelacionUnidadInteraccion |
RelacionMetaTarea| RelacionSocial |
RelacionOrganizacion | RelacionEntorno
RelacionFlujoTrabajo::= **WF** Identificador
RelacionAgente ::= **A** Identificador
RelacionInteraccion::= **I** Identificador
RelacionUnidadInteraccion ::= **UI** Identificador
RelacionMetaTarea::= **GT** Relacion
RelacionSocial ::= **AGO** Relacion
RelacionOrganizacion ::= **O** Relacion
RelacionEntorno ::= **E** Relacion

Ilustración 9. Expresión BNF para la los nombres de las relaciones

Por cada asociación entre una entidad *relationship* y un *object* existe una instancia de la primitiva GOPRR *Role*, nombrándose ésta con el mismo nombre que la relación pero antecedido por la letra **R** y terminado en una letra ***D*** u **O**, para indicar el sentido de la relación (D de Destino, O de Origen). En caso de tratarse de relaciones n-arias, el nombre se rescribe haciendo que después del nombre de la relación aparezca un identificador significativo del uso dado al extremo concreto, como *RUIIniciaEjecutarD*, que indica que se trata de un *role* que actúa como extremo de la meta-relación *UIInicia* y que el elemento que se asocia en el extremo se debe poder ejecutar (e.g. una tarea).

IdentificadorRolAsociacionBinaria::= **R**NombreRelacion(**O**|**D**)
IdentificadorRolAsociacionNAria::= **R**NombreRelacionIdentificador(**O**|**D**)

Ilustración 10. Expresión BNF para los nombres de los roles.

2.1.4 Entidades básicas

La metodología proporciona una jerarquía de conceptos básicos para el desarrollo de un SMA así como una notación para representar estos conceptos. La jerarquía comienza con la *Entidad MAS GRASIA* y la *Relación MAS GRASIA*, que reciben este nombre por el grupo de investigación en que han sido desarrolladas (GRupo de Agentes Software del departamento de sistemas Informáticos y prograMación o GRASIA). El motivo de comenzar con estos dos elementos es el poder incluir información acerca del sistema reutilizando términos del meta-modelo. La Ilustración 11 muestra parte de los elementos básicos que se utilizan en la metodología.

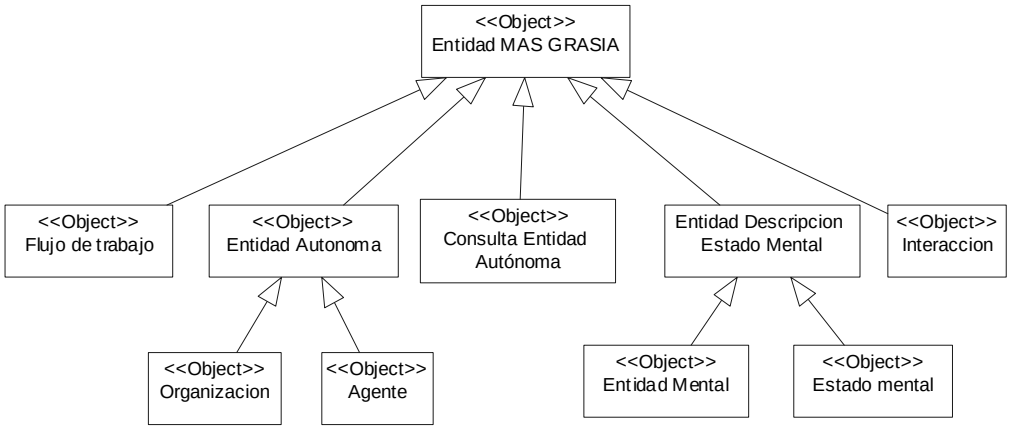


Ilustración 11. Entidades básicas de la metodología

Los meta-modelos que se van a presentar se fundamentan también en la existencia de relaciones contextualizadas en cuatro dominios: agentes aislados, organizaciones de agentes, interacciones entre agentes y el entorno de los agentes. Estas relaciones asocian especializaciones de la *Entidad MAS GRASIA*. Se trata de relaciones dirigidas que parten

de una entidad para ir a un conjunto finito de entidades destino. La Ilustración 12 muestra algunas de estas relaciones.

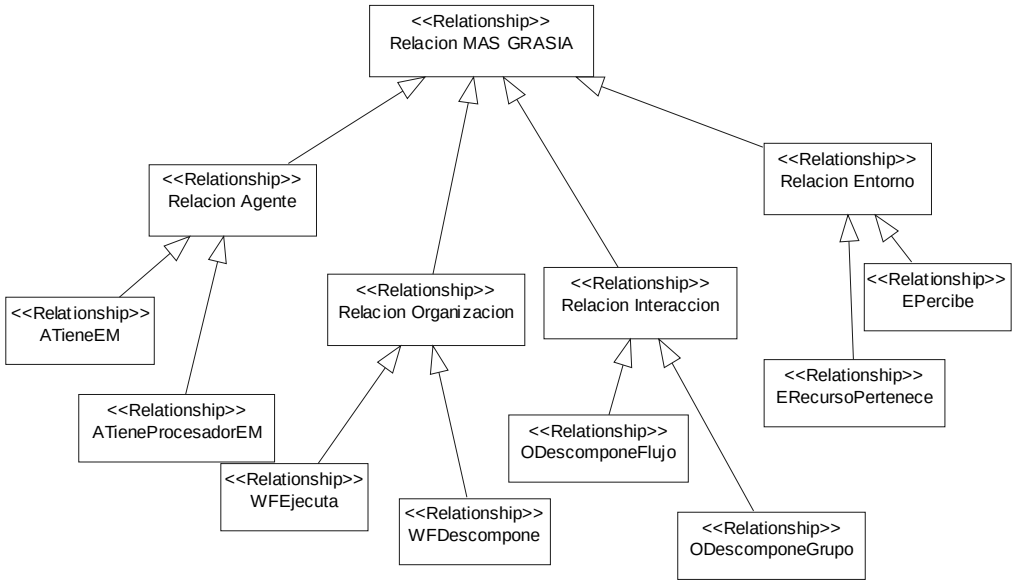
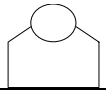
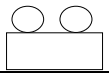
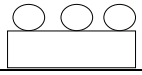
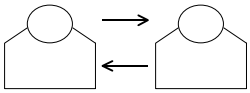
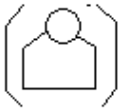
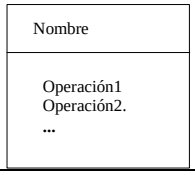
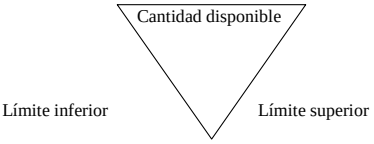


Ilustración 12. Relaciones básicas de la metodología

2.1.5 Notación

Los diagramas correspondientes al nivel del modelo (M2) (ver Ilustración 4) que se presentan a lo largo de este trabajo utilizan una notación desarrollada *ad hoc* con la herramienta METAEDIT+. Para entender los diagramas se debe revisar antes la notación de la Tabla 1.

	**Objetivo**. Se etiqueta con el nombre del objetivo
	**Rol**. Se etiqueta con el nombre del rol.
	**Procesador de estado mental**. Se etiqueta con el nombre del procesador.
	**Gestor de estado mental**. Se etiqueta con el nombre del gestor.
	**Agente**. Se etiqueta con el nombre del agente.

	**Grupo.** Se etiqueta con el nombre del grupo.
	**Organización**. Se etiqueta con el nombre de la organización.
	**Flujo de trabajo.** Se etiqueta con el nombre del flujo.
	Interacción. Se etiqueta con el nombre de la interacción y su naturaleza, como coordinación, planificación o negociación.
	**Consulta de entidades autónomas**. Se etiqueta con nombres concretos de agentes existentes o expresiones que denotan agentes existentes.
	**Unidad de interacción.** Se etiqueta con el nombre de la unidad y el acto del habla al que hace referencia, como *request*, *inform*, o *not-understood*.
	**Tarea.** Se etiqueta con el nombre de la tarea.
	**Aplicación.** Se etiqueta con el nombre de la aplicación y las operaciones soportadas.
	**Recurso.** Se etiqueta con el nombre del recurso, la cantidad disponible del mismo, el límite inferior y superior admisibles. Por debajo o encima de estos límites, el recurso se deshabilita.

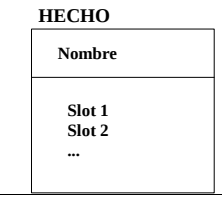
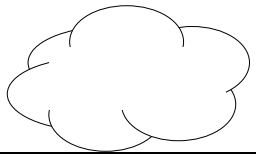
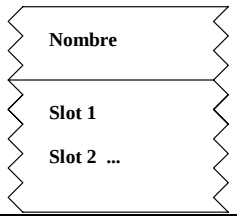
	**Hecho.** Se etiqueta con el nombre del hecho y los nombres de los slots identificados.
	**Creencia.** Se etiqueta con el nombre de la evidencia e información acerca de qué es lo que se está aceptando como cierto.
	**Evento.** Se etiqueta con el nombre del evento y los nombres de los slots identificados.

Tabla 1. Notación empleada en la representación de los modelos

2.2 Meta-modelo de Agente

El meta-modelo de agente se usa para describir agentes particulares excluyendo las interacciones con otros agentes. Este meta-modelo se centra en la funcionalidad del agente y en el diseño de su control. En este sentido, proporciona información acerca de los siguientes aspectos:

- **Responsabilidades**. Se trata de las tareas que sabe ejecutar y de los objetivos que se compromete a alcanzar. Generalmente se alude al término *rol* para agrupar la funcionalidad y las propiedades que aparecen con frecuencia en el diseño. Aunque es similar al concepto *interfaz* de la POO, el rol se diferencia principalmente en que se le puede asociar estado. En este caso, es de interés el conjunto de estados mentales asociados a un rol que participa en una interacción.

- **Comportamiento**. Existen diversas formas de expresar el comportamiento del agente. En las metodologías existentes, el comportamiento se entiende como un conjunto de llamadas a procedimiento (UML), paso de mensajes entre agentes (MaSE, MESSAGE) o transiciones en máquinas de estado (MaSE, ZEUS, UML). La tónica general es definir el comportamiento de los agentes en las interacciones, pero no es el único modo. En este trabajo se habla del control del agente, esto es, mediante qué mecanismos se va a asegurar la ejecución de tareas dentro de los parámetros acordados. Este control toma como entrada un conjunto de datos que se denominará *estado mental* [Shoham 93]. Además, se considerará el *estado mental*

como algo dinámico que evoluciona con el tiempo. Esta idea es necesaria para poder incluir el aprendizaje entre las capacidades del agente.

La justificación de dividir la definición del agente en responsabilidades y comportamiento proviene por un lado del estudio de puntos en común de las diferentes metodologías y desarrollos de armazones de SMAs y por otro de la experiencia en el desarrollo de SMAs en los proyectos Eurescom P815 [Eurescom P815 99], MESSAGE [Caire et al. 02] y PSI3 [PSI3 01]. Según la experiencia adquirida, estos dos aspectos son fundamentales desde el punto de vista de construcción del agente, ya que, al estudiarlos, el ingeniero se centra en que, individualmente, cada agente satisface las necesidades para las que fue diseñado. Para lograrlo, puede ser necesario incorporar aspectos referentes a la inteligencia o autonomía del agente.

La *inteligencia* surge al considerar mejoras en el control de ejecución de las tareas (e.g. planificación, parametrización de las tareas, generación de nuevas tareas como composición de existentes) o en las tareas en sí mismas (e.g. las tareas de comunicación que eligen colaboradores que se adecuen a las necesidades del momento o predicción del comportamiento del usuario). Gracias a las técnicas de inteligencia artificial, el agente puede adelantarse a los cambios en el entorno o a las acciones del usuario. Como la autonomía, la inteligencia puede surgir del diseño del propio agente, a través de heurísticas que consideren situaciones concretas, incorporando algoritmos que permitan la toma de decisiones [Decker 95] o como resultado de la combinación de algoritmos simples [Brooks 91a].

La *autonomía* viene soportada por el diseño y por la inteligencia del agente. El diseñador ha considerado las diferentes situaciones en que se encontrará el agente, teniendo en cuenta también posibles cambios en el entorno (e.g. aparecen nuevos agentes, la red no está disponible, los recursos de la máquina se han agotado). Interesa que el control del agente pueda adaptarse a los diferentes cambios del entorno de tal forma que ya no importe qué pueda o no pueda pasar.

Así pues, aunque es deseable incorporar métodos que añadan inteligencia o autonomía al agente, no se trata de algo imprescindible. Lo importante es que en el diseño del agente quede espacio para considerar la incorporación de elementos que otorguen al agente este tipo de habilidades.

En el resto de la sección se estudiarán aspectos íntimamente relacionados con la forma de definir el comportamiento del agente (tipo de control, especificación de estado mental y su evolución) y sus responsabilidades (asociación de tareas, objetivos y roles al agente). Como resultado de este estudio se plantea un meta-modelo de agente que puede expresar los aspectos anteriores. Para terminar, se ofrecen ejemplos de utilización y consideraciones acerca de la consistencia de los modelos que puedan generarse.

	Estado mental	Evolución estado mental	Control	Responsabilidades
JADE	No existe.	Codificado por el usuario dentro de las clases de comportamiento. Se permiten añadir/quitar comportamientos.	Ejecución de las instancias de clases de comportamiento siguiendo una política *round-robin* no expropiativa entre aquellos comportamientos concurrentes.	Asignación de tareas directa. Las tareas se codifican dentro de las clases de comportamiento.
RETSINA	Un conjunto de objetivos a satisfacer.	Satisfacción de objetivos.	Actuación de un planificador para extraer y generar planes de tareas que satisfagan los objetivos actuales. Un secuenciador (*scheduler*) se encarga de elegir las que se ejecutan.	Asignación de tareas directa. Las tareas permitidas así como su secuenciación (planes), se codifican en tiempo de diseño
Grasshopper	A definir por el diseñador.	A definir por el diseñador.	A definir por el diseñador.	A definir por el diseñador.
ZEUS	Sigue una codificación BDI. Los objetivos se definen durante el diseño. Se codifican ontologías que sirven para expresar reglas de comportamiento y que constituyen el equivalente a las creencias.	Modificación de las entidades del estado mental definidas por el diseñador. Actuación del planificador lleva a reconducir la estrategia del agente. Generación automática de compromisos según las relaciones entre los agentes.	Reglas que asocian entidades mentales a tareas. El motor de coordinación y el planificador deciden qué hacer a continuación en función de: compromisos adquiridos, horarios de finalización de tareas, reglas codificadas, estado de las tareas actuales.	Asignación de tareas dentro del IDE. Relaciones entre agentes para determinar con quién se puede hacer contratos.
ABLE	Codificaciones simbólicas (motores de reglas) y no simbólicas (redes neuronales, mapas de conexionado).	Las específicas de cada paradigma. El motor de reglas incorpora gestión de predicados. Las redes neuronales, algoritmos de aprendizaje.	Dependiente del paradigma.	A codificar directamente.
DESIRE	El equivalente a estado mental es lo que llaman *estados de información* que es el modelo parcial de un conjunto de predicados proposicionales.	Mediante la modificación del valor de verdad de los predicados.	Definición de reglas que modifican el estado actual de las tareas y que son responsables del flujo de entrada y salida de datos de una tarea a otra.	Las tareas se asignan directamente a los agentes.

Tabla 2. Soluciones de SMAs para el modelado del comportamiento y responsabilidades

2.2.1 Comportamiento y Responsabilidades de los Agentes

Existe una gran diversidad de modelos de comportamiento (control, estado mental y evolución del estado mental) y definición de responsabilidades de agentes (asignación de tareas y declaración de habilidades). La Tabla 2 resume las características que reportan algunas de las plataformas de desarrollo de SMA más relevantes. En el capítulo anterior se mencionaban JADE [Bellifemine, Poggi y Rimassa 01], ZEUS [Nwana et al. 99] y Grasshopper [Breugst y Magedanz 98]. Para abarcar otras formas de modelar SMA, en la Tabla 2 se han añadido RETSINA [Langley, Paolucci y Sycara 01;Sycara et al. 99] por su amplio espectro de aplicación, ABLE [IBM 02] por la construcción composicional de los elementos de control del agente y DESIRE [Brazier et al. 97] por representar enfoques basados en tareas. Para las metodologías se ha dispuesto la Tabla 3 donde se recogen las vistas en el capítulo anterior..

	Estado Mental	Evolución	Control	Responsabilidades
Agent0	Representación simbólica a través de lógica temporal reificada. Las entidades mentales son: creencia, obligación y capacidad.	Hacen referencia a la adición de conocimiento (*inform*) y a su gestión, que no queda detallada. Sólo habla de eliminar compromisos (*refrain*).	Reglas (acciones *IF*) asociadas al estado mental del agente. Las reglas ejecutan acciones en función de la satisfacción de una fórmula lógica cuyos términos son entidades del estado mental.	No aparecen como tal. Las tareas que se deben ejecutar se codifican directamente en las acciones IF. La capacidad del agente se expresa con el predicado *CAN*
CONGOLOG	Representación del estado del mundo a través de predicados. Son importantes los *fluents*. que son predicados modificables mediante tareas.	Las tareas modifican ciertas entidades del estado mental. Las entidades que son modificables se denominan *fluents*.	Las tareas se ejecutan si se satisfacen ciertos axiomas de precondición.	No contemplado explícitamente. De hecho, no hay una asociación directa entre tarea y agente.
Vowel Engineering	No contemplado explícitamente. Se habla de posibles arquitecturas de agente, que van desde arq. cognitivas a simples máquinas de estados, pero no se entra en la descripción interna de las mismas.	No contemplado explícitamente.	No se menciona el control de forma aislada. Aparece asociado a la forma de concebir interacciones con Redes de Petri.	Se habla de funciones implementadas por los agentes y funcionalidad colectiva. Dentro de los protocolos de interacción se habla de peticiones (*requests*) expresadas con notación similar a KIF.
MESSAGE	Existen ontologías para representar creencias, objetivos, compromisos y tareas. Se utiliza UML para representar el estado mental.	La única referencia a la evolución del estado mental es la de que las tareas satisfacen objetivos.	Se definen asociaciones tarea – objetivos. El control consiste en satisfacer un conjunto de objetivos que mantienen relaciones entre sí (*árbol de objetivos*).	A nivel global existe la definición de un flujo de trabajo en el que aparecen tareas y agentes asociados.
GAIA	Se habla de *expresiones de viveza* que describen situaciones que el agente busca alcanzar. Consta de: Objetivos que persigue el agente, Compromisos adquiridos, y Capacidades.	No hay ninguna referencia.	Sólo se mencionan precondiciones para la ejecución de actividades. No se explica cómo definir actividades. Se habla de procesos involucrados en las interacciones, pero no se detalla nada más.	Se maneja intensamente la idea de rol. El rol es la entidad que asume un conjunto de funciones y que liga a su cumplimiento.
MAS-CommonKADS	Aparecen objetivos, servicios y habilidades de los agentes. No se explicita cómo se representan computacionalmente. El trabajo se restringe a descripciones en lenguaje natural.	No se mencionan mecanismos explícitos para gestionar los objetivos del agente. Hay menciones vagas en el modelo de experiencia.	Modelo de experiencia para describir capacidades de razonamiento.	Las tareas son procesos conectados a objetivos. Las tareas toman su entrada de datos de los objetivos y depositan en ellos los resultados. Se asocian a los agentes mediante una relación de *responsabilidad*. *Servicios* como tareas a las que el agente puede comprometerse.
BDI	Creencias, intenciones, objetivos y planes.	Aunque no se explica cómo, se asume la capacidad de crear, modificar, y eliminar creencias, objetivos e intenciones.	Se ejecutan aquellas tareas que llevan a la consecución de los deseos del agente. Las tareas hacen que se modifique el estado mental. Las tareas se pueden agrupar en planes. No se explicita el mecanismo de selección de intenciones a ejecutar.	Existe la idea de *Servicio* para denotar una funcionalidad ofrecida a otros agentes y *role* en el sentido de agrupación de funcionalidad.

Tabla 3. Resumen de representaciones del estado mental y acciones en metodologías o arquitecturas de sistemas multi-agente

Respecto de las ideas iniciales de *comportamiento* y *responsabilidades*, es de notar que el concepto *estado mental* en el sentido de Shoham [Shoham 93] no es compartido por todos. Mientras que en los enfoques más formales (Agent0, CONGOLOG, BDI) se le da una importancia clave, en el resto se sigue un enfoque más de ingeniería, centrando su atención en las tareas y dejando el estado mental como un complemento (DESIRE[3], ZEUS, MaSE, JADE) . Como opción intermedia, MESSAGE hace un balance entre el peso de las tareas en el diseño y la importancia del estado mental. Un caso aparte es el tratamiento que hace

[3] El caso de DESIRE es especial porque todo el desarrollo se puede ver desde el punto de vista formal, hablando de *estados de información* y *composición de estados de información*, o desde un punto de vista de definición y composición de tareas.

ABLE, que incluye varias formas de implementar el control (lógica borrosa, razonamiento por encadenamiento hacia delante y atrás [Rich y Knight 90]) y soportar aprendizaje (árboles de decisión, redes Naïve-Bayes, mapas autoorganizativos [Rich y Knight 90]). En ABLE, control y aprendizaje son tratados de la misma forma: encapsulándolos dentro de *beans*. Esta forma de concebir el diseño de los agentes es similar al modelado que hace MAS-CommonKADS, que por herencia de CommonKADS [Tansley y Hayball 93], concibe el control del agente como un modelo aparte (modelo de *experiencia*).

La evolución del estado mental se suele especificar mediante dos operaciones que se ejecutan en la parte derecha de reglas: aserción y sustracción de entidades mentales (Agent0, CONGOLOG). Los otros trabajos, o bien hacen una gestión automática del estado mental (ZEUS), o bien se omite totalmente (MESSAGE, GAIA, Vowel Engineering). MAS-CommonKADS, por sus antecesores, debiera incorporar técnicas para la gestión del conocimiento, pero esto no ocurre.

Acerca de la autonomía e inteligencia de los agentes, existen en Inteligencia Artificial suficientes resultados metodológicos en áreas clave como para incorporarlos en las propuestas de ISOA existentes. En concreto se llama la atención sobre la planificación, sistemas expertos y aprendizaje, ya que con ellas se consigue diseñar la toma de decisión del agente de acuerdo con la situación actual y su capacidad de reaccionar teniendo en cuenta su experiencia pasada.

En general, los sistemas de planificación son sistemas capaces de describir un conjunto de acciones (o un plan) del que se espera que lleve al sistema a un estado concreto. Los planificadores suelen utilizar técnicas de búsqueda en un espacio de soluciones. Según el paradigma de planificación utilizado, cada solución será un estado del mundo o un plan, y el operador algo que modifica el estado del mundo o un transformador de plan. Según la literatura especializada (ver [Weld 99], [Hendler, Tate y Drummong 90] y [Rich y Knight 90]), hay algoritmos representativos de las tres clases de planificación que existen: planificadores lineales (STRIPS [Fikes y Nilsson 71]), no lineales (NOAH [Sacerdoti 77]) e independientes del dominio (GPS [Newell y Simons 63]). Aunque la implementación de estos algoritmos puede calificarse como agente (satisfacen la definición vista aquí), su propósito no es construir agentes sino demostrar la efectividad de una técnica de planificación frente a otras en dominios concretos. La aplicación a SMAs viene de la mano de otros dos algoritmos de planificación MuPAC [Chang, Day y Phiphobmongkol 93] y TAEMS [Decker 95]. Estos algoritmos asumen la existencia de múltiples agentes que saben ejecutar tareas y que imponen restricciones en su comportamiento.

El aprendizaje (*machine learning*) es otra de las características reseñables en los agentes. Existen múltiples técnicas de aprendizaje que se pueden aplicar. Son destacables el aprendizaje basado en inducción, orientado a la resolución de problemas, basado en explicaciones, por analogía, con redes neuronales o aprendizaje genético [Rich y Knight 90]. Este aprendizaje es función del tipo de control aplicado. Así, con control basado en reglas se aplicará prioritariamente aprendizaje por inducción.

Otro área que ha sido significativa en el desarrollo de los agentes es el trabajo realizado en Sistemas Expertos. Existen trabajos clásicos, como SOAR [Laird, Newell y Rosenbloom 87] y Prodigy [Veloso et al. 95]. También es muy importante la aportación de CommonKADS [Tansley y Hayball 93], que se trata de una metodología industrial para el desarrollo de sistemas expertos que interactúan con el usuario.

Es importante, antes de atacar el problema de la definición del control, el proporcionar medios para definir las responsabilidades del agente. La literatura repite con frecuencia la idea de *rol* y *servicio*. La primera define un conjunto de capacidades y comportamiento que es asociable a un actor. El segundo, define un conjunto de funciones que se ofrecen a otros agentes. A pesar de la similitud, el ámbito de *rol* engloba al de *servicio*, ya que un *servicio* se puede representar como un *rol*, pero no al revés.

Como conclusión de este estudio, el meta-modelo de agente debe tener en cuenta la necesidad de expresar por un lado el estado mental del agente y por otro su evolución. El *estado mental* estará compuesto por *entidades mentales* que tendrán que contemplar como mínimo creencias, compromisos y deseos.

Sobre este estado mental trabajará un control que debe:

- **Establecer una gestión del estado mental.** Cómo se crean, destruyen y modifican las entidades del estado mental. Las soluciones de la literatura son escasas en este aspecto. Destacan las *funciones para revisión de creencias* del *Modelado Computacional de SMAs* [Wooldridge 92].
- **Determinar el mecanismo de decisión.** El más común es el de definir reglas donde la parte izquierda haga referencias al estado mental y la derecha a las acciones que han de ejecutarse. Sin embargo, también puede añadirse complejidad incorporando planificación o cambiando el paradigma a lógica borrosa.

Debido a la multitud de formas en que puede presentarse el control, sería deseable separar su modelado del resto estableciendo restricciones mínimas que debe satisfacer el control. En este sentido se puede utilizar la representación del estado mental como guía del control del agente. Al control del agente se le pide únicamente que sea posible alcanzar la satisfacción de objetivos basándose en un estado mental de partida, pasando quizá por algunos intermedios.

Este enfoque permite la incorporación de inteligencia y autonomía al esquema dejando libertad para elegir la forma en que éstas se presentan. La restricción a satisfacer en la integración es la compatibilidad con el tipo de control elegido en términos de la gestión del estado mental y los mecanismos de decisión. Así, para el caso en que la decisión se base en la aplicación de reglas, es factible el aprendizaje por inducción y para el caso en que haya que optimizar algún valor, pueden utilizarse técnicas de planificación.

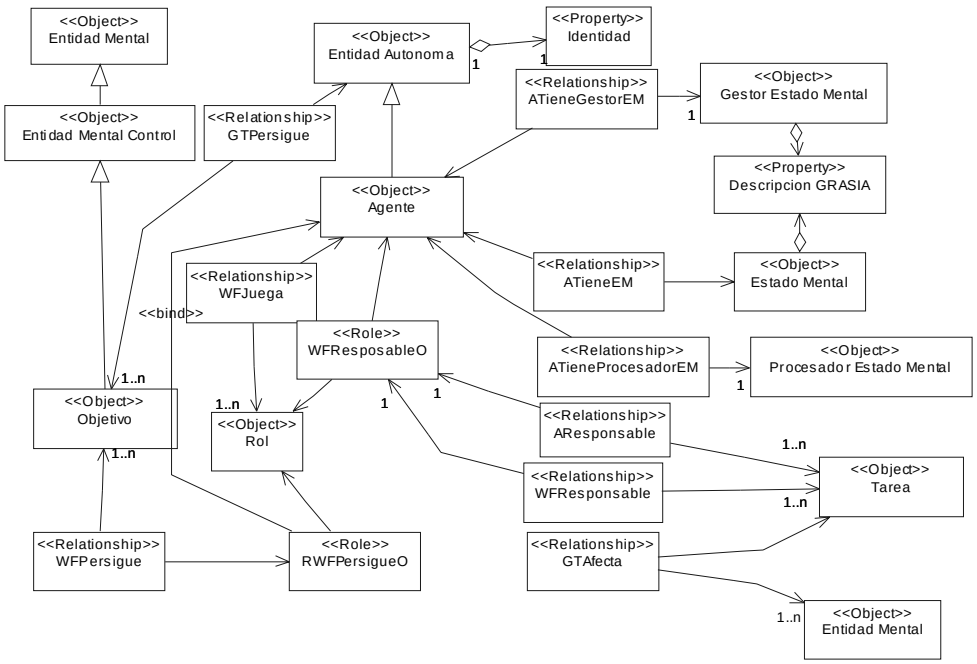


Ilustración 13. Entidades principales en el meta-modelo de agente

2.2.2 Presentación del meta-modelo de agente

En el meta-modelo propuesto (Ilustración 13), un *agente* es una *entidad autónoma*. Una *Entidad Autónoma* se caracteriza por tener propósitos y una identidad única. La entidad *agente*, por herencia de *Entidad Autónoma*, adquiere la capacidad de perseguir objetivos y, mediante las asociaciones con roles y tareas, de alcanzarlos. Un agente, siguiendo la definición de Newell [Newell 82] es una entidad autónoma que actúa en el *nivel de conocimiento* y que se basa en el *principio de racionalidad*. El principio de racionalidad dice que la entidad realiza sólo aquellas acciones que le llevan a satisfacer sus objetivos.

Por coherencia con este punto de vista, el meta-modelo refleja que existen asociaciones entre el agente, sus objetivos y las tareas desempeñadas. La primera asociación del agente con los objetivos le viene de su herencia de la *Entidad Autónoma*. Sin embargo, no es suficiente con decir que un agente tiene objetivos. Hace falta un entramado de conceptos que permita definir qué es lo que el agente está haciendo en un determinado momento, qué es lo que busca, y qué idea tiene acerca de lo que le rodea, *imagen del mundo* en BDI (e.g. entorno, organizaciones, agentes). Todo ello constituye el *Estado Mental*, que será explicado más adelante.

Para explicar mejor cómo es el meta-modelo de agente, se ha estructurado su presentación en tres sub-secciones. La primera habla del control del agente. Esta sección muestra cómo aparece el control a nivel conceptual y su relación con el estado mental del agente. También introduce la idea de *agente en ejecución* como solución para hablar de

propiedades de los agentes en el *nivel de información* (M4) dentro del *nivel del meta-modelo* (*M2*). La segunda explica cómo se definen las responsabilidades del agente. Para terminar, la tercera sección se centra en la definición del estado mental**.** Detalla la forma en que se representa el mundo utilizando entidades mentales y cómo explicitar los estados mentales intermedios que alcanzan agentes en ejecución.

2.2.3 Control del agente

El meta-modelo propuesto conlleva un paradigma de control (Ilustración 14) basado en objetivos y tareas. Este paradigma es similar a los presentados en [Ferber 99]. El agente juega cierto rol, el cual le lleva a perseguir unos objetivos. Estos objetivos se alcanzan cuando el mundo, tal como lo concibe el agente, se ha modificado. Las tareas son las encargadas de producir esta modificación. La modificación se expresa mediante la producción de evidencias, que pueden ser nuevas entidades mentales o eventos disparados por el entorno. Este esquema se incluye en el meta-modelo a través del *estado mental*, el *gestor de estado mental* y el *procesador de estado mental*.

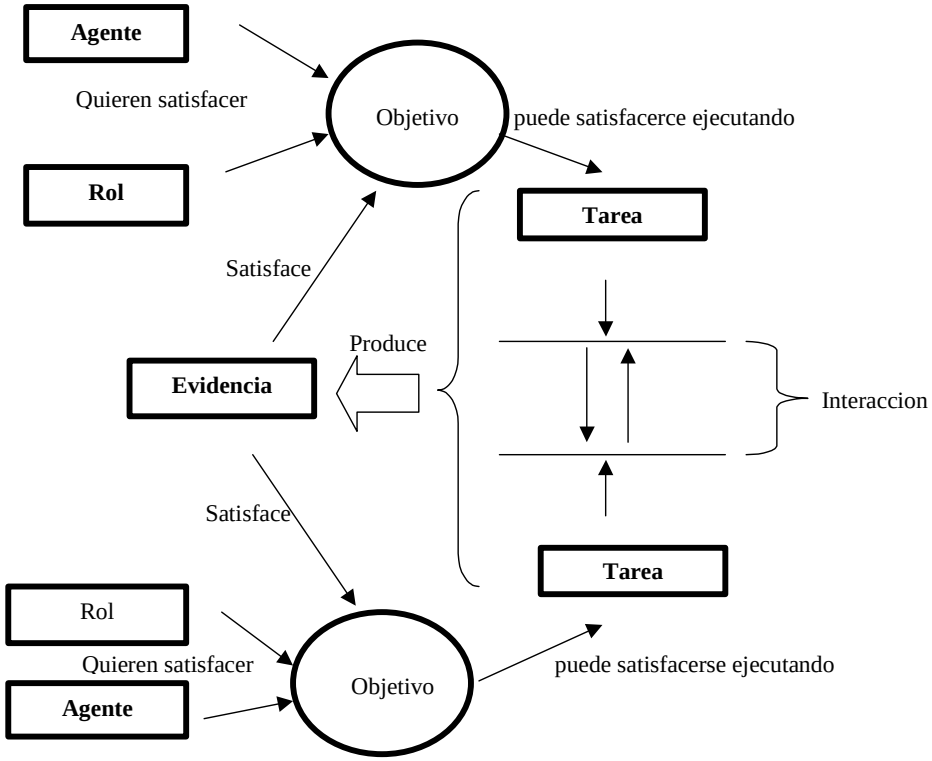


Ilustración 14. Conceptos relevantes en el control del agente

El *estado mental* puede verse como toda aquella información que permite al agente tomar decisiones. Esta información es gestionada y procesada para producir las decisiones del agente. Los trabajos relacionados de la literatura, muestran que la toma de decisiones a partir de un conjunto de datos no es trivial. Sistemas completos, como SOAR [Laird, Newell y Rosenbloom 87] o STRIPS [Fikes y Nilsson 71] se concentran en este tipo de aspectos.

Siguiendo la mentalidad integradora de este trabajo, se quiere dejar lugar a este tipo de desarrollos en la metodología. Así, se han incluido a nivel conceptual dos entidades, el *Gestor Del Estado Mental* y el *Procesador del Estado Mental*. El propósito del primero es desarrollar la evolución del estado mental mediante las operaciones de creación, destrucción, modificación y monitorización del conocimiento del agente. Es responsable de mantener la coherencia del conocimiento almacenado y de hacerlo evolucionar. El propósito del segundo es la toma de decisiones en sí, el control del agente. Las decisiones se pueden producir de forma algorítmica, como hace TAEMS [Wagner y Horling 01] en función de las tareas a ejecutar y el beneficio a obtener, u obedecer a la consecución de objetivos eligiendo secuencia de tareas a ejecutar, como en STRIPS [Fikes y Nilsson 71].

La ventaja de esta separación entre *gestor* y *procesador del estado mental* es que se desacoplan los mecanismos que implementan la autonomía e inteligencia del agente de la conceptualización de agente. El ingeniero puede decidir cualitativamente cómo va a ser el control, establecer qué debe incluir antes de implementarlo, expresarlo en función de manipulaciones del estado mental y ver cómo afecta a su desarrollo.

Para definir los requisitos que deben satisfacer el *gestor* y el *procesador* se utilizan tres herramientas: relaciones entre las tareas y entidades mentales, relaciones entre el agente y objetivos, y especificación de estados intermedios por los que pasa un agente en ejecución.

Las relaciones que asocian tareas y entidades mentales son instancias de la meta-relación *GTAfecta*. Las meta-relaciones *GTAfecta* tienen como propósito poner de manifiesto que la ejecución de una tarea afecta al estado mental del agente, satisfaciendo un objetivo o haciéndolo fallar, por ejemplo.

Las relaciones entre los objetivos y el agente, aparte de las heredadas de *Entidad Autónoma* (meta-relación *GTPersigue*) y de las obtenidas a través del *Estado Mental*, se vuelven a recalcar con la relación *WFPersigue*. Esta meta-relación coloca al objetivo y el agente en el contexto de la organización. La relación *WFPersigue* es parte de la definición del flujo de trabajo y se usa para justificar por qué se ejecutan las tareas en el seno de una organización.

Por último, en la asociación de estadios intermedios a agentes en ejecución, que se verá con detalle en la presentación del estado mental, hay que determinar cómo se referencian instancias de los agentes a nivel de meta-modelo. Se habla de referenciar instancias de entidades del modelo en lugar de utilizar directamente las instancias porque de otra forma se cometería una violación de los niveles de meta-modelado (ver Ilustración 4). Este tipo de solución también se aplica en los diagramas de secuencia y colaboración de UML donde se asigna un tipo y un identificador de variable para representar un objeto instancia de una clase.

Parte de la funcionalidad de estos gestores se puede expresar utilizando modelos de objetivos y tareas (ver sección 2.4.3). Así, para decidir qué hacer cuando un objetivo ha sido satisfecho o ha fracasado, se pueden definir tareas que tomen como entrada estos objetivos y modifiquen su estado, creen nuevos o simplemente los destruyan. Sería el equivalente a las meta-reglas de los sistemas expertos. Estas tareas estarían asociadas a objetivos especiales que denotarían propósitos de gestión.

2.2.3.1 Agentes en ejecución

En el meta-modelo de agente es necesario, para describir la evolución del estado mental hacer referencia a propiedades del agente en ejecución. Con este propósito se introduce en el meta-modelo la entidad *Consulta Entidad Autónoma*. Esta entidad sirve también para describir quién participa en las interacciones, jugando los roles correspondientes.

Consulta Entidad Autónoma se especializa en consultas acerca de las propiedades los agentes y en agentes concretos. Las primeras contienen expresiones de restricciones que un agente debe satisfacer, como "que desempeñe el papel de administrador de fondos de pensiones". Las segundas contienen expresiones que denotan agentes concretos, como librero@iiop://librería.com:1099/acc[4] que denota un agente único en el sistema.

Esta solución tiene varias ventajas:

- Permite seguir trabajando a nivel de meta-modelo con entidades en ejecución sin tener que bajar al nivel de instancias del modelo (M-3).
- Estas entidades pueden ser utilizadas dentro del estado mental como creencias acerca de la existencia de un agente con propiedades determinadas. La utilización de *Agente* no sería apropiada, ya que denota propiedades de un tipo de agente más que aludir a instancias de agentes
- Las expresiones utilizadas para identificar agentes pueden reutilizarse en el diseño de forma sencilla.
- Constituye una puerta de entrada a las tecnologías de *matchmaking* [Sycara et al. 99] en su integración con el desarrollo de SMAs. Aunque la identificación de agentes aparece en la mayoría de SMAs (RETSINA, JADE, GrassHopper), siempre lo hace en forma de servicio, sin indicar en qué forma pueden aprovecharse dentro del un agente.

[4] Localizador universal según el estándar FIPA

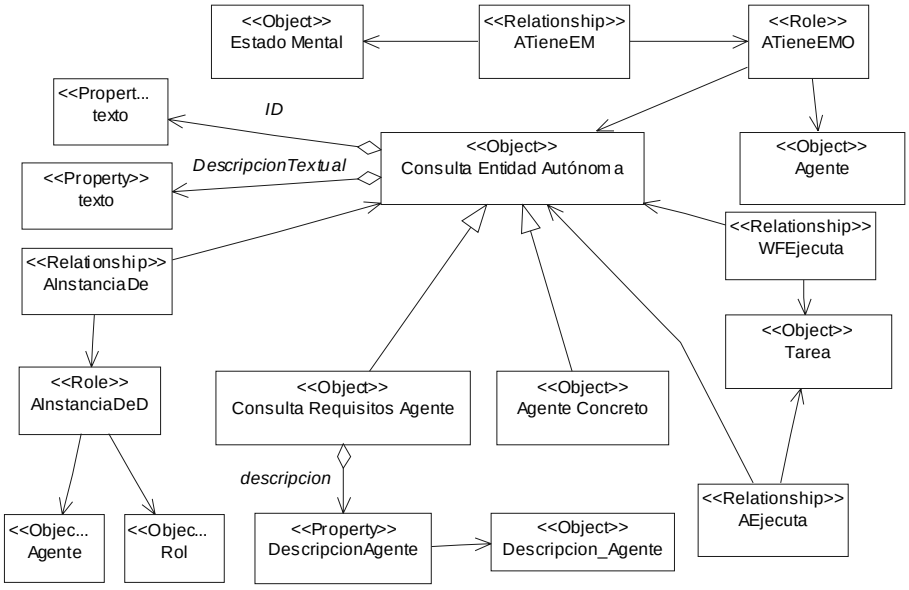


Ilustración 15. Representación de agentes en ejecución

Para poder representar completamente un agente en ejecución, por cada asociación en que participe la entidad agente, debe poder participar *Consulta Entidad Autonoma* (ver Ilustración 15). Se permite indicar si el agente en ejecución ha de ser de un tipo concreto (meta-relación *AInstanciaDe*). Esta restricción puede ser útil para hacer referencia a cualquier agente o rol de un tipo concreto. Para hacer referencia a otras cualidades del agente, se utiliza la propiedad *DescripciónAgente* (ver Ilustración 16).

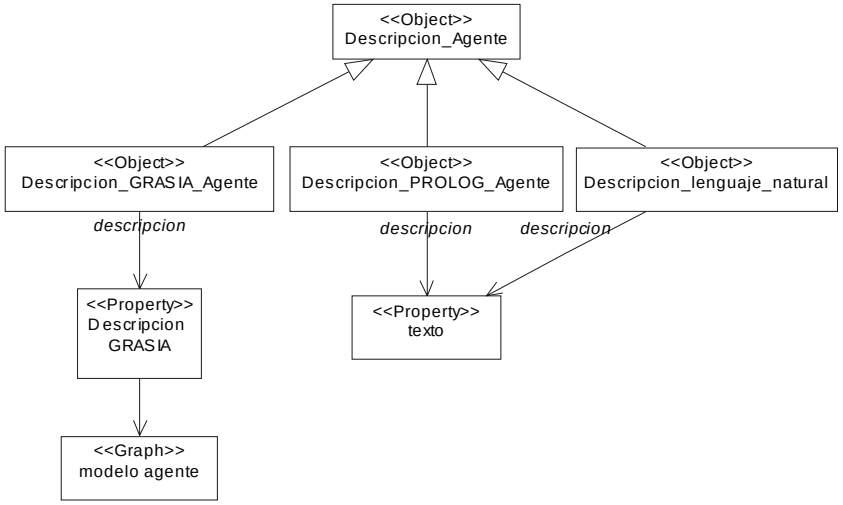


Ilustración 16. Posibles expresiones para designar un agente en ejecución

Esta propiedad se concreta en *Descripción GRASIA del agente*, que denota descripciones del agente utilizando instancias del meta-modelo de agente, y *Descripción PROLOG del agente*, que denota descripciones del agente utilizando predicados prolog donde los términos se corresponden con etiquetas que aparecen el modelo donde se encuentre. También se admite descripciones en lenguaje natural (*Descripción_lenguaje_natural*).

Los términos *Consulta Entidad Autónoma* se representan con el símbolo de un agente encerrado entre paréntesis. Este símbolo se etiqueta de diferente manera dependiendo de a qué se haga referencia:

- **Un agente concreto (*Agente Concreto*).** En este caso la etiqueta es el nombre del agente o bien una expresión que denota un único agente. Como ejemplo de aplicación, en los casos de estudio se han utilizado expresiones que utilizan referencias a resultados de tareas y *slots* de entidades mentales para denotar agentes singulares. Como caso especial, se tiene la etiqueta "*(ejecutor)*". Aparece al referenciar un agente que va a ejecutar una tarea, al definir un estado mental intermedio de un agente en una interacción, o al estudiar la satisfactibilidad de un objetivo. El *ejecutor* está unido a la tarea ejecutada con una instancia de la meta-relación *WFEjecuta* o *AEjecuta*, indicando ejecución de tareas en el contexto de un flujo de trabajo o por obligación del agente (las condiciones de ejecución no están supeditadas a otros agentes), respectivamente. En una interacción, la asociación entre *ejecutor* y el agente, o rol, se realiza mediante una propiedad de tipo *patron de estado mental* asociada al inicio de una unidad de interacción (*UIInicia*) o colaboración (*UIColabora*), como se verá más adelante en el meta-modelo de interacción.
- **Un conjunto de agentes (*Consulta Requisitos Agente*).** Otra vez la etiqueta define un conjunto de agentes por intensión. Se utiliza una expresión que denota las propiedades que se requieren de los agentes. Estas expresiones utilizan como términos predicados equivalentes a las relaciones representadas en los meta-modelos. Como indica la Ilustración 16, se contemplan expresiones en PROLOG (para incluir expresiones en lógica proposicional), lenguaje natural y descripciones con modelos de agente. Estas últimas tienen la ventaja de poder hacer referencia directa a elementos de otros modelos. Un ejemplo de descripción de agente seria el de la Ilustración 17.

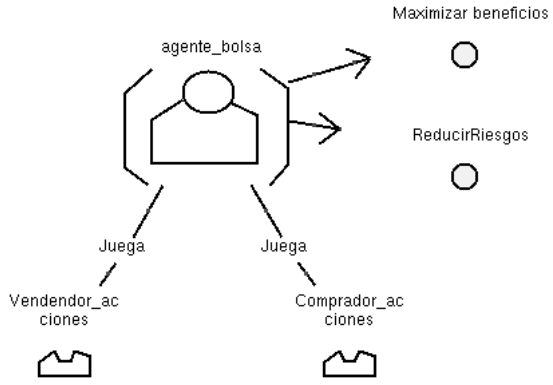


Ilustración 17. Descripción de un agente de bolsa en ejecución.

La Ilustración 17 muestra el ejemplo mencionado al principio de esta sección. Se busca un agente que sea capaz de vender y comprar acciones y que persiga maximizar beneficios y reducir riesgos. Para representar esta situación, se asocia un agente en ejecución con los objetivos pertinentes y con los roles identificados en el problema.

2.2.4 Asociación de responsabilidades

El meta-modelo contempla el uso de roles para asignar responsabilidades. El concepto de rol ha demostrado ser útil a la hora de realizar análisis de sistemas y de hecho existen metodologías que se basan en su utilización ([Kendall 95]). Su papel aquí consiste en separar la definición del agente de lo que se requiere de él. El rol del que aquí se habla es distinto del de UML. Como se indica en [Depke, Heckel y Küster 01], la especificación que hace UML de rol no se adecúa a las necesidades de la ISOA. En general, el rol es una abstracción de un conjunto de funciones, que puede tener estado y que para existir necesita de otra entidad no abstracta que lo desempeñe, en este caso el agente. De esta forma es posible aplicar los roles en ISOA para describir conjuntos de responsabilidades (las funciones asociadas) , protocolos (gracias a que el rol tiene estado implícito), o permisos de actuación (cambiando los roles de un agente, se cambia su capacidad de actuación).

Los roles aparecen en las interacciones y en las organizaciones. Los roles carecen de gestores y procesadores de estado mental, y, por lo tanto, de capacidad de toma de decisiones. Por su simpleza, es utilizado con frecuencia en lugar de los agentes, ya que hace más tratable el diseño de algunos aspectos del sistema, como las interacciones. Los roles también fomentan la reusabilidad, ya que existen para otorgar al que los juega sus cualidades. El uso de roles aparece con frecuencia en las metodologías existentes (GAIA, MaSE, MESSAGE). La relación entre roles y agentes se hace mediante la meta-relacion *WFJuega*. Esta meta-relación indica al diseñador que el agente adquiere todos los objetivos asociados al rol, todas sus responsabilidades y sus capacidades.

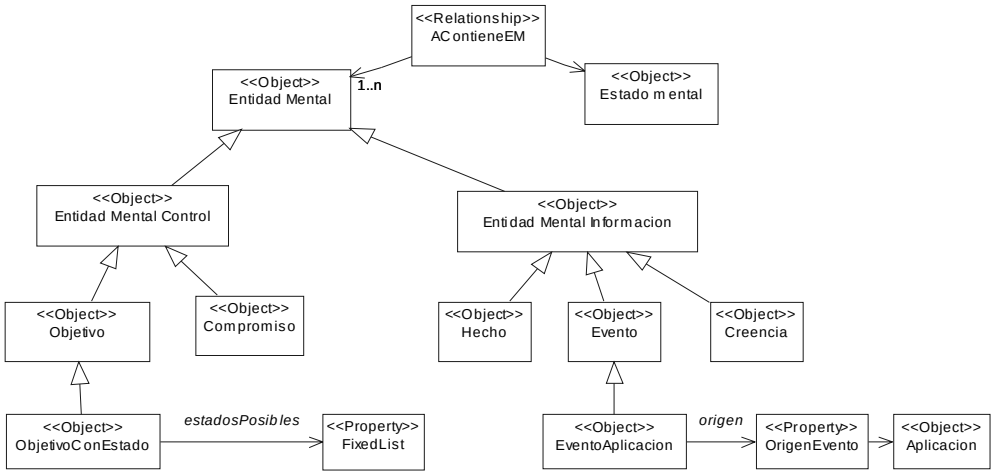


Ilustración 18. Meta-modelo de entidades mentales válidas

2.2.5 El estado mental

El estado mental se define como agregación de entidades mentales. Se admiten múltiples instancias del estado mental relacionadas con un agente. Al asociarlo directamente, se indica que el agente, al crearse, parte de ese estado. Cuando se asocia indirectamente, como en la Ilustración 21, se está definiendo un estado mental intermedio en la vida del agente. Para expresar estas situaciones, se admite que existen instancias del meta-modelo de agente que representan momentos concretos en la vida del agente. Para estos estados intermedios, se emplean entidades *Consulta de Entidad Autónoma*, ya introducidas en la secciones anteriores.

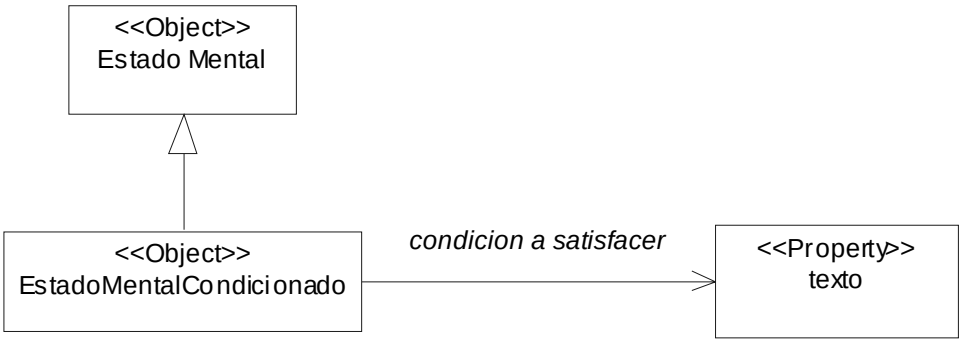


Ilustración 19. Asociación de condiciones al estado mental

En estas situaciones intermedias, a veces es importante añadir información acerca de qué se espera de las entidades mentales. Por ejemplo, si sus propiedades deben contener algún valor concreto o si se debe satisfacer cierta expresión. Para ello, se define un *estado mental condicionado* (Ilustración 19) que se caracteriza por contener expresiones que referencian las entidades mentales que existen en un momento concreto.

Las entidades mentales identifican los elementos en los que se basa el agente a la hora de tomar decisiones. Operacionalmente, la representación ofrecida es equivalente a un sistema de *marcos* con conocimiento heredable [Rich y Knight 90]. Definen, utilizando terminología de IA, una serie de ranuras (*slots*) que el diseñador ha de rellenar, pero a las que no se haya restringido, ya que se pueden introducir nuevas mediante herencia de las anteriores. Esto quiere decir que el diseñador tiene completa libertad para extender la estructura de entidades propuestas con el fin de adecuar la ontología a su problema concreto.

En el estado mental (ver Ilustración 18), la información se representa a alto nivel. Siguiendo las tendencias existentes [Shoham 93] [Lespérance et al. 96] [Wooldridge 92] [Caire et al. 02;Iglesias et al. 98;Wooldridge, Jennings y Kinny 00] , el estado mental se define a partir de entidades como *objetivos* (fundamental en este modelo), *creencias* y *compromisos*. Estas son las entidades compartidas por la mayoría de los modelos formales de agente. Adicionalmente, se plantean *hechos* (información cierta para el agente) y *eventos* (los cambios ocurridos, en el mundo que el agente capta). Los *eventos* deberían transformarse en *hechos* automáticamente en cuanto formaran parte del estado mental del agente. Sin embargo, en este modelo conviene más el mantenerlos así y dejar que sean los

procesos internos del agente los que decidan si debe realizarse tal transformación o simplemente deshacerse del evento.

Las entidades de control son el *objetivo*, porque indica al agente qué es lo que le motiva, y el *compromiso*, porque liga al agente con una acción requerida por otro actor. Los objetivos tienen asociado un estado mediante la propiedad *estados posibles*, que es una lista de constantes que denotan los estados por los que pasa. Los objetivos se satisfacen con la presencia de ciertas evidencias y pueden darse por fracasados cuando estas evidencias no se han producido. Hasta llegar a un objetivo satisfecho o fracasado, se pueden recorrer varios caminos. Aquí se propone que un objetivo pueda estar *pendiente, recogiendo evidencias, refinado, en proceso, satisfecho* o *fallido*. El ciclo de vida hace referencia a cómo se ejecutan tareas asociadas al objetivo (las que producen las evidencias que necesita). Las tareas no tienen estado en esta propuesta, por lo que, en caso de necesitar estructurar su ejecución, las tareas tienen que delegar su estado a otra entidad, el objetivo en este caso. La elección de este ciclo de vida no es obligatoria. Se sugiere este ciclo de vida en concreto de acuerdo con la experiencia en desarrollos fuertemente orientados a objetivos [Gomez-Sanz, Garijo y Pavon 00]

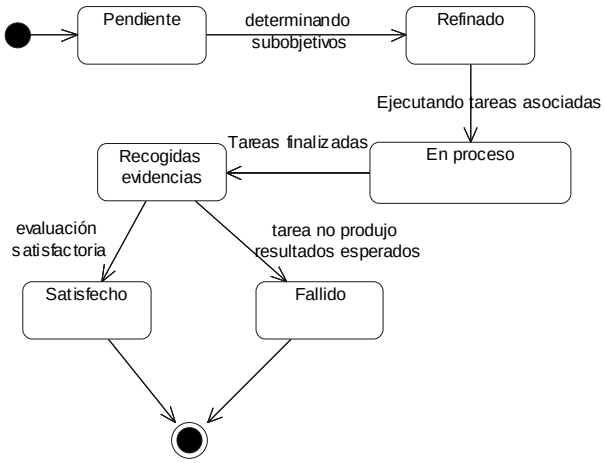


Ilustración 20. Ciclo de vida de un objetivo

El resto de entidades que definen el estado mental son entidades informativas que el agente utiliza para formase una imagen del mundo que le rodea. Los *hechos* reflejan información que es inherentemente cierta, como que "*el agua se evapora al aplicarle calor*" o bien información resultante de la ejecución de tareas. Las *creencias* reflejan información subjetiva en el sentido de que proviene de los elementos perceptivos del agente. Estas *creencias* son interpretaciones particulares hechas por el agente de lo que percibe. Las interpretaciones, pueden diferir por motivos técnicos como diseños diferentes de control de los agentes. Las *creencias* surgen de los *eventos* que simbolizan cambios ocurridos que han sido percibidos por el agente. Así ante el *evento "comunicación con el agente X interrumpida*" el agente puede elaborar la creencia "*no quiere comunicarse conmigo*" o bien "*ha ocurrido un fallo en la red*" o bien "*ha ocurrido un fallo en el ordenador que alojaba al agente X*".

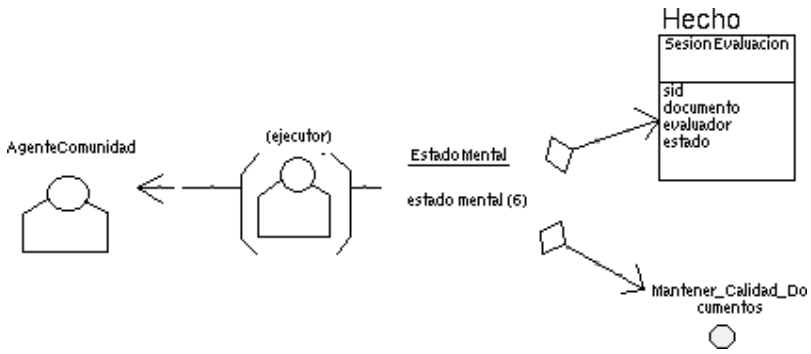


Ilustración 21. Ejemplo de estado mental requirido por un agente que decide si debe ejecutar una solicitud de alta en una fuente de información.

Como ejemplo, la Ilustración 21 corresponde al estado mental que debe tener un agente de comunidad que quiera procesar una petición de inscripción en la comunidad (ver el capítulo de experimentación). El diagrama dice que el agente persigue el objetivo *Mantener calidad documentos* y que además existe una sesión de evaluación en curso. El elemento *ejecutor* identifica al agente que ejecuta la tarea y la asociación con el agente indica que el ejecutor es del tipo *Agente de Comunidad*.

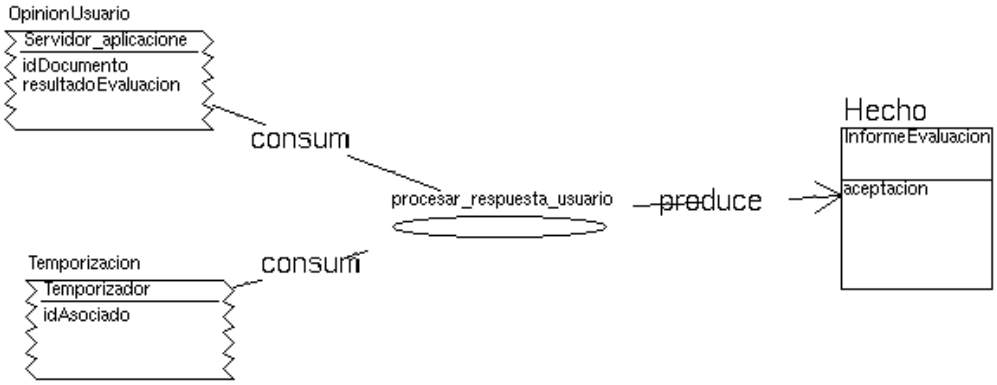


Ilustración 22. Ejemplo de utilización de eventos y hechos

La Ilustración 22 muestra una tarea cuyo propósito es generar un informe de evaluación acerca de la conveniencia de aceptar un nuevo usuario en una comunidad. La tarea *procesar respuesta usuario* puede arrancar porque el usuario haya evaluado la petición de suscripción (evento *OpiniónUsuario*) o porque haya saltado un temporizador por el retraso del usuario en ejecutar la evaluación (evento *Temporización*). De cualquier forma, estos dos eventos se transforman en un nuevo hecho (*Informe de Evaluación*).

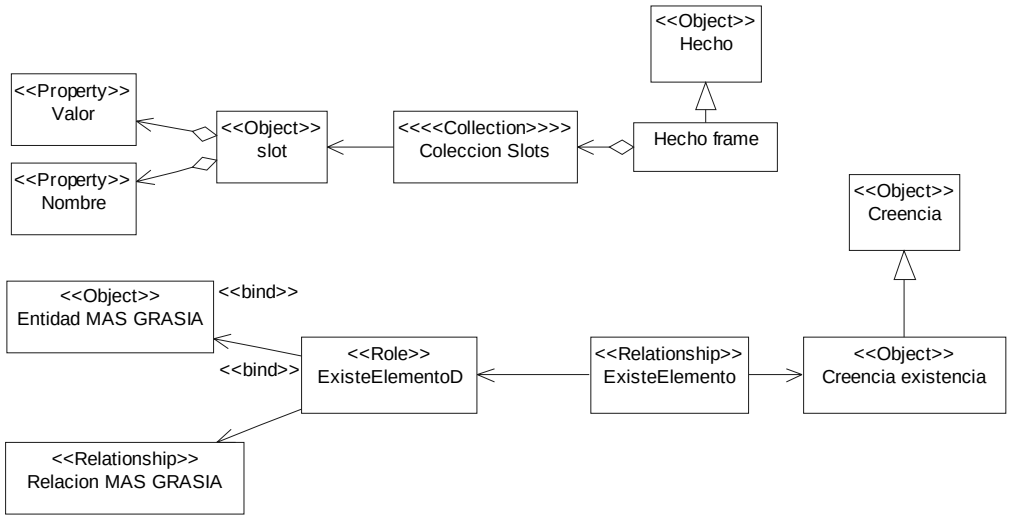


Ilustración 23. Extensión del meta-modelo para incluir creencias expresadas con conceptos MAS GRASIA y hechos representados con marcos (*frames*)

El estado mental es extensible para expresar entidades mentales más elaboradas. Por ejemplo, se plantean las dos extensiones de la Ilustración 23 que permiten incluir creencias acerca del estado mental de otros agentes o de sus capacidades o utilizar una representación más clásica de conocimiento basado en marcos (*frames*).

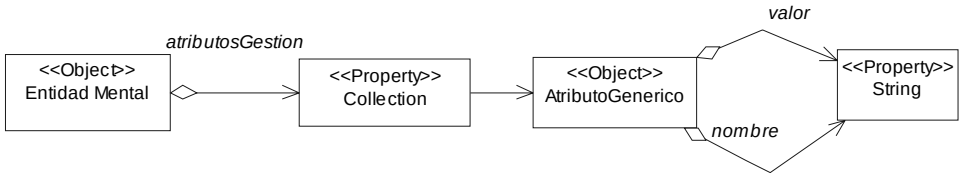


Ilustración 24. Atributos genéricos para la gestión de entidades mentales

Es deseable asociar atributos a las entidades mentales para propósitos de gestión del conocimiento (Ilustración 24). Ejemplos de tales atributos serían atributos temporales que especifiquen la fecha de creación de la entidad mental y hasta cuándo se estima que esa entidad tendrá validez. La naturaleza y propósito de estos atributos de gestión es variada. Los compromisos, por ejemplo, pueden ser válidos sólo hasta que se sobrepase una fecha, los eventos sólo durante unos segundos. La elección de los atributos de gestión comunes a todas las entidades mentales es función del tipo de gestión que se vaya a realizar. Por ello, se ha añadido en el meta-modelo sólo la asociación de atributos genéricos. Lo adecuado en este punto sería aplicar técnicas de ingeniería del conocimiento para conseguir una mejor estructuración de todas estas entidades, pero eso queda fuera del propósito de esta metodología.

2.2.5.1 Justificación de la definición de estado mental

La representación del estado mental tiene como fin permitir al diseñador definir diferentes situaciones en la configuración del estado mental. Siguiendo [Rich y Knight 90], una buena representación de conocimiento debe reunir las siguientes propiedades:

- **Suficiencia de la representación.** La capacidad de representar todos los tipos de conocimiento necesarios en el dominio.
- **Suficiencia deductiva.** La capacidad para manipular las estructuras de la representación con el fin de obtener nuevas estructuras que se correspondan con un nuevo conocimiento deducido a partir del antiguo.
- **Eficiencia deductiva.** La capacidad de incorporar información adicional en la estructuras de conocimiento con el fin de que los mecanismos de inferencia puedan seguir las direcciones más prometedoras.
- **Eficiencia en la adquisición.** La capacidad de adquirir nueva información con facilidad. El caso más simple es aquél en el que una persona inserta directamente el conocimiento en la base de datos. Idealmente, el programa sería capaz de controlar la adquisición de conocimiento por sí mismo.

La representación de estado mental elegida es equivalente a una representación basada en marcos (*frames*) clásica en IA. Una representación basada en marcos puede entenderse como una plantilla (el marco) donde se indica qué datos deben figurar en ella (cada una de las ranuras o *slots*). Existe una correspondencia unívoca entre la representación basada en marcos y las entidades de la Ilustración 18. La correspondencia se demuestra asociando a cada una de las entidades mentales una estructura de marco donde el nombre del marco es el nombre de la entidad mental y sus ranuras los atributos.

El conocimiento codificado con este esquema reune características de conocimiento heredable. Por su representación en GOPRR existe una relación de herencia que permite reusar y extender definiciones, permitiendo también estructuras de ranura y relleno (esto último se consigue asignando valores por defecto o valores de atributos ya definidos en los antecesores). Lo que no está contemplado en esta representación es la posibilidad de conocimiento deductivo o procedimental. Esto sería algo que debería aportar el *gestor de estado mental*.

Para justificar *la suficiencia de la representación* elegida para el estado mental se ha cotejado contra la literatura acerca de planificación, SMAs basados en paradigmas lógicos y metodologías existentes. Además, se han realizado prototipos de SMAs en varios dominios: servicios en agencia de viajes ([Caire et al. 02], asistencia para los usuarios de un sistema de desarrollo de proyectos software [Eurescom P815 99], y personalización de contenidos en internet [PSI3 01].

La *suficiencia deductiva* es equivalente a la que posee un sistema que trabaje con marcos (*frames*). Esta equivalencia se basa en una relación uno a uno entre un marco y la entidad mental donde el marco equivalente tiene tantas ranuras como atributos la entidad mental.

La *eficiencia deductiva* está asegurada porque el meta-modelo no establece que esta estructura sea fija, esto es, que una vez fijado el estado mental del agente, permanezca inmutable. El *gestor de estado mental* y las tareas tienen la potestad de añadir nuevo conocimiento, siempre y cuando el tipo de conocimiento añadido encaje en la ontología .

La *eficiencia en la adquisición* es función directa del tipo de gestor de estado mental elegido. Un gestor de estado mental que se corresponda con un armazón (*shell*) de sistema experto, puede admitir nuevo conocimiento de forma sencilla. Si el *gestor de estado mental* se implementa con una máquina de estados (el estado mental sería en este caso la codificación del estado de esta máquina), la adquisición de nuevo conocimiento puede darse por descartada.

2.2.6 Ejemplo: asistente personal

Se quiere modelar un agente de interfaz que asista al usuario en el manejo de un procesador de texto. El agente intercepta los eventos enviados por el usuario a las aplicación de la interfaz gráfica y utiliza esta información para asesorar al usuario acerca de secuencias de acciones complementarias que pueden mejorar los resultados obtenidos. Este ejemplo se centra en acciones relacionadas con el formateo de texto.

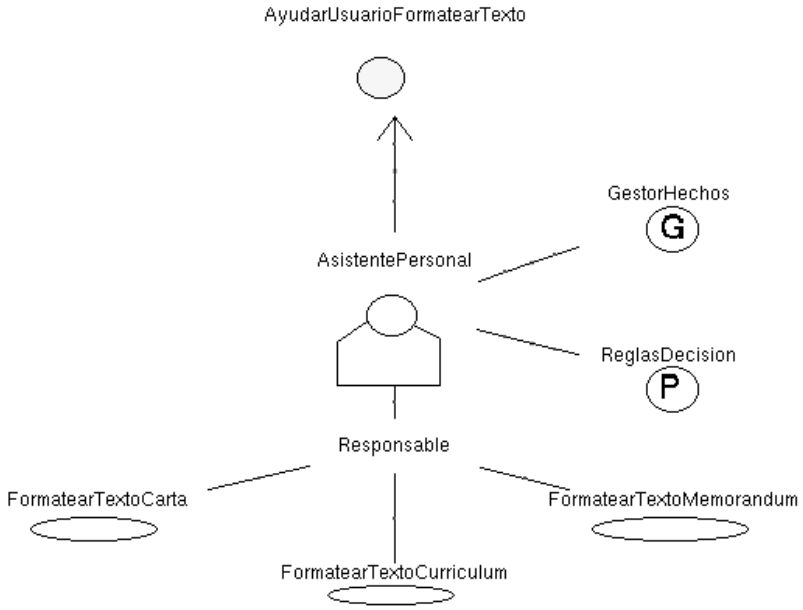


Ilustración 25. Objetivos del asistente personal (modelo de agente)

La Ilustración 25 muestra que el agente, *Asistente Personal*, tiene como objetivo el ayudar al usuario en labores de formateado de texto. Inicialmente, el asistente sabe cómo convertir el texto seleccionado en una carta (tarea *FormatearTextoCarta*), como un currículo (tare *FormatearTextoCurriculum*) o como un memorándum (tarea *FormatearTextoMemorandum*). El estado mental del agente se expresará como un conjunto de hechos, por lo que sólo se necesitan primitivas para asertar y eliminar hechos. El procesamiento del estado mental se hará mediante reglas de producción.

El agente percibe las acciones del usuario a través del procesador de texto. Para representar esto es necesario utilizar un modelos de entorno (Ilustración 26) (ver la sección 2.6).

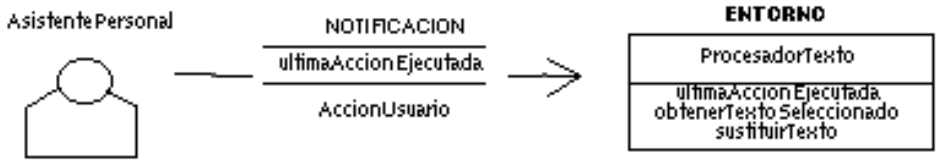


Ilustración 26. Percepción del agente utilizando el procesador de texto (modelo de entorno)

El agente necesita saber qué acciones ejecuta el usuario. Por ello se define una asociación de tipo notificación entre el *Asistente Personal* y el *Procesador de Texto*. Esta relación se interpreta como que el asistente personal debe ser notificado de cambios en el resultado de la operación *últimaAcciónEjecutada*.Cuando se produzca un cambio se creará un evento de la clase *Acción Usuario*. En este ejemplo se supone que el usuario selecciona texto en el procesador. El procesador modifica su estado para que con la operación *ultimaAcciónEjecutada* se devuelva la acción *selección de texto*. La percepción por notificación se activa y crea una nueva entidad mental de tipo *Evento*.

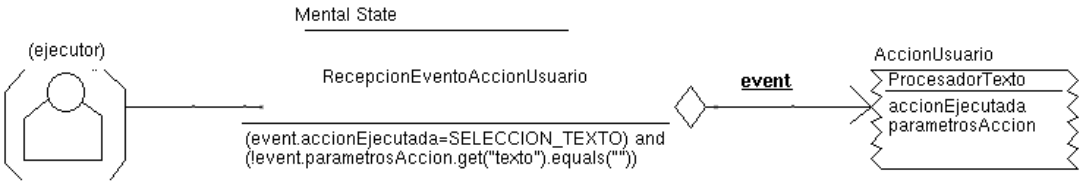


Ilustración 27. El agente conectado al procesador de texto recibe un evento (modelo de agente)

La Ilustración 27 expresa un estado mental significativo para el agente. Para concretar el estado mental se ha utilizado una expresión JAVA. En este estad se ha recibido un evento *AccionUsuario* cuyo slot *accionEjecutada* indica que se ha seleccionado texto en el procesador. Entre los parámetros de la acción del usuario (*parameterosAccion*) se incluye el parámetro *texto*, que ha de ser distinto de la cadena vacía.

Dentro de un SMA, el *Asistente Personal*, para mejorar la asistencia al usuario, puede contactar con otros agentes para preguntarles si saben realizar labores de formateo que él desconoce. Antes de iniciar la comunicación, se plantearía el estado mental de la (Ilustración 28).

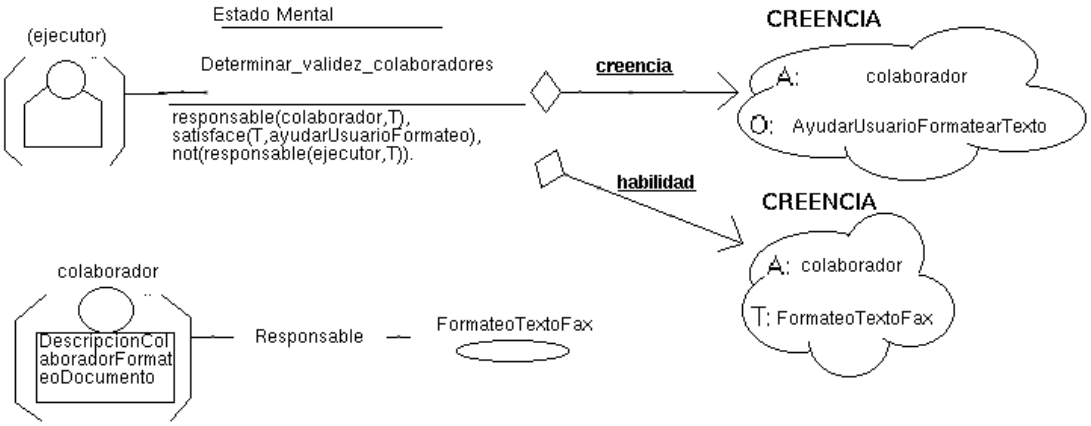


Ilustración 28. Creencias necesarias para expresar la conveniencia de colaborar con otro agente (modelo de agente)

El agente con el que se colabora se etiqueta *colaborador* en la Ilustración 28. Este *colaborador* se expresa con una *Consulta Requisitos Agente* que describe el agente colaborador mediante una expresión etiquetada como *DescripcionColaboradorFormateoDocumento*. Las creecias mostradas en la figura representan instancias de la Ilustración 23, indicando la existencia de instancias de *WFResponsable* y *GTPersigue* entre un agente y una tarea y entre un agente y un objetivo, respectivamente. En la primera instancia, se asocia un colaborador con la tarea *formateo texto fax*. En la segunda se asocia un colaborador con el objetivo *ayudar usuar a formatear texto*. Del agente colaborador no se puede esperar un tipo concreto, por ello se describen únicamente sus cualidades: que debe perseguir el objetivo *AyudarUsuarioFormatearTexto* y que debe tener alguna tarea que la satisfaga, *FormateoTextoFax* en este caso (Ilustración 29). Sin embargo, hace falta algo más, esto es, que las tareas que sepa ejecutar el colaborador no sea alguna conocida por el *Asistente Personal*. Esto se expresa dentro del estado mental de la Ilustración 28 en forma de predicado PROLOG. Se pide que *colaborador* sepa ejecutar una tarea *T* que satisfaga *AyudarUsuarioFormateo* que no esté disponible en el asistente *(not responsable(ejecutor,T)*).

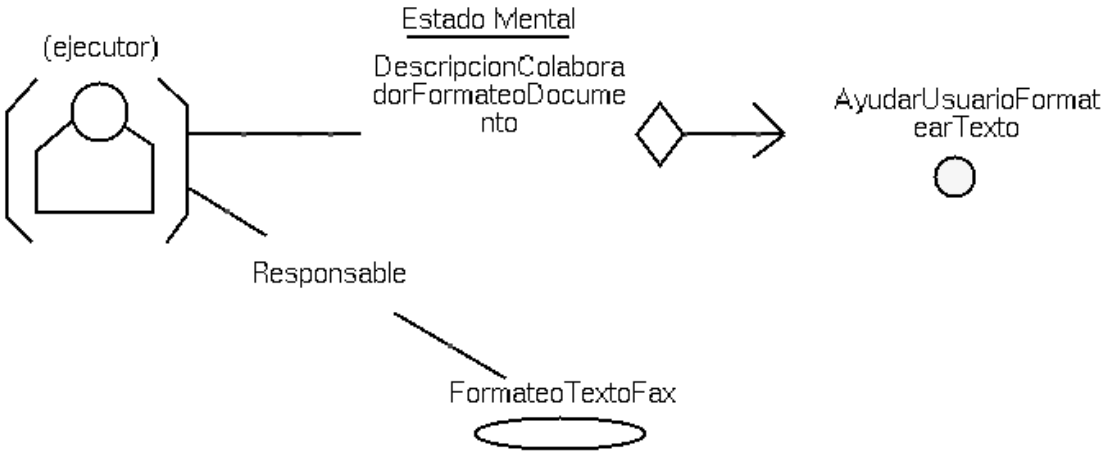


Ilustración 29. Descripción del agente colaborador (modelo de agente)

2.2.7 Integración con otros meta-modelos

El meta-modelo de agente contiene algunos de los elementos más comunes en todos los meta-modelos, como los *agentes* o *roles* (Ilustración 30).

En el meta-modelo de agente se asocian agentes con los roles, que luego participan en el meta-modelo de interacción. De esta participación se extraen un conjunto de estados mentales y de acciones que debe ejecutar el agente.

El meta-modelo de organización situa al agente con relación a otros agentes. Además, proporciona una descripción del papel del agente en los flujos de trabajo existentes. Mediante esta descripción se explica cómo colabora el agente con otros agentes para lograr objetivos comunes.

Para terminar, el meta-modelo de entorno proporciona una descripción de las aplicaciones y recursos requeridos por el agente o por las tareas de las que es responsable. También sirve como descriptor de la percepción del agente. Como se verá más adelante, en el meta-modelo de entorno, el agente se puede conectar a aplicaciones ya existentes para obtener información del entorno. La aplicación puede ser software como un procesador de texto o ser un recubrimiento de dispositivos hardware, como un controlador de una compuerta en una canalización de agua.

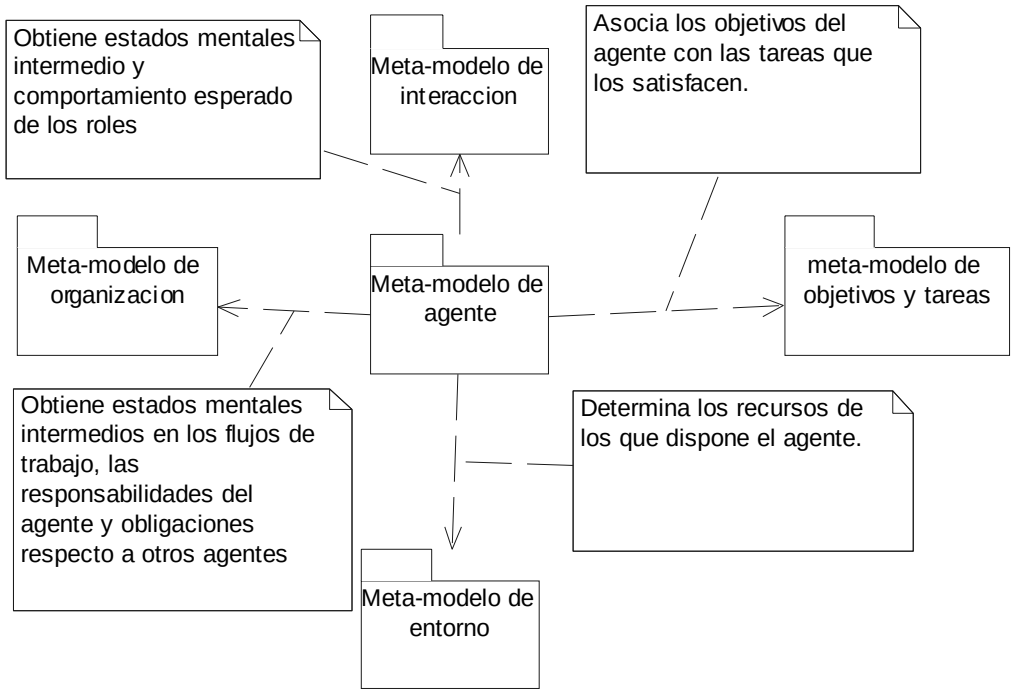


Ilustración 30. Dependencias del meta-modelo de agente respecto de otros meta-modelos

Para asegurar la consistencia de un modelo de agente respecto a otros modelos de un SMA es necesario satisfacer los siguientes requisitos:

1) Toda tarea asociada al agente debe aparecer en alguna instancia de meta-modelos de interacción, si implica interacción entre agentes, y en el modelo de organización, indicando su papel dentro de la estructura global de tareas.
2) Para cada tarea es necesario indicar cómo afecta al estado mental del agente. La ejecución de cualquier tarea siempre conlleva alguna modificación. Esta modificación debe aparecer en los modelos de tareas y objetivos.
3) El modelo del estado mental debe tener en cuenta la ejecución en paralelo de distintas tareas que modifiquen el estado mental. Es admisible que dos tareas tengan efectos contrarios sobre la misma entidad mental. Este hecho debe comprobarse en modelos de objetivos y tareas.
4) Para cada agente del modelo de organización, debe existir una instancia del meta-modelo de agente y viceversa.
5) Cada rol jugado por el agente debe aparecer en algún modelo de organización o en algún modelo de interacción
6) Un agente no puede jugar dos roles que interactúan entre sí.
7) Desde un punto de vista formal, sería deseable asegurar la consecución de objetivos. De forma somera, y tomando el ciclo *Evaluar objetivos - Evaluar tareas - Ejecutar tareas -*

Modificar Estado Mental, existe una similitud evidente con la concepción formal de un bucle: (7.1) actúa como función de progreso, el invariante es la función (7.2), y la condición de salida sería el postulado (7.3). Si se tienen en cuenta estas tres ideas, el agente alcanzará los objetivos fijados:

7.1) Cuando un agente se asocie a una tarea, los objetivos que se alcancen han de favorecer la consecución de los objetivos que persiga el agente (propios y obtenidos por asociación a roles). Esta información debe estar presente en los modelos de objetivos y tareas.

7.2) El agente siempre ejecuta acciones que le hacen alcanzar objetivos. Esta información se obtiene de este modelo.

7.3) Cuando un agente alcanza los objetivos fijados el problema pasa a ser un tema de control y como tal debe ser tratado. Dependiendo de la situación es posible:

i) Plantear nuevos objetivos que pueden ser réplica de los alcanzados. Aunque conceptualmente, se pueda tratar como un bucle, puede interesar que este sea infinito. La creación de nuevos objetivos puede tratarse a nivel del procesador de estado mental o como resultado de la ejecución de tareas (ver meta-relación *GTCrea* en meta-modelo de tareas y objetivos). Nuevamente, esta información debe aparecer en modelos de objetivos y tareas.

ii) Finalizar la ejecución. Alcanzados los objetivos fijados inicialmente, ¿existe motivo para continuar? Si no es así, quizá el agente deba dar por terminado su tiempo de vida.

2.3 Meta-Modelo de Interacción

El rol de las interacciones en la ISOA es fundamental ya que las interacciones identifican dependencias entre los componentes y contribuyen a la especificación del comportamiento de los componentes así como la funcionalidad asociada. Estos aspectos cubren un alto porcentaje de los asuntos a considerar cuando se diseña un sistema. De acuerdo con la experiencia del autor, el desarrollo de interacciones constituye un importante porcentaje del esfuerzo total, similar al invertido cuando se modelan interfaces de usuario (en este área las estimaciones dicen que el desarrollo de interfaces requiere un 47% del esfuerzo de desarrollo calculando este esfuerzo como la media aritmética entre el tiempo de diseño y de implementación [Myers y Rosson 92]). Debido a la similitud entre la interacción con el usuario y la interacción entre agentes, debería quedar clara la importancia del modelado de interacciones como un concepto clave en cualquier propuesta de ISOA. Sin embargo, la mayoría de las propuestas existentes no prestan la atención suficiente a este hecho. Alguno de ellos consideran que las interaciones son el resultado de ejecutar tareas, por lo que se centran en relacionar tareas (ZEUS [Nwana et al. 99]). Otros comienzan suponiendo que la solución está en la especificación basada en paso de mensajes (MESSAGE [Caire et al. 02]) Otros estudian qué se requiere de una interacción, pero no estudian cómo deberían ser diseñadas (GAIA [Wooldridge, Jennings y Kinny 00] [Zambonelly, Wooldridge M. y Jennings N.R. 00]). Para terminar, hay metodologías que consideran la construcción de SMA desde un punto de vista de ingeniería del conocimiento,

prestando poca atención a las interacciones ya que el problema principal es cómo modelar los objetivos del sistema (TROPOS [Bresciani et al. 01]).

De entre todos estos, destacan MAS-CommonKADS [Iglesias et al. 98] y *Vowel Engineering* [Demazeau 95] [Ricordel 01], porque tratan de integrar las interacciones en el conjunto de vistas del sistema haciéndolas parte indispensable de las metodologías. El trabajo presentado en esta sección se entiende como un complemento de los resultados alcanzados en ellas. Extiende MAS-CommonKADS y *Vowel Engineering* mediante la abstracción de las interacciones para estudiarlas desde diferentes puntos de vista, la naturaleza de la interacción, su ejecución, representación y contexto, integrándolas en el proceso de desarrollo (análisis y diseño), y estudiando elementos de diseño (como las diferentes formas de definir el control de la interacción) para que la implementación final sea más sencilla.

Esta sección no está orientada a la aplicación de lenguajes concretos de interacción como KQML [Finin et al. 94] o ACL [FIPA 01], ni al desarrollo de algoritmos de coordinación. Este trabajo establece cómo una interacción se puede integrar en el ciclo de desarrollo de un SMA, poniendo especial énfasis en actividades de análisis y diseño.

2.3.1 Análisis y diseño de interacciones

Las interacciones determinan el comportamiento de los agentes mostrando cuál es su reacción cuando actúan sobre ellos. Y cómo el comportamiento va a ser función de las objetivos de los agentes y las tareas a ejecutar, se puede concluir que existe un importante vínculo entre interacciones, objetivos y tareas.

El nivel de abstracción en el que se definen las interacciones cambia del análisis al diseño. En el Proceso Racional Unificado (*Rational Unified* Process) [Jacobson, Rumbaugh y Booch 99], un primer paso en el análisis es generar casos de uso que hagan referencia a interacciones clave, y quizás añadir diagramas de colaboración, secuencia o de actividades para mostrar cómo evoluciona el sistema. Posteriormente, en el diseño, estas interacciones se detallan con más diagramas de los antes mencionados, decorándolos con diagramas de estado y diagramas de clase para mostrar las interfaces relevantes.

En el área de agentes, sin embargo, esto no es tan simple. Aunque el análisis puede ejecutarse de una forma similar, esto es, definiendo algunos diagramas, el diseño no puede ser así. De acuerdo con la experiencia en [Eurescom P815 99], [Caire et al. 02] y [PSI3 01], la especificación completa de una interacción tiene que cubrir:

1. **Los actores que participan.** Un actor debería mostrar por qué está participando en la interacción. Esto es coherente con un modelo de agente que se basa en el *principio de racionalidad*.
2. **La definición de unidades de interacción.** La naturaleza de la unidad de interacción determina cómo tiene que procesarla el receptor. Una unidad de interacción puede ser tan simple como un paso de mensaje o tan compleja como la propuesta de [Ribeiro y Demazeau 98] que recibe el nombre de *mensajes activos*. Este tipo de mensajes se caracterizan por hacerse responsables de la interacción con el receptor de una forma personalizable dependiendo de las capacidades del receptor.

3. **Un orden sobre las unidades de interacción.** Las unidades de interacción se organizan siguiendo un protocolo estándar (como *contract-net* [Smith 80] o *FIPA-request* [FIPA 01]) o adecuado a una situación concreta. La representación del orden varía desde los diagramas de secuencia de mensajes, pasando por los diagramas jerárquicos de FIPA [FIPA 95], al actual diagrama de protocolos de Agent-UML (AUML) [Bauer, Müller y Odell 01].
4. **Acciones ejecutadas en la interacción**. Incluye detalles sobre:
    a. **Criterios para decidir cuándo ejecutar una tarea.** Para ejecutar una tarea no es suficiente con que alguien lo solicite, ya que el agente tiene libertad para abandonar su ejecución (dando por hecho que no existe ningún compromiso) o simplemente negarse a aceptar la solicitud.
    b. **Consecuencias de la ejecución de una tarea.** Al término de una tarea se esperan cambios en el estado mental del agente (como en Agent0 [Shoham 93]) o cambios en el estado del mundo (como en TAEMS [Decker 96]).
5. **Definición del contexto de la interacción.** El contexto consiste en detallar qué ocurre en el sistema cuando se inicia la interacción, mientras se desarrolla y a su conclusión. Esta información se proporciona, primero indicando qué actores participan, segundo qué motivos que los impulsan a hacerlo, tercero qué objetivos persigue la interacción y cuarto qué estados mentales se requieren de los agentes durante la ejecución de la interacción.
6. **Un modelo de control.** El control asegura que la interacción va a desarrollarse según fue definida. Este control debería tener en cuenta que en la mayoría de los casos se permite llevar varias interacciones en paralelo, i.e. se permiten diferentes conversaciones de los mismos protocolos (utilizando terminología de [Nowostawski, M., Purvis, M y Cranefield, S. 01]). Este control se tendría que orientar hacia la gestión de las actividades manejadas en la interacción, lo que terminaría siendo un mecanismo de coordinación [Malone y Crowston 94].

2.3.1.1 Interacciones en el análisis

Uno de los primeros resultados del análisis es la identificación de las interacciones que son relevantes a la funcionalidad del sistema. Aparecen cuando se consideran las relaciones entre las componentes del sistema, o al tratar de explicar como el sistema soporta la ejecución de algunas tareas.

Es importante capturar estos detalles utilizando las herramientas conceptuales existentes, como los diagramas de colaboración y secuencia [OMG 00d], diagramas de protocolo [Bauer, Müller y Odell 01], redes de transición [Demazeau 95], redes de Petri [Cost et al. 00], o diagramas de paso de mensaje [Iglesias et al. 98]. Como unidades de interacción se pueden manejar mensajes como en UML [OMG 00d], bien actos del habla como en FIPA [FIPA 02] o protocolos como en GAIA [Wooldridge, Jennings y Kinny 00].

En los enfoques actuales, siguiendo el trabajo de Kendall [Kendall 98] y la especificación de diagramas de colaboración de UML [OMG 00d], se resalta la importancia de la utilización de roles en la especificación de interacciones. En ambos trabajos se sugiere que la interacción se defina de forma genérica y que posterioriormente se detalle quién debe

jugar los roles indicados en la interacción. Otra influencia importante es la de FIPA en cuanto a la utilización de dos roles principales: el iniciador y los colaboradores. Sólo se admite un iniciador por interacción y al menos un colaborador.

Aunque la cantidad de información requerida para especificar una interacción pueda parecer demasiada, la mayoría de los aspectos mencionados pueden ser representados con un diagrama de colaboración UML. Tal diagrama dejaría sin especificar los puntos 4 y 6. Sin embargo, esto no debería suponer un problema, ya que el análisis no requiere tanto nivel de detalle.

2.3.1.2 Interacciones en el diseño

Disponer de toda la información necesaria acerca de las interacciones es una situación que se da raramente. Lo normal es comenzar con un boceto de la interacción y comenzar a pensar cómo las ideas del análisis pueden aparecer en el sistema final. Según se avanza hacia el diseño, la situación cambia. Se concretan componentes que sean capaces de soportar las interacciones al mismo tiempo que se aumenta el nivel de detalle en la especificación de las interacciones. El detalle se incrementa por lo general detallando cómo se ejecuta cada unidad de interaccion en la interacción. La ejecución se explicita describiendo qué información requieren y cómo se estructuran las distintas unidades de interacción.

Cuando se trata de mensajes, se definen el número y contenido de sus parámetros. Si se trata de llamadas a procedimiento, el número, tipo y valor de los argumentos de la llamada. En el caso de espacios de tuplas, como en Linda [Carriero y Gelernter 89] [Papadopoulos y Arbab 98], se detalla qué información se debe dejar en el espacio compartido y cómo esta información debe ser leída por los colaboradores.

En cuanto a la estructuración de unidades, se consideran UML, AUML y máquinas de estados. UML propone etiquetar los pasos de mensaje con *expresiones de secuencia* [OMG 00d] que denotan guardas, iteraciones, paralelismo o secuenciación. AUML, aboga por la utilización de notación específica para expresar estos aspectos y añade la encapsulación de los pasos de mensaje para dar lugar a protocolos reutilizables. Las máquinas de estados tienen implícita la secuenciación de unidades de interacción en su propia construcción.

En paralelo a este proceso de incremento de detalle, se tiene la selección y parametrización de las componentes que soportan la interacción. Cuando hay un lenguaje especializado, un armazón, o una herramienta para la generación automática de código, el proceso de generación de estas componentes es directo. Por ejemplo, los aspectos de interacción mencionados en la introducción son codificables casi directamente en Agent0 o ZEUS. Situaciones similares aparecen cuando las interacciones pertenecen a alguna categoría concreta. Por ejemplo, si el problema es categorizable como “ubicación de tareas” y necesita que se aplique negociación, existe la solución *contract-net* [Smith 80] y la representación computacional de FIPA, el *FIPA-contract-net* [FIPA 01] implementada en JADE [Bellifemine, Poggi y Rimassa 01]. Si el problema es categorizable como planificación, la planificación parcial global generalizada o GPGP (*Generalized Partial Global Planning*) puede usarse como especificación y como representación computacional JAF [Wagner y Horling 01] usando el armazón TAEMS [Decker 96].

Al margen de estas soluciones, se puede acudir a arquitecturas clásicas como la arquitectura de pizarra [Corkill 91] que soportan el paradigma de espacio de trabajo

compartido. Las variantes existentes de las arquitecturas de pizarra suelen asociarse con los lenguajes basados en Linda. En la práctica sin embargo, se haya más extendido el modelo establecido en FIPA [FIPA 01] que soporta paso asíncrono de mensajes ACL (*Agent Communication Language*) bajo diferentes tecnologías de comunicación. También es bastante común el encontrar arquitecturas de interacciones específicas utilizando paradigmas cliente-servidor directamente con CORBA [OMG 00a], DCOM [Microsoft 02] o RMI [Sun Microsistems 02]. Finalmente, si el diseñador desea desacoplamiento con la arquitectura de interacción, los métodos convencionales podrían proporcionar representaciones computacionales de alto nivel a través de los patrones de diseño *wrappers* y *proxies* [Gamma et al. 95].

2.3.1.3 Conclusiones

Los esfuerzos en el modelado de las interacciones pueden clasificarse en dos grupos: por un lado la especificación de alto nivel con diagramas de paso de mensajes (*Message Sequence Charts* [International Telecommunication Union 99 A.D.a]), diagramas de secuencia y colaboración (UML) y de protocolos (FIPA y AUML); y por otro lado, un diseño de más bajo nivel, sin llegar a la implementación, utilizando mecanismos de comunicación, como los espacios compartidos de tuplas, o de control, como redes de Petri[5]. En el caso de los primeros, existe el problema de cómo traducir estas especificaciones a elementos computacionales. En el caso de los segundos, el problema es el inverso, los mecanismos de control y comunicaciones son de demasiado bajo nivel como para expresar de forma sencilla situaciones complejas. Se puede decir que ambos enfoques son complementarios, sin embargo no es trivial pasar de uno a otro.

Hasta ahora, la notación para especificación de interacciones entre agentes más prometedoras son UML y AUML. Estas soluciones tienen el inconveniente de no contemplar todos los aspectos aquí mostrados. Estas técnicas proporcionan soluciones a la mayor parte de los aspectos expuestos en la sección 2.3.1 , dejando solo incompleta la parte de contextualización de las interacciones. La forma en que UML y AUML integran las interacciones en la especificación del sistema consiste en asociar las interacciones, expresadas como diagramas de secuencia y colaboración, a los casos de uso. Esta visión de las interacciones es limitada, ya que en un SMA haría falta hablar de por qué se inició la interacción, por qué decidieron participar agentes en ella y por qué los agentes envían mensajes y acceden a procesarlos. La respuesta a estas preguntas permite justificar la existencia a las interacciones más allá de la agregación a un caso de uso, definir de forma genérica el control del agente (un control adecuado del agente sería aquel capaz de responder estas preguntas) y validar la participación de agentes en la interacción (si las motivaciones de los agentes que participan en la interacción no están relacionadas, puede que no deban participar).

Persiguiendo una integración de técnicas de control y comunicación junto con el diseño de alto nivel de las interacciones, se propone una forma de ver las interacciones abierta a tanto nivel de detalle como requieran los mecanismos de control o comunicaciones y el grado de abstracción requerido por las labores de diseño. Para ello, se debe permitir que la

[5] Se habla de diseño de bajo nivel con redes de Petri, atendiendo a la capacidad de las redes de Petri a capturar mayor nivel de detalle que los diagramas de secuencia.

especificación de la interacción evolucione en paralelo con las tecnologías que la soportarán, partiendo de representaciones computacionales articuladas en torno a las necesidades del problema. En esta tesis se propone dividir el problema estudiando la interacción desde varios puntos de vista: el contexto, su naturaleza (como planificación o negociación), su ejecución (como estructura secuencial de los mensajes o estructurada con guardas o bucles) y representación de la interacción (como diagramas de protocolos o diagramas de secuencia).

2.3.2 Presentación del Meta-Modelo de Interacciones

El meta-modelo de interacciones propuesto busca simplificar:

- **La definición del contexto de la interacción.** Esto ayuda al ingeniero a saber qué se busca con la interacción y cómo va a afectar al sistema.

- **La generación de una especificación de comportamiento del sistema bottom-up / top-down.** Las interacciones no están aisladas. Es posible componerlas y relacionarlas para obtener especificaciones del comportamiento del sistema. Del mismo modo, también se puede partir de interacciones con el sistema y descender hasta las especificaciones de los componentes, que en este caso son agentes.

- **La definición de la ejecución de la interacción.** Ya sea utilizando paso de mensajes, protocolos FIPA o invocación de procedimiento remoto. De esta forma se logra independencia del medio de comunicación utilizado.

- **La representación de la interacción.** Al abstraerse de las posibles unidades de mensajes y de su ordenación, se pueden utilizar diferentes notaciones para representar lo mismo.

El meta-modelo de interacción 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) en las que hay un iniciador (emisor) y colaboradores (receptores). Además, se justifica la participación de los actores en la interacción y la existencia de la interacción en sí mediante objetivos.

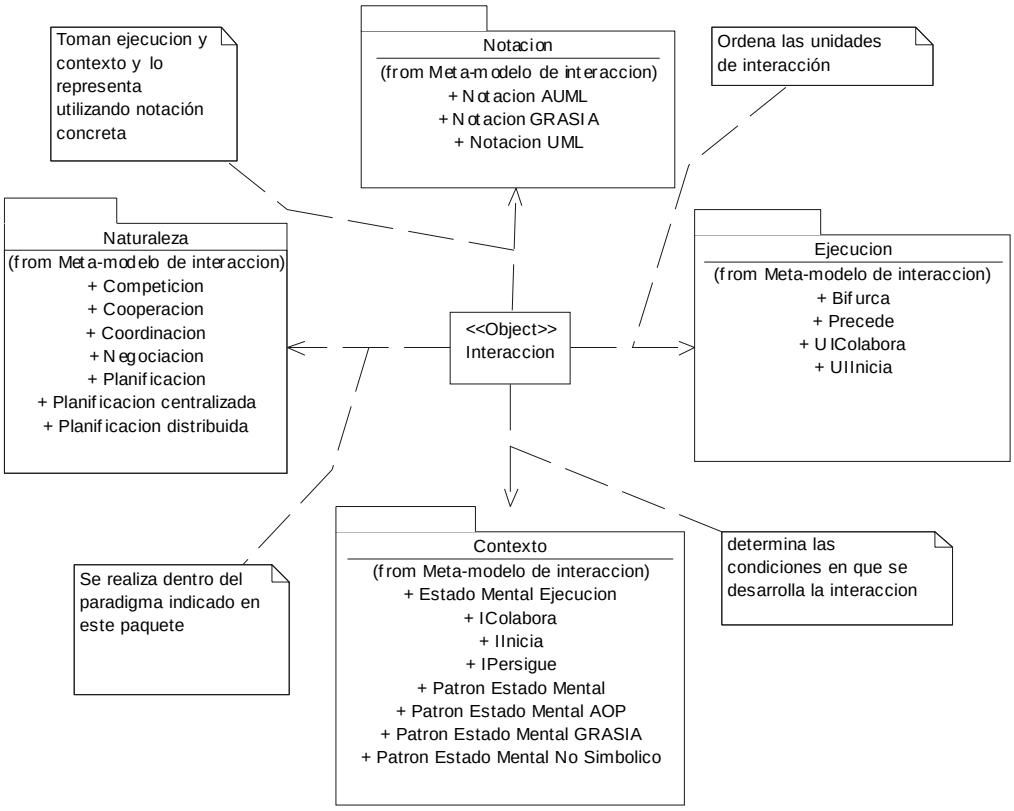


Ilustración 31. Relación entre los diferentes aspectos de la interacción

Estos elementos se interrelacionan mediante el conjunto de asociaciones de la Ilustración 31. En esta ilustración, se categorizan las asociaciones en los cuatro grupos ya vistos: naturaleza, contexto, ejecución y notación.

El *contexto* ayuda al ingeniero a situar la interacción en el marco del SMA, estableciendo en qué condiciones se iniciarán las interacciones. Esta información se utiliza para codificar el comportamiento del agente en forma de reglas de producción o para generar métodos de validación de la interacción, como métodos de detección de interbloqueos. En el metamodelo, el contexto se expresa con los estados mentales en ejecución que deben reunir los agentes y con las meta-relaciones *IPersigue*, *IColabora* y *IInicia*.

La *naturaleza* de la interacción indica qué clase de interacción se está considerando con respecto al tipo de control. Esta clase de control es de lo que trata la coordinación. La coordinación es la gestión de las dependencias entre actividades [Malone y Crowston 94] y se estudia dentro de la *teoría de la coordinación*. Esta teoría debería ser capaz de proporcionar taxonomías de coordinación, sin embargo, debido a su enfoque multidisciplinario, se ha preferido restringirse al ámbito de la informática tomando una taxonomía de Huhns [Huhns 00] (Ilustración 32).

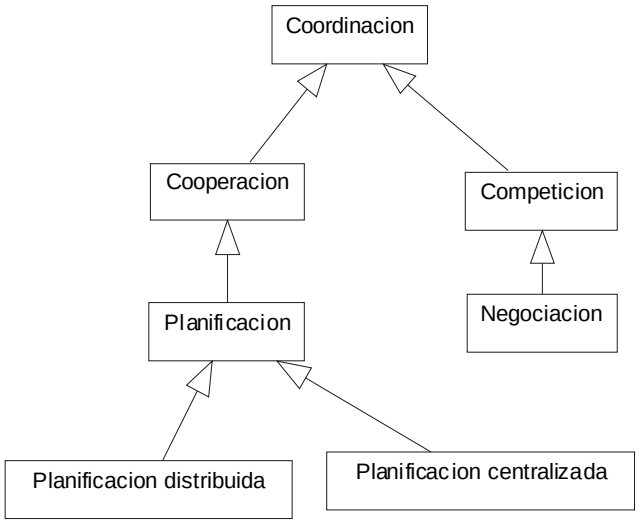


Ilustración 32. Taxonomía de métodos de coordinación entre agentes

La *naturaleza* de la interacción es importante porque determina qué algoritmos, herramientas y guías pueden ayudar al diseñador. Si la naturaleza es planificación, quizá el diseñador debería evaluar la adecuación de GPGP [Decker y Lesser 95] (ver [Durfee, Lesser y Corkill 89] para otras soluciones). El tipo de interacción también muestra qué elementos tienen que considerarse. Por ejemplo, si la naturaleza es competitiva entonces debería existir en la interacción una referencia al objeto por cuya posesión o uso se compite.

La *ejecución* de la interacción se refiere al conjunto de actividades requeridas a la hora de desarrollar la interacción. En la ejecución existe un orden impuesto sobre las acciones y *unidades de interacción*. El orden establece el protocolo de la interacción, mientras que las unidades representan mensajes o protocolos, dependiendo del nivel de abstracción.

Finalmente, la *representación* toma la información de la *ejecución* y la *naturaleza* para crear una representación gráfica o formalismo textual sencillo de manejar por los ingenieros. Por ejemplo, una negociación que siga *contract-net* [Smith 80] puede expresarse con diagramas de protocolo de Agent UML , diagrama de protocolos FIPA o varios diagramas de secuencia UML.

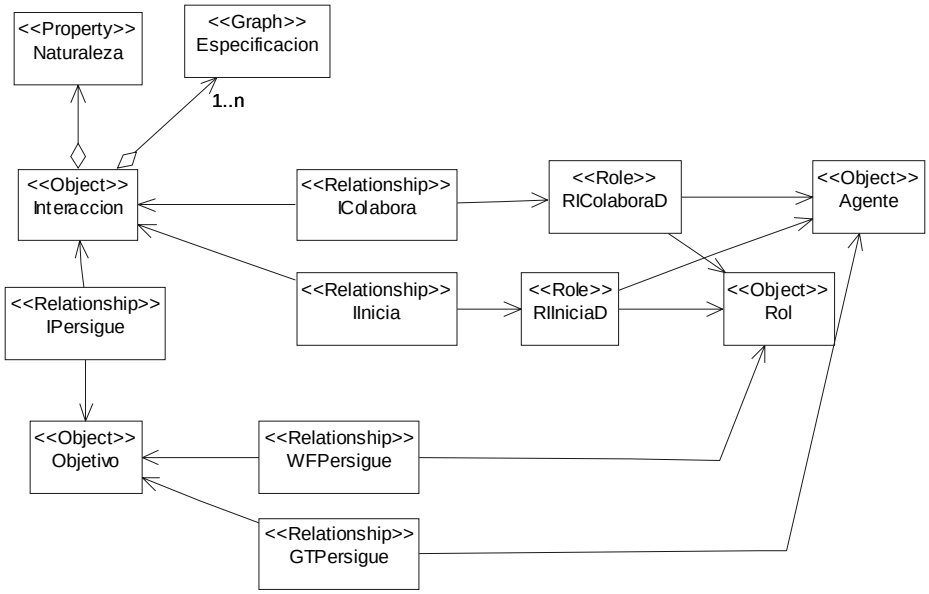


Ilustración 33. Meta-modelo de interacción

Las interacciones en este trabajo se definen con el meta-modelo de la Ilustración 33. Este meta-modelo cubre los aspectos mencionados referidos en la Ilustración 31: contexto, naturaleza, ejecución y representación. El contexto aparece como la motivación de la interacción (meta-relación *IPersigue*) y de los roles en el momento de participar en la interacción (meta-relaciones *WFPersigue* y *GTPersigue*). La naturaleza se asocia directamente como propiedad de la interacción. La ejecución y representación han sido agrupadas dentro la entidad *Especificación* que es del tipo *Graph*. Según GOPRR, esto implica que *Especificación* es un conjunto de *relaciones*, *objetos* y *roles*. Esta abstracción se usa para denotar las posibles formas de especificar la ejecución y tipo de las *unidades de interacción*. En este trabajo se ha experimentado con dos tipos de especificación:

- **Diagramas de colaboración UML.** En estos diagramas, las unidades de interacción son *mensajes* y la asociación entre emisor y receptor son las relaciones *UIInicia* e *UIColabora*. Las tareas pueden agruparse al igual que los agentes o roles, pudiendo recibir mensajes. El orden de ejecución así como las condiciones mentales se expresan con una *Expresión de secuencia* (ver [OMG 00d]).
- **Diagramas especializados GRASIA.** Los diagramas de colaboración UML no están pensados para el modelado de interacciones entre agentes. Así, la justificación de por qué se está ejecutando la interacción, detalles acerca de por qué se están aceptando ciertos mensajes y por qué transcurre la interacción de un modo concreto, no son fácilmente expresables. Por ello, se ha generado una *especificación GRASIA* adaptada a su uso dentro de la metodología (ver Ilustración 34). Esta última variante de especificación da cabida a la representación del estado mental del iniciador y de un colaborador en una unidad de interacción.

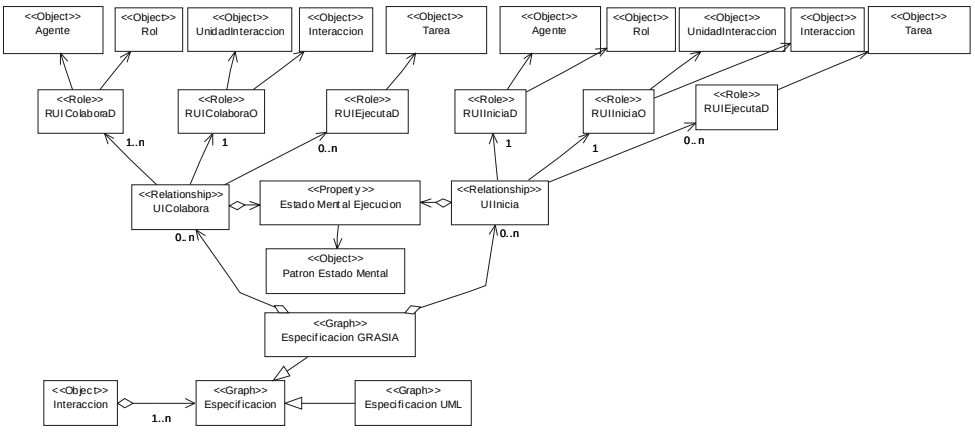


Ilustración 34. Unidades de interacción y su relación con tareas y roles dentro de una especificación.

Las unidades de interacción se detallan en la Ilustración 34. La composición de unidades de interacción siguiendo un orden determinado lleva a la especificación de la ejecución de la interacción. De forma similar a las interacciones, una *unidad de interacción* también describe quién la inicia (*UIInicia*), quién colabora (*UIColabora*) y qué tareas se ejecutan en cada una (*RUIEjecutaD*). Aunque no se menciona explícitamente, hay que entender que en una especificación GRASIA, también se tienen *tareas*, *agentes*, *roles*, *interacciones* y *unidades de interacción*.

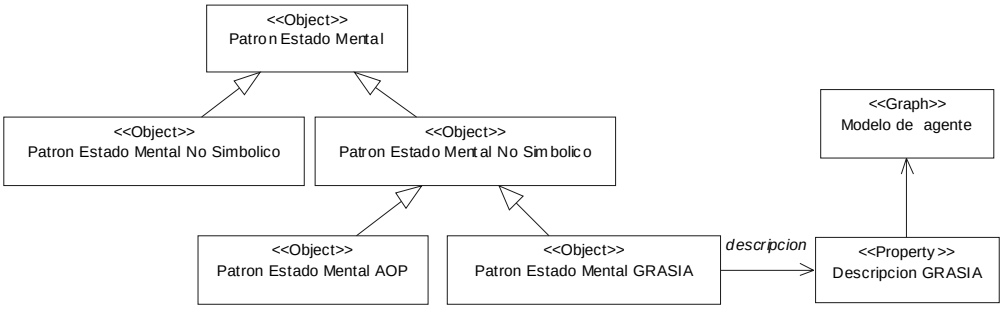


Ilustración 35. Patrones de estado mental asociables con las unidades de interacción

Los *patrones de estado mental* (ver Ilustración 35) se usan para expresar condiciones de comienzo y colaboración en términos de entidades de estado mental. La ejecución de tareas no tiene un *patrón de estado mental* asociado porque la condición implícita de ejecución de una tarea es la asociada a la meta-relación *UIInicia* o *UIColabora*. Así, cuando un agente comienza una unidad de interacción, también ejecuta las tareas asociadas mediante *RUIEjecutaD*.

Las condiciones que más se han estudiado en este trabajo son las expresables con la gramática expuesta en Agent0 [Shoham 93] (*Patron Estado Mental AOP*) y las expresables

como agregación de entidades mentales (*Patron Estado Mental GRASIA*), ya que su traducción a reglas de producción es sencilla. En ambas variantes, aparece una descripción de cómo se espera que sea el estado mental del agente en un momento concreto. En el caso de la entidad *Patron Estado Mental AOP* la descripción debe ser un término <*mntlpattern*> cuya construcción se expresa en BNF en el Anexo II. En el caso de *Patron Estado Mental GRASIA*, la descripción es un modelo de agente (ver sección 2.2) donde se indica qué entidades deben existir asociadas al estado mental del agente y qué condiciones se deben satisfacer. Cada entidad mental se etiqueta, en caso necesario, para referenciarlas cuando se exprese qué se requiere de estas entidades.

Finalmente, existe una meta-representación de los posibles órdenes que pueden aplicarse sobre las *unidades de interacción* (ver Ilustración 36). Los diseñadores pueden elegir el tipo de orden más apropiado para el problema que les ocupe, definir una instancia de dicho orden y dejar que el generador de código produzca un control de la comunicación adecuado. De esta forma, es factible definir un orden como el de los *behaviour* de JADE [Bellifemine, Poggi y Rimassa 01] usando las primitivas de comportamiento y ejecutando una traducción directa desde la especificación a las estructuras finales de JADE.

Para representar los posibles órdenes de ejecución de JADE, UML y AUML, se utilizan las guardas definidas en las instancias de *Estado Mental Ejecución* en la Ilustración 34 y tres primitivas: orden secuencial (*Precede*), orden no determinista (*UIConcurren*), selección (*bifurca*) e iteración (*UIIteracion*)

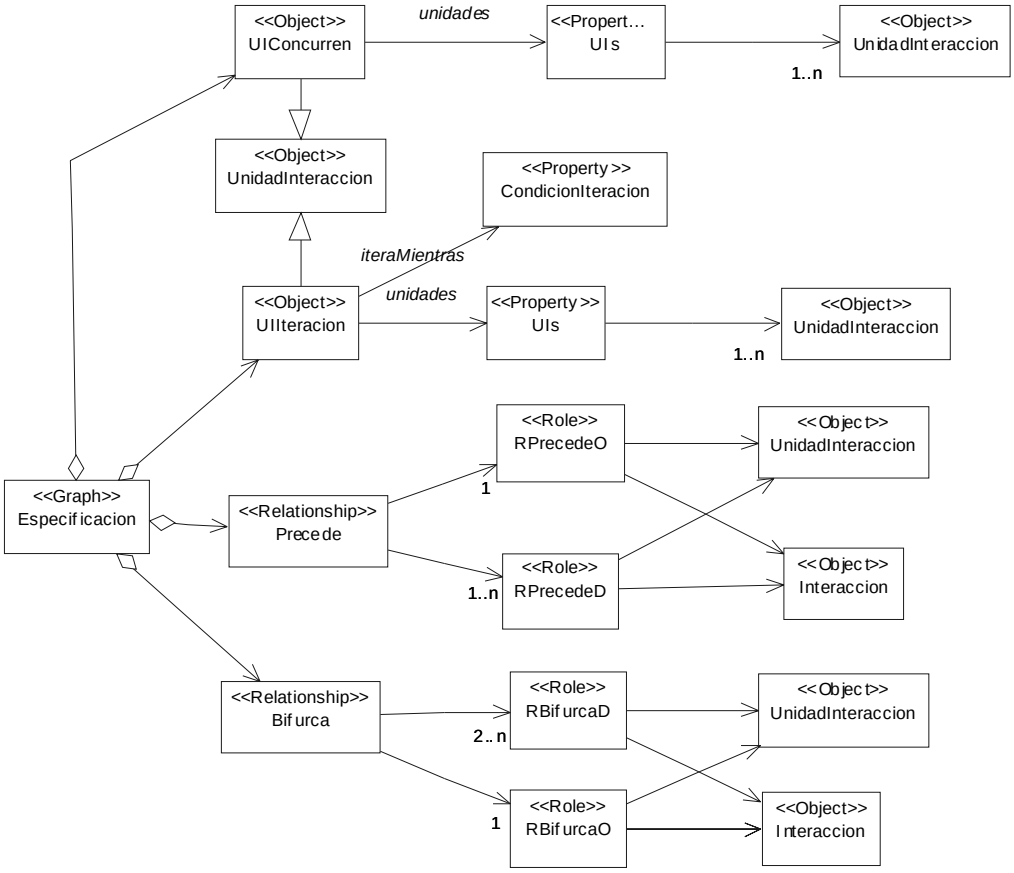


Ilustración 36. Ordenes de ejecución utilizables sobre las unidades de interacción

La meta-relación *Precede* define secuencias de unidades de interacción. Siendo *A* y $B_1,...,B_n$ unidades de interacción, la asociación A — *Precede* — $(B_1,...,B_n)$ significa que la unidad *A* tiene lugar siempre antes que la unidad B_i. La meta-relación *Bifurca* sirve para expresar condiciones de precedencia entre varias unidades de interacción. Siendo *A* y $B_1,...,B_n$ unidades de interacción, la asociación A — *Bifurca* — $(B_1,...,B_n)$ significa que sólo un B_i tendrá lugar después de A. Cuál será depende de las condiciones de colaboración o iniciación dadas por *Estado Mental Ejecución* en la Ilustración 34, ya que para que tenga lugar una unidad de interacción se han de cumplir las condiciones de colaboración e iniciación asociadas. En caso de que varias se cumplan, se elegirá una de forma no determinista. La entidad *UIIteracion* define la ejecución repetida de un conjunto de unidades de interacción. Estas entidades aparecen asociadas como una colección (*UIs*). El orden de ejecución dentro del bucle se determina aparte con las otras primitivas. La condición de terminación es que ninguna unidad de interacción de las contenidas sea ejecutable o bien que se cumplan las condiciones de inicio y colaboración de la unidad de interacción que se vaya a ejecutar a continuación. La entidad *Concurren* especifica un orden

de ejecución no determinista sobre las unidades de interacción asociadas. La unidad de interacción *Concurren* se da por terminada cuando todas las unidades de interacción que tiene asociadas hayan tenido lugar.

Instanciando el meta-modelo de interacción es determina cómo reaccionan los agentes frente a las acciones de otros agentes. La composición de estas instancias da lugar a la definición detallada del comportamiento del sistema. Esta composición se realiza en el marco de la organización y más concretamente dentro de los flujos de trabajo.

Para cada una de las instancias se define un contexto concreto, que es el fijado por el flujo de trabajo de las organizaciones. Al final, las interacciones entre agentes se inician porque se han ejecutado tareas que satisfacen un objetivo en la organización u organizaciones a las que pertenecen los participantes de cada interacción.

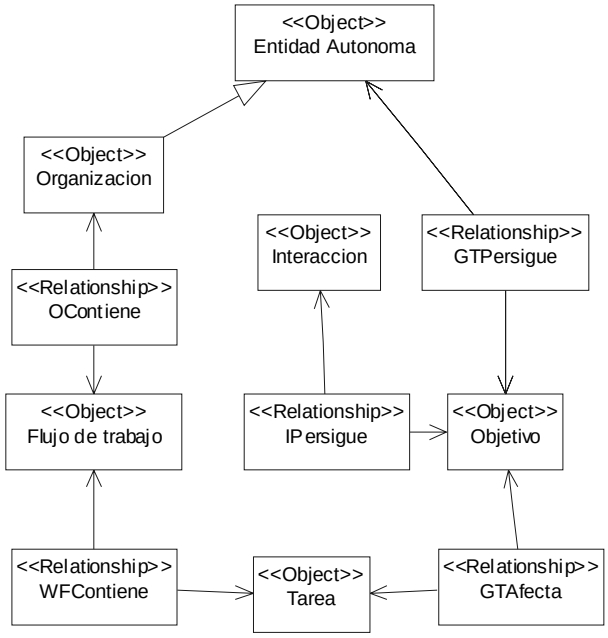


Ilustración 37. Relaciones entre la interacción y la organización

La Ilustración 37 muestra las dependencias entre la interacción y la organización. El nexo entre interacciones y organización es el objetivo perseguido por ambas. Como las tareas, la organización se vale de interacciones entre sus miembros para alcanzar sus objetivos.

2.3.3 Ejemplo: agente de bolsa

Se quiere diseñar un asistente para vender y comprar acciones en la Bolsa de Madrid. El asistente representa a un usuario con el que confirma las transacciones las acciones vía móvil. Existe la posibilidad de que el agente desarrolle su labor sin intervención del

usuario. Esta situación se da cuando el usuario ha configurado un margen de variación en el precio de las acciones que dispara la compra (en caso de que baje las acciones que interesen al usuario) y la venta (en caso de que suban las acciones del usuario). Para realizar las operaciones de compra y venta, la Camara de Comercio ha habilitado servidores con los que se puede obtener información en tiempo real del valor de las acciones.

Buscando influir sobre el precio de las acciones en bolsa, el asistente acude a directorios de localización de agentes para localizar a otros agentes de compra-venta de acciones con los que aliarse. Una vez localizados, nuestro asistente plantea alianzas estratégicas con las que hacer variar la bolsa.

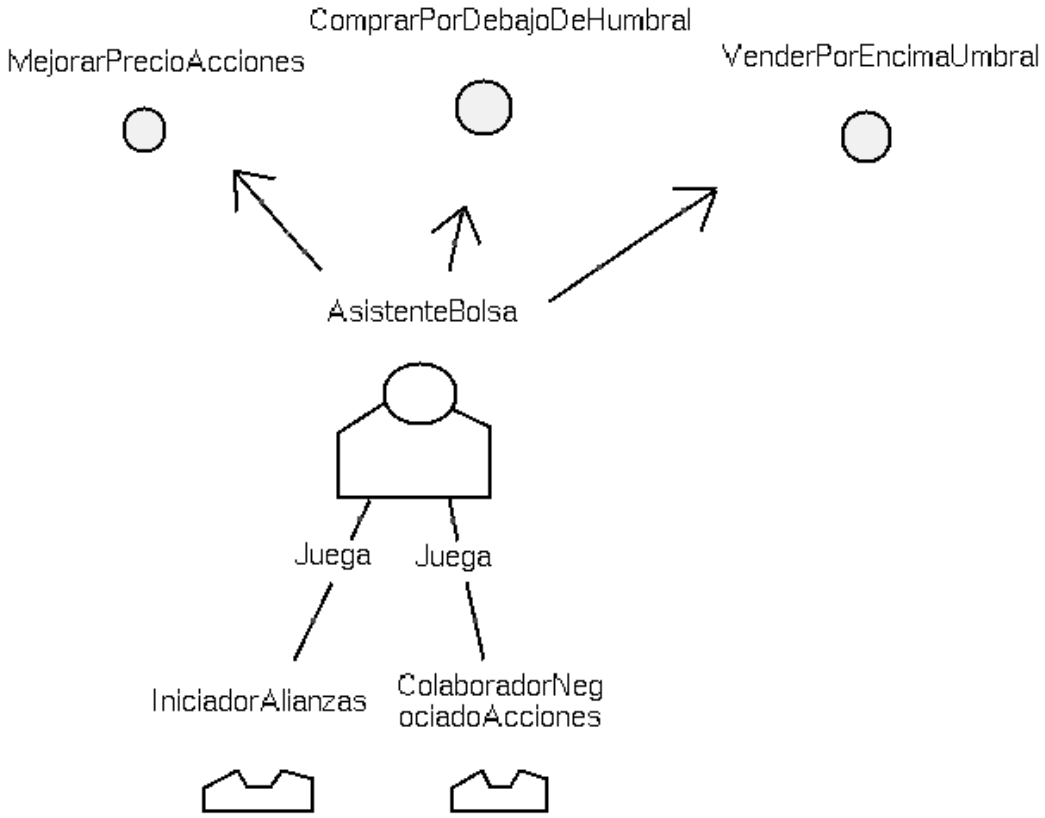


Ilustración 38. Definición del asistente de bolsa (modelo de agente)

Las alianzas se plantean mediante negociaciones con representantes de agentes ya aliados. En el caso de que no estén aliados, el agente con el que se contacta actúa como su propio representante. Tras asociarse con otro agente o agentes, hay que darse de alta en un registro de asociaciones, para que otros agentes puedan unirse a esta alianza. El protocolo de creación de alianzas se representa mediante una interacción *FormarAlianza*.

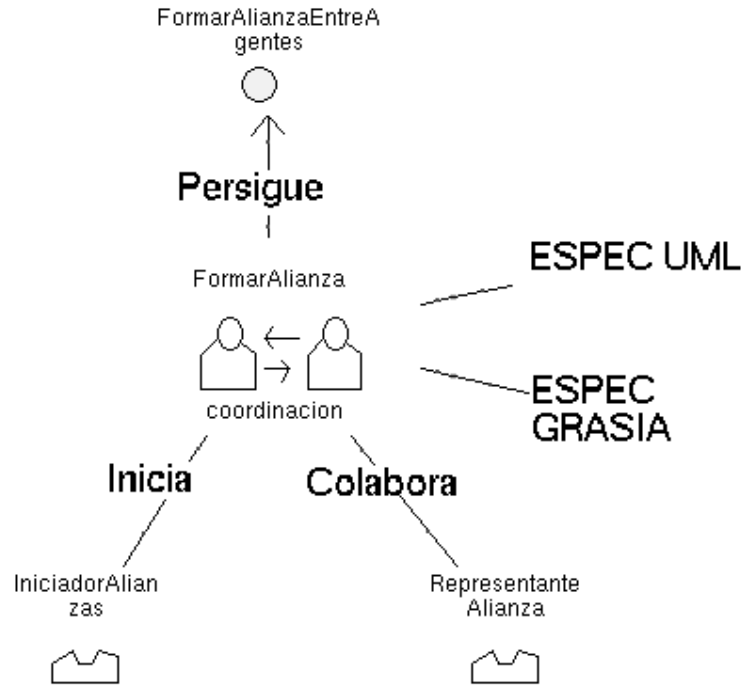


Ilustración 39. Interacción para la formación de alianzas

FormarAlianza persigue el único objetivo de formar una alianza entre agentes para conseguir mover las acciones de bolsa. La descripción del protocolo se describe inicialmente con un diagrama de colaboración (ver Ilustración 40).

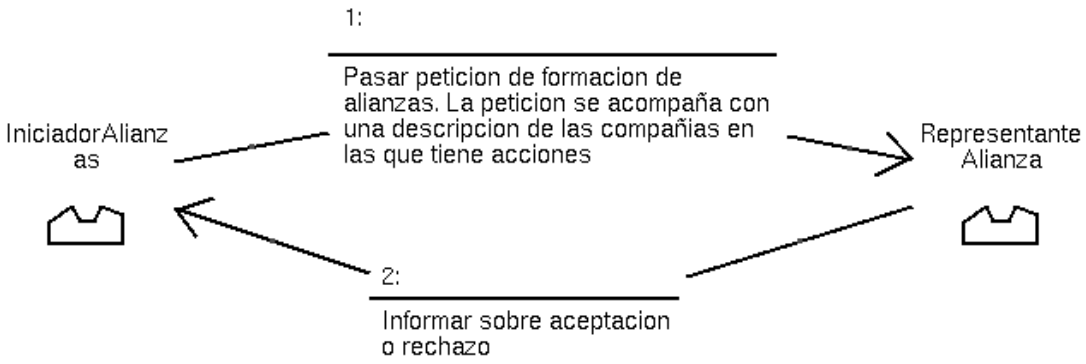


Ilustración 40. Descripción inicial de la interacción para formar alianzas con diagramas de colaboración

La descripción con diagramas de colaboración no reflejan, por ejemplo, el acto del habla involucrado en cada paso de mensaje. A la hora de dar más detalle, se elige una especificación GRASIA para representar la misma interacción (ver Ilustración 41).

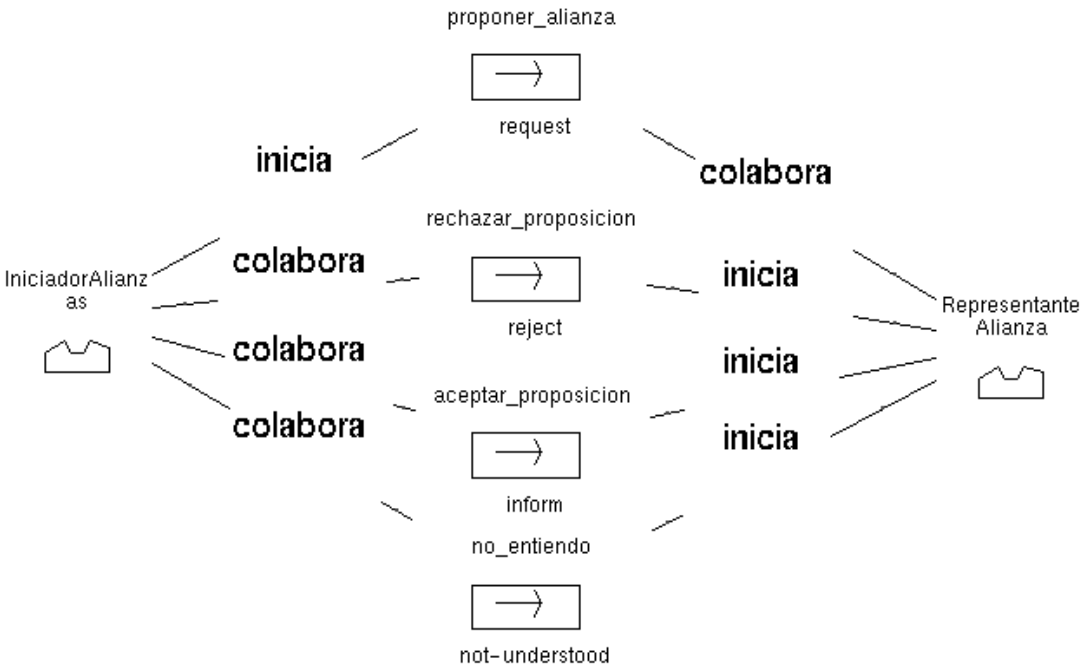


Ilustración 41. Boceto de descripción GRASIA de la interacción para formar alianzas

Con esta especificación se añaden las posibilidades de que la proposición sea rechazada, aceptada o simplemente no comprendida. Esta última posibilidad tiene que tenerse en cuenta en situaciones en que el software que interacciona es heterogéneo, como aquí. El *asistente de bolsa* no sabe quién ha hecho el otro agente ni que interfaces soporta. Se asume que existe una ontología compartida por los participantes en la que se definen un conjunto de contenidos válidos de mensajes. Al término de la negociación, el solicitante recibe un certificado personalizado que servirá para certificar futuras peticiones de la alianza.

Una vez se pertenece a la alianza, se plantean negociaciones en las que los agentes que pertenecen a la alianza acuerdan nuevos parámetros de compra-venta de acciones para lograr efectos significativos en bolsa. La negociación se hace mediante un protocolo FIPA contract-net [FIPA 01]. Este protocolo se encapsula dentro de otra interacción, *NegociacionMovimientoBolsa*.

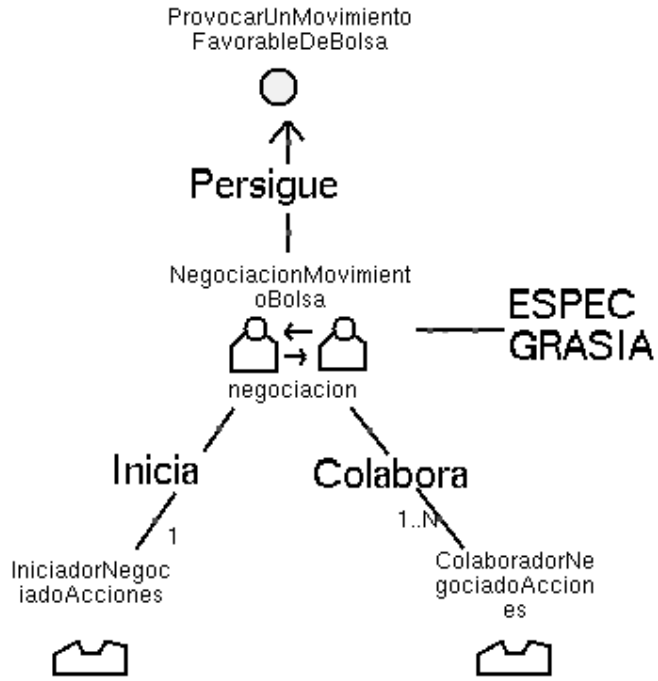


Ilustración 42. Interacción para conseguir movimientos en bolsa

Se definen las acciones sobre las que se va a actuar, su número y su valor. Después se acuerda un momento de actuación. Si llegado el momento se satisfacen los parámetros de actuación acordados en la negociación, entonces los agentes proceden a realizar las transacciones pertinentes con la Bolsa de Madrid.

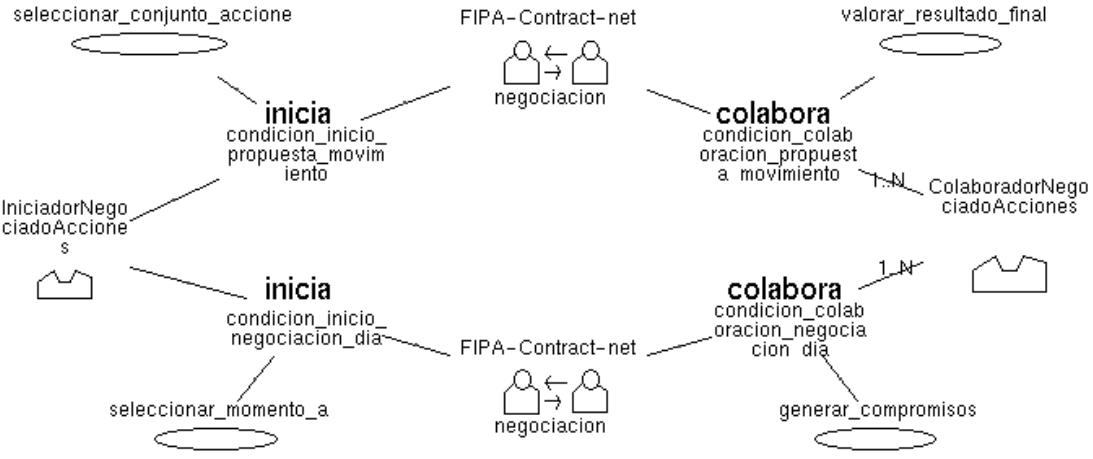


Ilustración 43. Descripción detallada de unidades de interacción involucradas en el acuerdo de acciones a realizar en la alianza

Para que el sistema funcione debe haber un número crítico de de agentes dentro de la alianza. Por ello se define que el iniciador de la negociación no comience hasta que se alcance un número crítico y se tenga la creencia de que se puede alcanzar el objetivo *Mejorar Precio Acciones*.

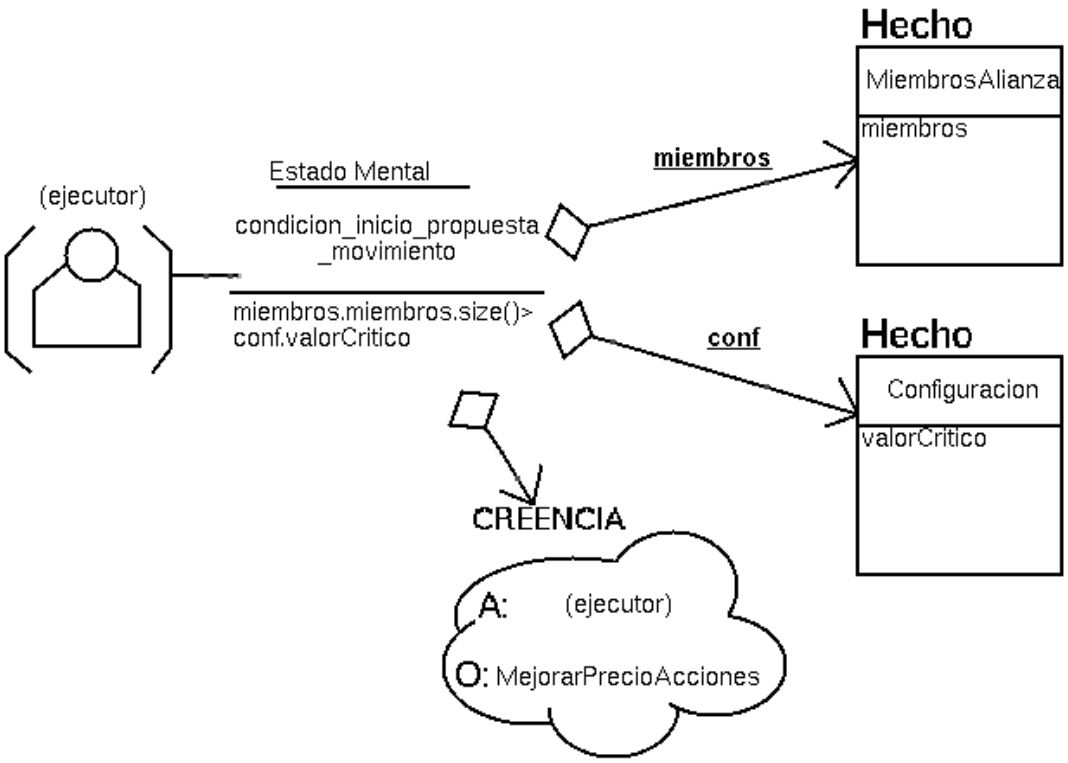


Ilustración 44. Condición mental contenida en *Condicion inicio propuesta movimiento*

La creencia antes mencionada sería el fruto de una tarea de análisis del estado de la bolsa. El colaborador en la interacción *NegociaciónMovimientoBolsa* participa en las negociaciones en virtud del compromiso adquirido al formar una alianza (ver Ilustración 45).

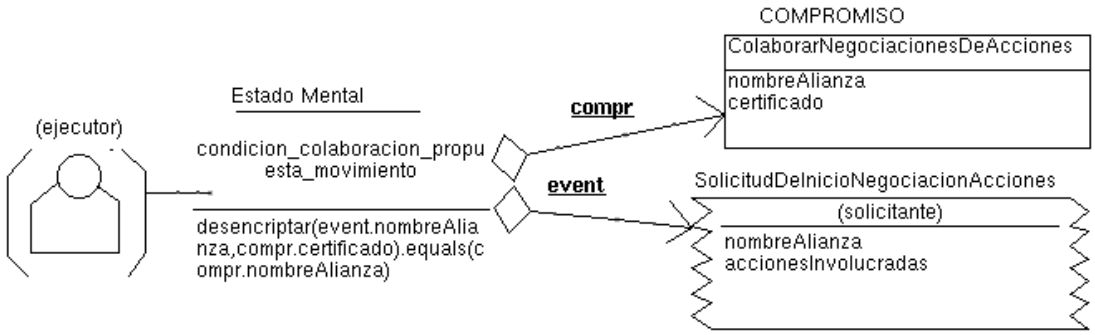


Ilustración 45. Estado mental requerido por un colaborador al inicio de las negociaciones

Para participar en una negociación para el movimiento de acciones, tiene que existir el compromiso generado en la interacción *FormarAlianza*. Este compromiso lleva implícito un certificado con el cual se puede verificar si una solicitud de participación es válida o no.

Al final de la interacción, en los agentes participantes se crea un compromiso de determinado día a determinada hora, si se cumplen las condiciones negociadas con anterioridad (umbral de beneficio dentro de los parámetros negociados y de lo configurado por el usuario), se realiza la venta o compra.

2.3.4 Integración con otros meta-modelos

El meta-modelo de interacción constituye una fuente de información importante para determinar cómo ha de ser el control del agente. Parte del comportamiento del agente se puede ver a través del conjunto de interacciones que definen cómo reacciona el agente ante peticiones de información de otros agentes. Mediante las interacciones, además, puede deducirse cómo se modifica el estado mental de los agentes que participan, ya que las tareas ejecutadas durante la interacción son en parte responsables de los cambios internos en los agentes.

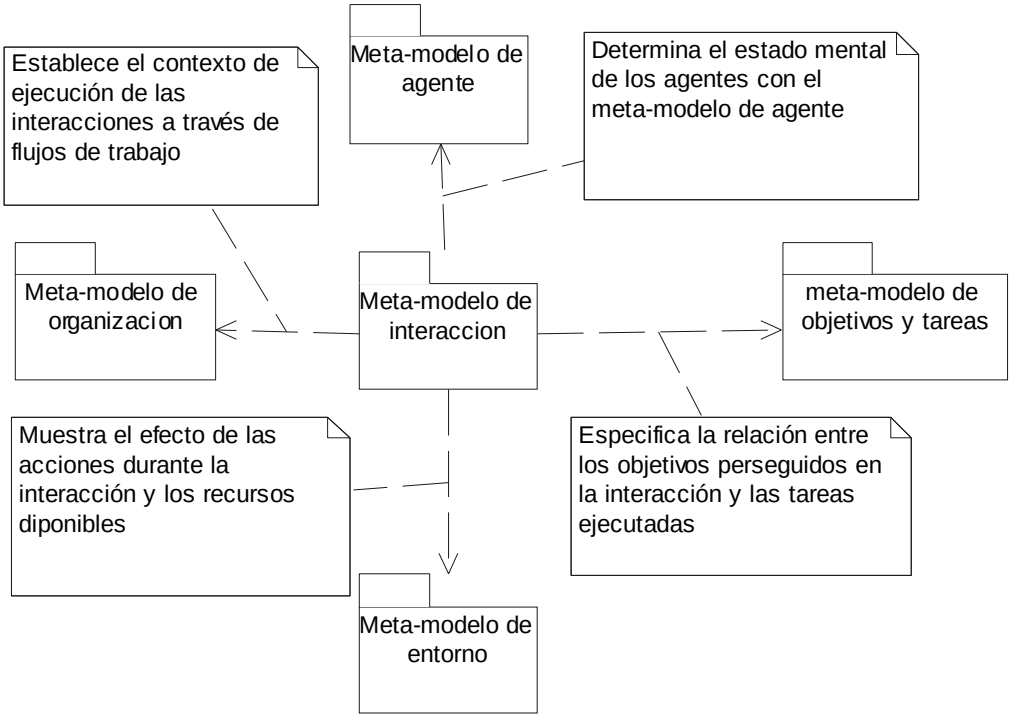


Ilustración 46. Relación del meta-modelo de interacción con otros meta-modelos

Las tareas ejecutadas en una interacción generalmente hacen uso de los recursos que configuran el entorno. La interacción, como en el caso del comportamiento del agente, también identifica los momentos en que se pueden disparar cambios en el entorno. Además, existe la posibilidad de que la propia interacción tome como medio de comunicación el propio entorno. En tal caso, el papel de los elementos recogidos en el meta-modelo de entorno es clave para reflejar qué cambios en el entorno son entendidos por otros agentes como actos de comunicación.

Finalmente, una interacción se asocia con las metas de la organización a la que pertenecen sus participantes. El contexto de la interacción se establece a través de *objetivos* y *patrones mentales* [Shoham 93]. Los objetivos se asocian directamente a las interacciones a través de la meta-relación *IPersigue* e indirectamente con las metas asociadas con la organización.

Basándose en estas relaciones, se plantean los siguientes criterios de consistencia del modelo de interacción:

1) Los objetivos perseguidos por la interacción (instancias de *IPersigue*) han de pertenecer a la organización o bien estar relacionados con estos por instancias de *GTDescompone* en modelos de objetivos y tareas. La relación entre los objetivos de la organización y los de la interacción se muestran en la Ilustración 37. Una organización de un SMA es una entidad con identidad y propósito que agrupa juntos

agentes, roles, recursos y tareas [Garijo, Gomez-Sanz y Massonet 01]. El propósito de la organización se expresa a través de objetivos que indican el motivo por el cual todos los elementos anteriores están juntos. De acuerdo con el modelo BDI [Kinny y Georgeff 97] y el diagrama de la Ilustración 37, los objetivos de una interacción se hallan íntimamente relacionos con los de la organización. Si los objetivos de la interacción son alcanzados, entonces los objetivos de la organización se hayan más próximos a su satisfacción.

2) Todos los participantes en la interacción persiguen objetivos relacionados con los de la interacción. Un criterio para determinar si dos objetivos están relacionados sería que existieran entre ellos relaciones de descomposición dentro de un modelo de tareas y objetivos (instancias de la meta-relación *GTDescompone*) o bien por igualdad de los mismos. Otro criterio posible sería determinar si las tareas ejecutadas para satisfacer uno de ellos proporcionan evidencias que ayuden a satisfacer los otros.
3) Las tareas ejecutadas a lo largo de la interacción permiten satisfacer los objetivos de la interacción. Para verificarlo, se utilizan las relaciones *GTSatisface* del modelo de objetivos y tareas y la descomposición de objetivos (*GTDescompone*).
4) Las interacciónes plantean estados mentales en los agentes que se pueden expresar con modelos de agente. Sería necesario comprobar si estos estados mentales son consistentes con las acciones que los producen. El estado mental del agente surge como resultado de la actuación de su *procesador de estado mental*, de su *gestor de estado mental*, y sobre todo de las tareas ejecutadas. Así, debiera justificarse en los modelos de agente de dónde vienen las entidades mentales referidas a lo largo de la interacción y si realmente es posible que tal estado mental se dé.

2.4 Meta-modelo de Objetivos y Tareas

El meta-modelo de objetivos y tareas tiene como propósito recoger las motivaciones del SMA, definir las acciones identificadas en los modelos de organización, interacciones o de agentes y cómo afectan estas acciones a sus responsables. Esta información constituye parte de la especificación de cómo se quiere que sea el control del agente a alto nivel. El meta-modelo de agente habla de Procesadores y Gestores de Estado Mental que establecen el proceso deliberativo y de mantenimiento del conocimiento del agente. El meta-modelo de interacción de los estados mentales requeridos por parte de un conjunto de agentes que interactúen. Aquí se trata de poder expresar cuáles son las consecuencias de ejecutar las tareas y de por qué se deberían llegar a ejecutar. Una vez proporcionada esta información, son las entidades anteriores las que determinan, dado un conjunto de tareas a ejecutar, cuál se elige.

La justificación del uso de objetivos como representación del control del agente, se encuentra en el *principio de racionalidad* ya enunciado con anterioridad. La asociación de los objetivos con las tareas es algo que surge en los comienzos de la planificación y cuyo uso ha sido ratificado con modelos teóricos como el BDI. Según el modelo BDI, los seres humanos tienden a enfocar sus acciones siguiendo una pauta concreta: cuál es la situación actual (creencias), qué es lo que se quiere (objetivos) y cómo se puede llegar a alcanzar (intenciones). Siguiendo también el *principio de racionalidad*, las acciones del agente se

justifican por los objetivos que persigue, lo cual lleva también a la asociación de tareas y objetivos. Finalmente, la inclusión de las tareas en este meta-modelo obedece a que, además de asociarse con objetivos, juegan un papel fundamental en la evolución del estado mental de sus responsables. El por qué aparece esta información aquí y no en el meta-modelo de agente o de organización se debe a que el ámbito de estos meta-modelos no es el de relacionar tareas con objetivos. El meta-modelo de organización fija el marco de ejecución de las tareas mientras que el meta-modelo de agente sólo informa de las características de un agente concreto.

Para desarrollar los contenidos de este meta-modelo, primero se estudia cómo son las tareas y objetivos en otras metodologías y herramientas de desarrollo de SMA, después se presenta el meta-modelo que recoge las ideas más relevantes de los trabajos estudiados, a continuación una guía de generación de modelos de tareas y objetivos. Para terminar, se presentan ejemplos de modelos de tareas y objetivos.

2.4.1 Tareas en los Sistemas Multi-Agente

El concepto *tarea* aparece repetidas veces en la literatura de agentes. En [Ferber 99] aparece un resumen de diferentes interpretaciones de qué es una tarea. Una tarea se puede ver como: transformaciones del estado global, respuestas a eventos, un proceso, una acción física o un comando.

De estas acepciones, se han elegido dos, que son complementarias: tarea como transformadora del estado global y tarea como proceso. La primera es útil porque concibe la tarea como pre-condiciones y post-condiciones (el cómo son estas pre y post-condiciones se discutirá más tarde). Esto permite su incorporación en mecanismos de planificación y razonamiento. La segunda es más pragmática y acorde con la realidad final: que la tarea será un conjunto de instrucciones que han de ejecutarse. En concreto, interesa la interpretación de OMG [OMG 00c], que ve la tarea como proceso integrado en un flujo de trabajo, planteamiento similar al que se adopta en el meta-modelo de organización.

En cuanto a la integración de tareas en los SMAs, se ha estudiado TAEMS y ZEUS. De TAEMS [Decker 96;Decker y Lesser 95;Decker 95] se ha tenido en cuenta la forma de dividir tareas, funciones de coste asociadas a tareas, influencias entre tareas y la integración de recursos como bienes necesarios para la ejecución de tareas. De ZEUS [Nwana et al. 99] se toma el modelo de ejecución de las tareas. En este modelo, las tareas se comienzan cuando se satisfacen sus precondiciones expresadas en términos del conocimiento del agente (hechos). Se incluye, como en Wooldridge [Wooldridge 92], la posibilidad de contabilizar como acción la actualización del conocimiento, que consiste en la adición o sustracción de uno o más hechos. Otra idea tomada de ZEUS es la asociación de atributos, concretamente coste y duración, a la tarea. Como en todos estos enfoques, cada tarea se asigna a un agente. Este agente tiene la responsabilidad de comenzar, controlar y ejecutar la tarea según los términos establecidos por el diseñador.

Se han dejado aparte otros desarrollos que podrían verse como relevantes. Agent Tool [DeLoach 01;Wood y DeLoach 00] se ha descartado por la forma de detallar el comportamiento de las tareas, similar al SDL [International Telecommunication Union 99 A.D.b] y por lo tanto poco adecuado para el diseño de agentes cognitivos. La metodología GAIA [Wooldridge, Jennings y Kinny 00] también ha sido dejada a un lado por concebir la

tarea como precondiciones y postcondiciones expresadas con fórmulas lógicas. Aunque este enfoque está de acuerdo con las tendencias convencionales de programación, plantea serios problemas a la hora de comprender su funcionamiento y de conseguir generación automática de código. Precondiciones y postcondiciones como fórmulas lógicas (como las que se muestran en GAIA), no son válidas si no se establece y formaliza un dominio para las mismas. Y formalizar el dominio, si bien es deseable, es muy costoso.

Sin querer prescindir de estas ideas, antes de hablar de precondiciones como en GAIA, se ha preferido utilizar precondiciones como en ZEUS, donde éstas se identifican con la parte izquierda de una regla y cuyos términos son entidades de conocimiento recogidas en una ontología. Al considerar las postcondiciones, debido a la integración de las tareas en los flujos de trabajo, se ha tomado la decisión de modelarlas como la producción o modificación de entidades mentales y la utilización de elementos del entorno, como en ZEUS. De esta forma, sólo quedaría por determinar cuál es la postcondición de la utilización de un elemento del entorno, pero ello se enmarca ya en la ingeniería convencional, como se verá en el meta-modelo de entorno.

Para simplificar el meta-modelo, se ha optado por restringir lo que se puede hacer en una tarea omitiendo el estado de la tarea. Así, la tarea existe como proceso durante un tiempo finito[6]. En caso de necesitar conservar el estado de una ejecución a otra, éste se puede expresar a través de entidades mentales.

2.4.2 Objetivos en los Sistemas Multi-Agente

En este trabajo, los objetivos se emplean para razonar acerca de las posibles alternativas que se le presentan a un agente en un momento dado. Para representar estas alternativas, existen dos tendencias:

- **Objetivos como agregación.** Este es el enfoque predominante en planificación clásica (STRIPS [Fikes y Nilsson 71]). El objetivo se ve como una descripción del estado del mundo a alcanzar. Cuando la descripción del estado del mundo se hace con un conjunto de predicados, el objetivo se convierte en una agregación de estos predicados.

- **Objetivos como entes.** Este enfoque se usa en el modelo BDI y en los paradigmas lógicos descritos en el primer capítulo. No se trata de agregaciones de predicados, como antes, sino de entes autorepresentativos. Un objetivo, según [Ferber 99], es una unidad mental resultado de impulsos (del propio agente) y solicitudes, de otros agentes. Según el modelo BDI, se trata de un deseo que se

[6] Las tareas de duración infinita no se incluyen como tales porque no son convenientes desde el punto de vista del control del agente expuesto en este trabajo. Como se verá más adelante, la ejecución de tareas suele requerir recursos. Si una tarea toma un recurso y no lo libera, como podría suceder con una tarea de duración infinita, se correrían el riesgo de tener un interbloqueo de tareas. La situación, no obstante es salvable extrayendo el estado de la tarea y depositándolo en el estado mental del agente. Así el control del agente sería responsable de ejecutar indefinidamente la tarea sin peligro de interbloqueos.

quiere satisfacer. El *principio de racionalidad* ve los objetivos como guía y justificante de las acciones del agente.

Aunque el término objetivo (*goal*) tiene el sentido descrito en la mayoría de los casos, existe otra interpretación sutilmente diferente: **objetivos como requisitos**. Algunas metodologías como *Kaos* [Bradshaw 96], TROPOS [Bresciani et al. 01] o *MaSE* [DeLoach 01;DeLoach, Wood y Sparkman 01;Wood y DeLoach 00] utilizan este planteamiento para reflejar en el diseño los requisitos que debe satisfacer el sistema.

En este trabajo, los objetivos se toman inicialmente como entes autorepresentativos que guían el comportamiento del agente. Para tener en cuenta el enfoque de planificación, hay que permitir el relacionar los objetivos con el conjunto de elementos (e.g. predicados) al que representa. En cuanto al último enfoque (objetivos como requisitos), la asimilación de objetivos con requisitos es puramente interpretativa. Es decisión del ingeniero el considerar un objetivo como requisito o no.

2.4.3 Presentación del meta-modelo de tareas y objetos

El meta-modelo de tareas y objetivos se usa para expresar la motivación que hay detrás de las tareas y qué opciones de actuación se le presentan a un agente en un momento dado. La Ilustración 47 muestra la parte del meta-modelo de tareas y objetivos dedicada a este fin.

En el meta-modelo se refleja la relación de los objetivos con los agentes, roles y organizaciones. Una organización, como *entidad autónoma* al igual que un agente persigue objetivos (*GTPersigue*). Los roles, por el contrario, se asocian con los objetivos en el marco de los flujos de trabajo, por ello se asocian con otra meta-relación, *WFPersigue*. Los agentes, por poder aparecer en los flujos de trabajo y ser una *entidad autónoma*, como la organización, pueden asociarse con cualquiera de las dos.

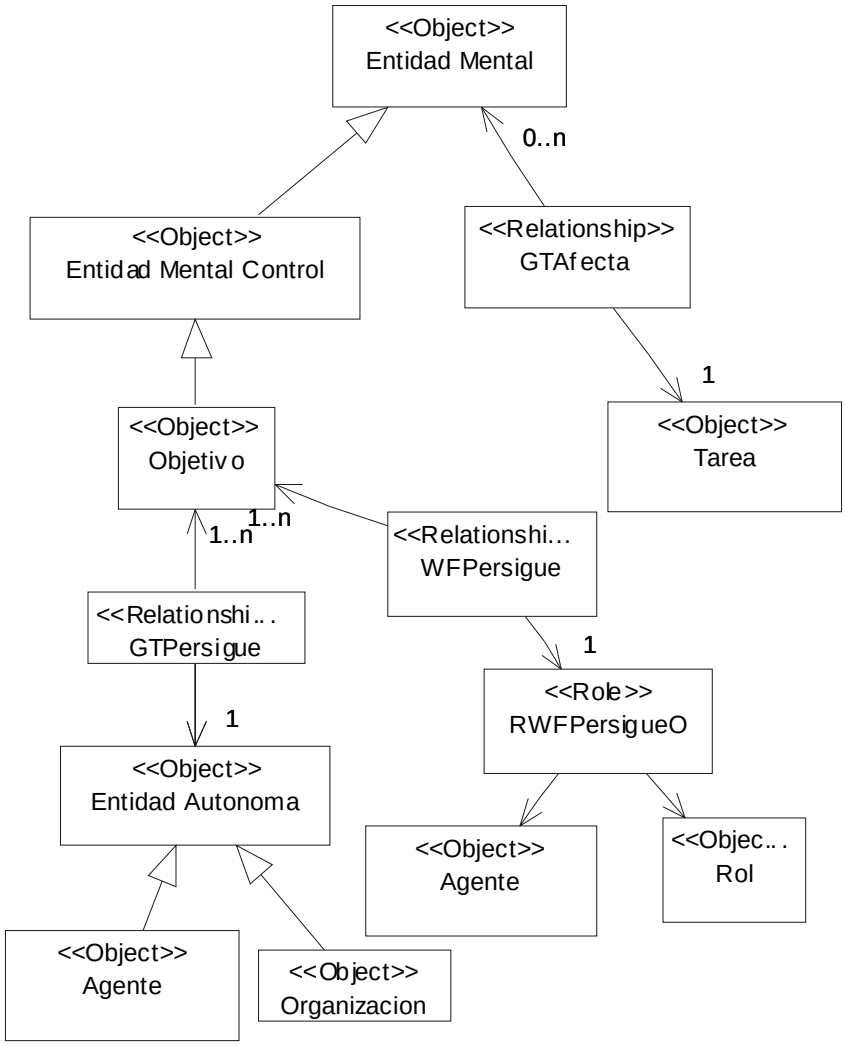


Ilustración 47. Relaciones entre objetivos y tareas

Las tareas se asocian con los objetivos mediante la meta-relación *GTAfecta* (ver Ilustración 48), cuya semántica es que la ejecución de la tarea afecta de una forma específica a una entidad mental. El ámbito de actuación de la tarea, en cuanto a *GTAfecta*, se limita a las entidades mentales del ejecutor de la tarea. La modificación del estado mental de otros agentes se modela haciendo que una tarea produzca una interacción con otro agente. Esta interacción provoca la ejecución en el otro agente de tareas que son las responsables finales de realizar el cambio.

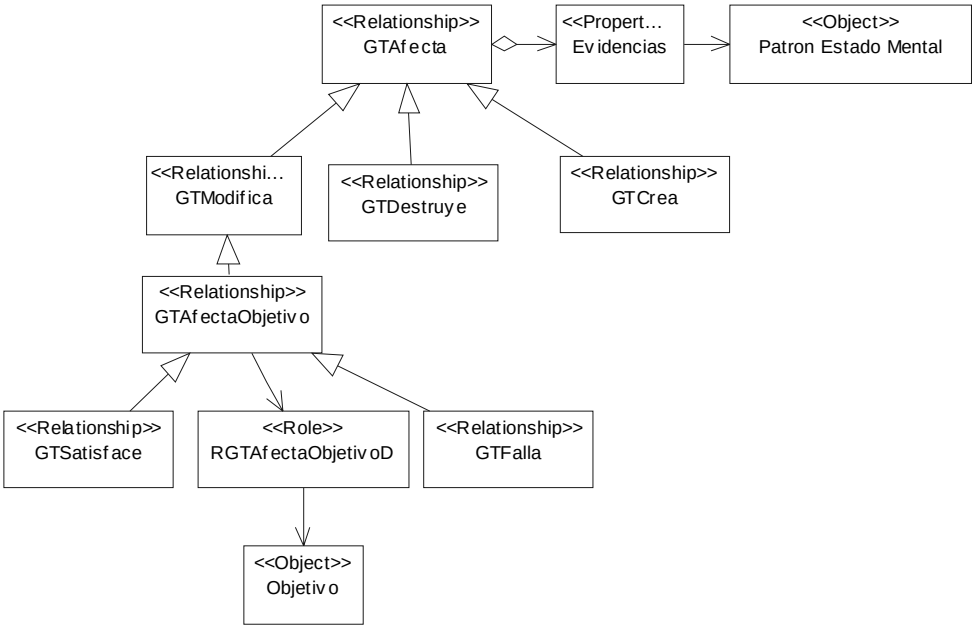


Ilustración 48. Formas en que una tarea afecta a un objetivo

Se distinguen tres formas de alterar el estado mental: destruyendo una o varias entidades (*GTDestruye*), creándolas (*GTCrea*) o modificándolas (*GTModifica*). Todas ellas se decoran con un *patrón de estado mental* para indicar en qué condiciones la creación, modificación o destrucción tiene lugar. Para el caso de la modificación y destrucción, se puede colocar dentro de estos patrones una entidad del mismo tipo que el que se va a ver afectado para reflejar dependencias de las entidades a modificar o destruir. La modificación de entidades mentales no se llega a detallar en profundidad. Sólo se señala que tal modificación existirá.

Para el caso de los objetivos, se ha especializado la relación *GTModifica*. Como el desarrollo que se propone se basa en la utilización de objetivos, es justificable el considerar cómo se satisfacen o fracasan los objetivos. Por ello se ha dispuesto de dos especializaciones que concretan la forma de modificar un objetivo: satisfaciéndolo (*GTSatisface*) o haciéndolo fallar (*GTFalla).* Estas dos últimas se condicionan a la existencia de ciertas evidencias que indiquen cuándo tiene lugar el fallo o la satisfacción del objetivo (propiedad *Evidencia*). Esta condición se expresa con *patrones de estado mental*, ya mencionados en el capítulo anterior. Queda pendiente el decidir qué hacer cuando se alcanza un objetivo o cuando este fracasa. Aquí se consideran dos enfoques:

1. **Delegar en el gestor y procesador de estado mental.** De forma genérica se establece cual es el tratamiento de los objetivos fracasados o satisfechos. Este tratamiento entraría dentro de las especificaciones del gestor y procesador de estado mental.

2. **Definir nuevas tareas para tratar con objetivos fracasados o satisfechos.** El cómo se realiza la gestión de entidades se detalla mediante tareas que destruyen, crean o modifican las entidades mentales existentes. La ventaja de este enfoque es que se hace explícito el tratamiento de gestión.

Para que este último enfoque funcione correctamente, es necesario definir relaciones de herencia entre entidades mentales. En concreto, se definen relaciones de herencia entre *hechos*. El resto de entidades mentales también podría necesitar este tipo de relaciones, pero de momento no ha sido necesario.

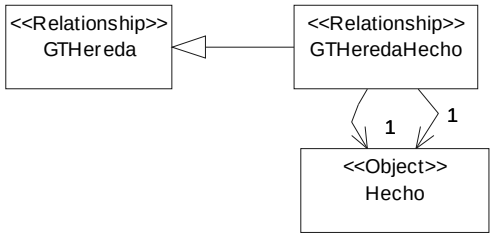


Ilustración 49. Relación de herencia entre hechos

La herencia definida es del tipo herencia simple y establece que lo que se hereda son las propiedades definidas en el ancestro.

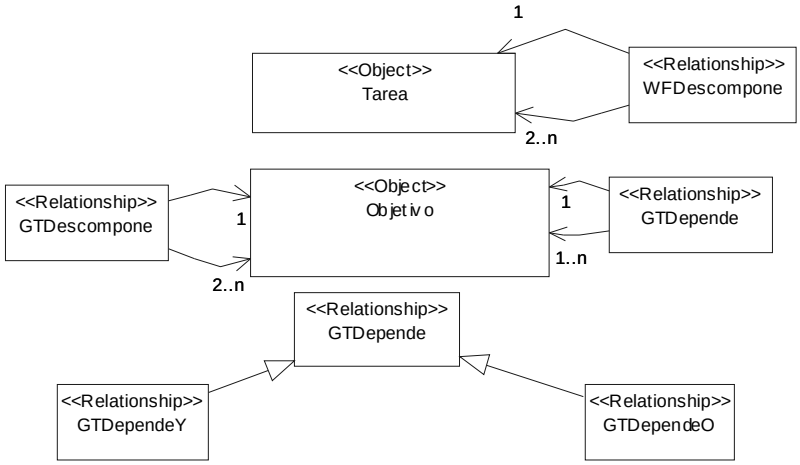


Ilustración 50. Meta-modelo de objetivos y tareas. Descomposición de tareas y objetivos.

Debido a la complejidad que pueden existir en los objetivos y las tareas, se han incorporado meta-relaciones que representan la descomposición de objetivos y tareas (*GTDescompone* y *WFDescompone*). Estas relaciones de descomposición dan lugar a grafos acíclicos donde se distinguen tareas, objetivos, sub-tareas y sub-objetivos. Este aspecto estructural se complementa con la expresión de dependencias entre objetivos. Nótese que no se incorporan dependencias entre tareas, aunque éstas son expresables mediante otras relaciones que se verán a continuación. La descomposición de objetivos da lugar a árboles

Y/O [Rich y Knight 90], que están soportados por instancias de *GTDependeY* y *GTDependeO*. *GTDependeY* asume que cuando todos los sub-objetivos (los definidos por esta relación) han sido satisfechos, el objetivo padre puede darse por alcanzado. La definición de *GTDependeO* se diferencia en que sólo se requiere que al menos uno de los objetivos referenciados haya sido satisfecho. Estas instancias también se usan para transmitir un fallo desde un sub-objetivo al objetivo padre. La transmisión de fallos desde los sub-objetivos al padre se hace en función del tipo de dependencia. *GTDependeY* hace que el objetivo padre falle si cualquiera de sus sub-objetivos falla. *GTDependeO* hace que el objetivo padre falle si todos sus sub-objetivos fallan.

Aunque según estas definiciones de dependencias entre objetivos, se podría asimilar descomposición a dependencia, en la realidad esta equivalencia no tiene por qué darse. Por ejemplo, se pueden definir un bosque de descomposición de objetivos y relaciones de dependencias Y/O entre objetivos pertenecientes a distintos árboles de descomposición estructural.

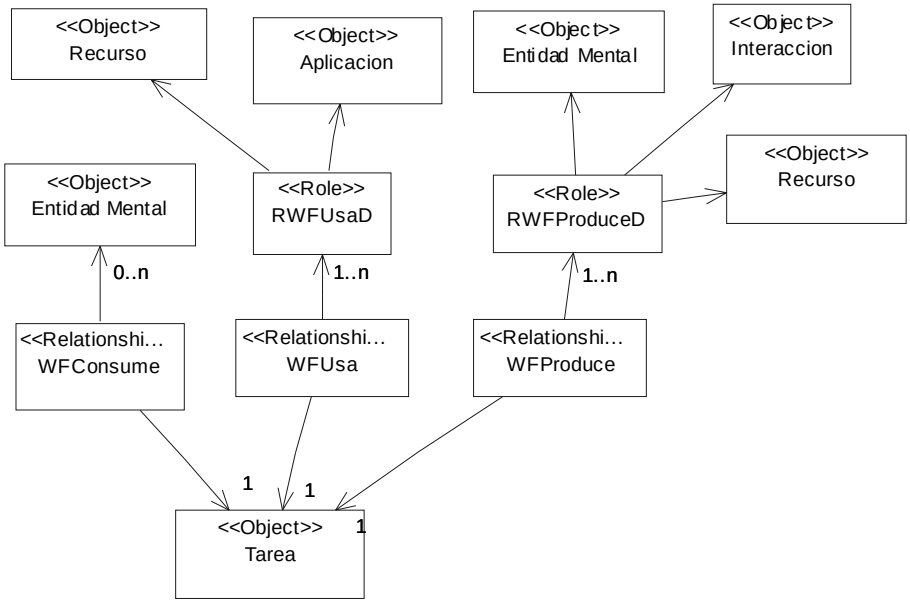


Ilustración 51. Meta-modelo de tareas y objetivos. Descripción de tareas

Las tareas (Ilustración 51) se describen mendiante meta-relaciones que definen precondiciones (*WFConsume, WFUsa, GTAfecta*) y postcondiciones (*WFProduce, GTAfecta*). *WFConsume* indica que para que se ejecute la tarea se necesita que existan determinadas entidades mentales (*objetivos*, *creencias, hechos,* etc.). La meta-relación *WFUsa* indica que se necesitan algunos de los recursos que se identifican en el meta-modelo de entorno. *GTAfecta* indica que se actúa sobre una entidad mental, por lo que se requiere de su existencia. Las postcondiciones se representan con *WFProduce* y *GTAfecta*. *WFProduce*

indica la creación de entidades mentales, interacciones o reposición de recursos en el seno de un flujo de trabajo. *GTAfecta* también puede señalar la creación de entidades mentales, pero se usa para indicar cambios en las entidades mentales no relacionadas con el flujo de trabajo en que se enmarca.

Aunque la ejecución de la tarea debe decidirla el *procesador de estado mental*, se asume que no se pueden ejecutar tareas que no satisfagan las precondiciones especificadas o que no dispongan de recursos o que, cuando se trate de tareas ejecutadas en el curso de una interacción, no se satisfagan las condiciones impuestas en instancias de *UIInicia* y *UIColabora*.

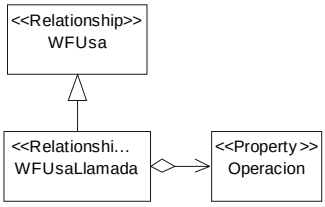


Ilustración 52. Meta-modelo de tareas y objetivos. Uso de entidades del entorno

Durante la ejecución de la tarea se ejecutan acciones sobre el entorno, que en este trabajo se considera como una agregación de agentes, aplicaciones y recursos. Las acciones sobre otros agentes se consideran como *interacciones* en el sentido de instancias del meta-modelo de interacción. Sin embargo, las accciones sobre otras aplicaciones o recursos están dentro de un orden distinto. Al principio del desarrollo, estas acciones se representan con *WFUsa* sin llegar a explicitar exactamente qué operación se está utilizando. En etapas avanzadas se emplea *WFUsaLlamada* donde se anexa como información qué operación se está utilizando.

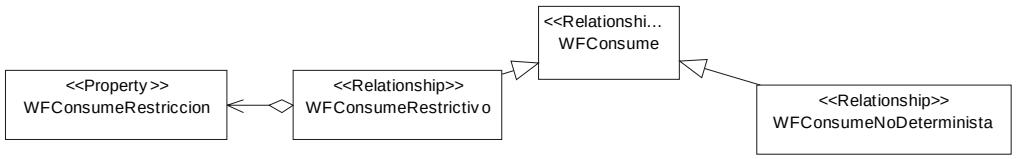


Ilustración 53. Meta-modelo de tareas y objetivos. Restricciones sobre elementos consumidos

Cuando existen múltiples instancias de las entidades mentales referenciadas, se emplea como ámbito el contexto de ejecución de la tarea dado por el flujo de trabajo. Si dentro del mismo flujo de trabajo existen varias instancias de la misma entidad mental, se hace una selección no determinista (*WFConsumeNoDeterminista*) o bien una selección basada en restricciones sobre los atributos asociados a la entidad (*WFConsumeRestrictivo*).

2.4.4 Ejemplo: un agente planificador de tareas

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* (Ilustración 54).

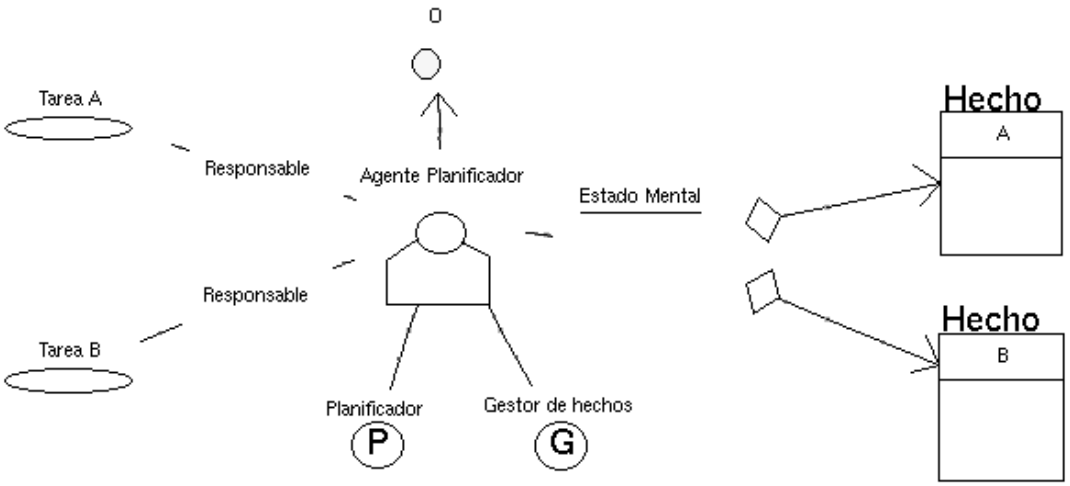


Ilustración 54. Modelo de agente para representar el agente planificador.

Inicialmente, el agente cuenta con los hechos *A* y *B*. Con esta información, el *procesador de estado mental* del agente, que es en realidad un planificador de tareas, se pregunta si será capaz de alcanzar el objetivo propuesto (*O*). Para determinar si será capaz o no, hace falta más información, concretamente bajo qué condiciones el objetivo *O* se alcanza y qué efectos tiene la ejecución de las tareas A y B.

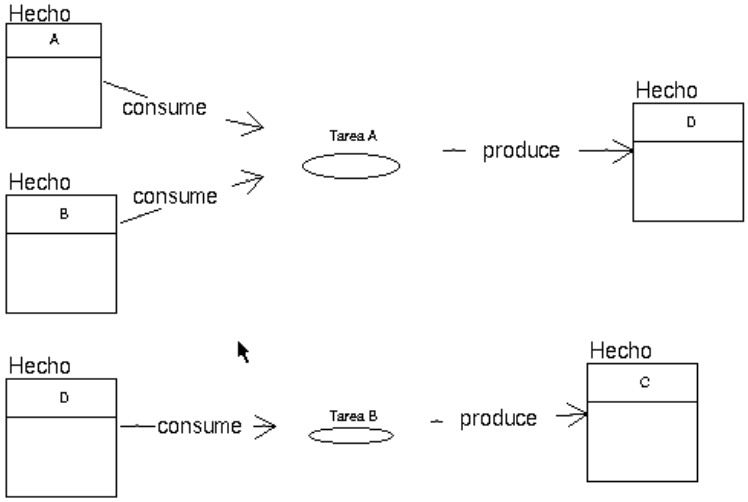


Ilustración 55. Modelo de tareas y objetivos para especificar las tareas A y B

La Ilustración 55 muestra las consecuencias de ejecutar las tareas A y B. En este caso, ambas se limitan a producir nuevas entidades mentales.

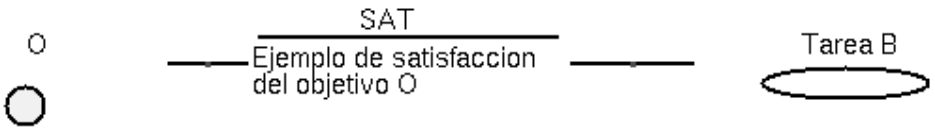


Ilustración 56. Modelo de tareas y objetivos para mostrar la satisfacción del objetivo *O* por la ejecución de la *Tarea B*

La asociación de objetivos con tareas es fundamental. Como se ha mencionado con anterioridad, hace falta justificar por qué el agente elige un curso de acciones. Aquí, la elección de la *Tarea B* se justifica porque con ello se alcanza el objetivo *O*. La relación se decora con un patrón de estado mental (*Ejemplo de satisfacción del objetivo O*) que denota el estado mental requerido para dar por satisfecho *O*.

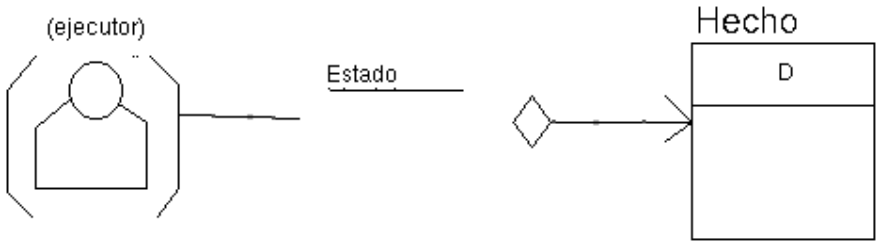


Ilustración 57. Modelo de agente para representar el estado mental requerido para alcanzar el objetivo *O*

La Ilustración 57 muestra lo que faltaba, una descripción del estado mental requerido para alcanzar el objetivo *O*. En este caso, se requiere que el que persigue *O* contenga también el hecho *D*. A partir de este momento, el *procesador de estado mental* del agente, que es un planificador, tiene información suficiente para deducir que debe ejecutar *Tarea A* antes de ejecutar *Tarea B* para alcanzar el objetivo *O*.

El problema se podría haber especificado de una forma alternativa. En lugar de asociar directamente el objetivo *O* con la *Tarea B*, se podría haber aplicado descomposición de objetivos y asociar los nuevos subobjetivos con las tareas existentes.

Ilustración 58. Modelo de objetivos y tareas para mostrar la descomposición de *O* en *O1* y *O2* (izquierda) y las dependencias entre éstos (derecha)

La Ilustración 58 muestra la descomposición del objetivo *O* en otros dos (*O1* y *O2*) con los que se establece una dependencia Y. Esto significa que cuando *O1* y *O2* hayan sido alcanzados, automáticamente, *O* se dará por alcanzado.

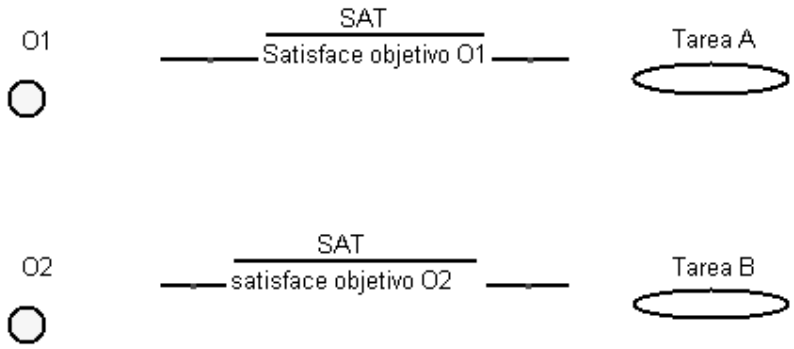


Ilustración 59. Modelo de objetivos y tareas para mostrar la asociación de las tareas *A* y *B* con los objetivos *O1* y *O2*

La Ilustración 59 sirve para asociar los objetivos *O1* y *O2* con las tareas que sabe ejecutar el agente. La descripción detallada del estado mental requerido por *O1* y *O2* se omite aquí. Baste decir que para alcanzar *O1* y *O2* se necesitan que existan los hechos *C* y *D*.

2.4.5 Integración con otros meta-modelos

Este meta-modelo se orienta principalmente a justificar la ejecución de tareas basándose en los objetivos, que a su vez, se van modificando tras dicha ejecución. Por ello, la mayoría de

relaciones con los modelos consiste en recoger, agrupar e interrelacionar los objetivos que aparecen en ellos, poniéndolos en el contexto de justificantes de la ejecución de tareas (ver Ilustración 60). Así, las relaciones con los meta-modelos de agente, organización e interacción se reduce a recoger todos los objetivos y tareas que allí aparecen y trabajar con ellos.

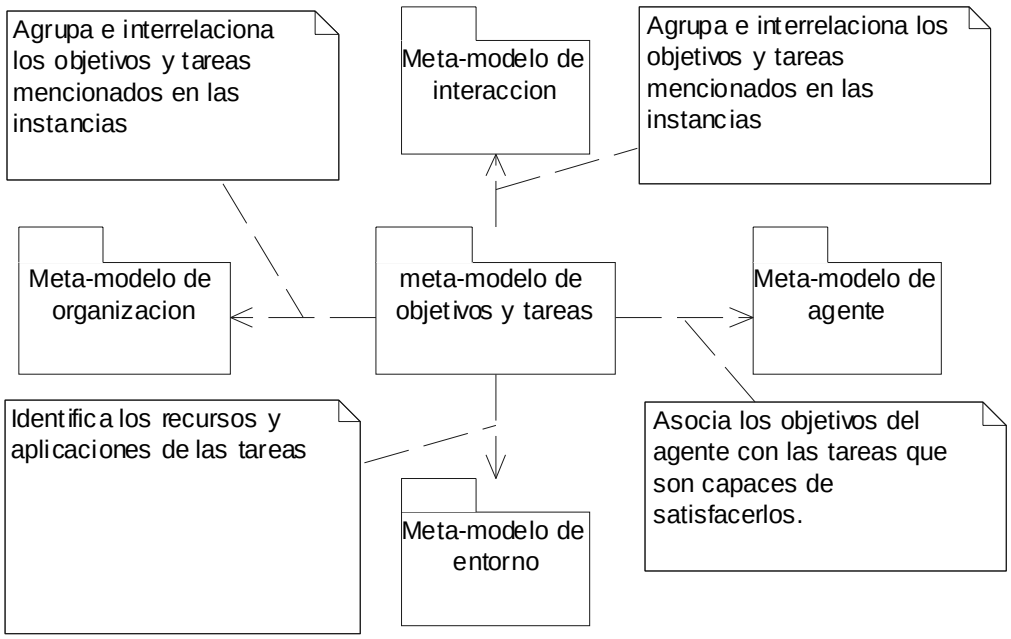


Ilustración 60. Dependencias del meta-modelo de objetivos y tareas respecto de otros meta-modelos

La única asociación de propósito diferente es la establecida con el meta-modelo de entorno. Este meta-modelo proporciona recursos y aplicaciones con las que habilitar las tareas.

Por lo tanto, la mayoría de los criterios de consistencia en este meta-modelo se restringen a verificar que se satisfacen las siguientes condiciones:

1. 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.
2. Todo objetivo que aparezca en un modelo de tareas y objetivos debe aparecer en un modelo de agente o en un modelo de organización.
3. Si una tarea produce una interacción, debe existir un modelo de organización donde se enmarque esta tarea dentro de un flujo de trabajo y un modelo de interacción para la interacción producida.
4. 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.
5. Todo recurso que aparece en este modelo debe aparecer en un modelo de entorno.

6. Las entidades mentales consumidas, producidas, modificadas o destruidas deben pertenecer al estado mental del agente ejecutor. Para modificar el estado mental de otro agente se utiliza una interacción.

2.5 Meta-modelo de Organización

El modelo de organización es el equivalente a la arquitectura del sistema en un SMA. El valor principal de un modelo de organización, como ocurre en las organizaciones humanas, son los flujos de trabajo que define. Del estudio de estos flujos surgen nuevas interacciones que reflejan con detalle cómo se coordinan los participantes del flujo. El modelo de organización también contribuye al modelo de tareas y objetivos identificando las tareas relevantes para la organización así como los objetivos que se persiguen globalmente. También define restricciones en el comportamiento de los agentes mediante relaciones como la de subordinación. Gracias a estas restricciones, el diseñador asegura que unos agentes obedecerán a otros o que se comprometerán a la ejecución bajo demanda de tareas respetando sus prioridades.

En esta sección primero se encuadra el trabajo relacionado para determinar qué debe aparecer en el meta-modelo. A continuación se presenta el meta-modelo que reune los elementos indicados por el estado del arte. Y para terminar, se introducen ejemplos de utilización del meta-modelo y un conjunto de consideraciones acerca de la consistencia del modelo generado.

2.5.1 Organizaciones de agentes

En general los trabajos realizados sobre la organización de SMAs no estudian en profundidad el problema del diseño de la organización, dedicándose a buscar organizaciones emergentes [Shoham y Tennenholtz 97;Walker y Wooldridge 95], tipos de organizaciones [Pattison 87] o cómo las organizaciones determinan el comportamiento del agente (*razonamiento social* [Sichman et al. 94] y el *nivel social* [Jennings N. y J. 97]). En MESSAGE se plantean, sin embargo, los aspectos de desarrollo, por lo que toma en consideración otras fuentes:

- **Empresas virtuales** [Ambvroszkiewizcz et al. 98]. La empresa se basa en tres vistas: vista de negocios, vista de la organización y la vista de flujo de información. La vista de negocios describe cómo se realiza la producción de la empresa en función de los agentes existentes. La vista de la organización establece roles y dependencias organizacionales (como relaciones de poder). La vista de flujo de información sirve para mantener la integridad organizacional así como para optimizar la producción. Hay más trabajo en el área de los agentes factoría [Van Dyke Parunak 87], aunque en ninguno la estructura de la organización fue tema de estudio.

- **Las organizaciones en MAS-CommonKADS** [Iglesias et al. 98]. Extienden los modelos de CommonKADS incorporando herencia de agentes, la noción de grupo y

relaciones de poder entre agentes. La estructura organizativa se ve como un medio de arbitrar en las interacciones entre agentes. Al tener cada agente un lugar en la estructura organizativa, es más fácil, en caso de conflicto, decidir quien tiene prioridad. En esta metodología, sin embargo, no existe la organización como entidad.

- **Common Enterprise Models Domain Task Force de OMG** [Adaptative Ltd.DSTC Gazebo Software Solutions 00]. Propone la especificación de interfaces para gestionar la creación, navegación, consulta, y otros usos básicos como la seguridad, importación/exportación de información en una *estructura organizativa*.

- **AALADIN** [Ferber y Gutknecht 98]**.** Define grupos, agentes y roles. Los agentes crean grupos y juegan roles dentro de ellos. Los grupos son agregaciones de agentes y las organizaciones agregaciones de grupos. Las in stancias de este modelo se denominan *estructuras de grupos* (*group structures*) caracterizadas por un conjunto de roles, de interacciones entre roles, y por agentes que desempeñan los roles.

Estas consideraciones fueron integradas dentro del meta-modelo de organización de MESSAGE basándose en dos principios:

- *Distinción entre agentes y organizaciones.* Las organizaciones son entidades en sí mismas en el nivel de análisis. Las organizaciones no pueden existir sin agentes, aunque el opuesto podría ser cierto. Al igual que en las organizaciones humanas, los agentes son los responsables finales de ejecutar tareas en una organización.

- *La Organización delega su estructuración en la Estructura Organizativa*. No hay relaciones de agregación entre las organizaciones. Existen diferencias entre los conceptos de *Organización* y *Estructura Organizativa* (o *Grupo*). La Organización tienen un conjunto de atributos que no tienen las Estructuras Organizativas. Además, su ciclo de vida es diferente. Las Estructuras Organizativas se crean y se destruyen dentro de una organización sin que ésta deje de existir. Los agentes y los recursos de la organización siempre pertenecen a la Organización, pero pueden ser asignados dinámicamente a diferentes Organizaciones Estructurales. Por último, si la Organización se disuelve, todas las Estructuras Organizativas desaparecen.

Tomando principalemente el trabajo de MESSAGE como punto partida, se ha aumentado la cohesión de los elementos de la organización con respecto a los elementos pertenecientes a otros meta-modelos. También se ha desarrollado la idea de flujo de trabajo, poco destacada dentro de MESSAGE, como tercer pilar del meta-modelo. El flujo de trabajo, siguiendo ideas de [Workflow Management Coalition 99], permite contextualizar la ejecución de las tareas e interrelacionarlas unas con otras independientemente de los ejecutores de las mismas.

2.5.2 Presentación del meta-modelo de organización

Para simplificar la presentación del meta-modelo de organización, se han separado tres aspectos del mismo, como muestra la Ilustración 61.

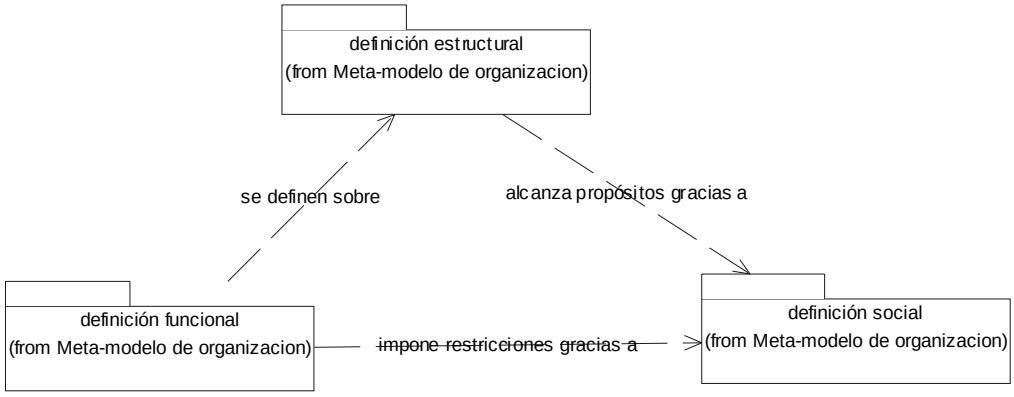


Ilustración 61. Las tres vistas de una organización

La estructura de la organización define los elementos principales que componen la organización y cómo se construye ésta a partir de ellos. Las relaciones sociales establecen relaciones de alto nivel entre los componentes para poner restricciones al comportamiento de la organización. Por último, la definición funcional establece qué ofrece la organización y cómo se lleva ésta a cabo.

2.5.2.1 Descripción estructural de la organización

De acuerdo con algunas tendencias filosóficas [WHITAKER 95] [Maturana 02] la organización es la entidad de más alto nivel en el sistema. La organización, desde el punto de vista estructural, es un conjunto de entidades asociadas por relaciones simples de agregación y herencia. En esta estructura se define el esqueleto donde van a existir los agentes, los recursos, tareas y objetivos. Sobre este esqueleto se definen una serie de relaciones que inducen la formación de flujos de trabajo y de restricciones sociales (ver siguientes secciones).

Según la Ilustración 62, la organización es una *entidad autónoma*, como lo es también un agente (ver sección 2.2.2). Los objetivos perseguidos por la organización son los objetivos comunes a los agentes que la componen y el motivo por el cual se han agrupado. No obstante, una organización no es un agente. La diferencia fundamental es que la organización no tiene capacidad de ejecutar tareas ni para tomar decisiones, son los agentes que la componen quienes se encargan de ello.

La visión estructural de la organización proporciona la descomposición de la organización en grupos (*OContieneOrganizacion*) y flujos de trabajo (*OContieneFlujoTrabajo*). La descomposición recursiva de flujos y grupos se realiza mediante *ODescomponeGrupo* y *ODescomponeFlujoTrabajo*, respectivamente.

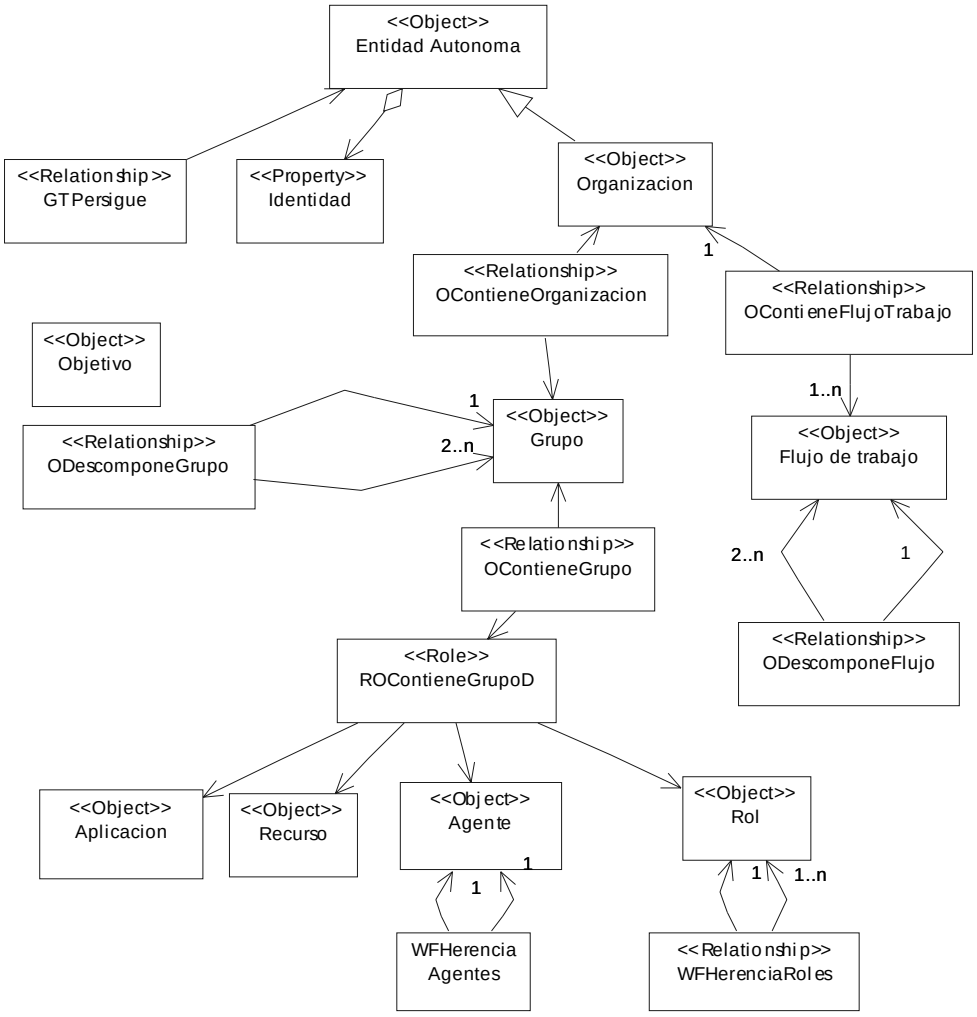


Ilustración 62. Meta-modelo de organización. Visión estructural

Cada grupo contiene (*OContieneGrupo*) agentes, recursos, aplicaciones o roles. La asignación de estos elementos a un grupo obedece a propósitos organizativos, esto es, están agrupados porque clarifica la creación de flujos de trabajo. Cualquiera de ellos puede pertener a otros grupos de la misma u otras organizaciones. La utilidad de los grupos aparece cuando el número de estos elementos (agentes, roles o recursos) empieza a ser poco manejable. Los grupos son herramientas inestimables para la estructuración de la organización, ya que permiten al desarrollador aplicar divide-y-vencerás para facilitar el problema de la definición del sistema, y también para modelar estructuras organizativas existentes en el mundo real. En cuanto a la gestión de los grupos, a nivel de meta-modelo no se expresa, ya que se pueden definir roles de tipo *gestor de comunidad* desempeñados por agentes que realicen estas funciones (como se sugiere en [Ferber y Gutknecht 98]).

Para aumentar la capacidad de abstracción e incrementar la reusabilidad de los agentes y roles, se ha incluido la posibilidad de extenderlos. Como en MAS-CommonKADS, se permite definir nuevos agentes o roles que extiendan los existentes. La herencia entre agentes es una herencia simple, mientras que con los roles se admite herencia múltiple. La semántica de la herencia es parecida a la semántica de la herencia de objetos. Así, si un agente A extiende un agente B, el agente A puede sustituir a un agente B en cualquier situación. Y si un rol A, extiende los roles B_1 hasta B_n, entonces A puede sustituir a cualquier B_i en cualquier situación. Como restricción, se exige que no se redefina ningún aspecto ya establecido en el ancestro (por ejemplo asociaciones con tareas, estados mentales, participación en flujos de trabajo) y que las nuevas capacidades de los agentes no creen conflictos con las de sus ancestros (por ejemplo asociando tareas que hagan lo contrario de otras que ya existan).

2.5.2.2 Descripción funcional

El objetivo del flujo de trabajo es establecer cómo se asignan los recursos, qué pasos (tareas) son necesarios para la consecución de un objetivo, y quiénes son los responsables de ejecutarlas. Según el la *WorkFlow Management Coalition* (WfMC), un flujo de trabajo [Workflow Management Coalition 99] es la *automatización de un proceso de negocio, en su totalidad o parcialmente, durante el cual los documentos, información o tareas son pasadas de un participante a otro, de acuerdo con un conjunto de reglas procedimentales*. En el flujo de trabajo, se habla de *actividades* en lugar de *tareas*[7], aunque en este contexto se pueden emplear indistintamente.

Dentro del flujo del trabajo interesa presentar dos tipos de información: cómo se asocian unas tareas con otras y cómo se ejecutan. La forma de presentar estos dos aspectos es a través de instancias de meta-relaciones que interconectan entradas y salidas de las tareas y los responsables de ejecutarlas (Ilustración 63).

[7] Una actividad [Workflow Management Coalition 99] es una descripción de un trozo de trabajo que forma un paso lógico dentro de un proceso. Una actividad puede ser manual, que no soporte automatización computerizada, o una actividad automatizada de flujo de trabajo. Una actividad es típicamente la unidad de trabajo más pequeña que se organiza dentro de una activación de proceso.

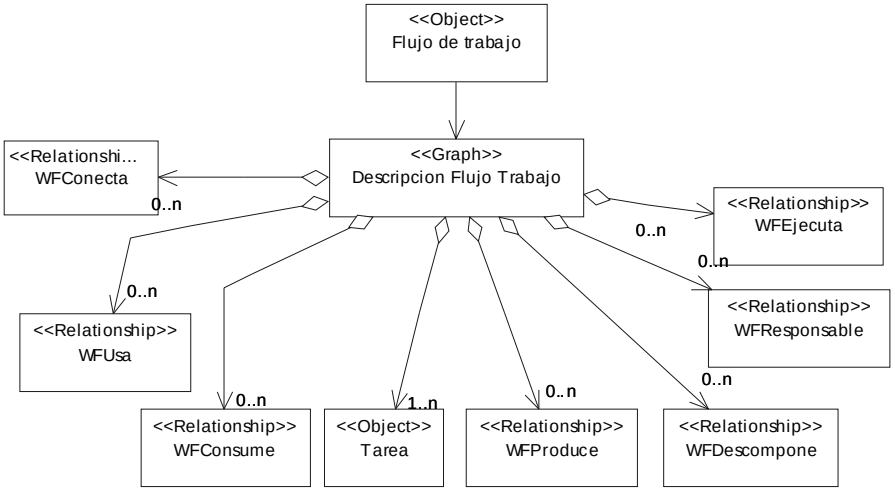


Ilustración 63. Meta-modelo de organización. Elementos que componen un flujo de trabajo

Algunas relaciones (concretamente *WFConsume, WFUsa, WFProduce, WFDescompone*) ya se han visto dentro del meta-modelo de objetivos y tareas. Las únicas nuevas son *WFEspecificaEjecucion* y *WFConecta* (ver Ilustración 64). *WFEspecificaEjecucion* indica que la especificación concreta de las condiciones de ejecución de una tarea se presenta dentro de una interacción. Se ha adoptado esta solución para correlacionar tareas ejecutadas por distintos actores. *WFConecta* se corresponde con la conexión de las salidas (instancias de *WFProduce*) de una tarea con las entradas de otra (instancias de *WFConecta*). La conexión no es completa en el sentido de que hay necesidad de que todo lo producido por una tarea deba ser consumido por la otra. Para determinar exactamente qué se consume, hay que considerar las instancias de *WFConsume*.

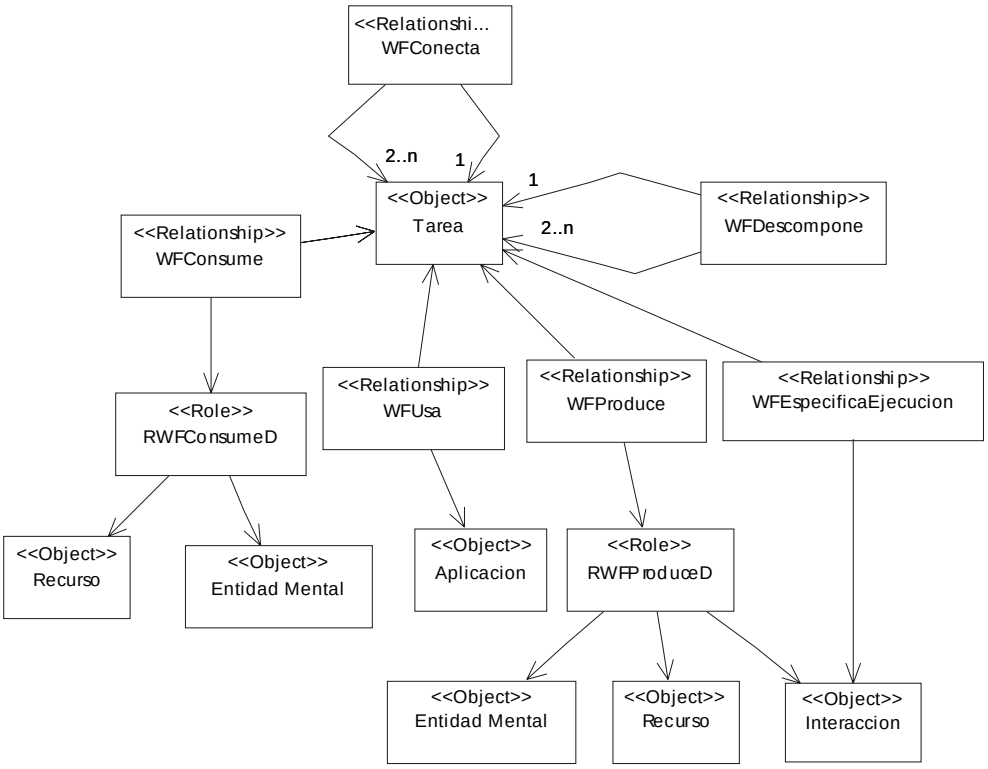


Ilustración 64. Meta-modelo de organización. Relación entre los elementos del flujo de trabajo

Para permitir descripciones incrementales, se permite que la entidad *Flujo de Trabajo* actúe como extremo de las relaciones *WFProduce, WFConecta* y *WFConsume* de la misma forma en que lo hace la entidad *tarea*. De esta forma, se pueden incluir flujos de trabajo dentro de la descripción de los flujos de trabajo.

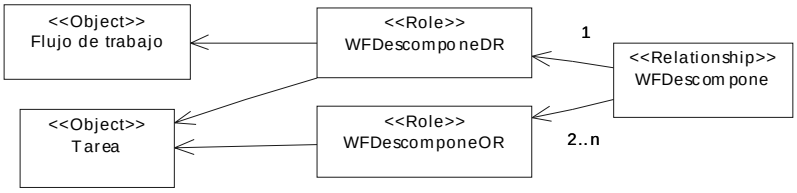


Ilustración 65. Descomposición de tareas y flujos de trabajo

Para terminar la inclusión de los *flujos de trabajo*, se permite que un *flujo de trabajo* se descomponga en otros *flujos de trabajo* o tareas. De esta forma se pueden generar definiciones incrementales de las tareas que componen el *flujo de trabajo*. No se permite la descomposición de una tarea en *flujos de trabajo.*

Las tareas son ejecutadas por agentes que son directamente responsables de ellas (mediante instancias de *WFResponsable*) o indirectamente a través de roles (mediante

instancias de *WFJuega*), como indica la Ilustración 66. Que un agente o rol sea responsable de la ejecución de una tarea, significa que el agente sabe ejecutar esa tarea.

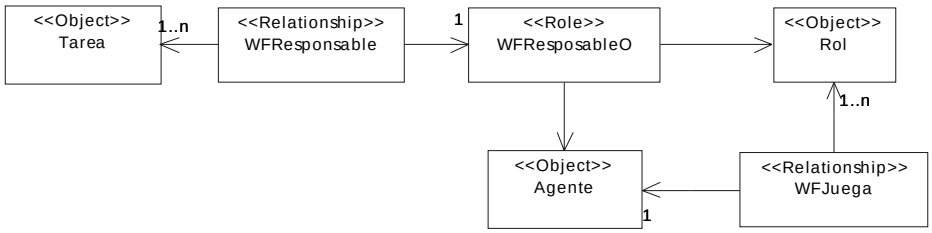


Ilustración 66. Meta-modelo de organización. Asociación de tareas con sus ejecutores.

La ejecución puede ser local o involucrar la ejecución de otras tareas en otros agentes. Ambas variantes se inician de la misma forma y tienen efectos locales idénticos: un agente inicia la tarea motivado por la satisfacción de un objetivo (*principio de racionalidad*) e indica lo que se produce por instancias de *WFProduce* y su efecto sobre el entorno con *WFUsa* (sobre aplicaciones y recursos). La diferencia entre ambos tipos es que la tarea que involucra ejecución de otras tareas en otros agentes produce *interacciones*. Las interacciones, indican bajo qué condiciones tiene lugar la ejecución de tareas (instancias de *UIInicia* y *UIColabora*), como se vio en la sección 2.3.2 . Aparecen en los modelos de organización dentro de flujos de trabajo como producto de la ejecución de tareas. Para concretar quien participa en las interacciones, se utilizan instancias de *Consulta Entidad Autónoma*. Estas instancias pueden asociarse a un agente o rol mediante la meta-relación*AInstanciaDe*. Las etiquetas asociadas a *Consulta Entidad Autónoma* harán referencia a elementos consumidos o producidos por las tareas Ilustración 75.

En cuanto al uso que se hace de los recursos, la principal preocupación es el evitar interbloqueos. En este trabajo se asume que no va a haber interbloqueos ya que se eliminan una de las cuatro condiciones de [Coffman, Elphick y Shoshani 71] para que se éste se presente (ver meta-modelo de entorno).

2.5.2.3 Descripción social

Las relaciones que se mencionan aquí entre Organizaciones, Agentes y Grupos, configuran restricciones sociales que limitan la ínteractuación entre estas entidades. En concreto, aquí se incluyen relaciones de subordinación, de prestación de un servicio y de cliente de un servicio. Estas relaciones se definen teniendo como extremos posibles organizaciones, agentes, roles y grupos.

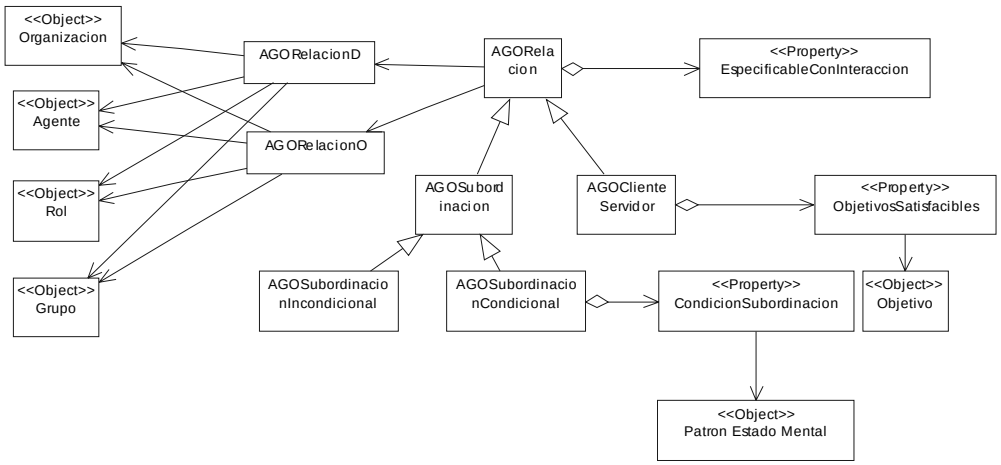


Ilustración 67. Meta-modelo de organización. Relaciones sociales entre los componentes de la organización

Las relaciones de subordinación (*AGOSubordinacion*) obligan al subordinado a obedecer en todo y permiten al subordinante dar cualesquiera órdenes. Esta relación contradice uno de los supuestos sobre los agentes, que tienen libertad de actuación. Aunque deseable en algunas ocasiones, es cierto que esta libertad añade nuevas variables al sistema (los grados de libertad del agente) haciéndolo mucho más complejo. El compromiso del diseñador en estos casos es utilizar desde un punto de vista racional las libertades del agente. Así, estas relaciones son interesantes desde el punto de vista de la ingeniería porque permiten al diseñador asegurar que un agente no deja de obedecer ordenes críticas. Esta obediencia se limita a los agentes que el diseñador establezca, respetando en otros la libertad de desobedecer.

El siguiente paso es descomponer esta relación en obediencia incondicional (*AGOSubordinacionIncondicional*) y condicionada (*AGOSubordinacionCondicional*). La incondicional consiste en obedecer todas y cada una de las órdenes. La condicional es asimilable a un contrato, donde si se infringen algunas de las condiciones impuestas, el contrato deja de tener validez. Las condiciones del contrato se establecen utilizando un *Patron de estado mental*. Con este tipo de patrón es posible expresar:

- **Acciones toleradas.** Aquellas permitidas en el marco de la subordinación. Las acciones se corresponden con las tareas cuya ejecución se ha requerido. Esta restricción previene contra el abuso en las relaciones entre agentes. Se expresarían en forma de creencias acerca de las capacidades del agente (como con los predicados *CAN* de Agent0). Según el meta-modelo de agente, esto se hace mediante instancias de *CreenciaExistencia* acerca de la existencia de relaciones *WFJuega* con roles que soporten las acciones requeridas.
- **Fecha hasta la que se mantiene la subordinación.** Se trata de mantener la relación sólo un periodo concreto de tiempo. El periodo se expresa como un intervalo de tiempo o con la duración del mismo, entendiendo que comienza en el momento de establecer la relación. Si se modela la subordinación como una *CreenciaExistencia* cuyo

tiempo de vida es limitado y se ha definido un período de validez de conocimiento en el *Gestor de Estado Mental*, basta con comprobar la existencia de la creencia anterior.

- **Estado mental que hay que mantener a lo largo de la subordinación.** Puede entenderse como el invariante de la relación. Si este estado mental no se mantiene, entonces la relación se puede dar por terminada. Se modelaría con una instancia directa de las entidades mentales que deben mantenerse.

Respecto a las relaciones Proveedor y Cliente (*AGOClienteServidor*), hacen referencia a que existe un servicio ofrecido (proveedor) y que se busca un servicio determinado (cliente). En la descripción de proveedor y cliente, y del servicio en sí, existen múltiples variantes. Aquí se propone una alternativa coherente con el meta-modelo ya descrito. A la hora de pedir o informar de la capacidad de dar o recibir un servicio, simplemente se informa qué se sabe satisfacer o qué se busca utilizando un objetivo. La técnica es similar a la más extendida de definición de ontología y a la utilización de un *matchmaking* [Sycara et al. 99]. La ontología la constituyen los objetivos a satisfacer. La utilización de los objetivos tiene la ventaja de orientar el sistema hacia la concepción de agente de Newell. Aquí, los agentes buscan satisfacer objetivos, y si se ponen en contacto unos con otros, es para conseguir satisfacer objetivos.

En cierto modo la subordinación condicional es similar a la prestación de un servicio, ya que en ambos se busca ejecutar una acción para otro. Sin embargo, son diferentes en la forma. En la prestación de un servicio, el proveedor puede denegar la petición, mientras que en la subordinación existe la obligación de ejecutar lo pedido.

2.5.2.3.1 Prioridad entre las relaciones sociales

Entre las distintas entidades de la organización (ver Ilustración 67) surgen dieciséis posibles combinaciones de dependencias entre organizaciones, grupos, roles y agentes. Para limitar y estructurar estas relaciones, se admiten sólo relaciones entre entidades del mismo tipo. Sobre las relaciones resultantes se plantea una política de aplicación. Se establece que entre dos entidades de la organización pueden existir como máximo relaciones a tres niveles. De mayor a menor nivel, estos niveles son:

- **Nivel organizativo.** Se trata de las relaciones organizacionales. Se dan entre las entidades de más alto nivel y son el equivalente a las relaciones inter-empresariales.
- **Nivel de estructura de organización.** Se trata de las relaciones estructurales. Serían las equivalentes a relaciones interdepartamentales.
- **Nivel de agente y rol.** Se trata de las relaciones de agencia. Serían equivalentes a relaciones directas entre el personal.

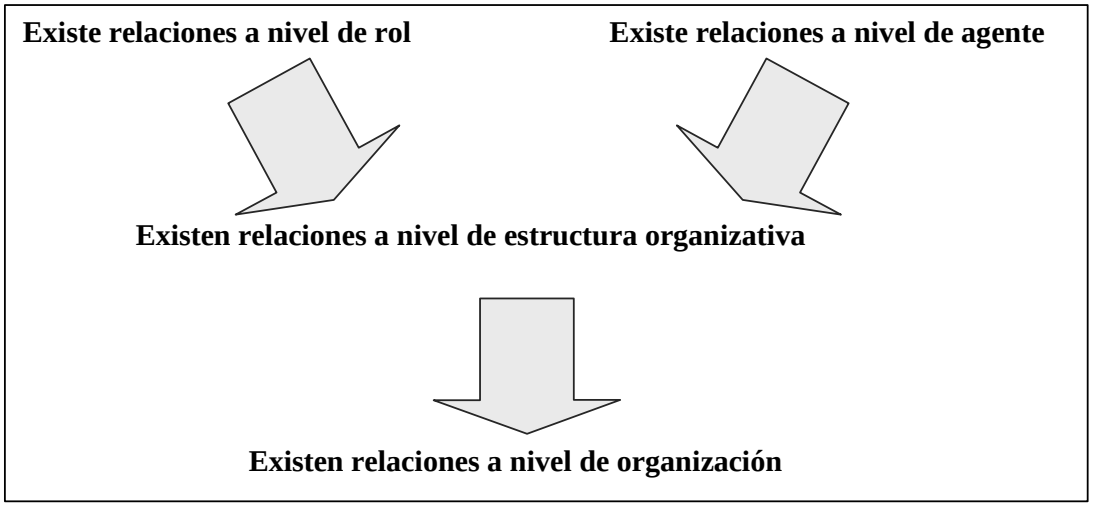


Ilustración 68. Meta-modelo de Organización. Semántica sobre las meta-relaciones sociales

No puede haber relaciones sociales entre dos entidades entre las cuales no exista una relación a nivel inmediatamente superior.

2.5.3 Ejemplos de modelo de organización

Este ejemplo se ha extraído del caso de estudio principal de este trabajo. Se quiere diseñar un sistema de filtrado colaborativo de documentos donde los usuarios se suscriban a grupos de interés para recibir documentos que coincidan con sus gustos. En este sistema, los usuarios actúan a través de *Agentes Personales*. Estos agentes se suscriben a los grupos o comunidades de interés interactuando con el representante de la comunidad, el *Agente de Comunidad*. Se admite que un usuario pueda estar suscrito a múltiples comunidades. Una vez suscritos, los usuarios pueden sugerir documentos a la comunidad y recibir sugerencias de otros usuarios de la misma comunidad a través de su *Agente Personal*. Como arquitectura del sistema de este SMA, se propone el modelo de organización de la Ilustración 69.

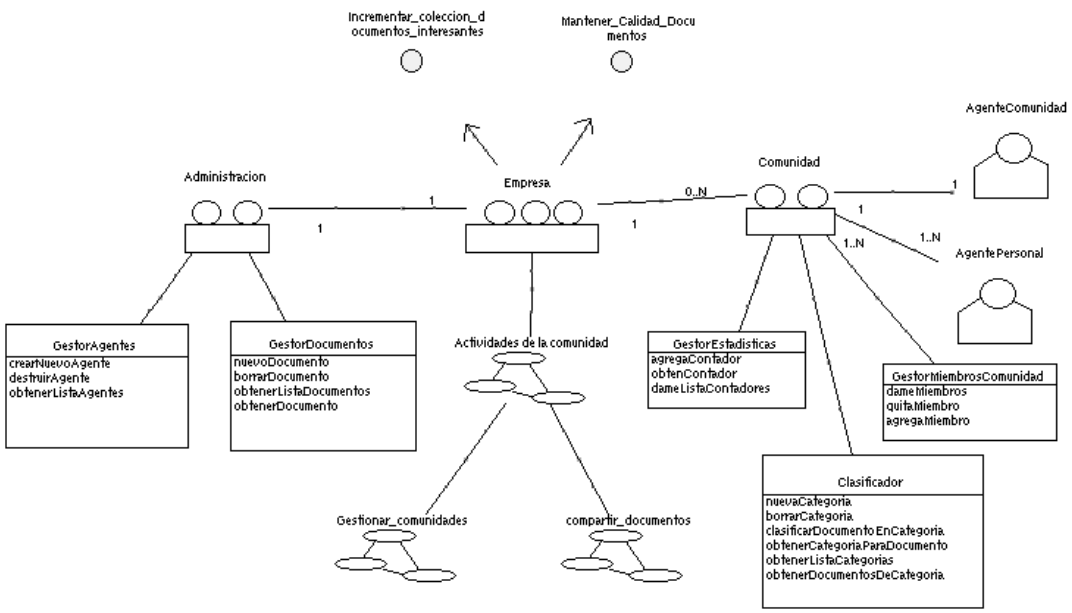


Ilustración 69. Estructura de una organización dedicada a la distribución de documentos de interés para sus usuarios

La empresa consta de de dos grupos, uno que representa la administración de recursos compartidos (*Administración*) y otro que representa comunidades de usuarios (*Comunidad*). Dentro del grupo *Comunidad* existe un agente responsable de su administración (altas y bajas de usuarios), el *Agente de Comunidad*, y otros agentes que actúan como miembros de la comunidad, *Agentes Personales*. Los *Agentes Personales* representan al usuario en los procesos internos del sistema. Aquí se verá como ejemplo su participación en la evaluación de nuevos candidatos desempeñando los roles *Miembro de la comunidad* y *Solicitante de suscripción*.

Los flujos de trabajo *Actividades de la comunidad*, *Gestionar comunidades* y *Compartir Documentos* aparecen para estructurar las tareas identificadas con modelos de interacción y con modelos de tareas y objetivos. En estos modelos se ha estudiado qué comportamiento básico se espera de los agentes a la hora de suscribirse a comunidades y sugerir documentos y cómo se alcanzan los objetivos iniciales asociados a los agentes.

Los flujos de trabajo se pueden refinar para obtener flujos de trabajo más especializados. En la Ilustración 70 se refina la gestión de comunidades en tareas básicas de gestión (añadir un miembro a la comunidad, quitar un miembro de la comunidad, monitorizar miembros de la comunidad), y flujos de trabajo para determinar bajas en la comunidad y altas en la comunidad.

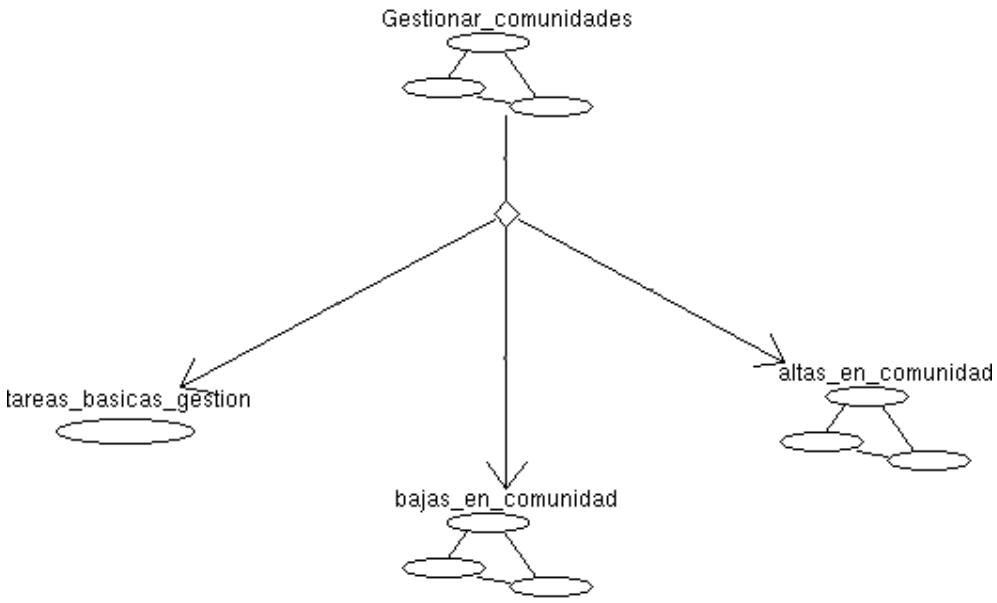


Ilustración 70. La gestión de la comunidad refinada en flujos

La descripción de los flujos de trabajo comienza indicando qué tareas forma parten del flujo. Como ejemplo, la Ilustración 71 muestra las tareas que componen el flujo de trabajo para dar de alta a un nuevo miembro en la comunidad.

Las altas en la comunidad se deciden primero en función de la comparación de los gustos del solicitante y los gustos conocidos de los miembros existentes, y después en función de la opinión del resto de los miembros de la comunidad. Para no molestar a todos, se elige un subconjunto de miembros para que emitan su voto. Si la mayoría vota afirmativamente, el solicitante es admitido como nuevo miembro.

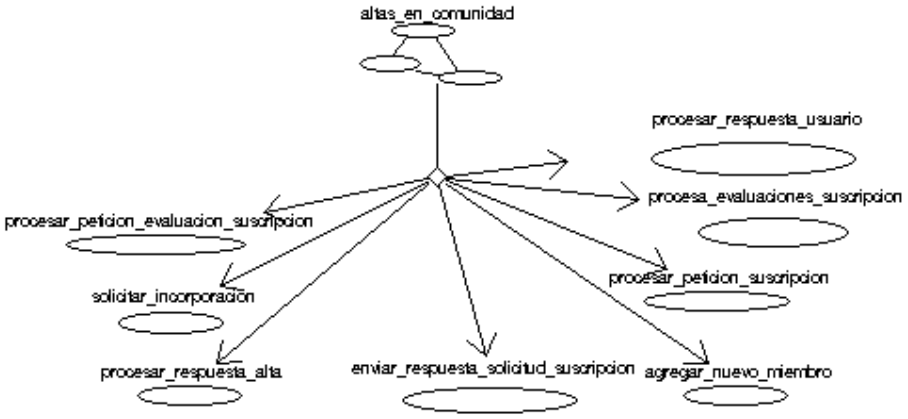


Ilustración 71. Tareas asociadas al flujo de trabajo para dar de alta en una comunidad

Las tareas anteriores se interconectan según las relaciones de la Ilustración 72. Estas relaciones muestran cómo se conectan las entradas y salidas de cada tarea. La primera tarea en ejecutarse es *solicitar incorporación*, ejecutada por el solicitante. A continuación, la comunidad procesa la solicitud con *procesar petición suscripción*. Si la solicitud coincide con los gustos de los usuarios ya registrados, se procede a ejecutar *enviar petición de evaluación de suscripción* para conocer la opinion de los miembros de la comunidad. Los miembros deciden qué hacer mediante *procesar petición evaluación suscripción*. La comunidad recolecta todas las opiniones de los miembros de la comunidad con *procesa evaluaciones de suscripción*, tras lo cual se procede a incluir el nuevo miembro (*agregar nuevo miembro*) y a informar al solicitante (*procesar respuesta alta*).

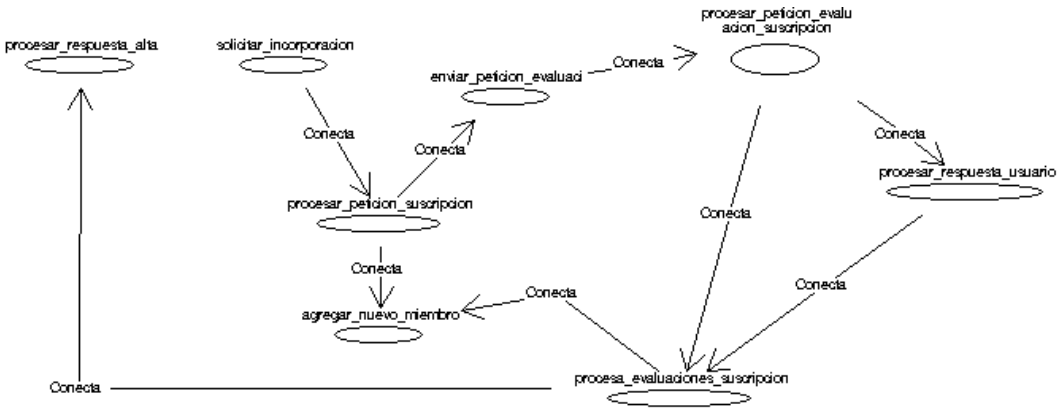


Ilustración 72. Asociación de tareas en el flujo de trabajo para dar de alta en una comunidad

Cada tarea de la Ilustración 72 es asignada a tres posibles roles: el *gestor de suscripciones de la comunidad*, el *solicitante de alta en comunidad* y los *miembros de la comunidad*. El *gestor de suscripciones* se encarga de tramitar las solicitudes de altas y bajas en la comunidad. El *solicitante de alta en comunidad* es el que ha pedido ingresar en la comunidad. El rol *miembro de la comunidad* designa a cualquier agente que pertenezca a una comunidad. La asociación de estos roles con las tareas que tienen que saber ejecutar se muestran en la Ilustración 73.

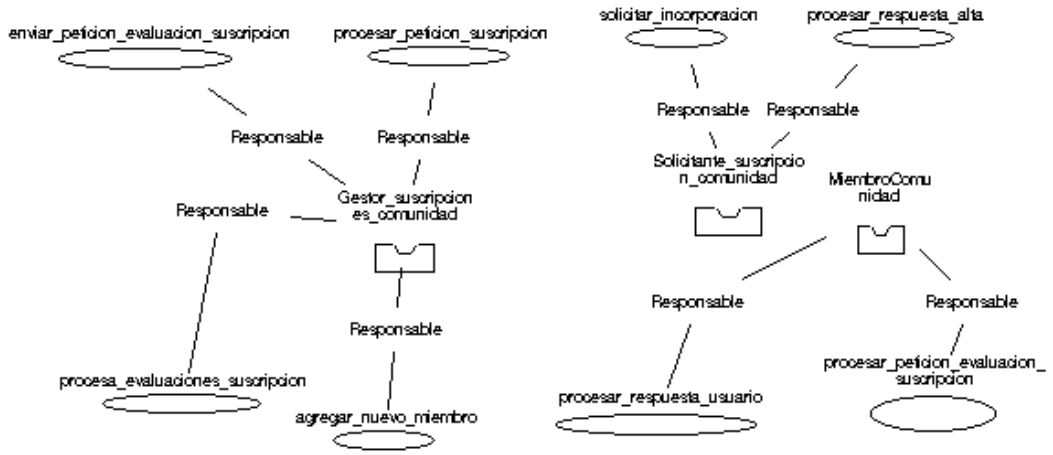


Ilustración 73. Responsabilidades asociadas a roles dentro de un flujo de trabajo

Como hay tareas que no se van a ejecutar por el mismo agente (no se espera que estos tres roles sean desempeñados por un único tipo de agente), se hace que la primera tarea que inicia el flujo de trabajo produzca una interacción donde se indica quién ejecuta qué y cuándo. En el flujo de trabajo se indica una tarea que ocasiona la interacción y otras tareas que se ejecutan como consecuencia de esta primera (Ilustración 74).

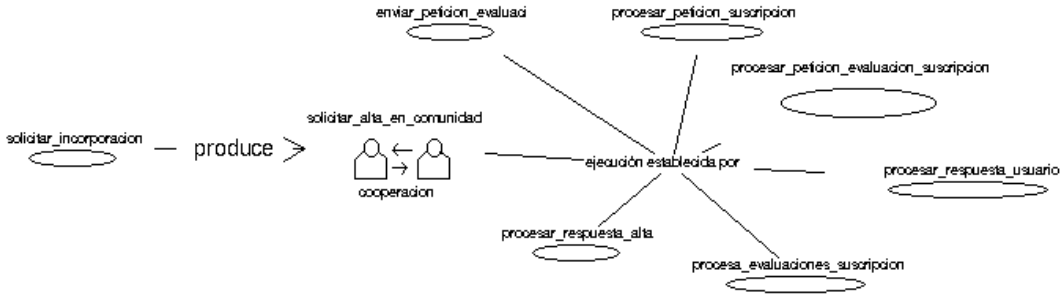


Ilustración 74. Tareas del flujo de trabajo asociadas a la interacción para dar de alta en una comunidad

La ejecución de los flujos de trabajo obedece a condiciones mentales de sus ejecutores. Estas condiciones mentales se explicitan dentro de la interacción, indicando por qué tiene lugar la primera y siguientes unidades de interacción.

Para completar la definición del flujo de trabajo, habría que indicar cómo se decide quién rellena los roles que participan en la interacción *solicitar alta en comunidad*. Esta interacción utiliza los roles *gestor de suscripciones de la comunidad*, el *solicitante de alta en comunidad* y los *miembros de la comunidad*, lo cual en el análisis es suficiente. Sin embargo, para completar una definición exhaustiva del flujo de trabajo hace falta señalar criterios para decidir quién desempeñará estos roles. Esta información se proporciona con instanciaciones de *Consulta Entidad Autónoma*. En este caso, los agentes concretos se determinan a través de hechos que deben existir en el agente ejecutor, como indica la Ilustración 75. La relación etiquetada como *Contiene* es simplemente azúcar sintáctico para

denotar que la entidad mental pertenece un estado mental asociado a *Consulta Entidad Autónoma*.

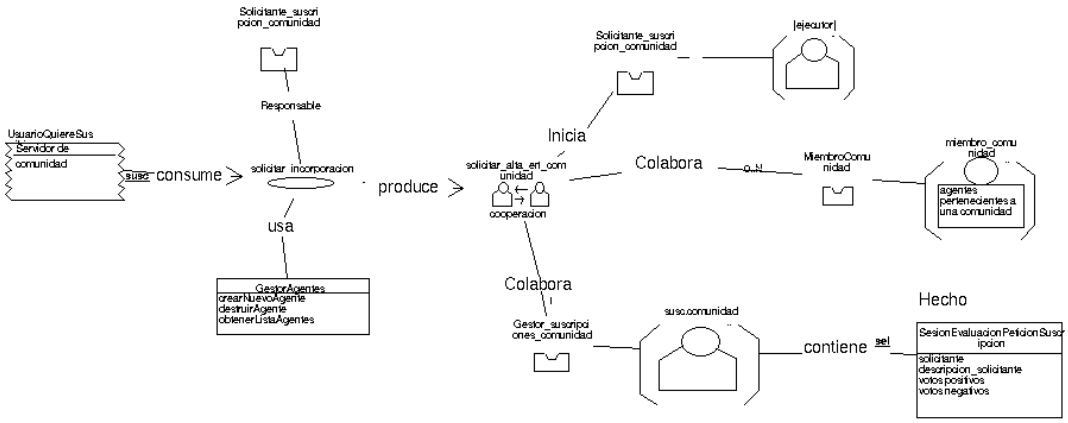


Ilustración 75. Descripción de qué agentes desempeñan los roles en la interacción que define la ejecución de tareas del flujo de trabajo para dar de alta en la comunidad

El *Agente Personal*, que jugará el rol *solicitante suscripción*, no puede saber quienes jugarán el rol de *miembro comunidad* porque no pertenece todavía a la comunidad. Sólo puede indicar quién actuará de *gestor de la comunidad*, que será el gestor asociado a la comunidad donde se quiere suscribir. Este gestor se incluye dentro de un slot (*comunidad*) del evento etiquetado con *susc*. La localización del agente se hace utilizando *Gestor Agentes* que proporciona la ubicación de todos los agentes del sistem.

El gestor de la comunidad fijará más tarde el resto de los roles con el hecho *SeleccionEvaluacionPeticionSuscripcion*. Este hecho contiene los colaboradores de en la interacción, que son los miembros de la comunidad a los que se preguntará acerca de la admisión. Estos miembros se expresan mediante una *Consulta Requisitos Agente* que describe los agentes involucrados. La descripción es una instancia de *Descripción Agente* denominada *agentes pertenecientes a una comunidad* que codifica los requisitos mediante un predicado PROLOG. Este predicado dice que los agentes seleccionados son aquellos cuyo identificador pertenece a la lista *sel.colaboradores* (todas las posibles unificaciones de p*ertenece(Miembro_comunidad, sel.colaboradores)*).

2.5.4 Integración con otros meta-modelos

El meta-modelo de organización mantiene fuertes dependencias con los otros meta-modelos, ya que muestra a nivel general cada entidad representada por los otros meta-modelos.

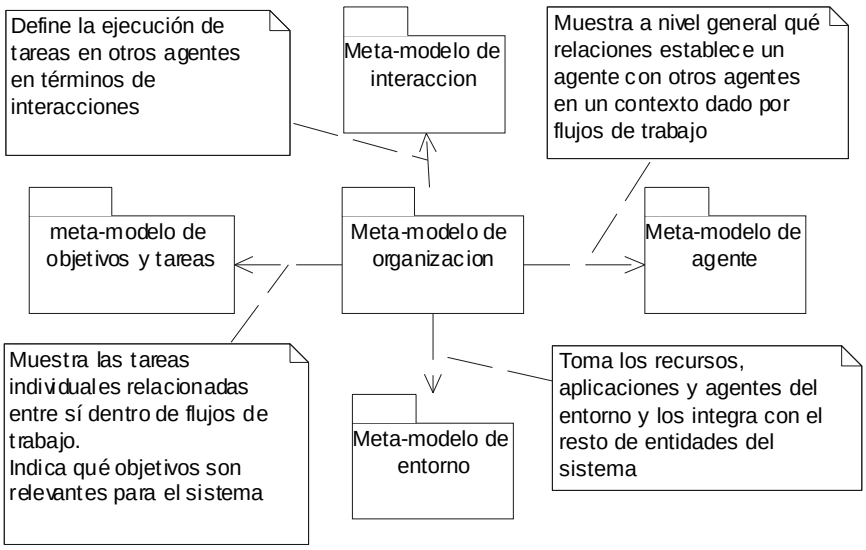


Ilustración 76. Meta-modelo de organización. Relaciones con otros meta-modelos

Cuando una tarea implica la ejecución de otras tareas en otros agentes, se necesitan instancias del modelo de interaccion que muestren quién ejecuta qué y cuándo. Estas instancias recogen dentro de diagramas de colaboración o diagramas GRASIA las distintas tareas ejecutadas indicando bajo qué condiciones se deben lanzar.

El meta-modelo de agente se centra en describir los agentes que aparecen en la organización, reproduciendo las capacidades que se descubren dentro de los flujos de trabajo, como el desempeño de roles, responsabilidades o condiciones mentales de subordinación a los deseos de otros agentes.

El meta-modelo de objetivos y tareas muestra cómo las tareas ejecutadas llegan a satisfacer los objetivos de la organización. También muestran de forma cómo son las tareas de forma aislada, qué consumen, qué producen y qué aplicaciones y recursos precisan. Por último, también sirve para relacionar las motivaciones de cada agente (instancias de *GTPersigue*) con las motivaciones asociadas a flujos de trabajo (instancias de *WFPersigue*).

El meta-modelo de entorno proporciona descripciones detalladas de los recursos, aplicaciones y agentes que existen separadamente del sistema en desarrollo. Estas entidades se utilizan para describir los flujos de trabajo y la estructura de la organización.

Las restricciones inducidas por estas relaciones son las siguientes:

1. Las tareas de las que se responsabiliza un agente, así como los roles que juega, deben aparecer también en el modelo de agente donde éste se describe.
2. Las tareas que aparecen en la organización han de ser asignadas a algún agente o rol.
3. Las tareas que se conectan con otras tareas asignadas a otros agentes o a roles que no puedan ser desempeñados por el mismo agente, deben aparecer en una *interacción*.

4. Las tareas que aparecen conectadas deben compartir al menos una entrada y una salida.
5. Todos los recursos y aplicaciones que aparecen en el modelo deben aparecer en algún modelo de entorno.
6. Todos los agentes que aparezcan deben tener un modelo de agente donde se les describa. En estos modelos, los agentes se asociarán a los roles identificados en los flujos de trabajo de la organización.
7. Las tareas deben especificarse con detalle dentro de un modelo de tareas y objetivos. En este modelo se describirán las entradas y salidas de la tarea, recursos consumidos y aplicaciones utilizadas. Además, la ejecución de la tarea se debe relacionar con la satisfacción de alguno de los objetivos previstos.

2.6 Meta-modelo de Entorno

En este meta-modelo el propósito no es generar representaciones del mundo en el que se ubica el SMA como hace, por ejemplo, [Ferber 99]. La experiencia en Inteligencia Artificial ha demostrado lo difícil de esta tarea. [Russell y Norvig 95] muestra que los entornos pueden ser accesibles (capacidad para percibir todo el entorno) o inaccesibles, deterministas (dado una codificación del estado del entorno y una acción ejecutada, se puede predecir el estado siguiente) o no-deterministas, episódicos (la experiencia del agente puede segmentarse en episodios independientes) o no episódicos, estático (el estado del mundo no cambia mientras el agente delibera) o dinámico, discreto (existe un conjunto finito de variables a observar y un conjunto finito de acciones posibles) o continuo. Lógicamente, el peor caso es un entorno inaccesible, no-episódico, dinámico y continuo, que suele ser la situación más común en problemas de control en tiempo real. El tipo de entorno influye en cómo se define la percepción del agente, sus acciones y su control.

Desde un punto de vista más pragmático, se ha decidido aplicar el enfoque de *Situated Automata* [Rosenschein y Kaelbling 95] y redes neuronales [Zilouchian 00]: discretizar el entorno utilizando un conjunto finito de variables observables. Desde este punto de vista, el problema se asemeja más a la situación con que se enfrentan los que diseñan sistemas empotrados. Esta discretización se aplica en dos direcciones: categorizar el tipo de entidades relevantes en el entorno y restringir la interacción con las mismas. Así, el entorno contendrá sólo recursos, aplicaciones y agentes, y se limitará la percepción y actuación de los agentes.

La aparición de recursos y su inclusión a los agentes se remonta a IRMA [Bratman, Israel y Pollack 88], donde la capacidad de razonamiento está limitada por los recursos disponibles, tiempo de procesador principalmente. Estos recursos van desde el número de procesos o hebras requeridos hasta el número de conexiones necesarias con bases de datos. Las aplicaciones pueden modelarse de distintas formas, en concreto, se estudiarán dos: abstracciones con objetos y con agentes. Por último, se considera la posibilidad de que existan otros agentes en el sistema. Para este caso, el comportamiento de estos agentes es modelable con los meta-modelos ya presentados.

Los mecanismos de percepción de los agentes son función directa de cómo se describen las entidades del sistema. La literatura muestra que hay dos mecanismos básicos: muestreo

(*polling*) y notificación. En ese meta-modelo de entorno se definen las asociaciones de los agentes con estas entidades y el tipo de mecanismo utilizado para recibir datos del entorno. Las acciones sobre el entorno se asume que son llamadas a las operaciones que se definen en estas entidades.

La evolución del entorno se considera como la suma de la evolución de sus componentes (recursos, entidades y agentes) influenciada por las dependencias entre ellos (una acción sobre un recurso puede afectar a otras entidades). No se pretende llegar a modelar este tipo de sistemas, sin embargo, se quiere dejar la puerta abierta a desarrollos especializados en estas áreas. Para ello se añaden asociaciones para indicar la existencia de efectos colaterales (*side-effects*) y, para tener un problema tratable, se ha simplificado el problema de la representación. El estado del entorno se modela bajo la hipótesis de que el comportamiento del sistema resultante de tener en cuenta estas asociaciones no es distinguible del que no las considera.

Con esta simplificación, se pueden concebir las tareas como transformadoras del estado del mundo. La concurrencia de diferentes tareas que modifican el estado del entorno es algo a considerar en la definición del flujo de trabajo de la organización. Al final de estas consideraciones, se puede llegar a la conclusión errónea de que se pretende modelar completamente un entorno y las repercusiones de las acciones sobre él. Ciertamente, sería ideal el poder tener una formalización completa de entorno y efectos de las tareas. No obstante, es harto complejo conseguir algo así, sobre todo cuando se manejan más a menudo especificaciones informales que formales y cuando existe la posibilidad de que el entorno sea modificado por otras entidades ajenas al sistema en desarrollo.

2.6.1.1 Aplicaciones

Las aplicaciones además de servir como actuadores y sensores de los agentes, se utilizan para integrar software en el desarrollo del SMA. Para definir aplicaciones se ha elegido una solución de ingeniería del software, concretamente del *Rational Unified Process* (*RUP*) [Jacobson, Rumbaugh y Booch 99]. El entorno aparece como subsistemas con unas interfaces que deben utilizarse para interactuar con ellos (requisitos funcionales), como notas asociadas en los casos de uso (requisitos no funcionales) o bien dentro de los diagramas de despliegue (*deployment*) (requisitos físicos).

Ahora bien, como en RUP, existen varias opciones para modelar una aplicación:

- **Usando objetos.** Las aplicaciones se recubren con una capa de objetos. La interactuación con estos componentes se define en términos de conceptos de UML (interfaces, diagramas de estados, diagramas de colaboración). Con esta solución, se integran subsistemas como sistemas propietarios o bases de datos, definiendo una interfaz y un comportamiento asociado al object.
- **Usando agentes.** Genesereth [Genesereth 94] propone tres formas de convertir en agente: por transductor (*transducer*), mediante recubrimiento (*wrapping*) y reescritura. El primero consiste en modelar la entidad como un objeto y asociar este objeto con un agente, el cual se encarga de mediar con el resto de agentes. El segundo consiste en incrustar la entidad dentro de un agente, de tal forma que el agente redirija las peticiones a la propia entidad. El tercero consiste básicamente en reescribir la aplicación.

Dependiendo del dominio, una tendrá ventajas sobre las otras. La experiencia dicta que la solución menos costosa es la primera, ya que la segunda implica, además de generar objetos, el generar un agente o bien rescribir completamente la aplicación, lo cual es, en la mayoría de los casos, inaceptable.

Las aplicaciones pueden emplearse para modelar servicios pasivos, esto es, un conjunto de operaciones que no requiere la interacción con ningún agente y que son empleadas por varios agentes. Ejemplos de tales aplicaciones serían servicios de nombrado (*Naming service*), servicios de emparejamiento (*matchmaking service*) o gestión del ciclo de vida de agentes (*Life cycle management*). Estos servicios pueden existir previamente en el entorno o no. Para el segundo caso, se hablará de *aplicaciones internas*.

De forma más frecuente, servirían como interfaz con el mundo real. Así la generación de estas aplicaciones estaría relacionada con el estudio del entorno para obtener, siguiendo a [Barber, Botti y Onaindia 94], un conjunto de variables de entrada, que representan datos accesibles en un momento dado mediante muestreo o por disparo de interrupciones, y salida, que sirven generalmente como parámetros de las acciones a ejecutar como reacción del sistema ante los cambios del entorno. Estas variables, aquí aparecerían como métodos asociados a las aplicaciones, para lectura de datos o actuación sobre el entorno.

2.6.1.2 Recursos

La integración de recursos en el desarrollo se toma de TAEMS [Decker 96;Decker y Lesser 95;Decker 95], donde los recursos se categorizan en:

- **Consumibles**. Son recursos que con el uso se agotan pero que son restituibles.
- **No consumibles**. El recurso se auto-restituye al término de la acción realizada sobre él.

Todos los recursos pueden pertenecer a un agente o ninguno (el que lo usa es el poseedor temporal). La capacidad de estos recursos se define con tres atributos del mismo tipo (real, entero, tipo compuesto). Los atributos son *estado*, *nivel de uso mínimo* y *nivel de sobrecarga*. El uso de un recurso hace que su estado decrezca hasta el nivel de uso mínimo, a partir del cual el recurso se inhabilita. Otra situación de inhabilitación es el restablecimiento excesivo del recurso, que incrementa su estado por encima del nivel de sobrecarga.

Las tareas se relacionan con los recursos mediante tres relaciones: consumición, producción y limitación. La relación de producción permite que una tarea restituya un recurso. La de limitación impone precondiciones para lanzar una tarea en función de la disponibilidad de recursos. Finalmente, la de consumición indica uso de un recurso, lo que conlleva una disminución del estado del recurso.

Esta forma de modelar los recursos, sin embargo, tiene un inconveniente: la aparición de interbloqueos, exclusión mutua e inanición. Se necesita incluir en la solución medios para gestionar los recursos de tal forma que todas las tareas tengan posibilidad de ejecutarse y que se elimine el riesgo de interbloqueo. La solución más sencilla consiste en centralizar la gestión de recursos e implementar una política FIFO (First In First Out) no expropiativa de gestión de tareas donde se ejecuten las tareas si existen todos los recursos necesarios para ello. Los recursos se piden de forma atómica, no por separado y se exije que no se pidan más recursos hasta la conclusión de la tarea, con lo que se rompen una de las cuatro condiciones de interbloqueo [Coffman, Elphick y Shoshani 71] (estrategia de prevención de interbloqueo

[Tanenbaum y WoodHull 97]). Esta solución es poco elegante, pero satisface las necesidades actuales de este trabajo.

Descripción del recurso	Atributos	Tipo (TAEMS)
descriptores de ficheros	número máximo, número utilizado	no-consumible
hilos de ejecución	número máximo, número utilizado	no-consumible
memoria	máxima disponible, memoria utilizada	no-consumible
dispositivos de salida	número de dispositivos disponibles, cantidad utilizada, exclusividad	consumible
dispositivos de entrada	número de dispositivos disponibles, cantidad utilizada, exclusividad	consumible
unidad de almacenamiento	espacio libre, espacio utilizado,	consumible
sockets	máximos disponibles, número utilizados	no-consumible
ancho de banda	máximo disponible, número utilizado	no-consumible.

Tabla 4. Enumeración de ejemplos de recursos

Por último, queda por determinar qué recursos son esperables en un sistema. Con este objetivo se ha elaborado una lista de recursos extraida de dos fuentes: los recursos reconocidos en la parametrización del núcleo de sistema operativo de Linux v. 2.4.X y el gestor de seguridad de Java. La elección del primero se basa la experiencia de implantación en diferentes tipos de máquinas que respalda Linux y en su condición de código abierto. La elección del segundo se fundamenta en la portabilidad de código que caracteriza JAVA y en la prioridad asignada a cuestiones de seguridad, que ha llevado a identificar recursos que necesitan los programas JAVA en múltiples plataformas. La lista se ha reducido a la tabla **Tabla 4**.

Los atributos mencionados son compatibles con la notación de TAEMS, ya que todos ellos son expresables con umbrales y el estado actual. De hecho, en la tabla se ha añadido una columna para categorizar el recurso según la taxonomía de TAEMS.

2.6.2 Presentación del meta-modelo de Entorno

La presentación del meta-modelo de entorno comienza restringiendo el tipo de elementos que van a aparecer. Se distinguen tres posibles tipos: agentes, recursos y aplicaciones. Por recurso se entiende toda aquel objeto del entorno que no proporciona una funcionalidad concreta, pero que es indispensable para la ejecución de tareas y cuyo uso se restringe a consumir o restituir. Cuando el uso sea más complejo, como la funcionalidad requirida de

una base de datos, se empleará el término *aplicación*. Por último, la denominación de agente se emplea cuando la entidad satisfaga el principio de *racionalidad*.

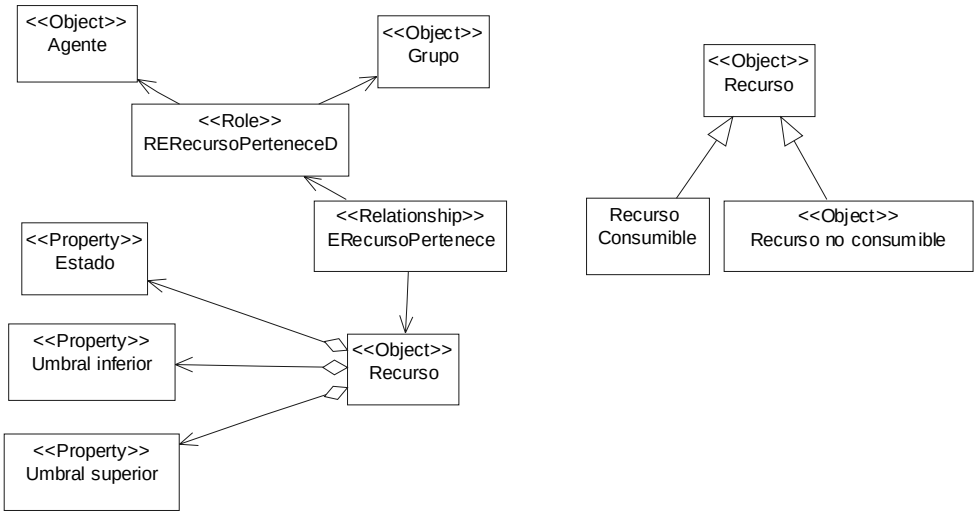


Ilustración 77. Meta-modelo de entorno. Recursos.

Los recursos pertenecen a un agente o a un grupo (*RERecursoPertenece*). Esta relación es similar a (**Tabla 4**

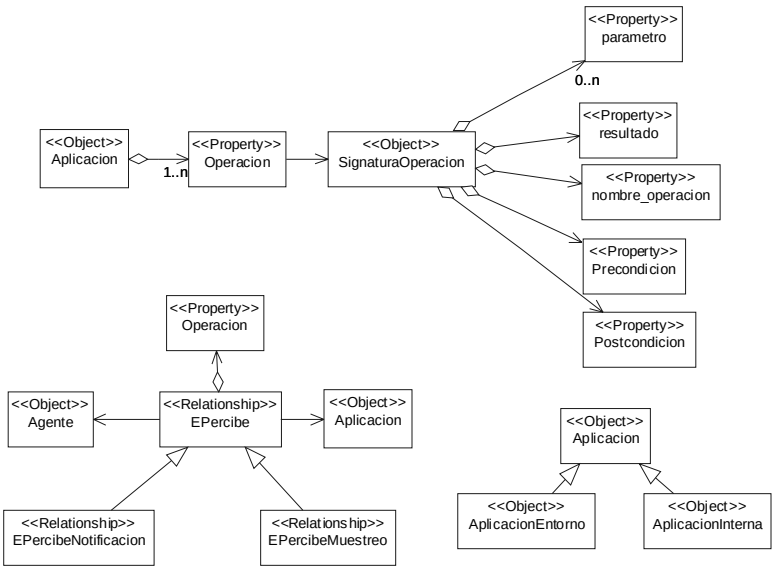


Ilustración 78. Meta-modelo de entorno. Aplicaciones y percepción

Las aplicaciones se caracterizan por poseer un conjunto de operaciones con una signatura convencional. Las precondiciones y postcondiciones se trasladan directamente de

las especificaciones de la aplicación (las aplicaciones ya están desarrolladas así como la interfaz para interactuar con ellas) cuando esta sea un software existente. Estas operaciones se utilizan para modelar la percepción del agente. Inicialmente se distinguen sólo dos tipos: percepción: por muestreo y por notificación. En ambos tipos se asocia la percepción (*EPercibe*) con una operación concreta. En el caso de muestreo, *operación* es una operación que se va a ejecutar con una frecuencia determinada. Tiene sentido cuando *aplicación* representa un dispositivo hardware que hay que muestrear. En el caso de notificación, se percibe únicamente si se recibe un resultado distinto del que se recibió en la última invocación.

Las aplicaciones pueden existir con anterioridad al desarrollo actual (*AplicacionEntorno*) o ser desarrolladas *ad-hoc* para los propósitos actuales (*AplicacionInterna*). Las primeras se obtienen de la captura de requisitos, mientras que las segundas se generan mediante técnicas convencionales de ingeniería del software.

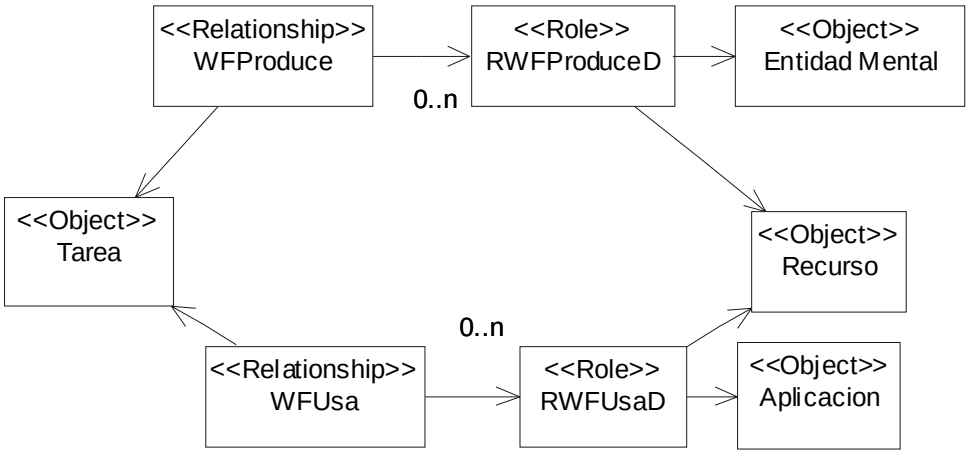


Ilustración 79. Meta-modelo de entorno. Tareas, recursos y aplicaciones

Tareas, recursos y aplicaciones se relacionan en la Ilustración 79. Como ya se ha visto en el meta-modelo de tareas y objetivos y en el meta-modelo de organización, la tarea consume y restituye recursos, pero también usa aplicaciones.

2.6.3 Ejemplos

Se diseña un asistente del sistema operativo que gestiona la ubicación de ficheros en el disco duro. La gestión consiste en reorganizar los ficheros automáticamente según criterios extraídos por el agente extraídos del estudio de la relación fichero-directorio en función del tipo de fichero, palabras clave presentes en el contenido del fichero o en el nombre del fichero.

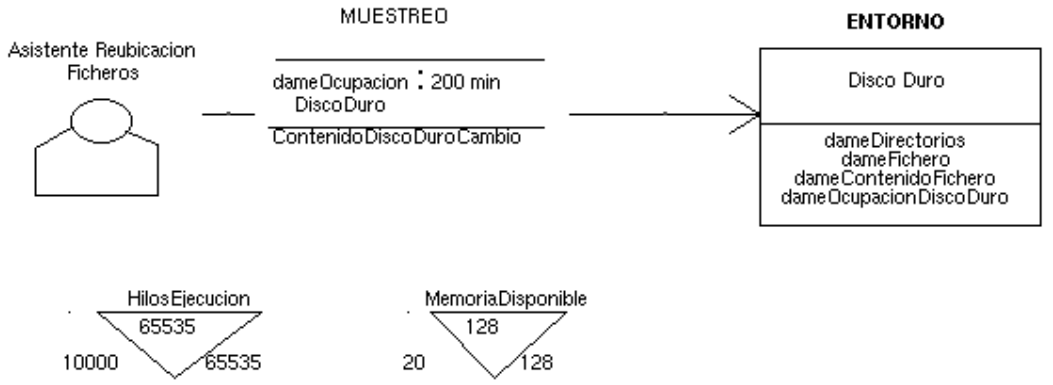


Ilustración 80. Descripción de aplicaciones y recursos para el caso de estudio

Para no afectar el funcionamiento normal del usuario, el agente atiende a la presencia de los recursos *Hilos de Ejecución* disponibles y *Memoria Disponible.* En la definición de los recursos se establecen el nivel mínimo de recursos que se debe observar (10000 hilos de ejecución y 20 Mb de memoria).

El agente respeta la organización de directorios existente ayudando a ubicar los nuevos ficheros en su lugar. El agente se activa cuando se produce un cambio en el espacio de disco ocupado (ver Ilustración 80). La consulta de espacio ocupado se realiza cada 200 minutos a través de la operación *dameOcupacionDiscoDuro* de una interfaz de acceso al disco duro (*Disco Duro*). Cuando se produce un cambio en el tamaño del disco duro, se genera un evento *Contenido Disco Duro Cambio* según lo establecido en la relación de percepción por muestreo de la Ilustración 80. Este hecho posibilita la ejecución de la tarea *Analizar Ficheros* tal y como indica la Ilustración 81.

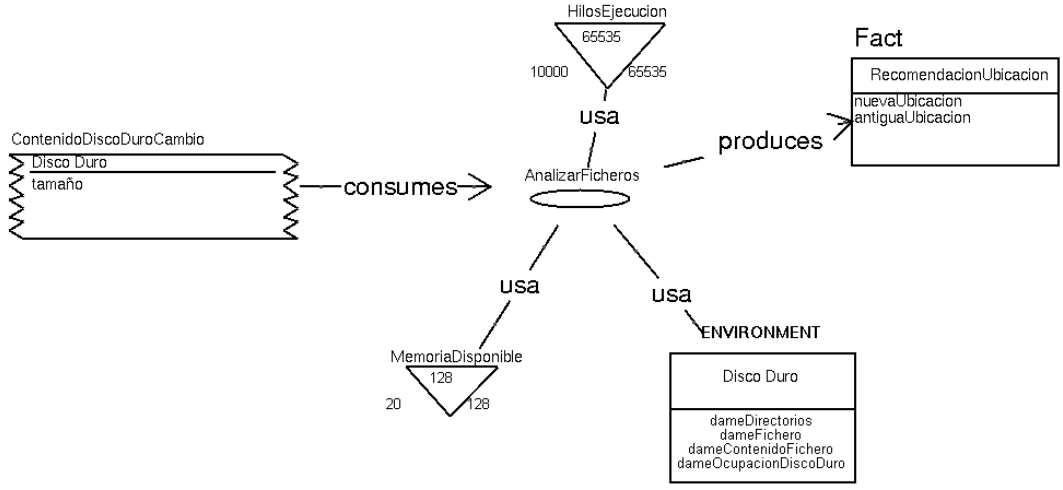


Ilustración 81. Tarea asociada al asistente para deducir la nueva ubicación de los ficheros

La tarea *Analizar Ficheros* consume los recursos *Hilos Procesador* y *Memoria Disponible* y el evento *Contenido Disco Duro Cambio*. Si no hubiera suficientes recursos, la tarea simplemente no se ejecutaría. *Analizar Ficheros* utiliza la aplicación *Disco Duro* (interfaz para acceder al disco duro) para obtener información acerca de qué existe en el dispositivo de almacenamiento. Como resultado de la tarea, se obtiene un hecho *Recomendación Ubicación* donde se indica qué movimientos de ficheros son necesarios.

El asistente puede coexistir con otros para lograr la automatización de tareas del Sistema Operativo. Los asistentes se dispondrían mediante un modelo de organización como el de la Ilustración 82.

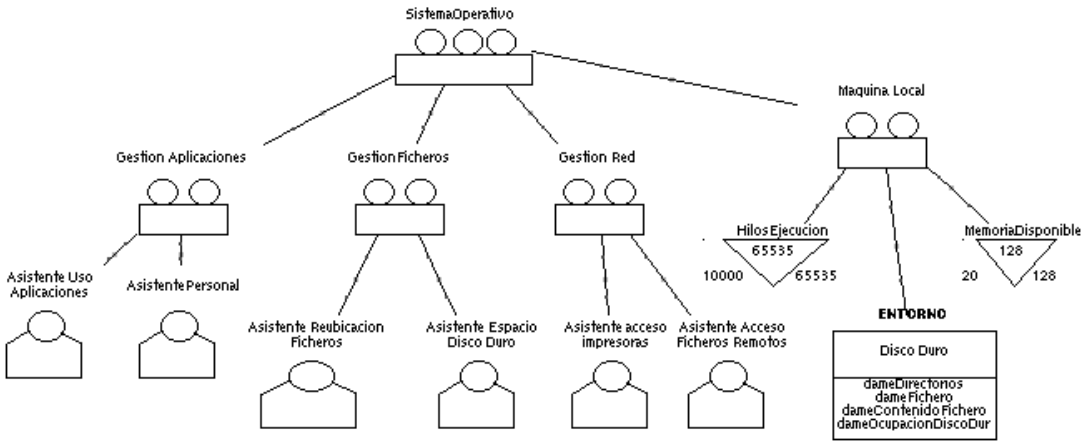


Ilustración 82. Organización de agentes asistentes en la máquina del usuario

Los recursos de la máquina del usuario se meten en el grupo *Maquina Local*. Estos recursos son utilizados por los agentes pertenecientes al resto de grupos (*Gestión Aplicaciones, Gestión Ficheros, Gestión Red*)

2.6.4 Integración con otros meta-modelos

La principal función del meta-modelo de entorno es identificar los elementos del entorno y relacionarlos con el resto de entidades del sistema. El meta-modelo de entorno se relaciona con tres meta-modelos (Ilustración 83). El entorno, tal y como ha sido planteado, no aporta nada a las interacciones, aunque se vea afectado por las tareas que son ejecutadas dentro de las mismas. No obstante estas consecuencias ya se reflejan en otros modelos.

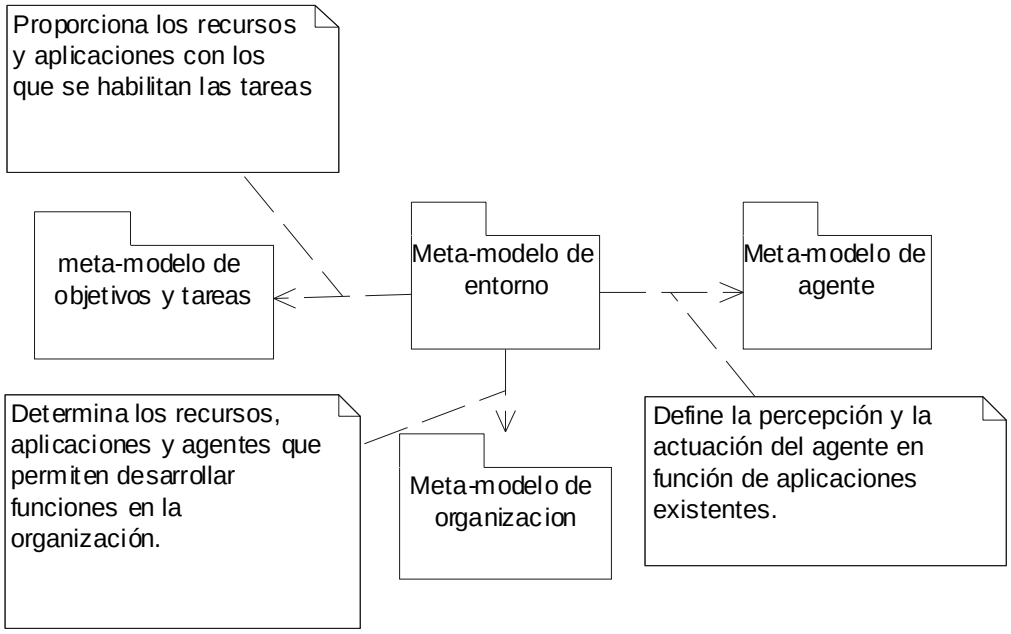


Ilustración 83. Meta-modelo de entorno. Relaciones con otros meta-modelos

Al meta-modelo de agente aporta la definición de la percepción de cada agente. Esta percepción, como se ha visto, se expresa mediante relaciones con aplicaciones. Al meta-modelo de organización le da el conjunto de recursos, aplicaciones y agentes disponibles. Por último, el meta-modelo de objetivos y tareas obtiene del entorno el conjunto de recursos que habilitan la tarea y las aplicaciones utilizadas para expresar las acciones realizadas.

Basándose en estas relaciones, se proponen los siguientes criterios de validación:

1. Todo recurso o aplicación que aparezca en los modelos de organización o tareas y objetivos, debe aparecer también en algún modelo de entorno.
2. Todo agente que aparezca en algún modelo de la organización o de agente y que según los requisitos del sistema deba percibir cambios en el entorno, debe aparecer en el de entorno asociado con una aplicación.
3. Las instancias de *aplicación* deben recoger el conjunto de operaciones descubiertas al detallar el funcionamiento de las tareas dentro de los modelos de tareas y objetivos.
4. Los recursos y aplicaciones deben asignarse a un grupo o agente. Esta información se indicar en un modelo de organización.

2.7 Conclusiones

Los meta-modelos presentados sirven como guía para construir modelos durante el desarrollo de un SMA. Por eso los meta-modelos deben comprender toda la información que a tener en cuenta en para especificar un SMA, aunque manteniendo abiertas distintas

estrategias de llevar el desarrollo. Es decisión del desarrollador decidir en qué orden se generan los modelos y con qué nivel de detalle. A este respecto, no existe ninguna restricción. De hecho, como se verá en el siguiente capítulo, se espera un desarrollo en paralelo de cada modelo teniendo en cuenta los criterios de integración presentados en cada meta-modelo

Los meta-modelos presentados incluyen resultados de la investigación en tecnología de agentes y áreas relacionadas. En cada uno se ha revisado brevemente qué parcelas de investigación habría que tener en cuenta y cómo influyen éstas en cada meta-modelo. También se han considerado aspectos de consistencia de cada meta-modelo con respecto a otros meta-modelos.

En la aplicación de los meta-modelos a los ejemplos y el caso de estudio final, se ha utilizado una herramienta de meta-modelado (METAEDIT +) en la que se han introducido los meta-modelos vistos a lo largo de este capítulo para poder probar la efectividad de las soluciones propuestas. Esta herramienta, además, asegura que los meta-modelos son seguidos con rigor.

Frente a otras metodologías, la novedad de la propuesta es que se considera el desarrollo de SMA en problemas pequeños que no impliquen un coste elevado y en problemas de naturaleza industrial donde sea necesario aplicar técnicas de ingeniería para abordarlos. El uso de los meta-modelos se ha ilustrado con ejemplos relacionados con temas de investigación actuales, como la planificación de tareas, filtrado colaborativo de información y diseño de agentes de interfaz. En el capítulo cuarto se mostrará un caso de estudio donde la aplicación de los meta-modelos se hace de acuerdo con las prácticas convencionales de ingeniería, más cercanas a la realidad industrial.

Capítulo 3. INTEGRACIÓN EN EL CICLO DE VIDA

Los meta-modelos son una gran ayuda en el proceso de desarrollo de SMA, ya que determinan qué entidades tienen que existir y cómo deben conectarse. Desde el punto de vista de ingeniería, lo que interesa de los meta-modelos, además de servir de guía, es cómo pueden ayudar a estructurar el desarrollo de SMA. Para estudiar la aplicación de los meta-modelos a procesos de ingeniería, se han tomado los meta-modelos como lenguaje de especificación del SMA, de forma similar a los diagramas UML como especificación de un desarrollo orientado a objetos. Partiendo de este lenguaje se estudian qué actividades y productos son necesarios para generar los modelos que constituyen la especificación del sistema. Estas actividades y productos son relevantes porque se pueden integrarse dentro de un proceso de ingeniería del software. Las actividades se pueden distribuir entre los diferentes componentes de un equipo de trabajo, estableciendo qué productos se esperan de cada actividad y cómo se combinan.

Como aplicación práctica de esta idea, en este capítulo se plantea la integración con el *Rational Unified Process* (RUP) [Jacobson, Booch y Rumbaugh 00] en las fases de análisis y diseño. Se ha elegido este modelo de desarrollo de software por las dependencias observadas entre los distintos meta-modelos. Al ser iterativo e incremental, es más sencilla la generación de modelos, ya que se pueden elaborar poco poco aplicando los criterios de integración de resultados de los modelos para asegurar su coherencia. Además, el RUP toma la arquitectura del sistema como guía del desarrollo, del mismo modo que en el capítulo anterior se señalaba que el modelo de organización constituía la espina dorsal de la especificación del SMA. Por último, el RUP utiliza *casos de uso* para determinar la funcionalidad del sistema y diagramas de colaboración y secuencia (entre otros) para describirlos. Ya se ha comentado la utilidad de las *interacciones* como generalización de diagramas de colaboración y diagramas de secuencia. Así, lo más lógico es utilizar *interacciones* como especificación del comportamiento de los *casos de uso*.

En este capítulo se presenta la integración propuesta definiendo primero el nivel de detalle a alcanzar en cada una de las etapas del RUP y después las diferentes actividades

requeridas para generar modelos. El nivel de detalle se obtiene a partir de la estimación hecha en el RUP para los modelos generados con UML. Las actividades se obtienen a partir de la experiencia obtenida en el desarrollo de casos prácticos, uno de los cuales se presenta en el siguiente capítulo, y se expresan con diagramas de actividades de UML. Como complemento al estudio de la generación de la especificación del sistema, en las dos últimas secciones se estudia cómo llevar a cabo la implementación del sistema de forma automatizada y qué elementos arquitectónicos deben existir en la arquitectura final del sistema.

3.1 Integración de los meta-modelos en el RUP

En el RUP, el esfuerzo del análisis y diseño se encuentra localizado en tres fases: inicio, elaboración y construcción. Dentro de cada fase se desarrollan las iteraciones (ciclos completos de desarrollo incluyendo análisis, diseño, implementación y pruebas) que construyen gradualmente el sistema. La Tabla 5 muestra el conjunto de actividades que según [Jacobson, Booch y Rumbaugh 00] se realizan en esas etapas. Como se puede apreciar, la principal diferencia en cada etapa consiste en el nivel de detalle alcanzado al final de las iteraciones que se realizan en cada una de ellas.

		FASES		
		Inicio	**Elaboración**	**Construcción**
	Objetivo	Hacer creíble que el sistema se puede construir	Hacer realidad la arquitectura	Hacer crecer el sistema
FLUJOS DE TRABAJO FUNDAMENTALES	**Análisis**	o Generar casos de uso y escenarios de casos de uso. o Expresar requisitos como casos de uso. o Elaborar un boceto de arquitectura	o Refinar casos de uso o Perfilar la resolución de los casos de uso relevantes para la arquitectura en función de paquetes, clases	o Estudiar resto de casos de uso
	Diseño	o Generar prototipo (interfaces gráficas, algoritmos que pueden demostrar la factibilidad del sistema)	o Diseño de las capas de la arquitectura. o Generar subsistemas y sus interfaces. o Elaborar bocetos del modelo de despliegue para sistemas distribuidos (nodos y red) o Expresar casos de uso en función de los subsistemas creados. o Diseñar las clases que componen los subsistemas	o No se añaden nuevos subsistemas. o Se diseña el resto de los casos de uso añadiendo nuevas clases o refinando las anteriores. o Generar un modelo de despliegue.
	Implementación		A partir de un conjunto reducido de casos de uso se plantea o Implementación de la arquitectura o Implementación de clases y de subsistemas relevantes para la arquitectura o Componer los elementos identificados en el diseño para el sistema.	o Lograr una arquitectura firmemente asentada, con todas sus clases y subclases implementadas o Realizar pruebas de que involucren varios módulos integrados para comprobar que la integración es completa y satisfactoria o Realizar planes de integración de los componentes en cada iteración
	Pruebas		o Seleccionar los objetivos que evaluarán la arquitectura o Diseñar procedimientos de prueba en base a esos objetivos o Realizar pruebas de integración entre componentes o Una vez integrado, plantear pruebas de sistema	o Continuar con las pruebas de la iteración anterior. o Añadir nuevas pruebas que tengan en cuenta el nuevo software integrado o nueva funcionalidad del software ya existente. o Evaluar las pruebas para verificar que se consiguen los objetivos planteados.

Tabla 5. Actividades a realizar en las etapas de inicio, elaboración y construcción.

La generación de modelos a partir de meta-modelos se guía por el conjunto de actividades que se presentarán a lo largo de este capítulo. Estas actividades no sustituyen las indicadas por el RUP, sino que se integran en el paradigma como complemento de los elementos de especificación ya existentes. De hecho, en las actividades de generación se mencionan *técnicas convencionales* refiriéndose a la utilización de notaciones como UML y modelos de desarrollo como RUP para atacar el diseño de elementos concretos.

Para orientar la integración, se plantean una serie de asociaciones entre elementos del RUP y elementos de los meta-modelos (ver Tabla 6).

Entidad MAS GRASIA	Entidad RUP
Agente	Clase
Organización	Arquitectura
Grupo	Subsistema
Interacción	Escenario
Roles, tareas y flujos de trabajo	Funcionalidad

Tabla 6. Asociaciones entre los elementos del RUP y entidades de MAS GRASIA

El *agente*, como la *clase*, define tipos. Lo que aquí se ha denominado *agente en ejecución* sería un *objeto* en el RUP. La *organización* equivale a la *arquitectura* en el RUP por su carácter estructurador. La *organización* da una visión global del sistema agrupando agentes, roles, recursos y aplicaciones y estableciendo su participación en los flujos de trabajo del SMA. El *grupo* es la unidad de estructuración utilizada en la organización. Su similitud con un *subsistema* se debe a que como éste, se utiliza para organizar elementos en unidades de abstracción mayores y define un conjunto de interfaces para interaccionar, los roles en este caso. La *interacción*, como se comentó en la introducción, se ve como una generalización de los diagramas de colaboración y secuencia. En el RUP, los diagramas de secuencia y colaboración se ven como *escenarios* que describen cada *caso de uso*. Por último, *roles*, *tareas* y los *flujos de trabajo* proporcionan el encapsulamiento de acciones que en el RUP dan *métodos* e *interfaces*.

		FASES		
		Inicio	**Elaboración**	**Construcción**
FLUJOS DE TRABAJO FUNDAMENTALES	**Análisis**	o Generar casos de uso e identificar realizaciones de los casos de uso con modelos de interacciones. o Esbozar la arquitectura con un modelo de organización. o Generar modelos del entorno para trasladar la captura de requisitos a los modelos	o Refinar casos de uso. o Generar modelos de agente para detallar los elementos de la arquitectura. o Continuar con los modelos de organización identificando flujos de trabajo y tareas. o Modelos de tareas y objetivos para generar restricciones de control (objetivos principales, descomposición de objetivos) o Refinar modelo de entorno para incluir nuevos elementos.	o Estudiar resto de casos de uso.
	Diseño	o Generar un prototipo con herramientas de prototipado rápido, como ZEUS o Agent Tool	o Centrar el modelo de organización en el desarrollo de flujos de trabajo. 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. o Expresar la ejecución de tareas dentro de modelos de interacción. o Generar modelos de agente para detallar *patrones de estado mental*.	o Generar nuevos modelos de agente o refinar los existentes. o Depurar la organización centrando el desarrollo en las relaciones sociales.

Tabla 7. Adecuación de las etapas del RUP a los meta-modelos presentados.

Estas asociaciones marcan prioridades a la hora de profundizar en la generación de los distintos modelos (agente, organización, interacción, entorno, objetivo y tareas). De acuerdo con estas equivalencias y el proceso del RUP, la generación de modelos se estructuraría

según indica la Tabla 7. Las distintas iteraciones consistirían en seguir las actividades de las subsecciones anteriores, ya dispuestas con un inicio y final.

Las fases de pruebas e implementación no se han incluido. La fase de pruebas no tiene por qué ser diferente de la del software convencional. El software final ha de satisfacer casos de uso identificados, y como indica la tabla Tabla 7, el concepto de casos de uso se mantiene. En cuanto a la implementación, se admite que el diseñador puede actuar como en diseños usuales, desarrollando elementos computacionales que hagan lo que está especificado. En este dominio, este enfoque no sería el más adecuado, ya que sería deseable aprovechar los desarrollos existentes, donde ya están resueltos los problemas de comunicación o implantación del control en los agentes. Por ello, se propone reutilizar el software de agentes personalizándolo con los modelos generados a partir de los meta-modelos de esta tesis. El proceso se denomina *parametrización* porque toma un software operativo y lo concreta con información extraída de los modelos generados. Desde este punto de vista, el software se ve como un resolutor general de problemas que ha de ser particularizado para el dominio actual. Aunque la *parametrización* puede realizarse manualmente, en este trabajo se propone automatizar en parte este proceso.

El proceso de *parametrización* implica ser capaz de recorrer los modelos generados y de poder extraer datos de ellos. Aunque la herramienta de soporte utilizada, METAEDIT+, no proporciona primitivas adecuadas para recorrer los modelos, permite traducir los modelos a otras representaciones. En este caso, se ha optado por traducir los modelos a términos PROLOG. Desde estos predicados, se ha diseñado un programa con el cual es posible volcar las especificaciones directamente en código fuente. Sólo se requiere que el código fuente que se parametriza esté marcado de una forma especial y que se generen las secuencias de datos a volcar. Para proporcionar más detalles de este proceso, puede consultarse la sección 3.7.

3.2 Generación de instancias del meta-modelo de agente

Al instanciar el meta-modelo de agente, se está describiendo cómo son los elementos fundamentales del sistema, los agentes. Debido a que lo normal es que la información reflejada en estos modelos sea una recolección de información presentada en los otros, no suele constituir un buen punto de partida.

La generación de instancias de este meta-modelo se organiza en torno a los resultados esperables durante el análisis y el diseño. Los resultados que se obtienen durante el análisis son los siguientes:

- **Funcionalidad del agente.** Un conjunto de roles que desempeña cada agente. Un conjunto de tareas asociadas a los agentes o a roles que estos jueguen.
- **Requisitos del agente.** Qué cualidades se esperan del agente. Como ya se ha mencionado, si se espera del agente que demuestre inteligencia o autonomía, hay que especificar con detalle qué tipo de inteligencia y qué tipo de autonomía se espera. A este nivel basta con una descripción en lenguaje natural acompañada de casos de uso que lo ejemplifiquen.

Los resultados a obtener durante el diseño se refieren al control del agente:

- **Restricciones de control.** Se representan mediante los estados mentales por los que pasa el agente a lo largo de las interacciones.

- **Medios de control**. Cómo es el control que asegura la transición entre los distintos estado mentales identificados. La especificación de estos medios puede hacerse en lenguaje natural o utilizando referencias a modelos de objetivos y tareas.

Para lograr estos resultados, se proponen las actividades de la Ilustración 85 e Ilustración 84:

1. **Identificar agentes usando el principio de racionalidad.** En el análisis, se buscarán entidades cuyo comportamiento se adapte al enunciado por este principio. Para ello, basta con ser capaz de explicar el comportamiento de la entidad con objetivos y tareas.

   - **Producto**. Un conjunto de instancias básicas del meta-modelo de agente con un único participante, el agente.

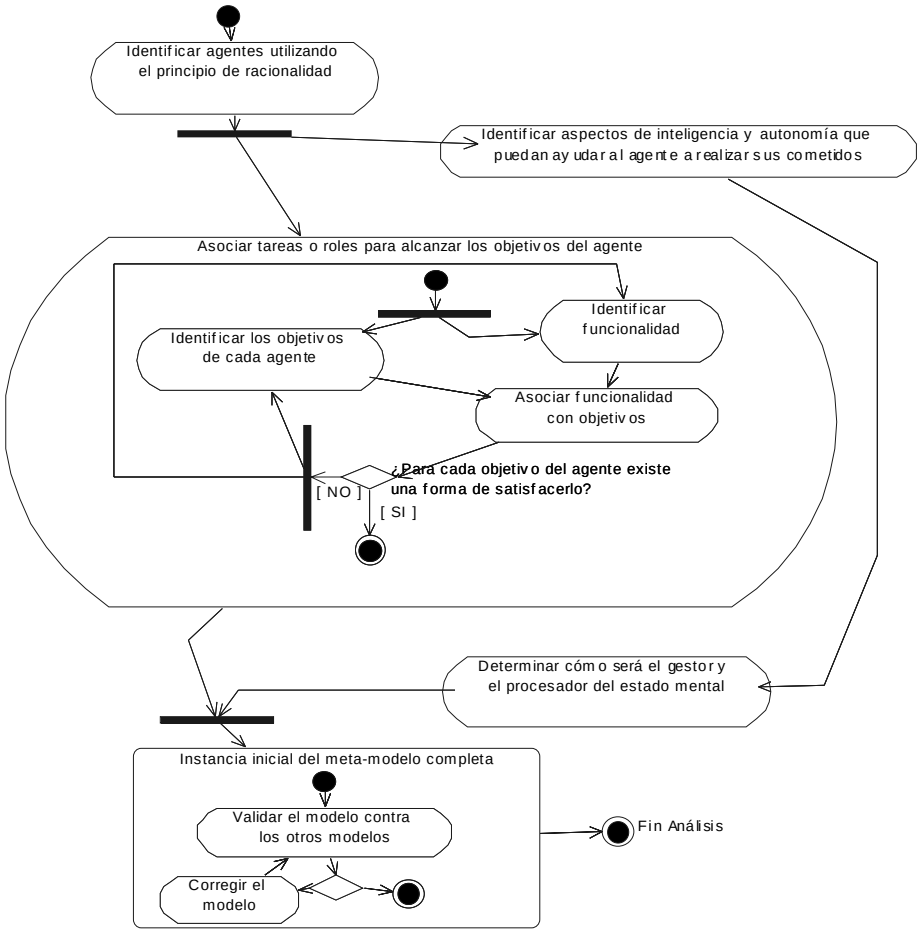


Ilustración 84. Actividades involucradas en la generación de modelos de agente en el análisis

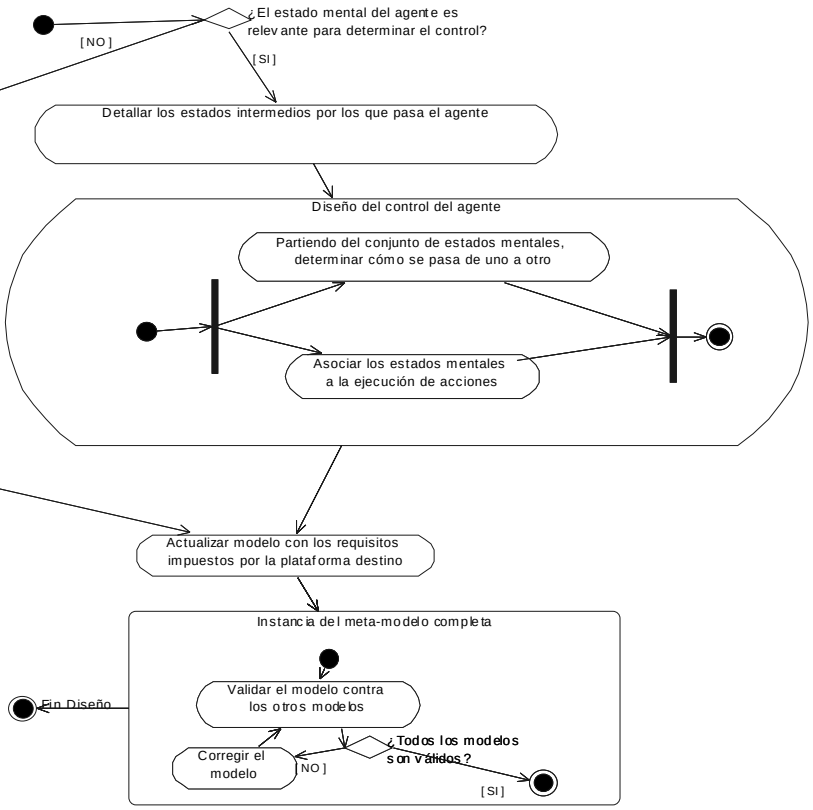


Ilustración 85. Actividades involucradas en la generación de modelos de agente en el diseño

2. **Asociar tareas o roles para alcanzar los objetivos del agente.** Según el esquema de control visto, los objetivos se alcanzan ejecutando tareas o desempañando roles que participan en interacciones que persiguen objetivos. Tras la identificación de objetivos, la asociación de tareas a objetivos o el desempeñar un rol sirve para clarificar el por qué una tarea existe.

2.a. **Identificar los objetivos de cada agente.** Siguiendo con la idea de principio de racionalidad, se hacen explícitas las metas del agente. La identificación de metas puede llevar implícita la identificación de tareas y viceversa.

- **Producto.** Un conjunto de objetivos asociados al agente por meta-relaciones *GTPersigue* o *WFPersigue*. Inclusión del objetivo en el estado mental si se trata del estado inicial del agente. Si pertenece a algún estado intermedio, indicarlo con insancias de *Consulta Agente* y asociar estas instancias con el tipo de agente.

2.b. **Identificar funcionalidad.** La funcionalidad del agente se determina mediante las tareas que sabe realizar, asignadas directamente (*AResponsable*) o como resultado de su participación en flujos de trabajo (*WFResponsable*), y los roles que desempeña. Mediante estos elementos se explica cómo se alcanzan los objetivos del agente. Una misma meta se puede alcanzar de diferentes formas. Pensando en las metas del agente, surgirán las tareas que deben ejecutarse o los roles que se desempeñan. Las tareas se asignan directamente al agente o se sobreentienden debido a asociaciones del agente con roles (si un agente desempeña un rol, entonces asume todas las tareas relacionadas con el rol). Los roles proceden del modelo de organización (de la especificación del flujo de trabajo) o bien del modelo de interacciones, al estudiar las diferentes interacciones que existen. La participación de roles en flujos de trabajo significa que tiene la responsabilidad de ejecutar tareas, mientras que la participación en interacciones significa que colabora en la consecución del objetivo de la interacción.

- **Productos.** Un conjunto de tareas y asociaciones *WFResponsable*, *AResponsable*. Un conjunto de roles y asociaciones *WFJuega*.

2.c. **Asociar funcionalidad a objetivos.** Justifica la ejecución de tareas o el desempeñar roles, asociando un sentido (el objetivo) a la funcionalidad del agente. Si se trata de tareas, el objetivo se asocia a las tareas mediante instancias de *GTAfecta*. Si lo que se ha identificado son roles, el objetivo se asocia al rol mediante instancias de *WFPersigue* o bien se puede omitir si se indica la participación del rol en interacciones, ya que las interacciones tienen asociado un objetivo. Más tarde, las tareas asociadas a los roles, dentro de flujos de trabajo o interacciones, se tendrán que asociar a objetivos relacionados, si no iguales, al del rol. Los objetivos también se pueden alcanzar mediante la satisfacción o fracaso de otros objetivos, esto es, sin necesidad de que se ejecuten otras tareas. Para determinar si es ésta la situación, hay que estudiar los modelos de objetivos y tareas buscando instancias de *GTDepende*. Hay aclarar que aunque algunos objetivos se puedan satisfacer de esta forma, el resto siguen teniéndose que alcanzar mediante tareas o roles.

- **Productos.** Asociaciones de tareas y objetivos (*GTAfecta*). Asociaciones de roles a objetivos *WFPersigue*.

3. **Identificar aspectos relevantes del comportamiento del agente.** Esta actividad se refiere a que hay que decidir qué cualidades debe tener el agente: si tiene

aprendizaje qué es lo que va aprender y para qué, si es inteligente qué tipo de inteligencia tendrá y para qué será usada.

- **Producto.** Descripción de las características deseables (tipo de inteligencia, tipo de aprendizaje, algoritmos y heurísticas aplicables) en el agente en lenguaje natural acompañada de un conjunto de casos de uso ilustrativos.

4. **Decidir, de acuerdo con las cualidades del agente, el tipo de procesador de estado mental.** Como se mencionó en al presentar el meta-modelo de agente, estas cualidades aparecen a nivel del meta-modelo encapsuladas dentro el *procesador de estado mental* y *gestor de estado mental*.

- **Producto.** Instancias de *Procesador* y *Gestor de estado mental* asociadas al agente mediante instancias de las meta-relaciones *ATieneProcesadorEM* y *ATieneGestorEM*. Estas entidades deben acompañarse de una descripción en lenguaje natural del tipo de gestión y procesamiento deseado utilizando la propiedad *Descripción*.

5. **Actualizar modelos con los requisitos de las plataformas destino.** Implica por lo general añadir información a la presentada en el modelo. Esta nueva información surge de la necesidad de instanciar un armazón software para que lleve a cabo las especificaciones del modelo. Según los requisitos impuestos por el armazón software, la especificación del sistema debiera modificarse. Así, un armazón software donde la comunicación entre agentes es importante, como JADE, influiría pidiendo un nivel de detalle alto en los modelos de interacciones.

- Productos. Nuevas entidades en el modelo o más detalle en las existentes.

6. **Detallar los estados intermedios por los que pasa el agente.** Los estados mentales intermedios provienen de los requisitos de los modelos de interacción y como resultado de la ejecución de tareas. En el primer caso, se tiene una referencia directa en las unidades de interacción a patrones de estado mental. En el segundo, hay que revisar las tareas del sistema para estudiar qué entidades mentales producen y qué entidades mentales se requieren.

- **Producto.** Nuevos modelos donde se reflejan los estados mentales requeridos.

7. **Diseño del control del agente.** El control del agente se diseña de acuerdo a las necesidades identificadas durante el análisis (inteligencia, autonomía, descripciones del *gestor* y *procesador de estado mental*). Es probable que el usuario decida dejar el control del agente en manos de un algoritmo, como en TAEMS [Decker 96;Decker y Lesser 95;Decker 95]. En tal caso esta actividad se delega en la actividad de generación de diseño convencional.

7.a. **Partiendo del conjunto de estados mentales, determinar cómo se pasa de uno a otro.** En esta actividad es determinante el tipo de *gestor de estado mental* elegido. A modo de guía, si se trata de un sistema basado en reglas, el gestor establecería qué entidades nuevas deben añadirse, cuáles modificarse y cuáles eliminarse en términos de operaciones *assert*, *retract* y *modify*. Para ver cómo se pasa de un estado a otro hay que estudiar las tareas asociadas con el agente, qué requieren en su ejecución y qué efectos tienen. Parte de los difgerentes estados mentales por los que pasa el agente se enumeran dentro de los diagramas GRASIA.

- **Productos**. Operaciones a realizar para asegurar la transición entre los diferentes estados mentales identificados. Las operaciones dependen del tipo de *gestor de estado mental* elegido.

7.b. **Definir la gestión de los objetivos de los agentes.** La gestión puede definirse en lenguaje natural si se considera apropiado. Será apropiado, por ejemplo, si la descripción provee de una definición sin ambigüedades de cómo se lleva a cabo. Si esta descripción no es posible, se propone detallar esta gestión con modelos de objetivos y tareas. En estos modelos, las tareas producirían, modificarían o destruirían objetivos. Estas tareas estarían guiadas por objetivos dedicados a la gestión de otros objetivos, de la misma forma en que existen meta-reglas en sistemas expertos que gestionan otras reglas. También habría que indicar cómo se gestionan los objetivos que gestionan objetivos. Para ellos se podría aplicar otra vez esta actividad. Se da por hecho que existe un conjunto de objetivos, por descubrir, cuya gestión es expresable en lenguaje natural de forma razonablemente no ambigua.

- **Productos.** Referencias a modelos de objetivos y tareas.

7.c. **Asociar los estados mentales a la ejecución de acciones**. Esta actividad depende del tipo de *procesador de estado mental* elegido. La decisión de ejecutar una acción puede ser fruto de la ejecución de una regla. La condición de ejecución se reflejaría en la parte izquierda de la regla mientras que la acción se codificaría en la parte derecha. La decisión también puede surgir como resultado de un proceso deliberativo complejo, como el propuesto por SOAR o IRMA. Estas alternativas pueden expresarse también utilizando modelos de objetivos y tareas.

- **Productos**. Descripción del proceso deliberativo que lleva a la ejecución de acciones en función del estado mental. La descripción depende del tipo de *procesador de estado mental* elegido. Es admisible la utilización de modelos de objetivos y tareas

8. **Validar el modelo contra los otros modelos.** La validación se realiza según los criterios de la sección 2.2.7 del capítulo segundo.

- **Productos.** Lista de inconsistencias

9. **Corregir el modelo.** Para hacer el modelo consistente con las instancias de otros meta-modelos

- **Productos.** Modificaciones a efectuar.

3.3 Generación de instancias del meta-modelo de interacciones

El meta-modelo de interaccion refleja el comportamiento deseado de un agente cuando recibe instrucciones de otro agente o bien requiere de otro agente para realizar una tarea. Como en los modelos de agentes, la generación de modelos de instancias se inicia en el análisis y se continúa en el diseño.

Para el análisis se ha fijado que los resultados esperables son:

- **Interacciones relevantes en el sistema.** Indicando participantes y objetivos perseguidos.

- **Esquema inicial de intercambio de mensajes.** Realizado con diagramas de colaboración o de secuencia UML. La realización de estos diagramas es menos costosa que los diagramas GRASIA.

Como en procesos convencionales de ingeniería, no es de esperar que los diagramas de colaboración resultantes sean definitivos. Estos diagramas sufrirán cambios cuando se entre en profundidad en la especificación de interacciones. Del diseño se espera :

- **Descripciones detalladas de las interacciones.** Han de ser tan detalladas como para permitir una generación automática de código. El nivel de detalle es función de las entidades que se escogen para implementar las interacciones.
- **Contexto de las interacciones**. Las interacciones contextualizadas en los flujos de trabajo de la organización y con información adicional acerca de las condiciones de participación en la interacción.
- **Conjunto de unidades de interacción asociadas a los participantes.** Cada participante en las interacciones comparte con los otros unidades de interacción. Esto forma parte de la descripción funcional del agente.

Para lograr estos resultados se proponen las siguientes actividades:

1. **Identificar los objetivos que se persiguen.** Se trata de determinar el principar producto de la interacción que es permitir a un conjunto de agentes alcanzar un objetivo que beneficia a todos los participantes. Cuál es este objetivo es un problema de estudio del contexto de la interacción. Así preguntándose cuándo tiene lugar la interacción, quién la inicia y dentro de qué flujo de trabajo estaría enmarcada, se deducen qué objetivos se están persiguiendo.

   - **Productos.** Un conjunto de objetivos asociados a la interacción mediante instancias de *IPersigue*.

2. **Identificar colaboradores e iniciadores de las interacciones.** Los participantes en la interacción aparecen si se estudia la interacción desde el punto de vista de la organización. En la organización, dentro del flujo de trabajo, las interacciones se producen como resultado de la ejecución de tareas. El ejecutor de las tarea que origina la interacción es el iniciador de la interacción (relación *IInicia*) y el resto de los participantes involucrados son colaboradores (*IColabora*).

   - **Productos**. Un conjunto de roles o agentes asociados a la interacción mediante relaciones *IInicia* e *IColabora*.

3. **Identificar la naturaleza de la interaccion.** La naturaleza de la interaccion influye en el tipo de control que se va a aplicar al agente. Si se necesita generar una secuencia de tareas para alcanzar un objetivo, la naturaleza es planificación. Si se trata de intercambiar mensajes en un orden concreto y un contenido predecible, se habla de cooperación. Si hay recursos limitados o existe algún bien que regula la predisposición de los agentes a colaborar, se tendrá negociación. Las negociaciones siguen reglas concretas, como el *contract-net* [Smith 80]. Cuando éstas no son rígidas y hay libertad para que un agente acapare recursos o bienes del sistema, se tendrá *competición*.

   - **Productos**. caracterización del tipo de interacción según la clasificación de [Huhns 00] (Ilustración 32).

4. **Generar una primera especificación con diagramas de colaboración o de secuencia de UML.** Los diagramas de colaboración y de secuencia son sencillos de identificar. Asumen que la interacción se realiza por intercambio de mensajes o bien por llamadas a procedimiento remoto. Aunque el contenido de la información intercambiada aún no es prioritario, sí que lo es el determinar de qué tipo son los mensajes que se están mandando y la secuencia en que son procesados. Esta información se ampliará más tarde con los diagramas de interacción GRASIA.

4.a. **Identificar mensajes intercambiados.** Para cada actor, determinar los mensajes intercambiados estableciendo el orden en que son pasados de un actor a otro y el propósito de los mismos.

- **Productos:** Un conjunto de pasos de mensaje etiquetados según la notación de UML (notación de secuencia, guardas, nombre de operación y parámetros)

4.b. **Identificar emisores y receptores.** Cada mensaje necesita de un un emisor y uno o varios receptores. Aunque aún no es momento de pensar acerca de la motivación de cada uno, sí que se puede determinar para cada participante qué mensajes envía y cuáles recibe.

- **Productos:** Se obtiene un diagrama de colaboración UML donde los actores son roles o agentes.

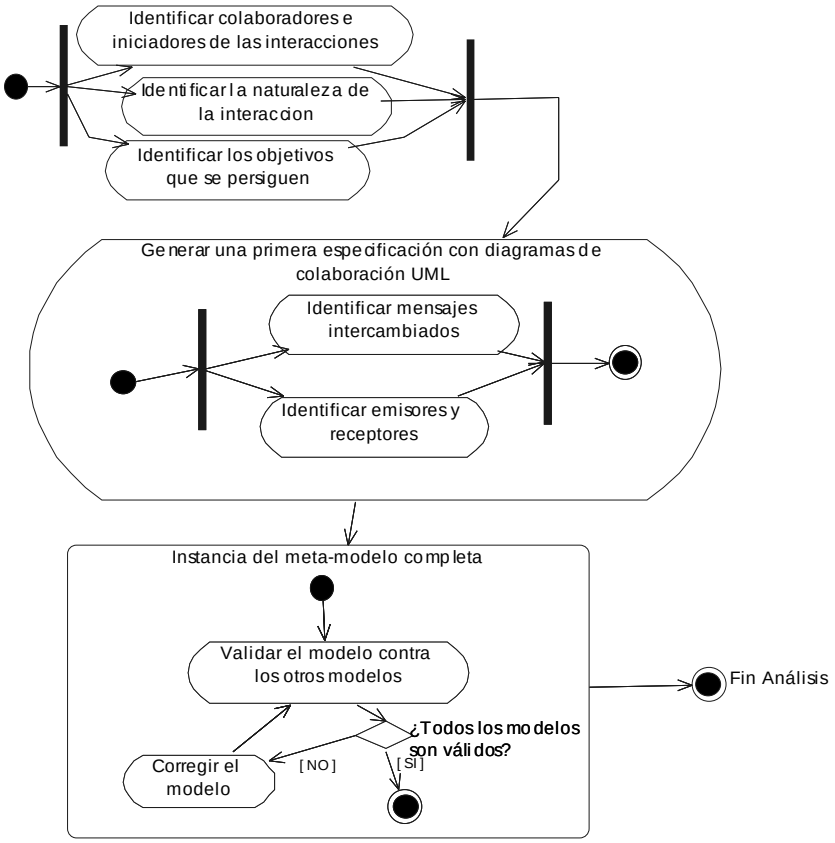


Ilustración 86. Actividades de análisis para la generación de modelos de interacciones

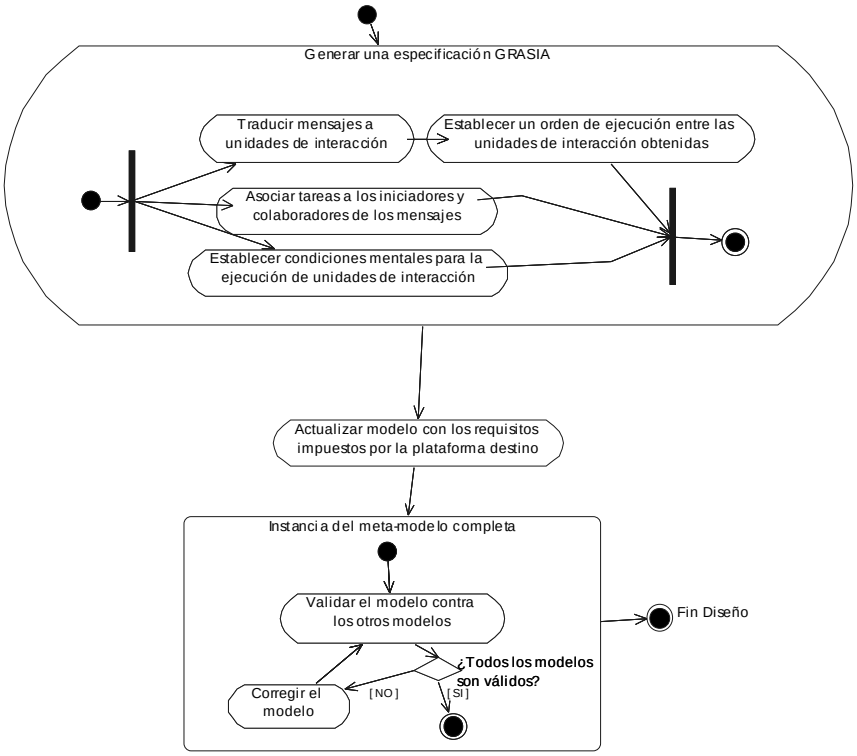


Ilustración 87. Actividades de diseño para la generación de modelos de interacciones

5. **Generar una especificación GRASIA.** La especificación GRASIA ha sido elaborada pensando en separar la ejecución de unidades de interacción de la asociación entre emisor y receptor. Esta separación es útil a la hora de generar código automáticamente.

5.a. **Traducir mensajes a unidades de interacción.** Cada mensaje se traduce a unidades de interacción adecuadas al medio de comunicación elegido (espacios compartidos de tuplas, paso de mensajes, llamada a procedimiento remoto).

- **Productos:** Un conjunto de unidades de interacción donde cada unidad de interacción se corresponde con uno o más mensajes.

5.b. **Asociar tareas a los iniciadores y colaboradores de los mensajes.** Las unidades de interacción son ejecutadas por un agente con el objetivo de que otro participe en su ejecución (recibiendo el mensaje, leyendo del espacio compartido de tuplas, recibiendo la llamada).

- **Productos:** Instancias de las relaciones *UIInicia* y *UIColabora* asociadas a las unidades de interacción.

5.c. **Establecer condiciones mentales para la ejecución de unidades de interacción.** Dentro de esta actividad se estudia la motivación de los participantes a nivel de unidad de interacción. La motivación consiste en una instantánea del estado mental de los agentes en el momento en que inician o colaboran en la ejecución de una unidad de interacción. En la representación del estado mental se tienen *patrones de estado mental* de entre los cuales destacan los *patrones de estado mental AOP* y los *patrones de estado mental GRASIA*. Los primeros se expresan con cadenas de texto que utilizan el BNF descrito por Shoham (ver sección 1.1 en el primer capítulo). Los últimos se expresan gráficamente con agregaciones de las entidades mentales que han de estar presentes.

- **Productos:** Un conjunto de *patrones de estado mental* asociados a las relaciones *UIInicia* y *UIColaba*.

5.d. **Establecer un orden de ejecución entre las unidades de interacción obtenidas.** Las unidades de interacción se ordenan de una forma determinada dependiendo de las necesidades establecidas por el flujo de trabajo representado. En el diagrama de colaboración, por defecto se tiene un orden secuencial de acciones. Sin embargo, también se permite la selección condicional y la iteración. Siguiendo UML, se han extraido cuatro primitivas fundamentales para determinar el orden de ejecución *Precede, UIItera, UIConcurren* y *Bifurca*.

- **Productos:** Instancias de las meta-relaciones *UIItera, UIConcurren, Precede* y *Bifurca* ordenando la ejecución de las unidades de interacción.

6. **Actualizar modelos con los requisitos de las plataformas destino**. Implica por lo general añadir información a la presentada en el modelo. Esta nueva información surge de la necesidad de instanciar un armazón software para que lleve a cabo las especificaciones del modelo. Según los requisitos impuestos por el armazón software, la especificación del sistema debiera modificarse. Así, un armazón software donde la comunicación entre agentes es importante, como JADE, influiría pidiendo un nivel de detalle alto en los modelos de interacciones.

- **Productos**. Nuevas entidades en el modelo o más detalle en las existentes.

7. **Validar el modelo contra los otros modelos**. La validación se realiza según los criterios de la sección 2.3.4 del capítulo segundo.

- **Productos**: Lista de inconsistencias

8. **Corregir el modelo**. Para hacer el modelo consistente con las instancias de otros meta-modelos

- **Productos**: Modificaciones a efectuar.

3.4 Generación de instancias del meta-modelo de tareas y objetivos

El meta-modelo de interaccion refleja el comportamiento deseado de un agente cuando recibe instrucciones de otro agente o bien requiere de otro agente para realizar una tarea. Como en los modelos de agentes, la generación de modelos de instancias se inicia en el análisis y se continúa en el diseño.

Para el análisis se ha fijado que los resultados esperables son:

- **Tareas y objetivos**. Los objetivos que se persiguen en el sistema, que han de ser resumen de todos los incluidos en los modelos de agente y organización. Se admite la descomposición estructural de tareas y objetivos (*GTDescompone* y *WFDescompone*). De hecho, se espera que se aplique con vistas a disminuir su complejidad. Esta descomposición se acompaña de dependencias entre objetivos (*GTDepende*)
- **Tareas asociadas a objetivos**. La asociación consiste inicialmente en instancias de *GTAfecta* que evolucionarán en el diseño hacia *GTSatisface* o *GTFalla*. Esta asociación constituye la justificación de la ejecución de la tarea.
- **Precondiciones y postcondiciones tentativas.** Las precondiciones mínimas son determinar qué entidades mentales se consumen (*WFConsume*), qué interacciones y entidades mentales se producen (*WFProduce y GTCrea*) y qué aplicaciones se usarán en el proceso (*WFUsa*).

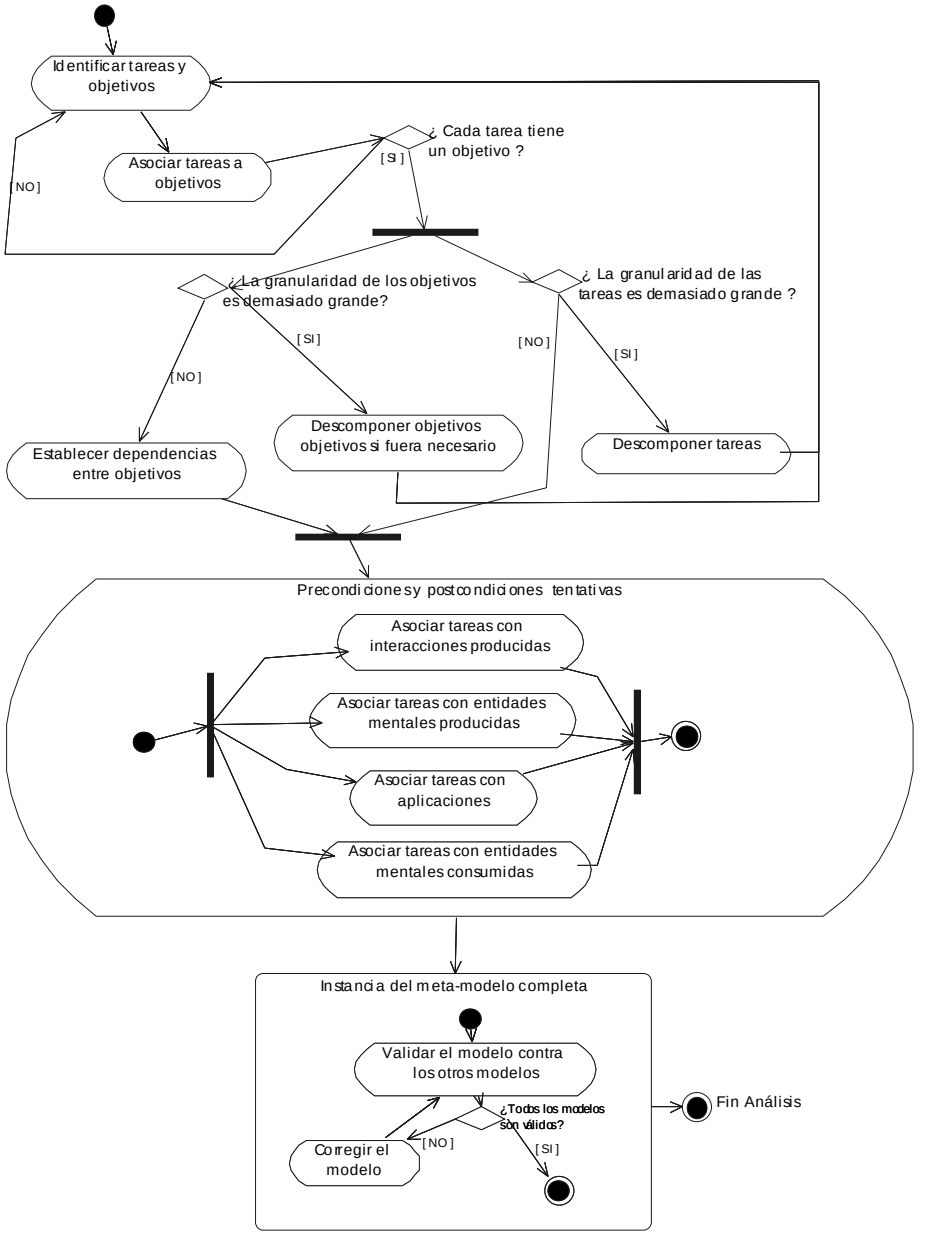


Ilustración 88. Actividades de análisis para la generación de modelos de tareas y objetivos

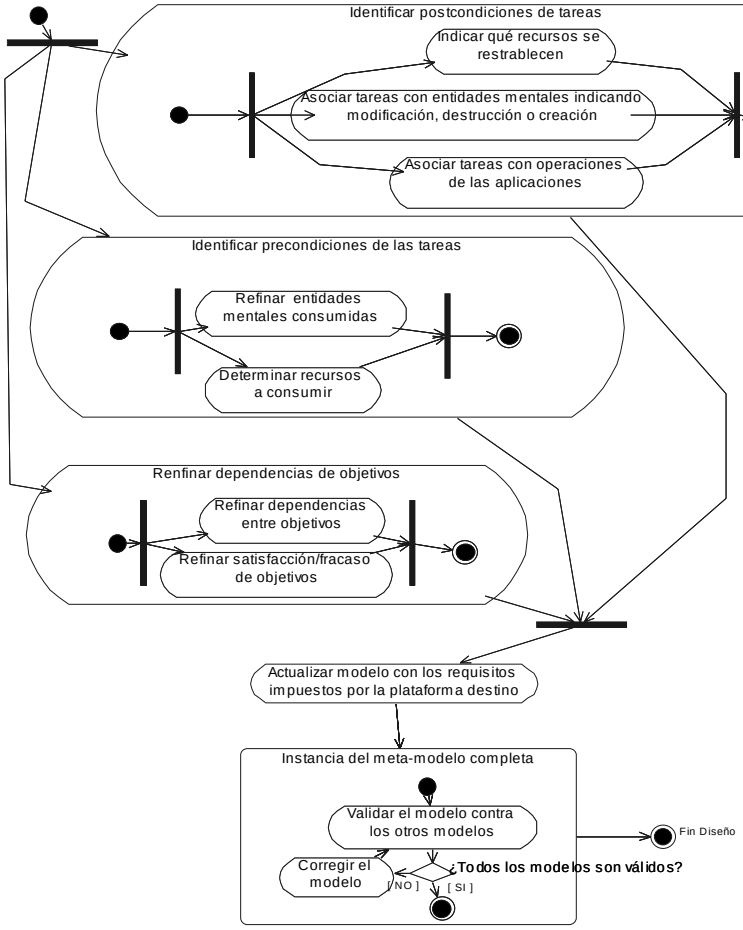


Ilustración 89. Actividades de diseño para la generación de modelos de tareas y objetivos

Para el diseño se ha fijado que los resultados esperables son:

- **Refinamiento de las dependencias entre objetivos.** Se detalla la dependencia indicando si se trata de dependencia *Y* (*GTDependeY*) o dependencia *O* (*GTDependeO*). Las dependencias pueden desaparecer con motivo de cambios en la dependencias estructurales entre objetivos.
- **Condiciones de satisfacción o fallo de los objetivos.** Las relaciones *GTAfecta* entre tareas y objetivos evolucionan a *GTFalla* y *GTSatisface*. En estos dos casos es necesario además indicar bajo qué condiciones se asume el éxito o fracaso del objetivo. Estas condiciones se expresan con *patrones de estado mental*.
- **Precondiciones detalladas.** El detalle en las precondiciones viene de añadir los recursos que necesita la tarea (*WFUsa)* y de incluir nuevas instancias de *WFConsume, GTDestruye, GTModifica*. Estas dos últimas se interpretan como precondiciones en el sentido de que requieren la existencia de la entidad mental con la que se relaciona.
- **Postcondiciones detalladas.** Se asocian nuevas instancias *WFProduce* con recursos para expresar la restitución de los mismos. Adicionalmente, se incluyen instancias *WFUsaLlamada* que expresan qué operaciones se están utilizando de las aplicaciones asociadas. Asimismo, se incluyen instancias de *GTDestruye, GTModifica y GTCrea* para mostrar cómo se modifica el estado mental. Se pueden asociar también entidades mentales con *WFProduce*, pero en dicho caso se interpreta la producción en términos de flujo de trabajo (información pasada de una tarea a otra).

Para conseguir estos resultados, se proponen las siguientes actividades:

1. **Identificar tareas y objetivos.** Los objetivos se pueden iniciar identificando, como hace MaSE, requisitos del sistema con objetivos. Otra forma es extraer qué deseos son asociables con el agente de otros meta-modelos (agente y organización). Por último, se pueden ver los objetivos como representantes del estado de las tareas con las que se asocian. En cuanto a las tareas, hay dos enfoques: relacionar las tareas con la funcionalidad requerida o bien sacar las tareas de los objetivos existentes.

   - **Producto**: Un conjunto de objetivos y tareas.

2. **Asociar tareas a objetivos.** Las asociación de tareas a objetivos se hace mediante instancias de meta-relaciones *GTAfecta*. Esta asociación determina cómo se satisface un objetivo, pero los objetivos también se pueden satisfacer mediante instancias de *GTDepende*. Por ello, no existe obligación de que todos y cada uno de los objetivos esté asociado con una tarea (el recíproco no es cierto).

   - **Producto:** Un conjunto de objetivos y tareas asociados por meta-relaciones *GTAfecta*.

3. **Descomponer tareas.** Es responsabilidad del ingeniero el determinar la complejidad que conlleva la realización de una tarea. Cuando la complejidad es excesiva, una técnica común es la de *divide y vencerás*. La descomposición de objetivos vuelve a sacar el problema de identificación de objetivos. La creación de nuevas tareas debería ir de la mano de la creación de nuevos objetivos. Así, cada nueva tarea procedente de la descomposición de la tarea original tiene que asociarse

con otro objetivo. La descomposición también puede entenderse al revés: la tarea se realiza porque el objetivo con el que está relacionado la tarea se ha descompuesto.

- **Productos:** Un conjunto de tareas asociadas con la relación *WFDescompone*.

4. **Descomponer objetivos objetivos si fuera necesario.** Como las tareas, los objetivos también se pueden descomponer. La descomposición obedece a la complejidad asociada con un objetivo (*divide y vencerás*) o bien por la descomposición de tareas asociadas.

- **Productos:** Un conjunto de objetivos relacionados por instancias de *GTDescompone*.

5. **Establecer dependencias entre objetivos.** Los objetivos se relacionan entre sí, por diversos motivos. En este trabajo se acepta que dos objetivos se relacionan entre sí únicamente con motivo de la existencia de una instancia de la meta-relación *GTDescompone*. En esta actividad se pide únicamente asociar los sub-objetivos con el padre mediante *GTDepende*. Para los casos en que las dependencias sean evidentes, se pueden indicar directamente en esta fase.

- **Productos:** Cada sub-objetivo relacionado con el objetivo padre mediante *GTDepende*.

6. **Precondiciones y postcondiciones tentativas.** Es necesario establecer a nivel de análisis qué es necesario para ejecutar las tareas y cuales son las consecuencias para el agente. Parte de las pre y postcondiciones se han fijado ya con instancias de meta-relaciones *GTDepende* y *GTAfecta*.

6.a. **Asociar tareas con interacciones producidas.** Las interacciones son posibles productos de las tareas. Que una tarea produzca una interacción significa que su ejecución conlleva la ejecución de tareas en otros agentes, lo cual implica que los efectos de una tarea se extienden más allá del agente. Esto es de suficiente importancia como para justificar la aparición de las interacciones en esta etapa.

- **Productos:** Asociaciones entre interacciones y tareas con *WFProduce*.

6.b. **Asociar tareas con entidades mentales producidas.** La ejecución de tareas conlleva la creación, destrucción o modificación de entidades mentales del agente ejecutor. A este nivel, sólo se resaltan las creadas. Hay que distinguir aquellas entidades mentales creadas por su utilización dentro de un flujo de trabajo y aquellas creadas localmente con otros fines. Las primeras se reflejan con *WFProduce* mientras que las segundas se representan con *GTCrea*.

- **Productos:** Asociaciones entre tareas y entidades mentales mediante instancias de *WFProduce* y *GTCrea*.

6.c. **Asociar tareas con entidades mentales consumidas.** Esta actividad se relaciona con la anterior. Se trata de la destrucción de entidades mentales como consecuencia de la ejecución de tareas. La destrucción de entidades mentales se hace con *GTDestruye*.

- **Productos:** Asociaciones entre tareas y entidades mentales mediante instancias de *GTDestruye*.

6.d. **Asociar tareas con aplicaciones.** Las tareas requerirán actuar sobre el entorno. Al tratarse de agentes software, las acciones sobre el entorno se traducen con

invocaciones de operaciones en aplicaciones software. A este nivel basta con indicar que existe una necesidad de actuar sobre la aplicación.

- **Productos:** Asociaciones entre tareas y aplicaciones mediante instancias de *WFUsa*.

7. **Identificar postcondiciones de tareas.** Las postcondiciones establecidas hasta ahora son sólo indicaciones de lo que puede pasar que no entran en detalles. Las actividades que se enmarcan dentro de ésta se orientan a la obtención de una descripción detallada de los efectos de las tareas.

7.a. **Indicar qué recursos se restablecen.** Una vez utilizados, los recursos se reponen bien indicando la finalización de su uso o bien restituyendo una cantidad predeterminada.

- **Productos:** Instancias de *WFProduce* asociando tareas y recursos.

7.b. **Asociar tareas con entidades mentales indicando modificación, destrucción o creación.** En actividades anteriores se mostraron relaciones entre tareas y entidades mentales (*GTCrea, GTDestruye* y *WFProduce*). Esta información se complementa en esta etapa con nuevas asociaciones para indicar modificación de entidades mentales (*GTModifica*) en concreto satisfacción y fallo de objetivos (*GTSatisface* y *GTFalla*). Estas asociaciones se caracterizan por estar condicionadas a la satisfacción de un patrón de estado mental. Este patron indica qué entidades mentales deben existir para dar por satisfecho o fallido el objetivo. Estas entidades pueden ser producidas por la tarea asociada o provenir del entorno como resultado de la actuación de la tarea.

- **Productos.** Instancias de *GTSatisface* y *GTFalla* asociando tareas y objetivos.

7.c. **Asociar tareas con operaciones de las aplicaciones.** Dentro de la tarea se realizan actividades sobre el entorno. Estas actividades se modelan utilizando operaciones concretas de las aplicaciones.

- **Productos:** Instancias de *WFUsaLlamada* indicando la operación utilizada sobre la aplicación.

8. **Identificar precondiciones de las tareas.** Las precondiciones en el diseño recogen información adicional respecto a los recursos consumidos y las entidades mentales que se utilizan como entradas para las tareas.

8.a. **Refinar entidades mentales consumidas.** Según progresa el diseño es normal que se matice el funcionamiento de las tareas, requiriendo en algunos casos, modificaciones en la información que se suministra a la tarea, esto es, cambios en las entidades mentales que se consumen. Estos cambios se reflejan en la aparición de nuevas entidades mentales, o modificación de las existentes.

- **Productos:** Nuevas entidades mentales, modificaciones en entidades existentes, instancias de *WFConsume*

8.b. **Determinar recursos a consumir.** Los recursos se consumen con instancias de *WFUsa*. Estas instancias significan la necesidad de un recurso concreto durante la ejecución de la tarea.

- **Productos:** Instancias de *WFUsa* asociando tareas y recursos.

9. **Actualizar modelos con los requisitos de las plataformas destino.** Implica por lo general añadir información a la presentada en el modelo. Esta nueva información surge de la necesidad de instanciar un armazón software para que lleve a cabo las especificaciones del modelo. Según los requisitos impuestos por el armazón software, la especificación del sistema debiera modificarse. Así, un armazón software donde la comunicación entre agentes es importante, como JADE, influiría pidiendo un nivel de detalle alto en los modelos de interacciones.

- **Productos.** Nuevas entidades en el modelo o más detalle en las existentes.

10. **Refinar dependencias de objetivos.** Los objetivos se alcanzan o bien por sus dependencias respecto de otros objetivos existentes o bien por sus asociaciones con tareas.

10.a. **Refinar dependencias entre objetivos.** Las instancias de *GTDepende* identificadas durante el análisis se refinan en *GTDependeY* o *GTDependeO*.

- **Productos.**Instancias de *GTDependeY* o *GTDependeO*.

10.b. **Refinar satisfacción/fracaso de objetivos.** La semántica de *GTDepende* determina *per se* la satisfacción o fracaso de un objetivo. Sin embargo, en el caso que el estado final del objetivo dependa de una tarea, el procedimiento es distinto. Se indica qué evidencias deben existir para considerar alcanzado o fracasado un objetivo. Esta información se proporciona dentro de instancias de *GTSatisface* y *GTFalla* en forma de patrones mentales.

- **Productos.** *Instancias de GTSatisface y GTFalla* asociadas con instancias de *patron de estado mental*.

11. **Validar el modelo contra los otros modelos.** La validación se realiza según los criterios de la sección 2.4.5 del capítulo segundo.

- **Productos:** Lista de inconsistencias

12. **Corregir el modelo.** Para hacer el modelo consistente con las instancias de otros meta-modelos

- **Productos:** Modificaciones a efectuar.

3.5 Generación de instancias del meta-modelo de organización

El meta-modelo de organización es comparable a la arquitectura del sistema en un SMA. Este meta-modelo relaciona las entidades activas (agentes, roles), pasivas (aplicaciones, recursos) y los elementos que proporcionan la funcionalidad del sistema (flujos de trabajo, tareas). Por ello, es aconsejable que este meta-modelo sea el primero en estudiarse para generar meta-modelos y que se vuelva a él con frecuencia a medida que van apareciendo nuevos elementos en los demás modelos.

Para el análisis se ha fijado que los resultados esperables son:

- **Estructura de la organización.** La idenficicación de los elementos del sistema es primordial. El detalle aquí alcanzado ha de ser el mayor posible, ya que ello posibilitará el tratamiento de las entidades encontradas en el resto de actividades.

- **Flujos de trabajo.** Aunque no se llegue a un gran detalle, hay que generar una lista de las tareas, con sus responsables correspondientes, conectadas unas con otras y resaltando aquellas tareas cuya ejecución afecte a otros agentes.
- **Dependencias sociales.** En el análisis basta con tejer una red de instancias de *AGORelacion* como se definieron en el meta-modelo de organización.

Para el diseño se ha fijado que los resultados esperables son:

- **Contexto de ejecución de las tareas.** Las tareas se asocian con sus ejecutores para indicar quien se ve afectado por ellas, qué recursos y entidades mentales se consumen y qué entidades de información se producen.
- **Refinamiento de las dependencias sociales.** Hay que detallar qué tipo de relaciones se están definiendo cuando se interconectan dos entidades con *AGORelacion*. Concretamente, se debe determinar si hay o no subordinación o si se trata de una relación cliente-servidor, indicando en cada caso las condiciones de prestación de servicio y el servicio en sí.

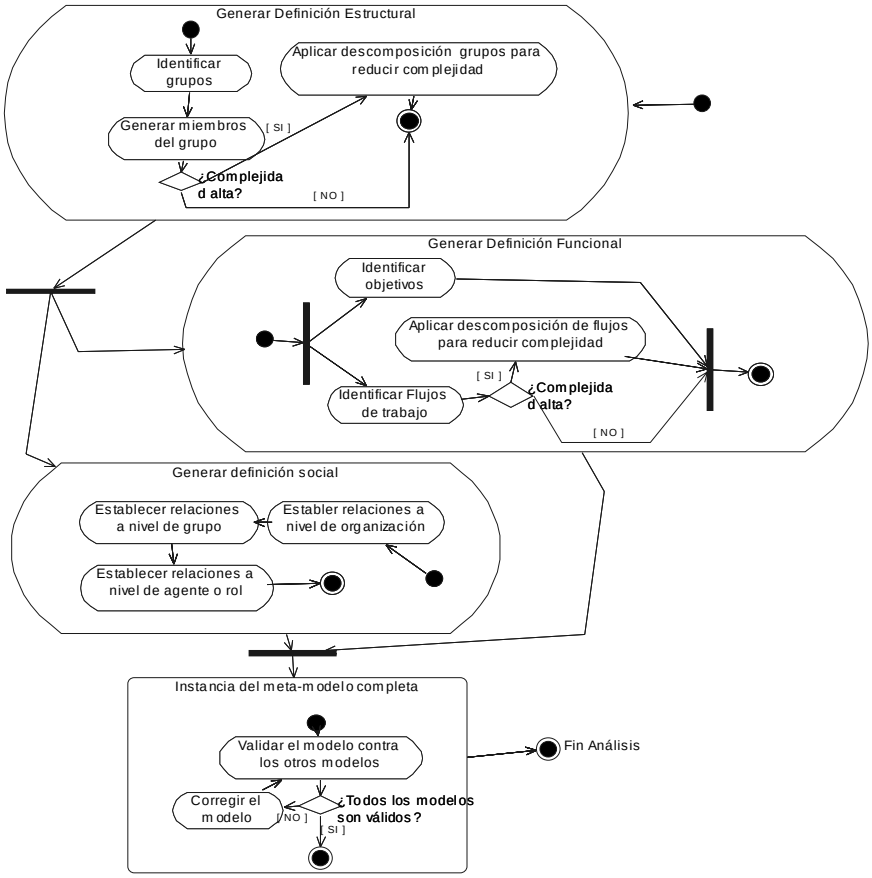


Ilustración 90. Actividades del análisis para generación de modelos de organización

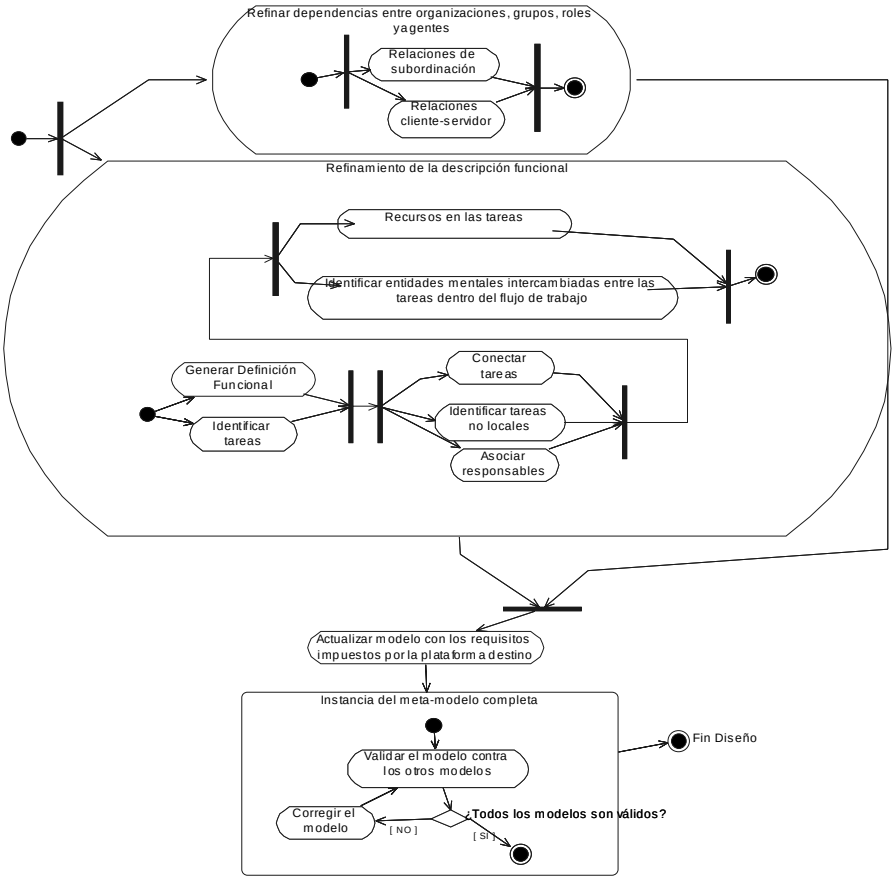


Ilustración 91. Actividades del diseño para generación de modelos de organización

Para conseguir estos resultados, se proponen las siguientes actividades:

1. **Generar Definición Estructural.** Antes de comenzar cualquier otra actividad, hay que identificar las entidades que componen el sistema, en este caso, agentes, roles, recusos, aplicaciones, grupos y organizaciones.

1.a. **Identificar grupos.** Los grupos aparecen cuando se intenta estructurar los distintos participantes de los distintos flujos de trabajo.

- **Productos:** Un conjunto de grupos asociados a la organización por instancias de *OContieneOrganizacion*.

1.b. **Generar miembros del grupo.** Los miembros de los grupos (agentes, roles, recursos y aplicaciones), aparecen al señalar los actores que aparecen en los flujos de trabajo (agentes y roles) y trasladar parte de los requisitos de la especificación (recursos disponibles, aplicaciones existentes con las que hay que integrar).

- **Productos:** Un conjunto de roles, agentes, recursos y aplicaciones asociados a uno o varios grupos mediante instancias de *OContieneGrupo*.

1.c. **Aplicar descomposición grupos para reducir complejidad.** La complejidad de los grupos que actúan en ellos puede ser demasiado elevada en algunos casos. Mediante la descomposición (aplicando *divide y vencerás*) el ingeniero asigna trabajo a los diferentes componentes del equipo de desarrollo para facilitar la construcción del SMA.

- **Productos:** Nuevos grupos asociados a los grupos existentes mediante instancias *ODescomponeGrupo*.

2. **Generar definición funcional.** La especificación funcional en el análisis se construye identificando las tareas más relevantes, relacionándolas con sus ejecutores y entre sí y destacando aquellas tareas que conllevan la ejecución de tareas en otros agentes.

2.a. **Identificar Flujos de trabajo.** Los flujos de trabajo surgen al considerar los requisitos funcionales del sistema.

- **Productos:** Un conjunto de flujos de trabajo asociados a la organización por instancias de *OContieneFlujoTrabajo*.

2.b. **Aplicar descomposición flujos de trabajo para reducir complejidad.** La complejidad de los flujos de trabajo que actúan en ellos puede ser demasiado elevada en algunos casos. Mediante la descomposición (aplicando *divide y vencerás*) el ingeniero asigna trabajo a los diferentes componentes del equipo de desarrollo para facilitar la construcción del SMA.

- **Productos:** Nuevos flujos de trabajo asociados a los flujos de trabajo existentes mediante instancias *ODescomponeFlujo*.

2.c. **Identificar objetivos.** La organización tiene un conjunto de objetivos que justifican la colaboración de los agentes. La relación exacta entre estos objetivos y los asociados con los agentes y roles individuales se establece dentro de modelos de tareas y objetivos.

- **Productos:** Objetivos asociados a la organización mediante instancias de *GTPersigue*.

3. **Generar definición social.** La definición social en el análisis se limita a identificar la existencias de relaciones sociales a cada uno de los tres niveles organizativos expuestos al presentar la definición de relaciones sociales en el meta-modelo de organización.

3.a. **Establer relaciones a nivel de organización.** Consiste en asociar las distintas organizaciones entre sí mediante instancias de *AGORelacion*.

- **Productos:** Instancias de *AGORelacion* entre organizaciones.

3.b. **Establecer relaciones a nivel de grupo.** Consiste en asociar los grupos existentes entre sí mediante instancias de *AGORelacion*.

- **Productos:** Instancias de *AGORelacion* entre grupos

3.c. **Establecer relaciones a nivel de agente o rol.** Se relacionan los agentes y roles entre sí mediante instancias de *AGORelacion*.

- **Productos:** Instancias de *AGORelacion* entre agentes o roles.

4. **Refinar dependencias entre organizaciones, grupos, roles y agentes.** Durante el diseño, las relaciones sociales del análisis se refinan en dependencias concretas del tipo *AGOSubordinacionCondicional, AGOSubordinacionIncondicional* o *AGOClienteServidor*.

4.a. **Relaciones de subordinación.** Las relaciones de subordinación posibles son *AGOSubordinacionCondicional* y *AGOSubordinacionIncondicional*. La primera se aplica cuando el agente no puede comprometerse completamente en la obediencia a otro agente. La segunda, cuando no existe tal restricción. Las condiciones de obediencia se fijan en términos de *patrones de estado mental*.

- **Productos:**Instancias de *AGOSubordinacionCondicional*, asociadas a instancias de *patrones de estado mental* para indicar las condiciones de obediencia, o *AGOSubordinacionIncondicional*

4.b. **Relaciones cliente-servidor.** Con estas relaciones se representa la noción de *servicio* aparecida en otras metodologías. El servicio se asimila con un *objetivo* dando a entender que lo que se ofrece es la capacidad de satisfacer el *objetivo* ofertado.

- **Productos:** Instancias de *AGOClienteServidor*

5. **Refinamiento de la descripción funcional.** La descripción funcional en el diseño ha de ser lo más detallada posible desde el punto de vista de interacción con otras tareas. De forma individual, las tareas se describen en el meta-modelo de tareas y objetivos.

5.a. **Identificar tareas.** Se trata de identificar tareas dentro de los flujos de trabajo definidos en el análisis. Las tareas a ejecutar se pueden obtener mediantes técnicas convencionales de ingeniería del software a partir de la especificación del problema o bien obtenerse de otros modelos (modelo de tareas y objetivos, modelo de agente). Estas tareas se asocian a los flujos de trabajo mediante relaciones de pertenencia y relaciones de flujo de datos (*WFProduce*, *WFConecta*, *WFConsume*).

- **Productos:** Un conjunto de tareas asociadas a flujos de trabajo.

5.b. **Conectar tareas.** Las tareas generadas en esta o futuras iteraciones se interrelacionan por medio de las entradas requeridas y las salidas producidas dentro del flujo de trabajo.

- Productos: Instancias de *WFConecta* conectando las distintas tareas.

5.c. **Identificar tareas no locales.** Las tareas que conlleven la ejecución de otras tareas en otros agentes se identifican asociándoles una *interacción*. Se asume que en un flujo de trabajo, donde cada tarea previsiblemente se ejecutará en distintos agentes, sólo produce una *interacción* la tarea que da comienzo al flujo de trabajo. Estas tareas se asocian con la interacción con instancias de *WFEspecificaEjecucion*.

- **Productos:** Instancias de *Interaccion* asociadas con tareas mediantes instancias de *WFProduce* y *WFEspecificaEjecucion*.

5.d. **Asociar responsables.** Las tareas son ejecutadas directamente por agentes o indirectamente a través de los roles desempeñados. Para cada tarea, se ha de decidir quién es su responsable, teniendo como posibilidades alguno de los agentes o roles existentes.

- **Productos:** Instancias de *WFResponsable* asociando agentes o roles y tareas

5.e. **Recursos en las tareas.** Los recursos disponibles en el sistema provienen de la especificación de requisitos inicial. Estos recursos se asignan a las distintas tareas en función de lo que el diseñador considere necesario. Los recursos que la tarea necesita para poder ejecutarse se asocian con instancias *WFUsa*. Los recursos que restituye la tarea se asocian con instancias de *WFProduce*. El conjunto de recursos consumidos es un subconjunto estricto de los recursos restituidos. Idealmente, por cada recurso consumido debe existir una restitución del mismo, pero para algunos recursos (los no consumibles según lo establecido en la sección 2.6.1.2) puede omitirse entendiendo que estos recursos se restauran automáticamente a la finalización de la tarea.

- **Productos:** Instancias de *WFUsa* y *WFProduce* asociando recursos a las tareas.

5.f. **Identificar entidades mentales intercambiadas entre las tareas dentro del flujo de trabajo.** Las entidades mentales son los elementos pasados de una tarea a otra durante la ejecución. Estas entidades se identifican de forma detallada dentro del meta-modelo de tareas y objetivos. No obstante, no todas las entidades que aparecen en las instancias de ese meta-modelo son válidas aquí. En este contexto se mencionan únicamente aquellas entidades mentales que o bien activan el flujo de trabajo o bien sirven para que éste continúe su ejecución. Una tarea se conecta con las entidades mentales consumidas en el flujo de trabajo con *WFConsume* y con las entidades producidas con *WFProduce*. Las tareas productoras de entidades mentales están unidas a las consumidoras con *WFConecta*.

- **Productos:** Instancias de *WFConsume* asociando a las tareas entidades mentales requeridas. Instancias de *WFProduce* asociando instancias de entidades mentales generadas por las tareas.

6. **Actualizar modelos con los requisitos de las plataformas destino.** Implica por lo general añadir información a la presentada en el modelo. Esta nueva información surge de la necesidad de instanciar un armazón software para que lleve a cabo las especificaciones del modelo. Según los requisitos impuestos por el armazón software, la especificación del sistema debiera modificarse. Así, un armazón software donde la comunicación entre agentes es importante, como JADE, influiría pidiendo un nivel de detalle alto en los modelos de interacciones.

- **Productos.** Nuevas entidades en el modelo o más detalle en las existentes.

7. **Validar el modelo contra los otros modelos.** La validación se realiza según los criterios de la sección 2.5.4 en el capítulo segundo

- **Productos:** Lista de inconsistencias

8. **Corregir el modelo.** Para hacer el modelo consistente con las instancias de otros meta-modelos

- **Productos:** Modificaciones a efectuar.

3.6 Generación de instancias del meta-modelo de entorno

La mayoría de la información para la generación de instancias del meta-modelo de entorno proviene de la captura de requisitos, que es la entrada para el análisis.

Para el análisis se ha fijado que los resultados esperables son:

- **Entidades del entorno.** Una enumeración de las entidades que preceden al sistema a desarrollar. Estas entidades se categorizan como aplicaciones, agentes o recursos. Cada entidad se acompaña de una descripción de sus funciones, que se entiende han de extraerse de la captura de requisitos.
- **Percepción del agente.** Aunque se detallará más adelante, la percepción del agente se asocia con las aplicaciones del entorno.

Para el diseño se ha fijado que los resultados esperables son:

- **Configuración de la percepción del agente.** Cada agente asociado con aplicaciones explicita cómo va a percibir, si va a utilizar mecanismos de muestreo o notificación y qué es lo que espera recibir.
- **Relaciones con tareas.** Aunque se considera en otros meta-modelos, se identifican asociaciones de producción y utilización de aplicaciones y recursos con tareas.
- **Definición detallada de los recursos.** Consiste en especificar exactamente cómo es cada recurso, detallando su categoría concreta, el límite inferior de consumo, el superior y el estado en que se encuentra inicialmente.

Para conseguir estos resultados, se proponen las siguientes actividades

1. **Identificar aplicaciones.** Las aplicaciones se conciben como objetos. Será una aplicación todo software o hardware con que el sistema debe interactuar y que no puede ser modelado (o no conviene) como agente.

1.a. **Identificar aplicaciones del entorno.** Las aplicaciones del entorno se extraen la captura de requisitos del sistema. Se identifican con el software propietario o paquetes estándar con que hay que interactuar.

- **Productos:** Conjunto de instancias de *AplicacionEntorno*

1.b. **Identificar aplicaciones internas.** Las aplicaciones internas se extraen de las necesidades contempladas a la hora de diseñar los agentes en otros meta-modelos. Se trata de servicios que han de mantenerse centralizados y que no muestran necesidad de incorporar comportamiento similar al de los agentes.

- **Productos:** Conjunto de instancias de *AplicacionInterna*.

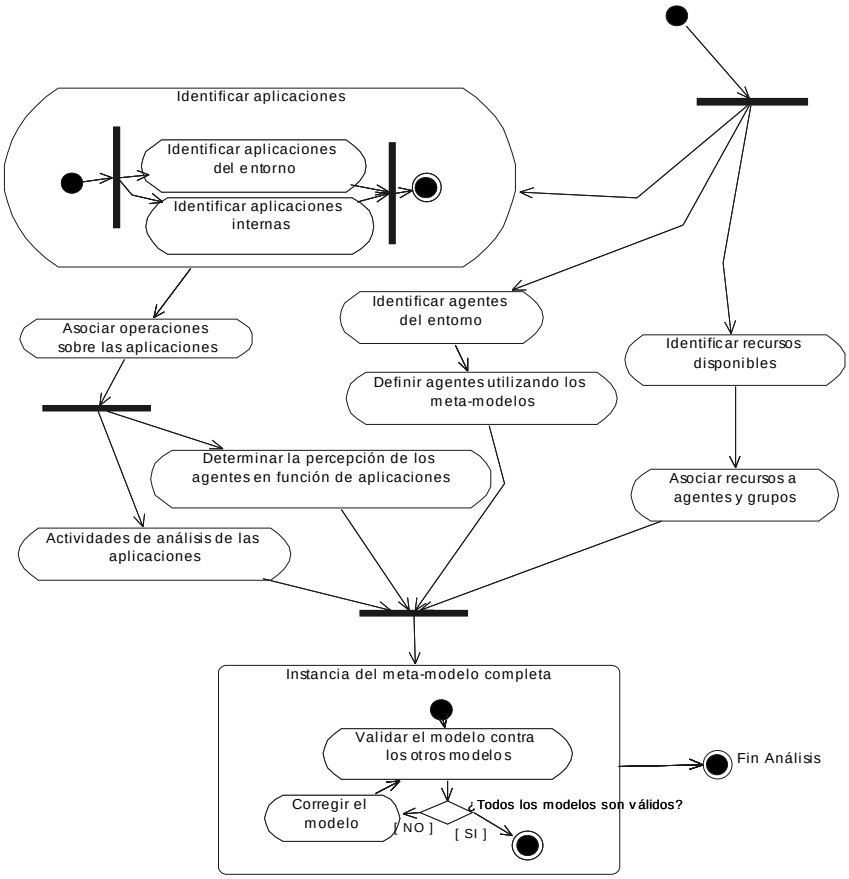


Ilustración 92. Actividades de análisis para la generación de modelos de entorno

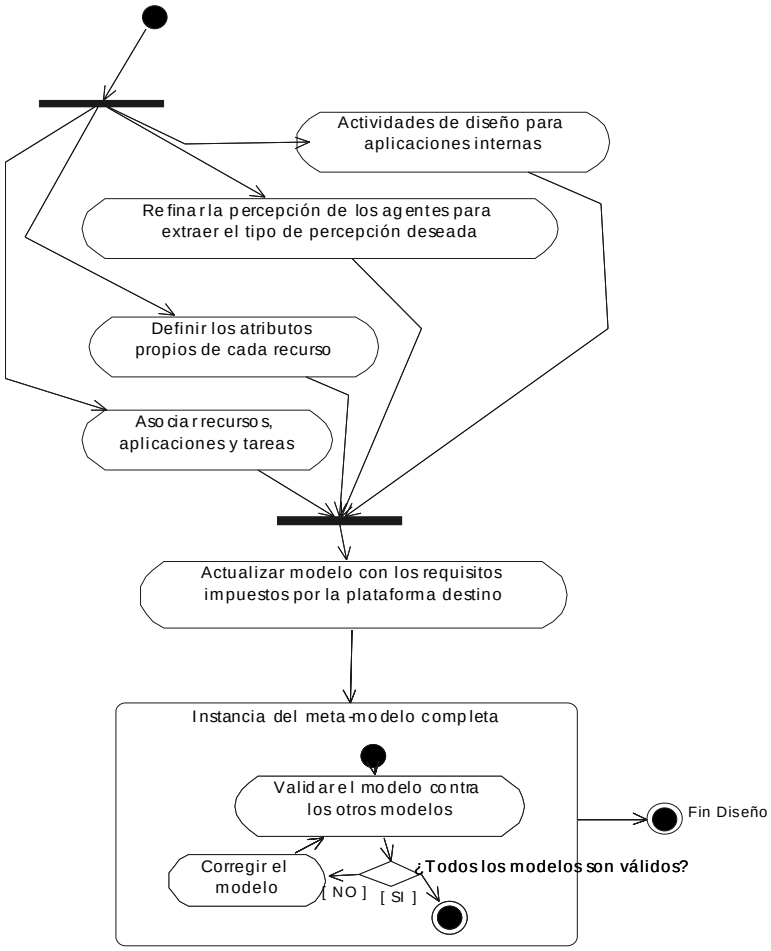


Ilustración 93. Actividades de diseño para la generación de modelos de entorno

2. **Asociar operaciones sobre las aplicaciones.** Sobre las aplicaciones se pueden realizar un conjunto de operaciones que son definidas durante la captura de requisitos. Estas operaciones se caracterizan por tener una signatura (nombre, tipo devuelto, parámetros) más unas precondiciones y postcondiciones. La identificación de estas operaciones es una tarea convencional de ingeniería.

- **Productos:** Un conjunto de instancias de *Operación* asociadas a las aplicaciones.

3. **Actividades de análisis de las aplicaciones.** La especificación concreta de las aplicaciones internas se realiza con métodos convencionales. De aplicar un análisis orientado a objetos, por ejemplo, sería conveniente detallar el estado interno de las aplicaciones y el efecto de las operaciones sobre este estado. Las aplicaciones del entorno no deberían necesitar este esfuerzo ya que se presuponen ya analizadas.

- **Productos:** Los establecidos por el método elegido

4. **Identificar agentes del entorno.** Los agentes del entorno son entidades similares a las instancias de *aplicación* solo que son modelables utilizando los meta-modelos aquí presentados. Principalmente, se aplica el *principio de racionalidad* para distinguirlas del software convencional, aunque se cuenta con que dentro de la captura de requisitos ya aparecen identificados.

- **Productos:** Un conjunto de agentes

5. **Definir agentes utilizando los meta-modelos.** Para el caso en que se trate de software convencional que hay que recubrir con una capa de software de agentes, se aplican los meta-modelos aquí expuestos para definir el nuevo agente.

- **Productos:** Instancias de los meta-modelos de agente, organización, tareas y objetivos, entorno e interacción para los nuevos agentes.

6. **Determinar la percepción de los agentes en función de aplicaciones.** Inicialmente basta con unir agentes con aplicaciones con instancias de *EPersigue* sin indicar nada más.

- **Productos:** Instancias de *EPercibe* asociando agentes a instancias de *aplicación*.

7. **Identificar recursos disponibles.** Los recursos disponibles surgen de la captura de requisitos del sistema. Los recursos se categorizan en consumibles y no consumibles. De forma orientativa, en la sección 2.6.1.2 se muestra una lista de los recursos más comunes. Esta lista se corresponde con posibles instancias de *Recurso*.

- **Productos:** Instancias de *Recurso*.

8. **Asociar recursos a agentes y grupos.** Los recursos se asocian a agentes o roles, indicando quién es el responsable de su gestión. Cuando se asocian a un grupo, no hay un responsable directo, salvo que el ingeniero acuerde tácitamente que determinado agente o rol es responsable de los recursos de su grupo.

- **Productos:** Instancias de *ERecursoPertenece* asociando recusos a agentes o grupos.

9. **Refinar la percepción de los agentes para extraer el tipo de percepción deseada.** La percepción del agente se refina indicando si se tiene un muestreo o una notificación respecto de los valores devueltos por una operación de la aplicación.

- **Productos:** Instancias de *EPercibeNotificacion* y *EPercibeMuestreo* donde antes había solo instancias de *EPercibe*. Es obligatorio el indicar la operación asociada.

10. **Actividades de diseño para aplicaciones.** Las aplicaciones se diseñan utilizando métodos convencionales de ingeniería del software.

- **Productos:** Los establecidos por el método elegido

11. **Definir los atributos propios de cada recurso.** Cada recurso se define mediante umbrales y el estado inicial. Durante el diseño, y en vista de los resultados del análisis en otros modelos, se fijan los recursos que se necesitan.

- **Productos:** Determinar el valor de los umbrales de los recursos y su estado inicial

12. **Asociar recursos, aplicaciones y tareas.** Durante el diseño, los modelos de tareas y objetivos mostrarán asociaciones con recursos y aplicaciones que conviene repetir aquí.

- **Productos:** Instancias de WFUsa asociando tareas y aplicaciones. Instancias de WFConsume y WFProduce asociando tareas y recursos

13. **Actualizar modelos con los requisitos de las plataformas destino.** Implica por lo general añadir información a la presentada en el modelo. Esta nueva información surge de la necesidad de instanciar un armazón software para que lleve a cabo las especificaciones del modelo. Según los requisitos impuestos por el armazón software, la especificación del sistema debiera modificarse. Así, un armazón software donde la comunicación entre agentes es importante, como JADE, influiría pidiendo un nivel de detalle alto en los modelos de interacciones.

- **Productos.** Nuevas entidades en el modelo o más detalle en las existentes.

14. **Validar el modelo contra los otros modelos.** La validación se realiza según los criterios de la sección 2.6.4 en el capítulo segundo

- **Productos:** Lista de inconsistencias

15. **Corregir el modelo.** Para hacer el modelo consistente con las instancias de otros meta-modelos

- **Productos:** Modificaciones a efectuar.

3.7 Fase de implementación

La fase de implementación en esta tesis se plantea como la parametrización de un armazón software ya existente. Existen múltiples desarrollos que pueden servir como armazón software, por ejemplo, los ya vistos, JADE, ZEUS o Grasshopper. Sobre estos armazones se construrían esqueletos genéricos que serían concretados más tarde con los datos de los modelos que especifican el SMA.

Este planteamiento es diferente del adoptado por la mayoría de las herramientas de análisis y diseño, donde la generación automática de código es fija y sin posibilidad de modificación. Herramientas como Rational Rose o TogetherJ trabajan con elementos como

diagramas de secuencia y de estados para los que existen representaciones únicas dentro de un lenguaje destino a elegir por el desarrollador (JAVA , Smalltalk, C++).

A diferencia de estas herramientas, en este trabajo se busca traducciones de los modelos en elementos conocidos como máquinas de estado, protocolos o arquitecturas de agente. Estos elementos se suponen organizados para formar un armazón de SMA. Para facilitar esta traducción, se propone un método genérico de parametrización de armazones basado en el marcado con XML del código del armazón. A grandes rasgos, el proceso consiste en sustituir el código marcado en el armazón por información elaborada a partir de los modelos que componen la especificación.

La selección de un armazón adecuado es importante. El armazón destino debe reunir ciertos elementos recogidos en los modelos, por ejemplo, elementos a cargo de la gestión del estado mental o de la toma de decisiones.

En esta sección, primero se describe informalmente el proceso completo de generación del sistema para ubicar la parte de parametrización del armazón. A continuación se describen los elementos que se utilizan en la parametrización, las *plantillas* y las *secuencias*. El proceso en sí se presenta después utilizando diagramas de decisión. Para terminar, algunas consideraciones sobre el armazón sobre el que se aplicará el proceso.

3.7.1 Actividades

La generación de código se realiza de acuerdo con las actividades de la Ilustración 94.

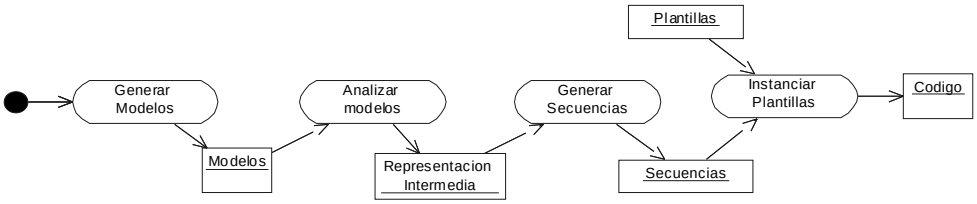


Ilustración 94. Actividades para la generación automática de código

La generación de modelos consiste en el diseño del sistema utilizando los meta-modelos de este trabajo. Se soporta con la herramienta METAEDIT+, que genera editores de modelos personalizados para cada uno de los meta-modelos.

Al *análizar los modelos* se transforma la representación de METAEDIT+ en un conjunto de predicados PROLOG más sencillos de procesar. La transformación se hace también desde PROLOG.

La *generación de secuencias* se hace teniendo en cuenta los parámetros que admite el armazón sobre el que se va a trabajar. Las secuencias son listas de listas de términos *clave = valor*. Esta parte es dependiente del armazón destino, por lo que debe ser programada especialmente para cada tipo de armazón.

Por último, la *instanciación del armazón* se realiza utilizando el intérprete de lenguaje de marcado de *PiLLOw* (*Programming in (Constraint) Logic Languages on the Web*)

[Departamento de Inteligencia Artificial (UPM) 02]. La instanciación consiste en sustituir marcas (*tags*) en el armazón por el valor indicado en las secuencias del paso anterior.

El resultado final es un armazón parametrizado donde los programadores tienen que insertar su código. El proceso se realiza mediante la utilización de *plantillas* marcadas.

3.7.2 Plantillas y Secuencias

Como ya se ha comentado, parte del código dentro del armazón sobre el que se desarrolla está marcado. Las partes que están marcadas se denominan *plantillas*. Las plantillas pueden estar escritas en cualquier lenguaje de programación, siembre que se respeten las marcas establecidas. Las marcas indican los lugares donde se ubicarán los datos extraídos de los modelos. Las *plantillas*, entonces, se conciben como documentos XML que satisfacen la especificación de la Ilustración 95.

Para rellenar las plantillas, se utilizan la misma solución que en los lenguajes de que integran SQL en páginas HTML. La información de las instancias del meta-modelo se pasa a un formato de listas de listas de términos igualdad (ver Ilustración 96) . Estas secuencias se incrustan en las plantillas mediante dos clases de etiquetas: *repeat* y *v* (de *variable*).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="qualified">
    <xsd:element name="repeat">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="v"/>
                <xsd:element ref="repeat"/>
            </xsd:choice>
            <xsd:attribute name="id" use="required" >
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="v">
    </xsd:element>
</xsd:schema>
```

Ilustración 95. Especificación de una plantilla con marcado con esquemas XML.

La etiqueta *repeat* tiene un atributo *id* cuyo valor sirve para identificar las secuencias que contienen datos relevantes para el fragmento de código actual. Estas secuencias se caracterizan por contener un identificador *id* con el mismo valor asociado que en la plantilla. Para *repeat* anidados, se admite la existencia de múltiples *id* siempre y cuando no se repitan sus valores. La etiqueta *v* señala la existencia de una variable cuyo identificador aparece en la secuencia considerada.

```
Listas::=[Lista]|[Lista, Lista]
Lista::= [Terminos]|[]
Terminos ::= Termino | Termino,Terminos
Termino::= Identificador = "Valor"
```

Ilustración 96. Expresión BNF para definir secuencias de datos.

Las secuencias que describen la parametrización del armazón se especifican con la Ilustración 96. Se trata de listas de listas de términos *identificador = valor*. El identificador se utiliza para identificar la variable a sustituir (marca *v* de la Ilustración 95). Existe un identificador reservado. Este identificador es *id* y se utiliza para identificar secuencias, posiblemente anidadas, de parametrizaciones.

3.7.3 Rellenando las Plantillas

El proceso para la reescritura de las plantillas está escrito íntegramente en PROLOG (ver Anexo I.). La idea general es que hay que sustituir las etiquetas de las plantillas por información de los modelos. La información de los modelos viene en forma de lista de listas de términos que han de ser cotejados contra los identificadores asociados a las etiquetas *repeat* y *v* en la plantilla. El proceso se basa en la librería PiLLOW. Esta librería lee cualquier documento de texto y es capaz de extraer las marcas. El resultado es una lista de términos donde hay etiquetas o texto. El proceso (ver Ilustración 97) comienza analizando la plantilla.

Tras el análisis, se procede a estudiar la lista resultante, comprobando si el término a analizar es texto o una etiqueta. Si es texto, se imprime por pantalla. Si es una etiqueta y además un *repeat*, se filtran las secuencias que contienen la información de los modelos dejando solo aquéllas que contengan un término *id* cuyo valor coincida con el del atributo *id* de la etiqueta *repeat* encontrada. Con las secuencias filtradas, se vuelve a aplica recursivamente el análisis. Cuando la etiqueta es *v*, se realiza una sustitución de la etiqueta por el valor asociado dentro de las secuencias de datos de los modelos.

Al término del proceso, todas las etiquetas de la plantilla han desaparecido y sólo quedan los valores extraidos de los modelos ya situados en el código del armazón. Como ejemplo de aplicación, la Ilustración 98 muestra cómo se instancia una plantilla de regla de producción con una secuencia de datos ficticia.

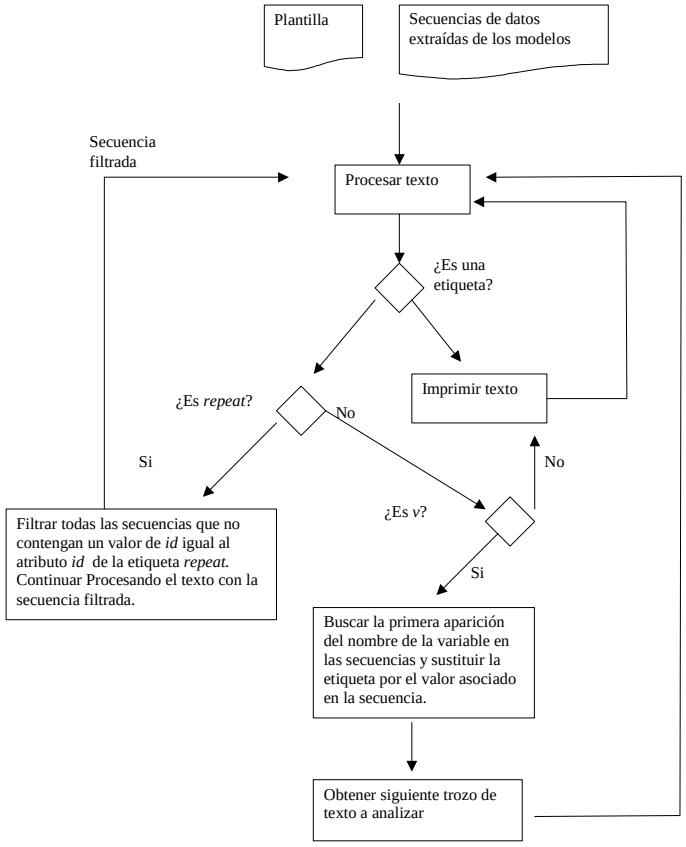


Ilustración 97. Diagrama de decisión del proceso de instanciación de plantillas

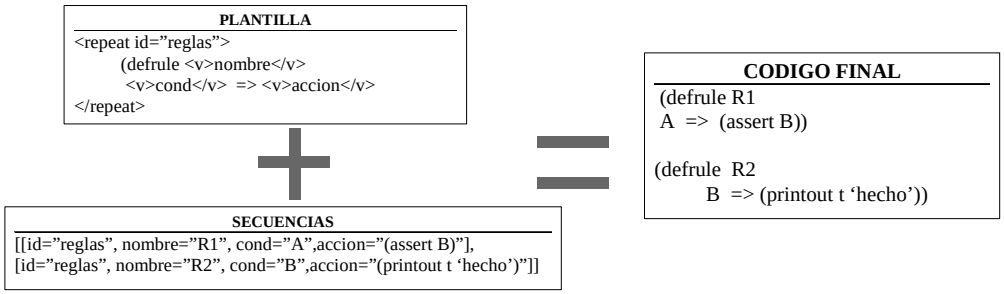


Ilustración 98. Ejemplo de aplicación de secuencias a plantillas.

3.7.4 Patrones arquitectónicos de diseño

Los modelos generados para especificar un SMA definen implícitamente un conjunto de patrones arquitectónicos que deben soportarse en la arquitectura final. Los patrones

arquitectónicos se entienden como abstracciones de arquitecturas del sistema y sus componentes. Al identificar el conjunto de elementos necesarios para definir el agente y el SMA, se está aludiendo a qué elementos han de estar presentes en la arquitectura que soporte el SMA, cómo se relacionan y qué funcionalidad deben aportar.

Respecto de la arquitectura del agente, los meta-modelos indican que deben existir componentes que representen:

- **El estado mental.** Contiene toda aquella información que permite al agente tomar decisiones. A lo largo de la tesis, se ha expresado el estado mental como agregación de entidades mentales utilizando instancias del modelo de agente. Se ha distinguido entre el estado mental inicial del agente, asociando el agente a una instancia de *estado mental*, y los estados mentales intermedios, asociando una instancia de *consulta autónoma* a una instancia de *estado mental*. El paso del estado mental inicial por cada uno de los intermedios se indica mediante la ordenación de tareas dentro de flujos de trabajo e interacciones, que son las que modifican el estado mental. El estado mental, en este trabajo, se ha descrito de forma simbólica, por lo que se espera que en la arquitectura aparezca de igual forma. Sin embargo, hay desarrollos precedentes que muestran que no es necesario. Así, en los *Situated Autómata* se establecen procedimientos para transformar especificaciones simbólicas en autómatas equivalentes [Rosenschein y Kaelbling 95]
- **El gestor de estado mental.** Describe cómo se gestionan las entidades mentales. Este gestor completa la definición de la evolución del estado mental estableciendo, por ejemplo, qué ocurre con los objetivos una vez se alcanzan, qué entidades ya no son válidas y si se pueden añadir nuevas entidades mentales mientras se está tomando una decisión. Estos aspectos se definen utilizando texto y modelos de tareas y objetivos.
- **El procesador de estado mental.** Es el encargado de tomar las decisiones a partir del estado mental. A la descripción textual de este elemento se añade una descripción en forma de modelos y tareas. A veces es suficiente con decir que se trata de un planificador STRIPS, otras hay que detallar un conjunto de tareas que describan cómo se activan otras tareas.
- **Percepción.** La percepción del agente obliga a que la arquitectura considere la comunicación con elementos anteriores a la construcción del sistema o con nuevos elementos identificados durante el análisis y el desarrollo. La percepción se puede encapsular en un componente o aparecer distribuida en un conjunto de componentes especializados.
- **Otros componentes.** Durante el desarrollo aparecen instancias de *aplicación* asociadas al agente dentro del modelo de entorno. Estas asociaciones se trasladan a la arquitectura como componentes casi directamente ya que se trata de elementos cuyo desarrollo se hace utilizando técnicas convencionales, como tecnología de objetos.

Alguno de estos elementos son combinables, como el *procesador* y *gestor de estado mental*, dentro de la arquitectua final. De hecho, los motores de reglas, como se verá más tarde, son combinación de los elementos de gestión y procesamiento del estado mental.

Respecto a patrones arquitectónicos del SMA, su tratamiento difiere de los de los agentes. Estos patrones, no implican la existencia de una entidad SMA como componente, ya que en la mayoría de desarrollos existentes, los componentes asociables al SMA como tal se distribuyen entre los agentes del sistema, buscando sobre todo la descentralización. De cualquier forma, la arquitectura que soporte el SMA, debe tener en cuenta:

- **Coordinación.** Las interacciones constituyen el comportamiento del agente cara a otros agentes. Dependiendo de cómo se planteen las interacciones, la arquitectura deberá soportar comunicación síncrona, asíncrona, dirigida por datos o dirigida por mecanismos de control [Papadopoulos y Arbab 98]. Además, hay que tener en cuenta que el agente puede participar en varios procesos de coordinación simultáneos, por lo que serán necesarios mecanismos de gestión de sesiones de comunicación.
- **Restricciones sociales.** Las restricciones sociales se observan en la implementación mediante elementos que limiten las actuaciones o las percepciones de los agentes. Estas restricciones pueden aparecer como condiciones adicionales para la ejecución de tareas o como componentes arquitectónicos que inhiben al agente.
- **Funcionalidad.** La funcionalidad del sistema se recoge principalmente en tareas, flujos de trabajo y objetivos. La existencia de tareas en la arquitectura final es difícil de evitar. Sin embargo, en el caso de los flujos de trabajo y los objetivos, se pueden obviar. MaSE consigue omitir los objetivos igualando un rol a un objetivo durante la implementación,. Los flujos de trabajo, aunque no aparezcan explícitamente, estarán presentes en la arquitectura como asociación entre tareas.
- **Estructuración del sistema.** El modelo de organización proporciona una estructuración de elementos que sirve de partida a la hora de elaborar la arquitectura final del sistema. La estructuración en *grupos* es similar a la estructuración en paquetes aplicada en JAVA. A diferencia de los paquetes JAVA, la inclusión de los grupos en la organización se rige por la conveniencia de la agrupación en tanto se facilite la la consecución de los objetivos de la organización. La arquitectura final del sistema respetará esta estructura y proporcionará, si fuera necesario, mecanismos de gestión de los miembros de la organización así como servicios de localicación y, en caso necesario, *matchmaking*.

Para facilitar la revisión de estos aspectos de la especificación, se ha organizado la documentación del sistema siguiendo una estructura similar a la enumeración ya vista. Parte de estos aspectos ya han sido estudiados dentro de MESSAGE, identificando elementos computacionales que parecen especialmente adecuados como implementación de las especificaciones aquí mostradas. Estos componentes se utilizaron durante el desarrollo de MESSAGE como implementación de las especificaciones iniciales:

- **Motores de reglas.** Se usan para definir el comportamiento del agente. Se han utilizado especialmente motores de reglas de inferencia con encadenamiento hacia delante y hacia atrás. Estos motores permiten gestionar eficientemente la ejecución de un conjunto grande de reglas. La definición de reglas así como su gestión se consigue a través de un lenguajes de definición de reglas específico del motor. Las reglas constan de guardas, tambien conocidas como parte izquierda de la regla o LHS

(*Left Hand Side*), y acciones, tambien conocidas como parte derecha de la regla o RHS (*Right Hand Side*). Su funcionamiento se limita a chequear la parte izquierda de la regla contra la memoria de trabajo del motor, si se satisfacen las guardas, entonces se ejecuta de forma mutuamente exclusiva la parte derecha de la regla. Las reglas pueden utilizarse para propósitos de gestión y procesamiento del estado mental mientras que las entidades mentales se pueden mostrar como hechos. Las restricciones sociales también se presentan dentro de un motor de reglas como reglas de alta prioridad o como guardas adicionales en las reglas existentes.

- **Máquinas de estado para los protocolos.** Las interacciones se pueden codificar en forma de máquinas de estados, donde dentro de cada estado o en la transición de un estado a otro se ejecutan tareas. Es importante la reutilización y flexbilidad de la implementación de la máquina de estados. Para el caso de MESSAGE, ya que se implementaban muchos protocolos, se diseñó un intérprete para plantillas de definición de máquinas de estado en XML. Gracias a este intérprete, las modificaciones en la comunicación tenían un impacto reducido en el diseño.
- **Comunicación CORBA y FIPA ACL.** Se utiliza para implementar las unidades de interacción. La comunicación mediante paso de mensajes utilizando FIPA ACL puede ser una buena opción desde las especificaciones de esta metodología, ya que en cada unidad de interacción se están transportando entidades mentales de un agente a otro. FIPA ACL se adecúa a este uso con facilidad, ya que el paso de mensajes es asíncrono y además contempla el transporte de entidades mentales de un agente a otro. Sin embargo, diseñar un agente que se comunique correctamente utilizando FIPA ACL no es trivial. Uno de los mayores problemas consiste en verificar la estructura y contenido de los mensajes, ya que es muy fácil que, debido a errores de programación, se malformen los mensajes. En este sentido, uso de CORBA es más sencillo y seguro ya que define interfaces de forma convencional y no requiere arquitecturas que soporten mensajería asíncrona.
- **Objetos.** La POO es conveniente para representar y encapsular el comportamiento de entidades como las tareas, los objetivos o los eventos. Para la representación de tareas, OMG proporciona un conjunto de interfaces compatibles con el uso de las tareas tal y como se ha mostrado aquí [OMG 00c]. Para los objetivos, reutilizando la experiencia de proyectos anteriores [Gomez-Sanz, Garijo y Pavon 00], se diseñaron estructuras de datos arbóreas que facilitaban la gestión de los mismos. Estas estructuras se basaban en los arboles Y/O ya comentados en la presentación del meta-modelo de objetivos y tareas (ver sección 2.4).

3.8 Conclusiones

Este capítulo se ha mostrado una integración de la generación de la especificación del sistema como una actividad más en el modelo de desarrollo del RUP. La integración se ha planteado por un lado definiendo qué productos son deseables en cada iteración y fase, y por otro qué actividades se involucran en cada fase de desarrollo.

En este trabajo, estas actividades han sido tomadas como guía de los productos que deberían generarse y de las posibles dependencias que se pueden encontrar los usuarios finales de esta metodología. Aunque su número es elevado, no se plantea en ningún

momento que el proceso de desarrollo tenga que seguir tantos pasos. Los usuarios pueden decidir unir varias actividades para generar otras más genéricas siempre y cuando se respeten los productos que se deben generar. Además, las actividades identificadas constituyen un punto de partida para futuros trabajos. Su número y propósito concreto deberán ser probados en distintos desarrollos.

En cuanto a la composición de las actividades dentro de los diagramas de actividades de análisis y diseño, no se prevén cambios en estos diagramas al variar las iteraciones. Los diagramas marcan las dependencias entre las actividades y, por lo tanto, el orden en que debieran producirse cada parte del modelo completo. Si el desarrollador en una iteración encuentra que el nivel de detalle alcanzado por una actividad es suficiente, entonces, puede dar la actividad por concluida.

La implementación final del sistema se ha estudiado desde dos puntos de vista: como un proceso automatizado de parametrización de arquitecturas software y como un problema de aplicación de patrones arquitectónicos de diseño. Dentro del primer enfoque, se ha proporcionado un medio de instanciación de arquitecturas software que toma como entrada los modelos que forman la especificación del sistema. Dentro del segundo, se ha estudiado qué patrones arquitectónicos son más relevantes y qué elementos computacionales pueden soportarlos.

Capítulo 4. EXPERIMENTACIÓN

Esta sección muestra una versión alternativa al trabajo desarrollado en el proyecto PSI3 [PSI3 01]. El objetivo del proyecto PSI3 es la personalización de contenidos en Internet. La personalización consiste en presentar al usuario contenidos que se adapten a sus gustos. Para ello se utilizan técnicas de filtrado colaborativo junto con categorización estadística basada en redes bayesianas.

El capítulo comienza con un resumen de la especificación del trabajo a realizar. A continuación se presentan varias iteraciones de las fases de análisis y diseño a lo largo del RUP. En concreto, se presentan los resultados de las etapas Análisis-Inicio, Análisis-Elaboración, Diseño-Elaboración, Análisis-Construcción y Diseño-Construcción. Dentro de cada etapa, se detalla qué actividades de las identificadas en el capítulo anterior se aplican y qué resultados se obtienen. Las actividades se referencian mediante el nombre y número asignado en el capítulo tercero para identificarlas más fácilmente. Para no oscurecer el ejemplo, se han separado ambos aspectos dentro de cada etapa.

La especificación completa del sistema es demasiado larga para presentarla dentro de este capítulo. El propósito del capítulo es ilustrar la aplicación de las ideas del capítulo tercero. Por ello se ha incluido en esta sección sólo parte del trabajo realizado. La especificación completa puede revisarse en http://grasia.fdi.ucm.es/metodologia.

4.1 El sistema a desarrollar

El sistema a desarrollar es un complemento a los servicios que se ofrecen en un web comercial. El beneficio que obtiene la empresa que lo instala es un sistema de información personalizada en base a los contenidos que la empresa elija y una segmentación de sus usuarios en grupos de intereses comunes.

El sistema se concibe como un Sistema Multi-Agente en el que el usuario es representado por un *Agente Personal* que se agrupa con otros *Agentes Personales* para formar comunidades, representadas a su vez por *Agentes de Comunidad*. El motivo de que se dispongan estas comunidades es favorecer la segmentación de intereses (un tema por comunidad) porque con esta segmentación es más sencillo asegurar la calidad de los documentos proporcionados a cada usuario.

Las comunidades pueden verse como fuentes de información a las que el usuario se suscribe buscando información relevante a sus intereses. Una vez suscrito, el usuario comienza a recibir información de la comunidad. La información recibida se origina en los propios miembros de la comunidad o en la asociación de otras fuentes de información a la comunidad no especificadas. La información que llega al usuario debe pasar por una serie de filtros para asegurar su calidad. Cuando un usuario proporciona una sugerencia a la comunidad, la comunidad primero coteja la sugerencia con el perfil asociado a la comunidad. Si la comparación es satisfactoria, el documento pasa a ser evaluado por un conjunto de miembros de la comunidad. Antes de acudir al usuario, cada Agente Personal intenta decidir si el documento puede interesar o no a su usuario. En caso afirmativo, la petición de evaluación pasa al usuario para que este evalúe el documento recibido. En caso negativo, se emite automáticamente un voto en contra.

Tras la votación, la comunidad difunde la sugerencia sólo si la mayoría de los usuarios consultados han votado a favor del documento. Las evaluaciones positivas y negativas se contabilizan durante el proceso, influyendo en la aceptación de futuras sugerencias por parte de la comunidad y sus miembros. Puede llegar el caso en que sea necesario cuestionarse la pertenencia a la comunidad de un usuario que insiste en proporcionar información que a nadie interesa. Ello plantea una serie de políticas de permanencia en la comunidad y de aceptación de sugerencias.

- **Echar a los usuarios que sólo sugieren documentos evaluados negativamente.** Se trata de usuarios molestos cuyos intereses no coinciden con los de la comunidad.
- **Echar a los usuarios que evalúan negativamente demasiados documentos.** Son usuarios que aunque no hayan suministrado información criticable, sí que han demostrado que no les interesa el tipo de información que aquí se proporciona.

Agentes de Comunidad y Agentes Personales se caracterizan por un perfil que puede ser un conjunto de documentos, palabras clave o categorías. Los perfiles evolucionan en ambos agentes según las actuaciones de sus usuarios (miembros de la comunidad en el caso de los Agentes de Comunidad). La evolución para el caso del conjunto de documentos se describe por una ventana de documentos de anchura N, esto es, los N últimos documentos evaluados positivamente por el usuario. Para el resto de los casos, palabras clave o categorías, se admiten únicamente modificaciones realizadas por el usuario.

El sistema a desarrollar ha de admitir la incorporación de nuevas fuentes de información, en concreto foros de noticias y réplicas de este sistema. Nuevas noticias publicadas en un foro pueden ser de interés para los miembros de una comunidad cuya temática esté relacionada. Otras réplicas de este sistema podrían colaborar entre sí para intercambiar información interesante. Debe permitirse el intercambio de información siempre y cuando haya sido debidamente autorizado por el administrador del sistema.

Ambas alternativas están sirven para inyectar información adicional en las comunidades, salvando así una evantual pasividad de los usuarios.

Los agentes deben poder ubicarse en distintas máquinas para distribuir la carga del sistema. Además, los agentes no podrán estar operativos permanentemente, ya que el número de usuarios es demasiado grande como para tener un número igual de Agentes Personales permanantemente activados. Se ha comprobado que cada usuario se conecta una media de una hora al día. Así, en un sitio web con diez mil usuarios registrados, cada hora se pueden encontrar 300 usuarios conectados. Esto permite que la carga media que debe soportar el sistema no sea de diez mil agentes, sino sólo trescientos. Para lograr esto, los agentes han de poder pararse, almacenando su estado previamente, para reanudar más tarde su funcionamiento.

Los usuarios se conectan con los agentes mediante una interfaz web que les permite:

- Sugerir documentos
- Evaluar documentos
- Ver documentos sugeridos por las distintas comunidades

Ver estadísticas de funcionamiento Adicionalmente, existe la figura de administrador del sistema que se encarga de:

- Definir cuántos agentes pueden existir en el sistema.
- Anexionar nuevas máquinas al sistema, incrementando así el número posible de agentes activos.
- Eliminar máquinas del sistema.
- Crear nuevas comunidades de usuarios.
- Eliminar comunidades con un número de usuarios bajo.
- Eliminar agentes cuyos usuarios han permanecido inactivos demasiado tiempo.
- Configurar parámetros de ejecución de los agentes. Umbrales de aceptación de documentos, veces que un usuario puede evaluar negativamente un documento antes de ser echado de la comunidad, número de usuarios a seleccionar cada vez que haya que evaluar un documento.

4.2 Análisis – Inicio

En esta etapa el objetivo es convencerse de que el desarrollo es posible. Para ello se identifican los casos de uso más importantes que reflejen los problemas principales que se van a encontrar y cuáles van a ser los componentes del sistema que participarán en su resolución.

4.2.1 Estructuración de la etapa

La identificación y gestión de los casos de uso se hace con UML según lo establecido en el RUP. La identificación de los componentes se hace mediante un modelo de organización y un modelo de entorno. Serían necesarios también modelos de agente para identificar los

agentes si no fuera porque la especificación del problema ya establece qué agentes existen. El modelo de organización dará una vista global de cómo se organizan estos elementos, mientras que el de entorno en esta etapa establecerá con qué elementos del sistema previo se necesita interactuar.

El modelo de entorno se genera ejecutando las actividades *Identificar aplicaciones del entorno* (1.a) que produce las aplicaciones mencionadas por la especificación (*ServidorAplicaciones* y *BaseDatos*), *Asociar operaciones sobre aplicaciones* (2) que producen las operaciones asociadas a los elementos anteriores, y *determinar la percepción de los agentes* (6) que produce la asociación entre el *Agente Personal* y *Servidor de Aplicaciones*.

Actividad	**Tipo de resultado**	**Referencia**
Identificar aplicaciones del entorno (1.a)	Un conjunto de *aplicaciones*	Ilustración 100
Asociar operaciones sobre aplicaciones (2)	Operaciones sobre las aplicaciones	
Determinar la percepción de los agentes (6)	Asociaciones entre agentes y *aplicaciones*	

La organización se determina mediante *identificar grupos* (1.a) que produce la estructuración de la organización en comunidades, *generar miembros* (1.b) que produce la asociación de los agentes a las comunidades y la asociación de las aplicaciones al grupo *administración*. Las relaciones sociales no se consideran en esta etapa y en cuanto a la funcionalidad, se restringe a la identificación de casos de uso relevantes y a la ejecución de la actividad *identificar objetivos* (2.c). Los objetivos obtenidos se corresponden con los objetivos generales del sistema, ayudar en la distribución de documentos y mantener la calidad de los documentos proporcionados.

Actividad	**Tipo de resultado**	**Referencia**
identificar grupos (1.a)	Un conjunto de *grupos*	Ilustración 101
generar miembros (1.b)	Elementos que componen los *grupos*	
identificar objetivos (2.c).	Objetivos asociados a las organizaciones	

4.2.2 Resultados obtenidos

Los casos de uso relevantes se muestran en la Ilustración 99. Corresponden a las situaciones que se pueden presentar en una comunidad. Se han omitido los casos de uso de administración para restringir el desarrollo a la parte de agentes. Los casos de uso tratan principalmente del desarrollo del trabajo en las comunidades

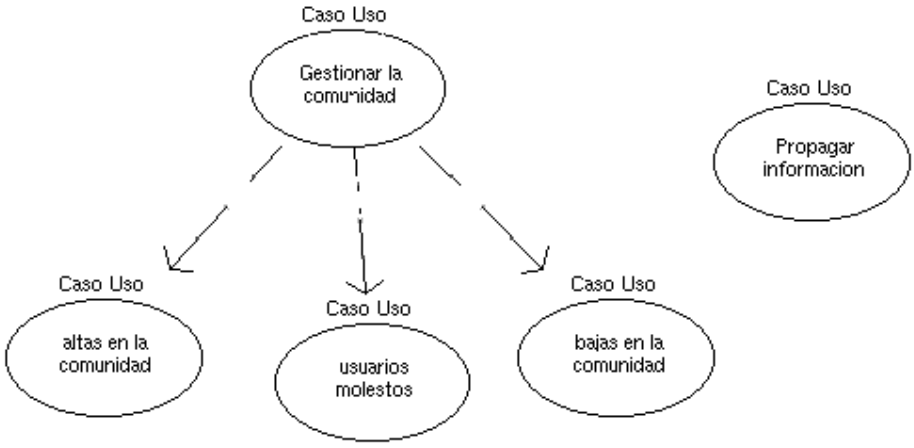


Ilustración 99. Casos de uso asociados al problema

- **Gestionar la comunidad**. La comunidad debe poder tratar altas, bajas y asegurar que los usuarios que tengan un comportamiento molesto sean echados de la comunidad.
  - **Usuarios molestos.** Los usuarios molestos son aquellos que hacen que el resto de miembros de la comunidad reciban información que no gusta o bien que se dedican a criticar negativamente todo cuanto reciben.
  - **Altas en la comunidad.** Las altas de la comunidad se realizan de forma democrática consultando a los miembros existentes. Para asegurar la adecuación del nuevo miembro, éste pasa por un proceso de selección similar al de evaluación de sugerencias.
  - **Bajas en la comunidad.** Las bajas se realizan a petición del usuario y no son sometidas a ninguna discusión
- **Propagar información**. La información se propaga entre los usuarios de la comunidad de forma selectiva siguiendo los pasos indicados en la especificación.

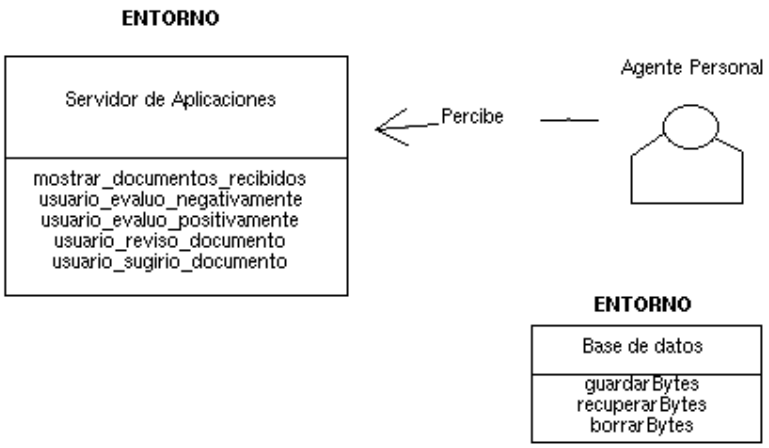


Ilustración 100. Entidades identificadas en el entorno

De la fase de captura de requisitos, se tiene que existen en el entorno (Ilustración 100):

- **Base de datos.** Donde se debe almacenar el estado de los agentes
- **Servidor de aplicaciones**. Representa al conjunto de programas que se utilizan para interaccionar con el usuario en la actualidad. Esto incluye la interfaz web. Dado que la percepción del agente personal está relacionada con las acciones que ejecuta el usuario y que éstas se procesan en el interfaz web, debe existir una relación de percepción entre el agente y el servidor de aplicaciones.

Con esta información se propone la siguiente arquitectura del SMA. La arquitectura muestra que debe existir una comunidad representada por un *Agente de Comunidad* y un conjunto de *Agentes Personales*, tal y como indica la especificación.

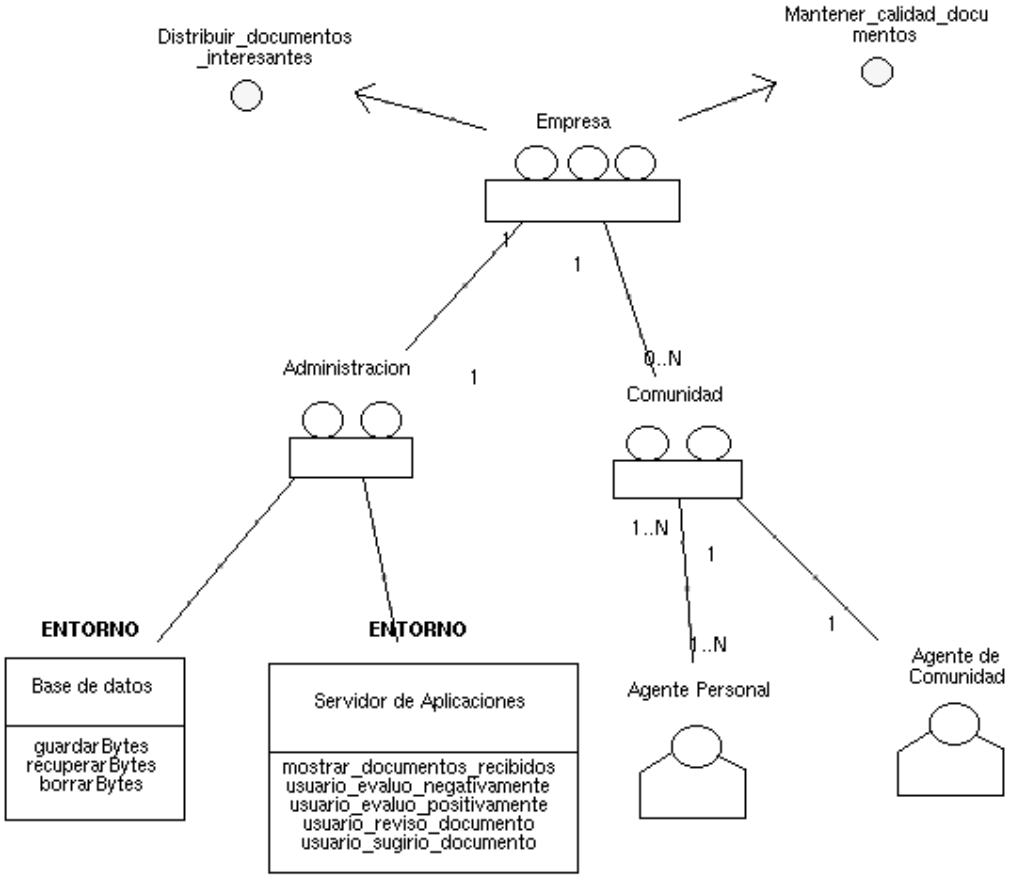


Ilustración 101. Arquitectura tentativa del sistema

La organización *Empresa* inicialmente persigue dos objetivos: incrementar la colección de documentos interesantes para los usuarios (*Distribuir_documentos_interesantes*) y mantener la calidad de los que ya posee (*Mantener_calidad_documentos*). Esto se justifica porque si aumentan los documentos, y la calidad de los que llegan es mala (de acuerdo con la opinión de los usuarios), entonces la calidad conjunta es mala.

Dentro de esta organización se define el grupo de *Administración*, encargado de gestionar recursos comunes, y *Comunidad*, cuyo número no está acotado. Pueden existir tantas comunidades como se desee. Por cada comunidad existe un único *Agente de comunidad* y varios *Agentes Personales*, mientras que un *Agente Personal* puede pertenecer a varias comunidades. Las tareas de gestión de la comunidad (altas y bajas de los miembros) serán responsabilidad del *Agente de Comunidad*.

4.3 Diseño-Inicio

Según el RUP, en esta fase hay que generar un propotipo del sistema. En el capítulo tercero se proponía elaborar este prototipo con un entorno de desarrollo como ZEUS o agentTool. Sin embargo, en este dominio concreto, la funcionalidad principal se muestra mejor con un prototipo elaborado en HTML. En este prototipo se mostraría cómo interactúa el usuario con la aplicación y qué tipo de resultados se obtendrían.

4.4 Análisis-Elaboración

Se analizan los casos de uso que se consideran relevantes para obtener la arquitectura del sistema, en este caso, la estructura de la organización y los flujos de trabajo fundamentales. Los casos de uso de la etapa anterior se refinan y se asocian con modelos de interacciones para estudiar cómo se llevan a cabo.

4.4.1 Estructuración de la etapa

Cada modelo de interacción se inicia con la actividad *identificar los objetivos que persigue la interacción* (1), que puede partir del conjunto de objetivos identificados en la organización, *identificar su naturaleza* (3), que dados los objetivos no puede ser otra cosa que cooperación, e *identificar los participantes* (2), en este caso roles para agrupar la funcionalidad que se irá descubriendo. La especificación inicial de las interacciones se hace mediante diagramas de colaboración de UML, siguiendo la actividad *generar una primera especificación* (4).

Actividad	**Tipo de resultado**	**Referencia**
Identificar los objetivos que persigue la interacción (1)	Un conjunto de objetivos	Ilustración 103 Ilustración 106 Ilustración 109
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	Ilustración 107 Ilustración 110 Ilustración 104

Para conocer mejor los agentes de la organización de la etapa anterior, se elaboran modelos de agente. La actividad *Identificar agentes utilizando el principio de racionalidad* (1) no es necesaria ya que se conocen de antemano qué agentes van a existir, no obstante se puede utilizar esta actividad para verificar la corrección de la decisión. Los objetivos del agente, actividad *identificar los objetivos* (2.a) se deducen a partir de los objetivos de la organización y del papel que se prevee van a jugar los agentes en ella. La funcionalidad

surge asociando tareas a los objetivos, ya identificados, y roles al agente según la actividad *identificar funcionalidad* (2.b) y *asociar funcionalidad con objetivos* (2.c). Los aspectos de autonomía e inteligencia son función directa de la especificación del comportamiento del agente (modelos de interaccion, organización, tareas y objetivos). Por ello, antes de ejecutar la actividad *identificar aspectos de autonomía e inteligencia* (3) y *determinar cómo será el gestor y procesador de estado mental* (4) sería conveniente avanzar los otros modelos.

Actividad	**Tipo de resultado**	**Referencia**
Identificar agentes utilizando el principio de racionalidad (1)	Un conjunto de agentes	Ilustración 118
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	Texto adjunto
Determinar cómo será el gestor y procesador de estado mental (4)	Descripción en lenguaje natural	Texto adjunto

Todos los objetivos y tareas identificados se organizan dentro de modelos de tareas y objetivos. Estas tareas y objetivos aparecen tras la ejecución de las actividades *Identificar tareas y objetivos* (1) y *asociar tareas a objetivos* (2). Tras estas actividades, se plantea la descomposición de objetivos y tareas, que se realiza mediante las actividades *descomponer objetivos* (4) y *descomponer tareas* (3) respectivamente. En este caso se ha preferido ahondar en la descomposición de objetivos para relacionar con tareas los objetivos hoja del grafo acíclico resultante. La satisfacción de los objetivos intermedios se realiza mediante dependencias según la actividad *establecer dependencias entre objetivos* (5). Las tareas se detallan más indicando si producen interacciones, *asociar tareas con interacciones* (6.a), si producen o consumen entidades mentales, *asociar tareas con entidades producidas* (6.c) y *consumidas* (6.b), y qué *aplicaciones* requieren, *asociar tareas con aplicaciones* (6.d).

Actividad	**Tipo de resultado**	**Referencia**
Identificar tareas y objetivos (1)	Un conjunto de tareas y objetivos	Referidos en otras ilustraciones
Descomponer objetivos (4)	Asociaciones de descomposición entre objetivos	Ilustración 111 Ilustración 112
Asociar tareas a objetivos (2	Asociaciones entre objetivos y tareas	Ilustración 113

Descomponer tareas (3)	Asociaciones de descomposición entre tareas	Ilustración 108
Establecer dependencias entre objetivos (5).	Dependencias	No es necesaria
Asociar tareas con interacciones (6.a)	Instancias de *WFProduce* asociadas a entidades *Interacción*	Ilustración 114 Ilustración 115
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	

Una vez decidido cómo se alcanzan los objetivos, surgen nuevas necesidades que han de representarse con modelos de entorno. Se trata de las aplicaciones requeridas por las tareas para ejecutar su cometido. Esta vez, se ejecutan las actividades *identificar aplicaciones internas* (1.b), *asociar operaciones* (2) y *actividades del análisis de las aplicaciones* (3) (los resultados de esta actividad no se han incluido) para describir estas aplicaciones requeridas por las tareas. Esta vez, la información de la que se dispone es mayor, ya que se tienen interacciones en las que se enmarcan las tareas y asociaciones con objetivos señalando qué se espera de ellas.

Actividad	**Tipo de resultado**	**Referencia**
Identificar aplicaciones internas (1.b)	Un conjunto de instancias de *aplicación interna*.	Ilustración 119
Asociar operaciones (2)	*Operaciones* sobre las *aplicaciones*	
Actividades del análisis de las aplicaciones (3)	Diagramas UML para expresar comportamiento	Omitido
Asociar aplicaciones a grupos	Un conjunto de aplicaciones	Ilustración 119
Identificar recursos	Un conjunto de recursos	Ilustración 120

Para terminar, el modelo de organización recoge todos estos resultados y procede a su estructuración. Se asocian las nuevas aplicaciones con la actividad *generar miembros* (1.b). Cómo ya se tienen un conjunto razonable de tareas, procede agruparlas en flujos de trabajo,*identificar flujos de trabajo* (2.a) y refinar estos grupos de trabajo, *aplicar descomposición de flujos* (2.b). Las relaciones sociales todavía no son necesarias.

Actividad	Tipo de resultado	Referencia
Generar miembros (1.b)	Entidades asociadas a instancias de *grupo*.	Ilustración 121
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	Ilustración 122

4.4.2 Resultados obtenidos

Los casos de uso iniciales se refinan en la Ilustración 102. El caso de uso *Usuarios molestos* se refina en dos nuevos: *Monitorización de los usuarios* y *Echar usuarios*. *Propagar información* también se refina creando el nuevo caso de uso *Sugerencia directa de documentos*. Por último, para decidir si la información suministrada es de calidad, se plantea la necesidad de *evaluar la información*.

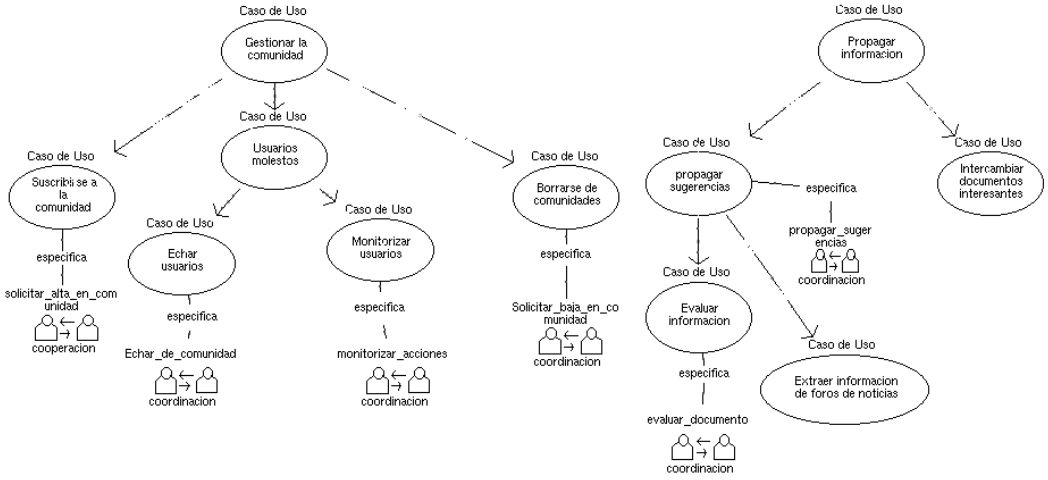


Ilustración 102. Casos de uso asociados

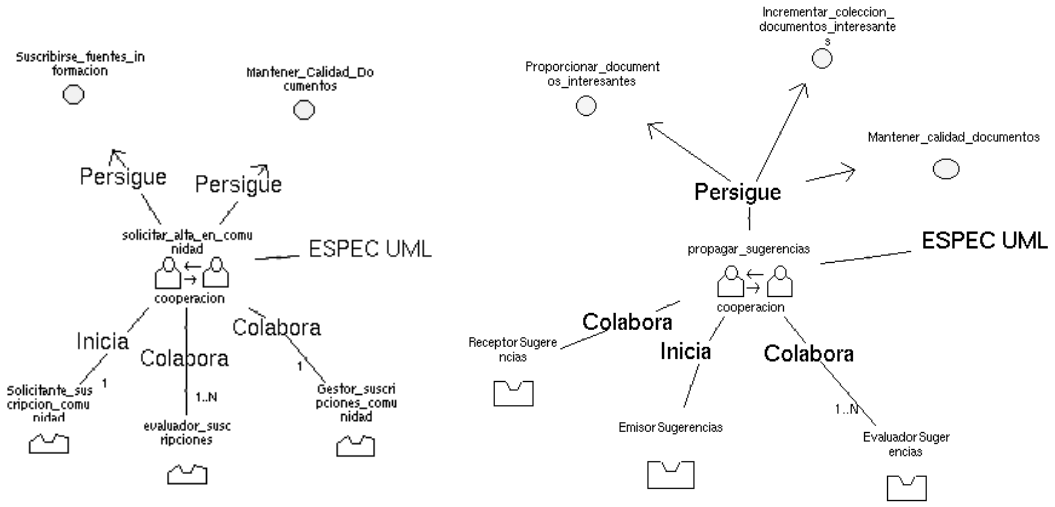
A continuación se muestra una descripción más detallada de los casos de uso:

- **Monitorizar usuarios**. Los usuarios molestos se detectan recogiendo estadísticas acerca de sus acciones. En concreto interesan:
  - Número total de veces que un documento del usuario ha sido evaluado negativamente.
  - Número total de veces que un documento del usuario ha sido evaluado positivamente.
  - Número total de veces que un usuario ha evaluado positivamente algún documento.
  - Número total de veces que un usuario ha evaluado negativamente algún documento.

- **Echar usuarios molestos.** A los usuarios molestos se les echa de la comunidad dándoles detalles acerca de los motivos de la expulsión.
- **Propagar sugerencias** Se permite a los usuarios sugerir información en las comunidades a las que están suscritos.
- **Evaluar información.** Define el proceso de filtrado de información tomando como referencia un único evaluador. Este caso de uso se reutiliza para decidir si las sugerencias se propagan a los usuarios finales.
- suscribirse a la comunidad
- **borrarse de comunidades.**
- **Extraer información de foros de noticias**. Se trata de utilizar un foro de noticias como fuente de información. Dado un foro cuya temática se considere interesante de acuerdo con los gustos de los miembros de una comunidad, sería apropriado que los nuevos anuncios insertados en el foro se retransmitieran al resto de la comunidad.
- **Intercambiar documentos interesantes**. El intercambio de información no tiene por qué ser restrictivo a una organización. Varias organizaciones pueden, de mutuo acuerdo, intercambiar información de sus comunidades.

De estos casos de uso, se toman como relevantes para la arquitectura del sistema sólo los cuatro primeros. Para la extracción de noticias, se puede tomar un agente personal modificado que extienda su percepción al foro de noticias y así poder observar nuevos anuncios. Para el intercambio de documentos, habría que crear un nuevo agente con la capacidad de participar en las comunidades actuando como representante de una organización. En el funcionamiento de este agente serán necesarios elementos no contemplados con anterioridad, sin embargo, estos cambios no afectarán a la arquitectura básica del sistema, de hecho se apoyarán en ella.

Para detallar lo que se pretende con cada caso de uso, se utilizan los modelos de interacción que se muestran a continuación. Los roles que se muestran en cada una de las interacciones provienen de los diagramas de colaboración generados como especificación inicial.



(A) (B)

Ilustración 103. Interacciones para detallar la realización de casos de uso *altas en comunidad* (A), y *propagar sugerencias* (B).

La interacción *solicitar alta en comunidad* (ver Ilustración 106 (A)) se inicia para disponer de más fuentes de información (objetivo *suscribirse fuentes de información*) y se ejecuta de forma que el nuevo usuario sea compatible con la temática tratada en la comunidad (objetivo *mantener calidad de documentos*).

La interacción *propagar sugerencias* (ver Ilustración 106 (B)) se inicia para incrementar la cantidad de documentos relevantes para los usuarios (*incrementar colección de documentos interesantes*) pero manteniendo la calidad de los documentos (*mantener calidad documentos*). Los usuarios colaboran en el mantenimiento de calidad porque buscan documentos que les interesen (*buscar documentos interesantes*).

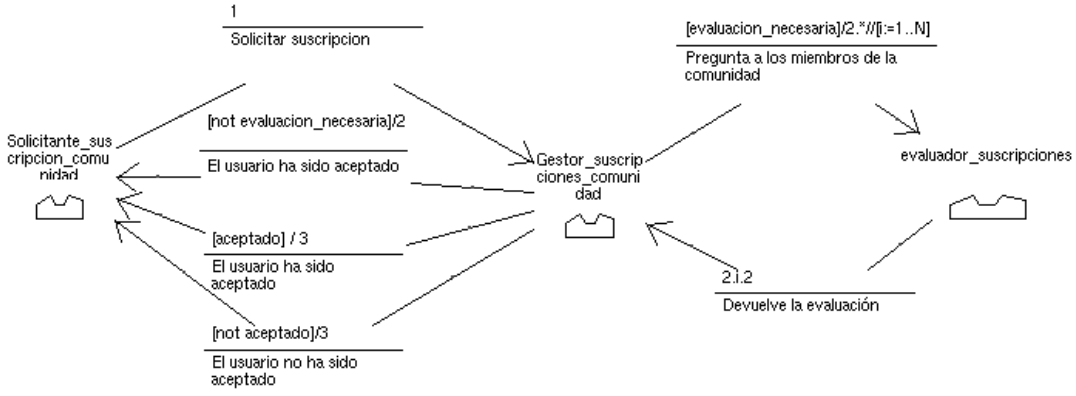


Ilustración 104. Diagrama de colaboración para la interacción *solicitar alta en comunidad*

Para garantizar la idoneidad de un nuevo miembro con respecto a los gustos de una comunidad (ver Ilustración 104) se sigue un proceso similar al de propagación de sugerencias (ver Ilustración 105). El usuario, al solicitar su entrada en la comunidad, proporciona un resumen de los gustos que le caracterizan. Este resumen es comparado con un perfil de la comunidad, para determinar si merece la pena seguir adelante y consultar a los otros miembros. Si es así, se inicia el proceso de evaluación para N miembros seleccionados de entre los miembros de la comunidad. Al término de las evaluaciones, se decide si aceptar finalmente al usuario o no.

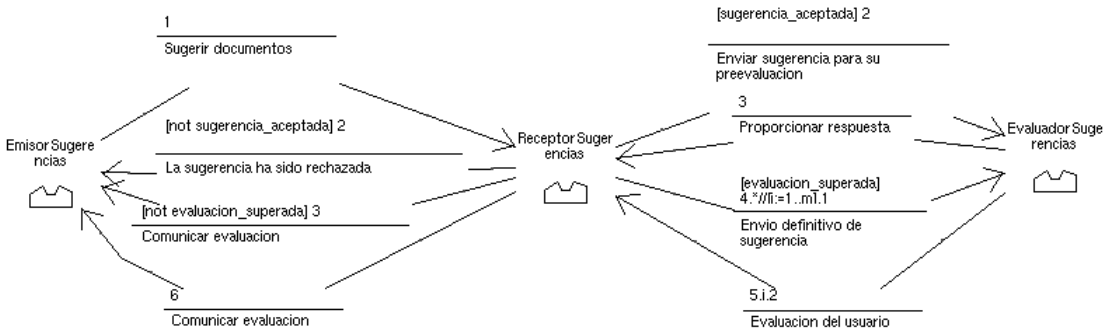
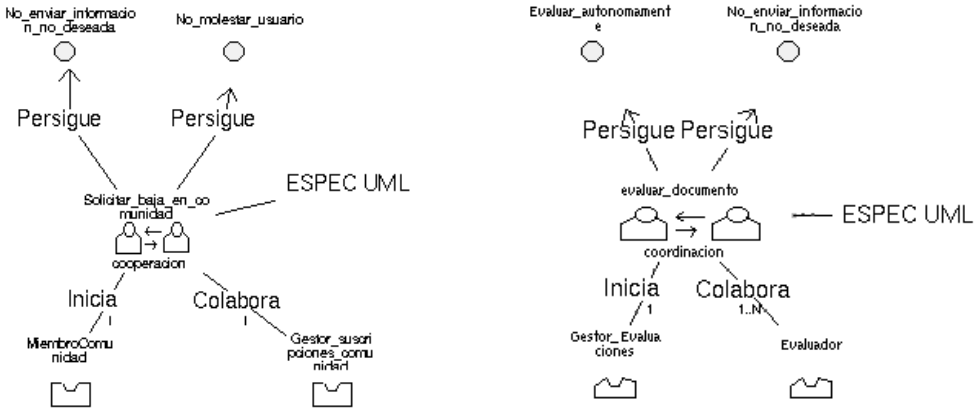


Ilustración 105. Diagrama de colaboración para la interacción *propagar sugerencias*.

El proceso de tramitación de sugerencias es similar al otro en cuanto que se precisa de la opinión de los miembros de la comunidad. La diferencia radica en que las sugerencias, si son evaluadas positivamente por usuarios y comunidad, se radian al resto de miembros y se recogen sus opiniones nuevamente. El resultado es evaluado en el emisor de *sugerencias* de forma que se preserva el anonimato de los evaluadores.



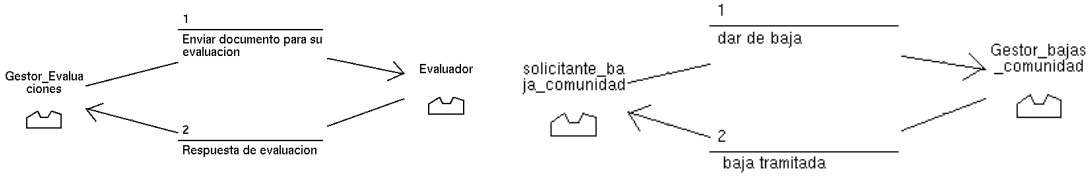
(A) (B)

Ilustración 106. Interacciones para detallar la realización de casos de uso *bajas en comunidad* (A) y *evaluar información* (B).

Ambos procesos, donde se solicita la opinión de los usuarios, convierten el sistema de gestión de la comunidad en una democracia donde todos opinan alguna vez sobre lo que se hace en la comunidad.

La interacción *solicitar baja en comunidad* (Ilustración 106 (A)) muestra el proceso que permite a un miembro de la comunidad darse de baja. En principio este proceso obedece a la voluntad del usuario, sin embargo también puede ser decisión del agente motivada por una serie de evaluaciones negativas por parte del usuario de documentos procedentes de una comunidad. En cualquiera de los casos, la baja es por insatisfacción del usuario. Por ello, se han indicado que la interacción persigue no molestar más al usuario (objetivo *no molestar usuario*) y no recibir más información de una comunidad (objetivo *no enviar información no deseada*).

La interacción *evaluar documento* (Ilustración 106 (B)) expresa la evaluación de un documento. El propósito de esta interacción es mostrar cómo se consigue no molestar al usuario con el proceso de evaluación. Así, se tienen como objetivos el evaluar autónomamente la información que llega (*evaluar autónomamente*) con la menor intervención posible del usuario y que esta evaluación no termine en información no deseada por el usuario (*No enviar información no deseada*).



(A) (B)

Ilustración 107. Diagrama de colaboración para la interacción *solicitar baja en comunidad* (B) y la interacción *evaluar información* (A)

Los diagramas de colaboración en este caso son limitados a propósito. El caso de la Ilustración 107 (A) es comprensible ya que una baja en una comunidad no reviste un problema aparente. Sin embargo para el caso Ilustración 107 (B), la viabilidad del proceso no está clara. Para detallarlo un poco, se utilizará un modelo de objetivos y tareas (Ilustración 108) donde se muestren las tareas involucradas.

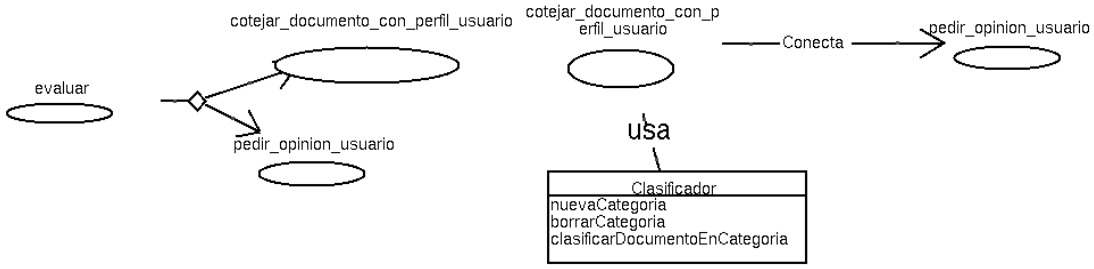
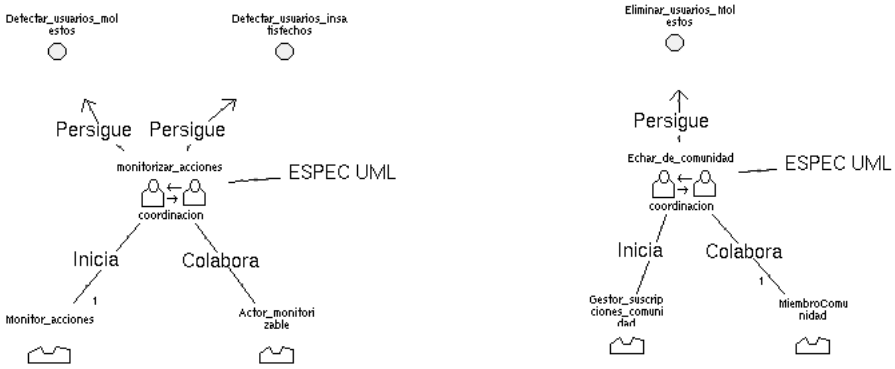


Ilustración 108. Tareas involucradas en la evaluación de documentos

El proceso de evaluación autónoma es posible mediante la utilización de clasificadores que permitan determinar en qué medida un documento pertenece a una categoría. Esta

categoría es una abstracción de un sub-conjunto de los gustos del usuario expresados como conjuntos de páginas.

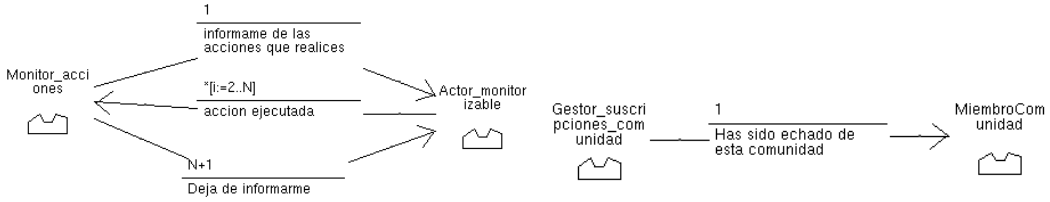


(A) (B)

Ilustración 109. Interacciones para la realización de los casos de uso *monitorizar usuarios* (A) y *echar de comunidad* (B)

La interacción *monitorizar acciones* (Ilustración 109 (A)) tiene como objetivos la monitorización de las acciones de los usuarios con vistas a detectar usuarios que disturben el orden de la comunidad (objetivo *detectar usuarios molestos*) y usuarios que no estén satisfechos con la información recibida (objetivo *detectar usuarios insatisfechos*).

La interacción *echar de comunidad* (Ilustración 109 (B)) es similar a la de *solicitar baja en la comunidad* (Ilustración 106 (A)) salvo que ésta se invoca cuando la presencia del usuario no es bien recibida. El único motivo que justifica el echar a un usuario es que se trate de un usuario molesto (objetivo *eliminar_usuarios_molestos*).



(A) (B)

Ilustración 110. Diagramas de colaboración para las interacciones *monitorizar usuarios* (A) y *echar de comunidad* (B)

Para lograr los objetivos anteriores se plantean los diagramas de colaboración de la Ilustración 110. El monitorizar usuarios se ve como un *patrón observador* [Gamma et al. 95]. El diagrama de colaboración de la Ilustración 110 (A) utiliza las primitivas de expresiones secuenciales de UML para reflejar que existe un número no determinado de notificaciones antes del final del cese de la observación. El echar a usuarios de la comunidad no requiere mayor comentario.

Los objetivos encontrados en las interacciones y arquitectura inicial del sistema se estructuran utilizando un modelo de tareas y objetivos.

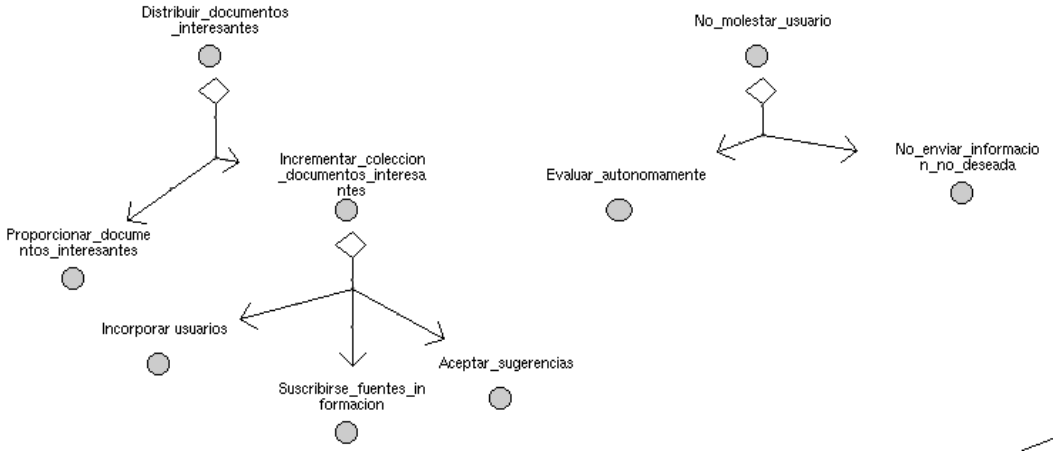


Ilustración 111. Estructuración inicial de objetivos (I)

La distribución de documentos interesantes se descompone en la voluntad de proporcionarlos (*Proporcionar documentos interesantes*) y la voluntad de aceptarlos (*incrementar colección de documentos interesantes*).

En paralelo, se persigue molestar al usuario lo menos posible (*no molestar usuario*) que aquí se entiende como el evaluar autónomamente cuando sea posible (*evaluar autónomamente*) y que el usuario no reciba información que, según las estimaciones realizadas, no le van a gustar (*no enviar información no deseada*).

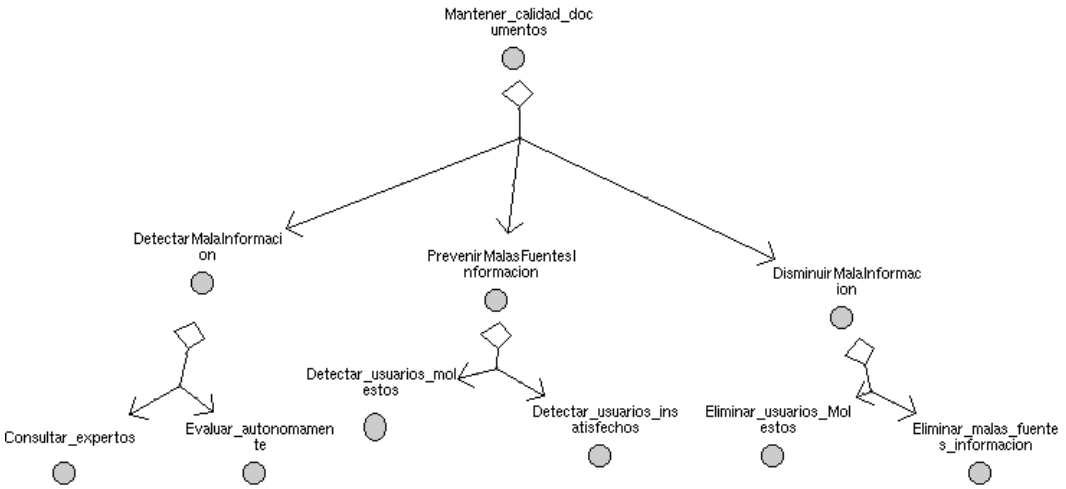


Ilustración 112. Estructuración inicial de objetivos (II)

Para terminar, se busca mantener la calidad de los documentos del sistema (*mantener calidad documentos*), para lo cual se plantean tres frentes: detectar malas fuentes de información (*detectar mala información*), prevenir que usuarios se puedan convertir en malas fuentes de información (*prevenir malas fuentes información*) y disminuir la mala información en la organización (*disminuir mala información*). Para detectar malas fuentes de información se debería perseguir o bien una evaluación automática de la información

contenida en esta fuente (*evaluar autónomamente*) o bien consultar a expertos, los usuarios, para que den su opinión acerca de la calidad de la fuente (*consultar expertos*). Para prevenir que las fuentes existentes se corrompan, se debería detectar qué usuarios molestan con la información que sugieren (*detectar usuarios molestos*), qué usuarios no proporcionan información acorde con la temática de la comunidad (*detectar usuarios molestos)* o que evalúan negativamente toda la información que les llega (*detectar usuarios insatisfechos*). El último frente, disminuir la mala información, se consigue echando de la comunidad a aquellos usuarios identificados como molestos (*eliminar usuarios molestos*) y haciendo que los usuarios puedan salirse de la comunidad cuando no estén satisfechos con los resultados (*eliminar malas fuentes de información*).

Las dependencias entre objetivos (instancias de *GTAfecta*) no se muestran aquí ya que en este caso coinciden exactamente con las relaciones de descomposición. Para satisfacer estos objetivos, se definen tareas que más tarde se incorporarán a los flujos de trabajo de la organización.

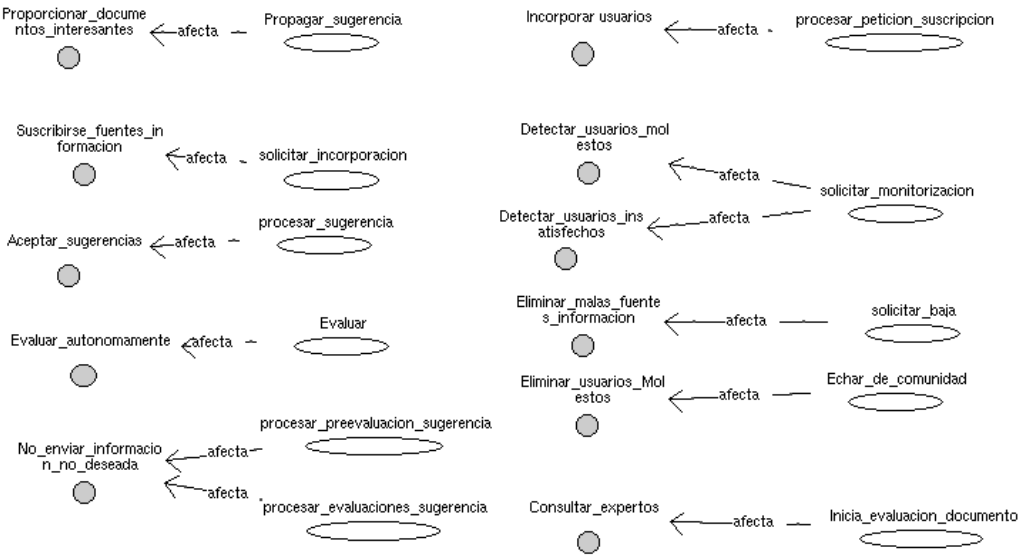


Ilustración 113. Tareas asociadas a los objetivos

La identificación de tareas ha sido guiada por los objetivos que se han encontrado en los modelos de interacción. Para cada objetivo se han creado tareas cuya ejecución pueda enmarcarse dentro de los diagramas de colaboración anteriores. La forma exacta en que intervienen estas tareas se definirá en la siguiente sección:

- **Solicitar monitorización**. Ordena a un agente que informe de todas sus actividades a otro agente. Se utiliza para elaborar estadísticas acerca del comportamiento de los usuarios. Gracias a estas estadísticas, se pueden detectar los usuarios molestos o insatisfechos.
- **Propagar sugerencia.** Un documento suministrado por el usuario se traslada a una comunidad. Mediante esta tarea se busca difundir información que pueda ser interesante para los miembros de una comunidad.

- **Procesar sugerencia.** Toma una sugerencia y la compara contra los gustos conocidos de los miembros actuales de la comunidad. Si el grado de similitud es elevado, se procede a preguntar a un subconjunto de usuarios por su parecer acerca de la calidad de la sugerencia recibida. Esta tarea se ejecuta tras haber recibido una sugerencia de un miembro de la comunidad.
- **Procesar preevaluación sugerencia.** Tras haber recibido una sugerencia, se pregunta a los miembros de la comunidad por su opinión acerca de la calidad del documento. En respuesta, los agentes que representan a los usuarios generan evaluaciones positivas y negativas. El cometido de esta tarea es decidir, una vez recibidas las evaluaciones, si la sugerencia se difunde al resto de los miembros de la comunidad o bien si su calidad no es la suficiente. El criterio es comparar los votos a favor y en contra para comprobar si a la mayoría le gusta el documento sugerido.
- **Procesar evaluaciones sugerencia**. Tras haber difundido la sugerencia, se procesan las reacciones de los usuarios. Esta nueva información también se incorpora al historial.
- **Inicia evaluación del documento**. Esta tarea la ejecuta un *agente de comunidad* para averiguar la opinión de un miembro de la comunidad acerca de la calidad de un documento. La evaluación la puede generar el *agente personal* que representa al miembro consultado o bien el propio miembro.
- **Evaluar.** Evalúa un documento dentro de un *agente personal* de acuerdo con el perfil de su usuario o, si esto no es suficiente, preguntando directamente cual es la opinión de su usuario.
- **Solicitar incorporación.** Pide a una comunidad que le acepte como miembro. La aceptación está condicionada por la opinión de los miembros y por el perfil de la comunidad. Si la comunidad tiene como temática el fútbol y al usuario sólo le interesa, de acuerdo con su perfil, la economía nacional, entonces la solicitud puede ser rechazada. Este hecho puede ser detectado de forma automática, mediante el *clasificador* (ver Ilustración 108) o bien por los propios miembros, mediante votación.
- **Solicitar baja**. La baja en una comunidad se pide cuando el usuario no está satisfecho con la información que recibe. Para procesar la baja, además de eliminar al usuario de la lista de miembros de la comunidad, hay que dar de baja al solicitante en los procesos de evaluación en curso y otros relacionados, como el proceso de monitorización.
- **Procesar petición de suscripción.** Se ejecuta después de que un usuario solicite su incorporación a una comunidad. Esta tarea realiza la comparación de los gustos del usuario, incluidos en la petición, contra los gustos de la comunidad. Si existe un grado de similitud suficiente, la solicitud pasa a ser revisada por un subconjunto de los miembros de la comunidad. Si éstos consienten, el usuario es aceptado.
- **Echar de comunidad**. Echa a un usuario de una comunidad. Para ello se debe descartar al expulsado de cuantos procesos de comunicación existieran, en especial, del proceso de monitorización de acciones de los usuarios, iniciado mediante la tarea *solicitar monitorización* (ver Ilustración 115).

De estas tareas las que producen interacciones son las que se muestran en la Ilustración 114 y la Ilustración 115. Estas se muestran con detalle a continuación. El resto de tareas se describe en http://grasia.fdi.ucm.es/metodologia.

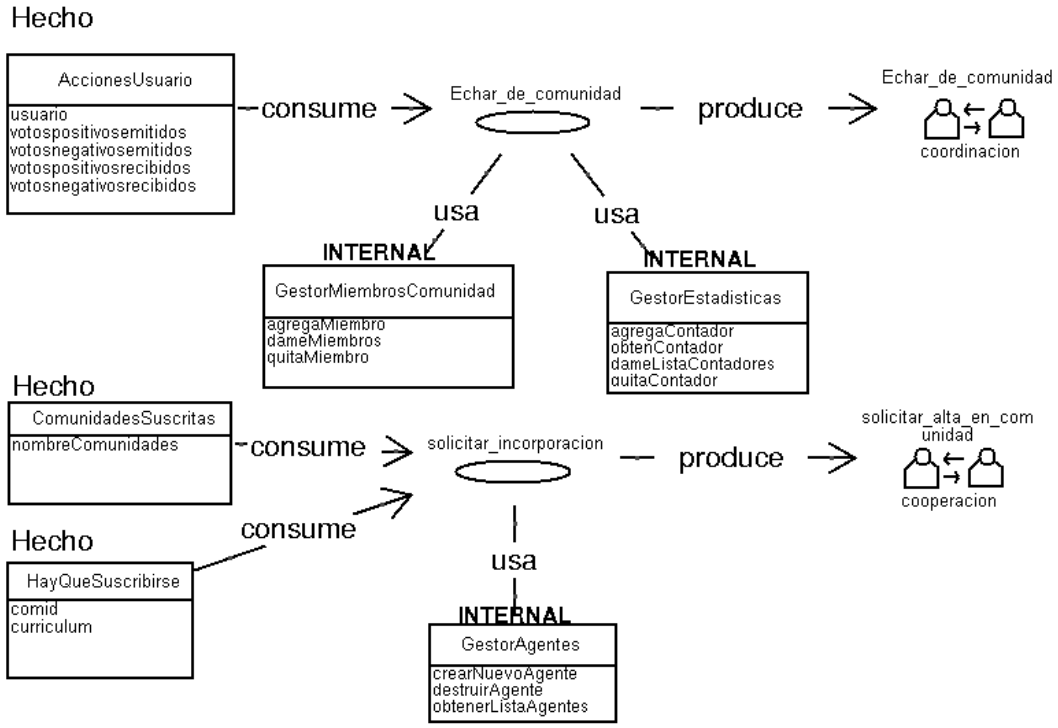


Ilustración 114. Entidades producidas y consumidas por tareas (I)

La tarea *echar de comunidad* toma como entrada las acciones realizadas por el usuario hasta ahora, que se suponen han sido extraídas del *Gestor de Estadísticas*. El criterio para decidir si se echa se indica en un *patrón de estado mental* asociado a la unidad de interacción *echar usuario* en la interacción *echar de comunidad*, que se menciona en la siguiente sección. Para dar de baja al usuario en la comunidad hay que usar el *gestor de miembros de la comunidad* y el *gestor de estadísticas*. La notificación se envía al usuario mediante la interacción *echar de comunidad*.

La tarea *solicitar incorporación* necesita saber a qué comunidad se quiere suscribir el usuario (hecho *hay que suscribirse*) y a qué comunidades pertenece ya (hecho *comunidades suscritas*), para no suscribirse a una comunidad en que ya está suscrito. Para localizar al agente de comunidad correspondiente, se utiliza el *Gestor de Agentes*. La suscripción se realiza dentro de la interacción *solicitar alta en comunidad*.

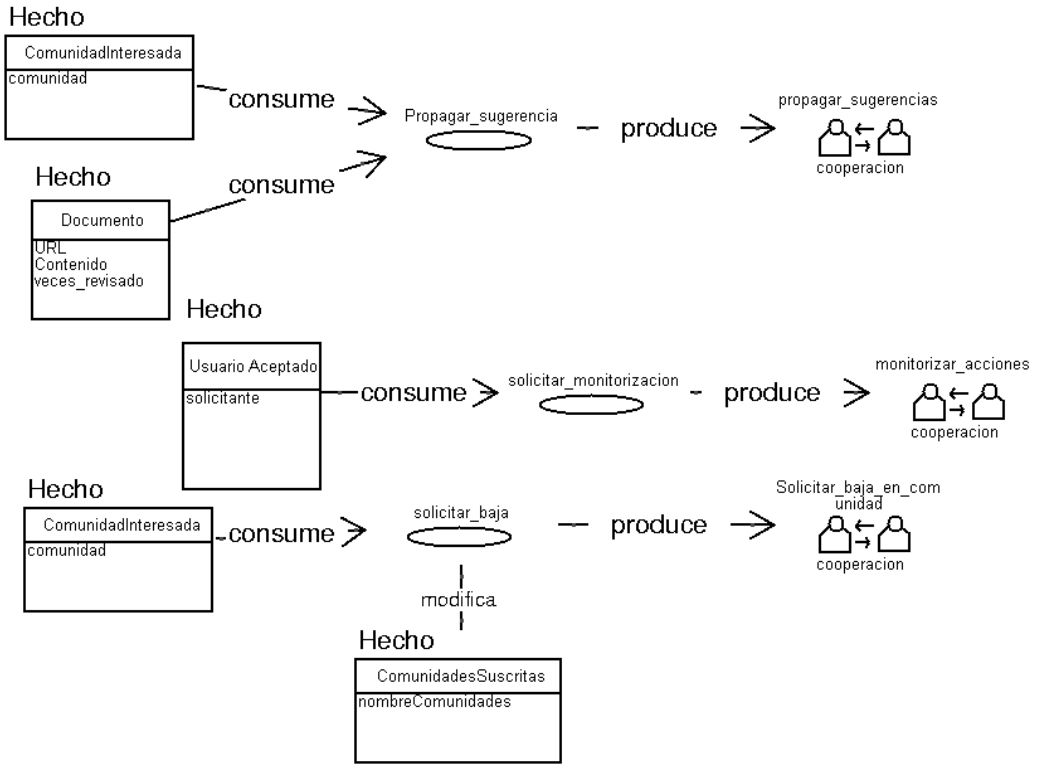


Ilustración 115. Entidades producidas y consumidas por tareas (II)

La tarea *propagar sugerencia* consume el documento a sugerir (hecho *documento*) y la comunidad a la que hay que enviarlo (hecho *comunidad interesada*). Con estos datos, la tarea inicia una interacción *propagar sugerencias*.

La tarea sólo necesita conocer cuándo ha sido un usuario aceptado en una comunidad. Esto se conoce cuando aparece el hecho *usuario aceptado*. Esto es posible porque las tareas implicadas en los procesos de alta en la comunidad producen este hecho automáticamente cuando se ha finalizado el proceso de admisión.

Por último, la tarea *solicitar baja* da de baja a un agente personal de una comunidad. El motivo puede ser por solicitud del usuario o instatisfacción del mismo por la información obtenida de ella hasta ahora. La comunidad se indica con el hecho *comunidad interesada* y la tarea tiene como efecto la modificación del hecho *comunidades suscritas*. La baja se realiza dentro de la interacción *solicitar baja en comunidad*.

La participación de los roles en las interacciones está sujeta a la relación entre los objetivos que persiguen y los asociados a la interacción. Las ilustraciones siguientes muestran la asociación de los objetivos con roles. Se puede comprobar que existe relación directa con los objetivos de las interacciones utilizando como referencia la Ilustración 111 y Ilustración 112.

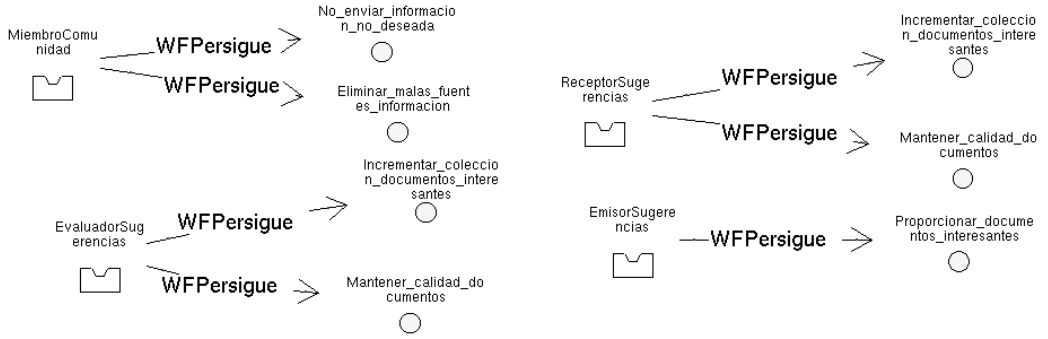


Ilustración 116. Asociaciones entre objetivos y roles (I)

La Ilustración 116 muestra la motivación de los roles *miembro de comunidad*, *evaluador de sugerencias*, *emisor de sugerencias* y *receptor de sugerencias*. El primer rol describe los objetivos generales que persigue un miembro de la comunidad. A estos se añaden los incorporados a los roles *evaluador sugerencias* y *emisor sugerencias*, ya que se trata de roles concretos de los que *miembro de comunidad* es una generalización. El rol *receptor de sugerencias* lo desempeñan *agentes de comunidad* y se encarga de guiar el proceso de propagación de sugerencias (interacción *propagar sugerencias*).

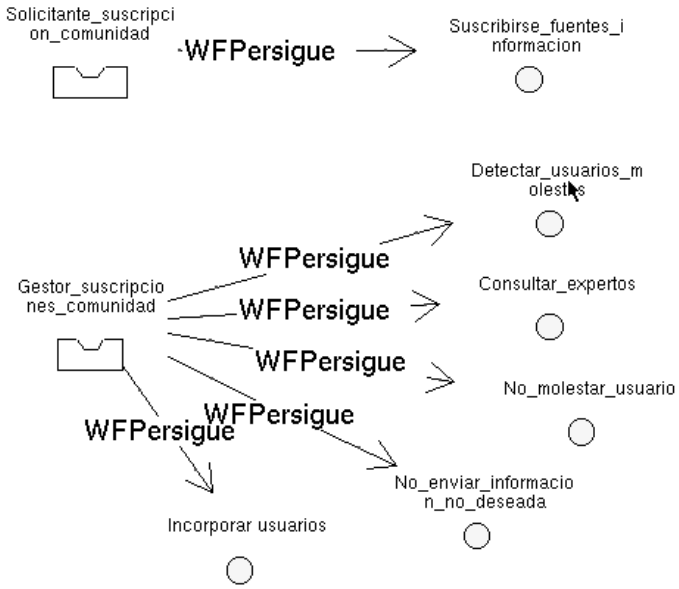
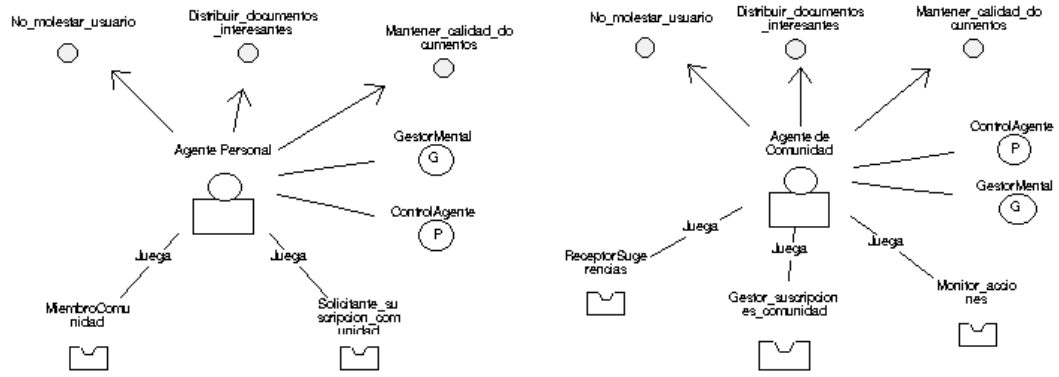


Ilustración 117. Asociaciones entre objetivos y roles (II)

La Ilustración 117 muestra los objetivos perseguidos por un *solicitante de suscripción a la comunidad* y un *gestor de suscripciones a la comunidad*. El primero solicita su entrada a la comunidad, satisfaciendo el objetivo *suscribirse fuentes de información*. El gestor de suscripciones colabora en el proceso porque desea *incorporar usuarios*. El resto de objetivos describen bajo que condiciones tendrá lugar el proceso de suscripción y la permanencia del

solicitante en la comunidad. Los procesos de evaluación de la petición del solicitante aseguran que *no se molesta al usuario* y que, además, el usuario no *enviará información no deseada*. Además, el gestor se encargará de vigilar al nuevo miembro de la comunidad porque desea *detectar usuarios molestos*.

Los roles identificados en las interacciones se asocian con los agentes que los desempeñarán. Inicialmente sólo hay dos clases de agentes: *Agentes Personales* y *Agentes de Comunidad*. El número reducido de roles que desempeñan los agentes contrasta con el número de roles identificados en las interacciones. Sin embargo, hay que comentar que alguno de estos roles, como el rol *miembro de la comunidad* agrupan otros roles, por lo que el reducido número no implica el que no se tengan en cuenta los ya identificados.



(A) (B)

Ilustración 118. Modelos de agente para el Agente Personal (B) y Agente de Comunidad (A).

Los Agentes Personales (Ilustración 118 (A)) y Agentes de Comunidad (Ilustración 118 (B)) asumen los objetivos de distribuir documentos interesantes y el de mantener calidad de los documentos. Adicionalmente, cada agente se responsabiliza de no molestar a su usuario (Agente Personal) o a los miembros de la comunidad (Agente de Comunidad). Estos objetivos se pueden completar gracias a que los agente despempeñan roles de las ilustraciones Ilustración 117 y Ilustración 116. Los procesadores y gestores de estado mental son del mismo tipo para ambos agentes.

El procesador de estado mental en los dos casos se basa en el algoritmo RETE. El mecanismo deliberativo utiliza reglas de producción. El gestor de estado mental tiene como restricción el que la introducción de nueva información se debe hacer escalonadamente. Sólo se mete una nueva entidad de información cuando el agente ha terminado de tomar decisiones.

La inteligencia del agente se concreta en la forma en que ambos se adaptan a los gustos de sus usuarios utilizando como alimentación el resultado de las evaluaciones ejecutadas. La comunidad obtiene información de sus miembros mediante los procesos de monitorización (ver Ilustración 109). Los agentes personales, en los procesos de propagación de sugerencias, modifican el perfil del usuario para constatar que determinado documento le gusta a su usuario.

Del diseño de los diagramas de colaboración ya se ha deducido la existencia de una entidad *Clasificador*. En este punto se puede extrapolar la existencia de otras tres

entidades: un *gestor de estadísticas*, un *gestor de miembros de la comunidad* y un *gestor de agentes*. La necesidad del primero se deduce de la monitorización de usuarios, donde se deben almacenar las veces que se ejecuta cada acción. La necesidad del segundo viene implícita del proceso de alta y baja en la comunidad, lo cual implica llevar cuenta de quién pertenece a la comunidad en un momento dado. El tercero sirve para localizar agentes en el sistema. Es útil cuando el agente personal no pertenece a ninguna comunidad, ya que permite localizar las comunidades existentes.

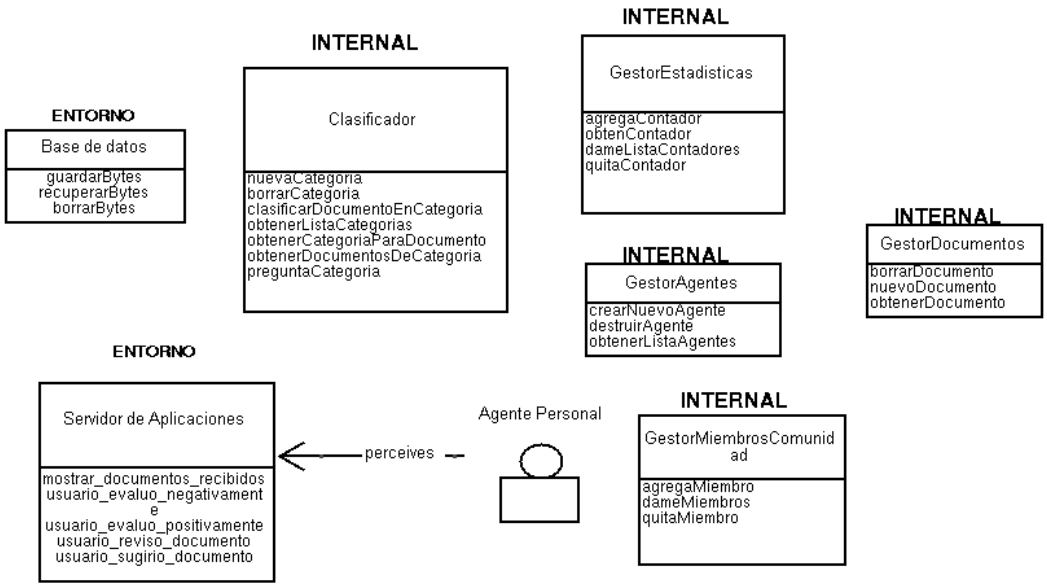


Ilustración 119. Nuevos elementos identificados en el entorno

Estos elementos se asocian a grupos de la organización siguiendo un criterio de utilización: aquellas aplicaciones que cualquier miembro de la organización pueda utilizarla, pertenecerá al grupo *administración*, mientras que las aplicaciones que sólo puedan ser utilizadas por los miembros de una comunidad, pertenecerán al grupo *comunidad*.

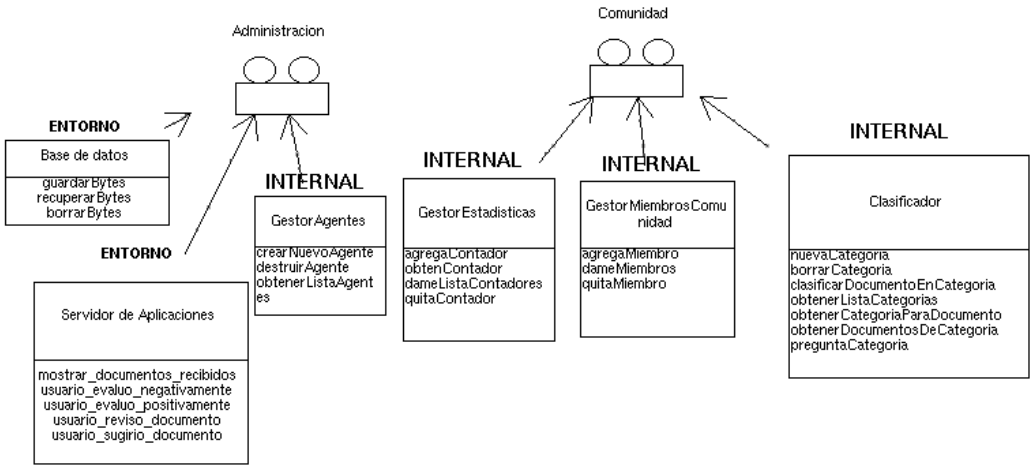


Ilustración 120. Recursos identificados y asociación de aplicaciones a grupos

La descripción de estos elementos (Ilustración 119) se muestra a continuación:

- **Clasificador.** Es una herramienta que permite en base a una colección de documentos, crear una categoría y decidir con posterioridad si un documento encaja en alguna de las categorías creadas.
- **Gestor de Estadísticas.** Para registrar las actividades del usuario se definen contadores asociados con claves. El desarrollador definirá claves que se identifiquen con las distintas acciones y para cada una se crearán contadores.
- **Gestor de Miembros de la comunidad.** Esta aplicación gestiona los miembros de una comunidad, permitiendo al agente llevar cuenta de quiénes están asociados.
- **Gestor de agentes.** Sirve para localizar, crear y destruir agentes en el sistema. Mediante esta aplicación se puede ordenar la creación de agentes específicos así como obtener información de los existentes.

Con esta información, se revisa la arquitectura del sistema propuesta anteriormente:

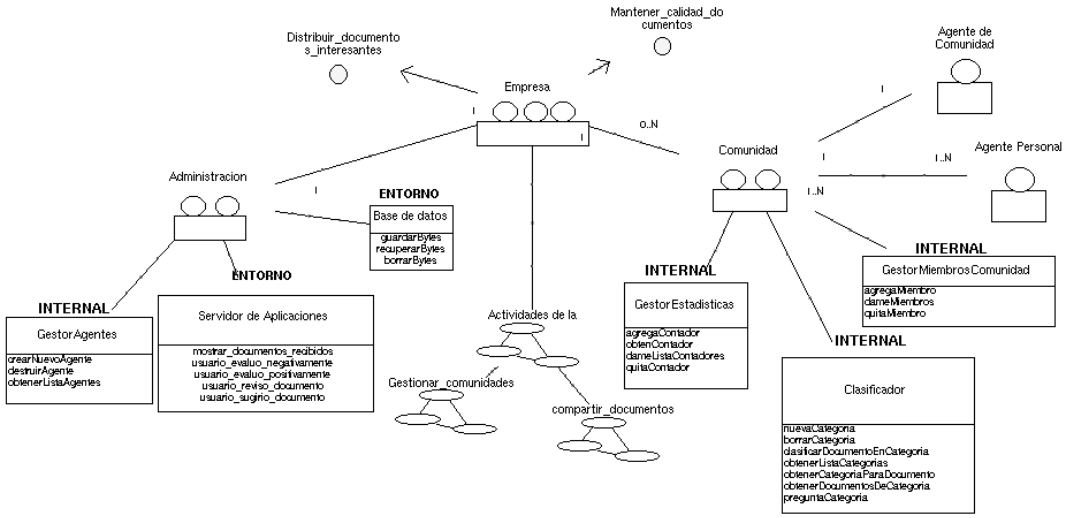


Ilustración 121. Representación del sistema Multi-Agente para el sistema personalizado de distribución de documentos

Los agentes en la organización se disponen por grupos. El más relevante es *Comunidad* que agrupa un conjunto de agentes personales, un único agente de comunidad y los recursos propiedad de la comunidad. Estos recursos son utilizados para gestionar los miembros de la comunidad, las estadísticas de funcionamiento y la categorización estadística de documentos.

El grupo *Administración* reúne recursos comunes a todos los agentes. En este caso se trata de un *gestor de agentes* y un *gestor de documentos*. El *gestor de agentes* actúa como factoría de agentes, permitiendo la creación o destrucción de *Agentes Personales* o *Agentes de Comunidad*.

Dentro de la organización se distinguen dos flujos de trabajo: los destinados a la gestión de la comunidad (*Gestionar Comunidades)* y aquellos que soportan la distribución de documentos (*Compartir documentos*). La existencia de estos flujos se debe a la necesidad de organizar las tareas que se han identificado.

Los flujos de trabajo *Gestionar comunidades* y *Compartir documentos* se refinan en más tareas y flujos de trabajo. Este refinamiento se avanzarán en el diseño con la interconexión de tareas y la definición de su ejecución dentro de interacciones (cuando la ejecución de las tareas involucra a otros agentes).

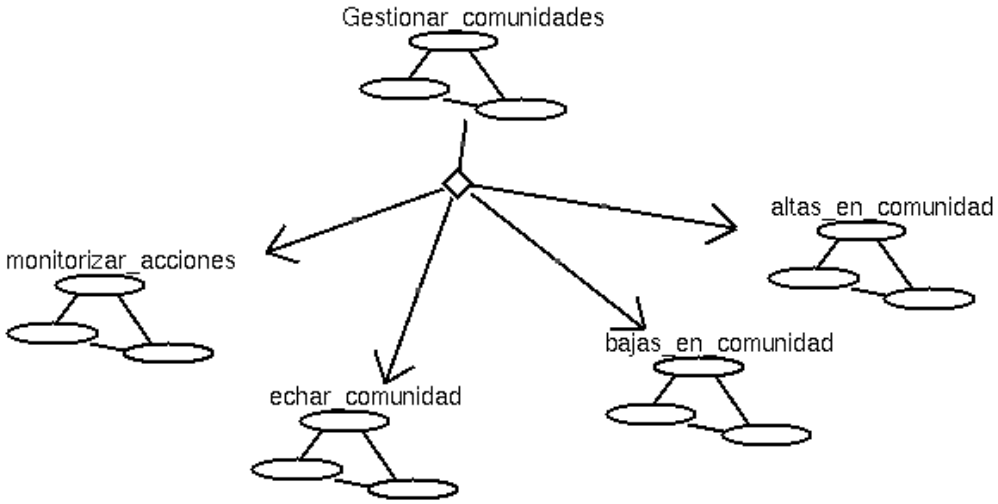


Ilustración 122. Refinamiento del flujo de trabajo para la gestión de comunidades

El flujo de trabajo *Compartir documentos* se refina distinguiendo entre *evaluación de documentos, monitorizar acciones, echar comunidad* y *propagación de las sugerencias*. En la siguiente sección se ofrecerán más detalles respecto a cómo tiene lugar esta descomposición.

4.5 Diseño-Elaboración

En esta etapa se aumenta el nivel de detalle en la especificación determinando cómo se llevan a cabo los casos de uso identificados. Los resultados se centran en refinar los flujos de trabajo y las tareas asociadas, las interacciones, cómo es el control del agente y cómo se satisfacen los objetivos del sistema.

4.5.1 Estructuración de la etapa

Al detallar las interacciones y los flujos de trabajo, es normal que aparezcan nuevas tareas. Idealmente, con estas tareas se deberían estudiar en otra iteración dentro de la etapa de análisis-elaboración. En esta ocasión, este estudio se omite.

En esta etapa, la documentación es extensa, por ello se ha preferido mostrar sólo aquellos aspectos más relevantes y dejar el resto para la consulta de los lectores en una página web. Así, la documentación completa se puede consultar en http://grasia.fdi.ucm.es/metodologia.

Los modelos de organizaciones se refinan asociando tareas a los flujos de trabajo, actividad *identificar tareas* (5.a). Estas tareas se obtienen de modelos de objetivos y tareas y de las interacciones esquematizadas en el análisis. Las tareas se conectan entre sí uniendo aquellas que produzcan entidades con aquellas que las consuman, actividad *conectar tareas*

(5.b). Se describen también qué recursos se consumen, actividad *recursos en las tareas* (5.e), detallando qué tareas implican la ejecución de otras tareas en otros agentes, *identificar tareas no locales* (5.c), y quién se encarga de ejecutarlas, *identificar responsables* (5.d). Las instancias *WFConecta* se complementan con la información elaborada por la actividad *identificar entidades mentales* (5.f), que produce los elementos intercambiados dentro de los flujos de trabajo.

Actividad	**Tipo de resultado**	**Referencia**
identificar tareas (5.a).	Un conjunto de tareas asociadas a un flujo de trabajo	Ilustración 123 Ilustración 131 Ilustración 137 Ilustración 144 Ilustración 150 Ilustración 154
conectar tareas (5.b).	Instancias de *WFConecta*	Ilustración 124 Ilustración 132 Ilustración 138 Ilustración 145 Ilustración 151 Ilustración 154
identificar tareas no locales (5.c),	Tareas que producen interacciones	Hecho en análisis-elaboración
identificar responsables (5.d)	Instancias de *WFResponsable* asociando roles o agentes y tareas	Ilustración 149 Ilustración 143 Ilustración 136 Ilustración 130
identificar entidades mentales (5.f),	Asociaciones de las tareas con entidades mentales mediante instancias de *WFProduce* y *WFConsume*.	Ilustración 125 Ilustración 133 Ilustración 139 Ilustración 146 Ilustración 152 Ilustración 155

Los modelos de interacción se relacionan con los flujos de trabajo a través de las entidades *interacción*. Estas entidades especifican la ejecución de tareas no locales tomando como guía los diagramas de colaboración existentes. La especificación se presenta con un conjunto de diagramas GRASIA. Estos diagramas parten de los diagramas de colaboración de la etapa anterior. La conversión se inicia traduciendo pasos de mensajes a unidades de interacción, actividad *traducir mensajes* (*5.a*). En las unidades de interacción se explica por

qué un agente decide mandar el mensaje,actividad *establecer condiciones mentales* 5.c, y qué tareas se ejecutan al enviar o recibire el mensaje, actividad *asociar tareas* (5.b). Por último, se organizan las unidades de interacción en un diagrama de flujo para explicar en qué orden se ejecutan, actividad *establecer orden de ejecución* (5.d).

Actividad	**Tipo de resultado**	**Referencia**
traducir mensajes (*5.a*)	Diagramas GRASIA donde se asocian roles con *unidades de interacción*	Ilustración 128 Ilustración 135 Ilustración 141 Ilustración 148 Ilustración 153 Ilustración 156
establecer orden de ejecución (5.d).	Relacionar las unidades de interacción mediante primitivas *UIPrecede, UIBifurca, UIItera, UIConcurren*	
establecer condiciones mentales 5.c	Instancias de *patrón de estado mental* asociado a *unidades de interacción*	Ver especificación completa en http://grasia.fdi.ucm.es/metodologia
asociar tareas (5.b)	Asociar tareas a instancias de *unidad de interacción*	Hecho en análisis-elaboración

Los modelos de agente no se modifican en este caso, aunque sí que se detalla cómo se espera que sea su control. Se estudian los estados mentales extraídos de las interacciones y los requeridos por los flujos de trabajo, actividad d*etallar los estados intermedios* (6). Obtenidos estos estados mentales, hay que estudiar cómo se pasa de uno a otro, actividad d*eterminar cómo se pasa de un estado mental a otro* (7.a), y detallar cómo decide el agente qué tarea ejecutar a continuación (7.c).

Actividad	**Tipo de resultado**	**Referencia**
Determinar cómo se pasa de un estado mental a otro (7.a).	Referencias a diagramas GRASIA y a modelos de tareas y objetivos	Ver especificación completa en http://grasia.fdi.ucm.es/metodologia
Detallar los estados intermedios (6)	Instancias de modelos de agente donde se asocia una instancia de *agente concreto* a instancias de *estado mental*	Ver especificación completa en http://grasia.fdi.ucm.es/metodologia
Detallar cómo se gestiona el estado mental (7.b)	Instancias de modelos de tareas y objetivos donden las tareas producen, destruyen o modifican objetivos	Ilustración 162 Ver especificación completa en http://grasia.fdi.ucm.es/metodologia

Asociar los estados mentales a la ejecución de acciones (7.c).	Instancias de modelos de tareas y objetivos donden las tareas producen, destruyen o modifican objetivos	Ver especificación completa en http://grasia.fdi.ucm.es/metodologia

Los modelos de entorno se refinan atendiendo al tipo de percepción de los agentes y detallando cuales son los parámetros de los recursos. Los recursos se asocian con tareas cuando sea necesario indicar necesidades relevantes, como tareas que consumen mucho tiempo de procesador. Aquí se utilizan principalmente para refinar la percepción de los agentes, actividad *refinar la percepción de los agentes* (9). Para determinar los parámetros adecuados de los recursos, actividad *definir los atributos propios de cada recurso* (11), conviene conocer antes qué necesidades se tienen. Para clarificar esta visión es apropiado aplicar la actividad *asociar recursos, aplicaciones y tareas* (12), a partir de lo cual será más sencillo determinar exactamente qué se necesita.

Actividad	**Tipo de resultado**	**Referencia**
refinar la percepción de los agentes (9).	Instancias de EPercibeNotifica y EPercibeMuestreo asociando agentes y aplicaciones	Ilustración 163 Ver especificación completa en http://grasia.fdi.ucm.es/metodologia
definir los atributos propios de cada recurso (11)	Instancias de modelos de agente donde se asocia una instancia de *agente concreto* a instancias de *estado mental*	Ver especificación completa en http://grasia.fdi.ucm.es/metodologia
asociar recursos, aplicaciones y tareas (12)	Instancias de modelos de tareas y objetivos donden las tareas producen, destruyen o modifican objetivos	Ver especificación completa en http://grasia.fdi.ucm.es/metodologia

Los objetivos y tareas se estudian para indicar exactamente cómo se resuelven los objetivos y cómo se ejecutan las tareas. Para las tareas se describen las postcondiciones y precondiciones del análisis con detalle. Así, se indican qué recursos se consumen, actividad *determinar recursos a consumir* (8.b), cuáles se restablecen, actividad i*ndicar qué recursos se restablecen* (7.a), qué ocurre con las entidades mentales requeridas, actividad *asociar tareas con entidades mentales* (7.b), si se necesitan nuevas entidades mentales, actividad *refinar entidades mentales consumidas* (8.a), qué operaciones se están invocando sobre las aplicaciones, actividad *asociar tareas con operaciones de las aplicaciones* (7.c). Los objetivos, por otro lado, se refinan indicando qué dependencias existen, actividad *refinar dependencias de objetivos* (10.a), y cómo se satisfacen o fracasan, *refinar satisfacción/fracaso de objetivos* (10.b).

Actividad	**Tipo de resultado**	**Referencia**
determinar recursos a consumir (8.b)	Instancias de *WFUsa* asociando tareas y recursos	Ilustración 157 Ver especificación completa en http://grasia.fdi.ucm.es/metodologia
i*ndicar qué recursos se restablecen* (7.a)	Instancias de *WFProduce* asociando tareas y recursos	
asociar tareas con entidades mentales (7.b)	Instancias de *GTSatisface y GTFalla.*	
refinar entidades mentales consumidas (8.a)	Instancias de *WFConsume*	
asociar tareas con operaciones de las aplicaciones (7.c)	Instancias de *WFUsaLlamada*	
refinar dependencias de objetivos (10.a)	Determinar dependencias Y/O ente objetivos	Ilustración 160 Ilustración 161
refinar satisfacción/fracaso de objetivos (10.b)	Instancias de *GTFallaObjetivo* o *GTSatisfaceObjetivo* asociando objetivos y tareas.	Ilustración 157 Ilustración 159 Ilustración 158

4.5.2 Resultados obtenidos

La *gestión de comunidades* se ha descompuesto en una serie de tareas atómicas que implementan primitivas de gestión (*tareas basicas de gestion*), y en dos flujos de trabajo destinados a la tramitación de altas (*altas en comunidad*) y bajas en la comunidad (*bajas en comunidad*).

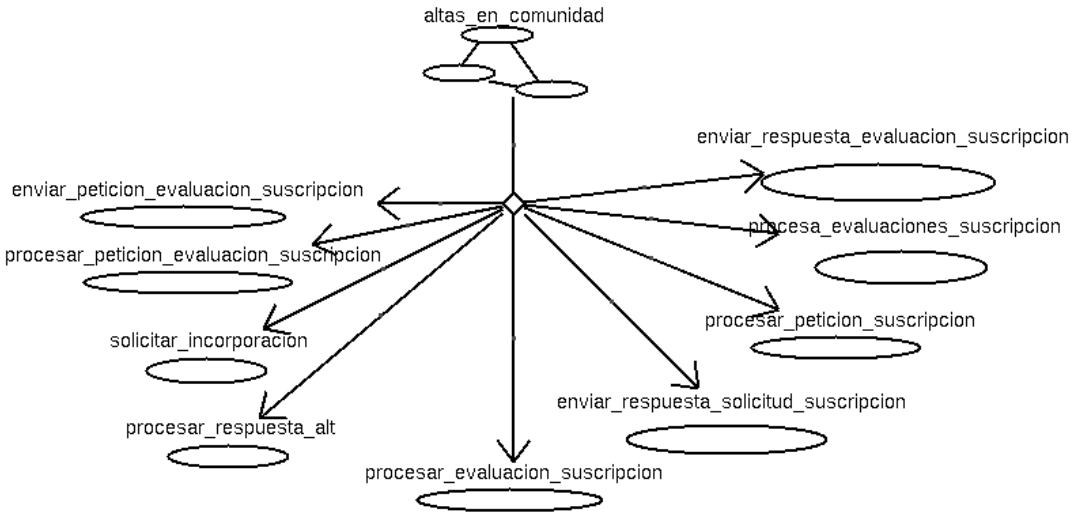


Ilustración 123. Tareas que componen el flujo de trabajo *altas en la comunidad*

La identificación de tareas para el flujo *altas en la comunidad* (Ilustración 123) se basa en el diagrama de colaboración de la Ilustración 104. El número final se debe a refinamientos con motivo de la definición de diagramas GRASIA que se mostrarán más tarde. La ordenación de estas tareas se muestra en la Ilustración 124.

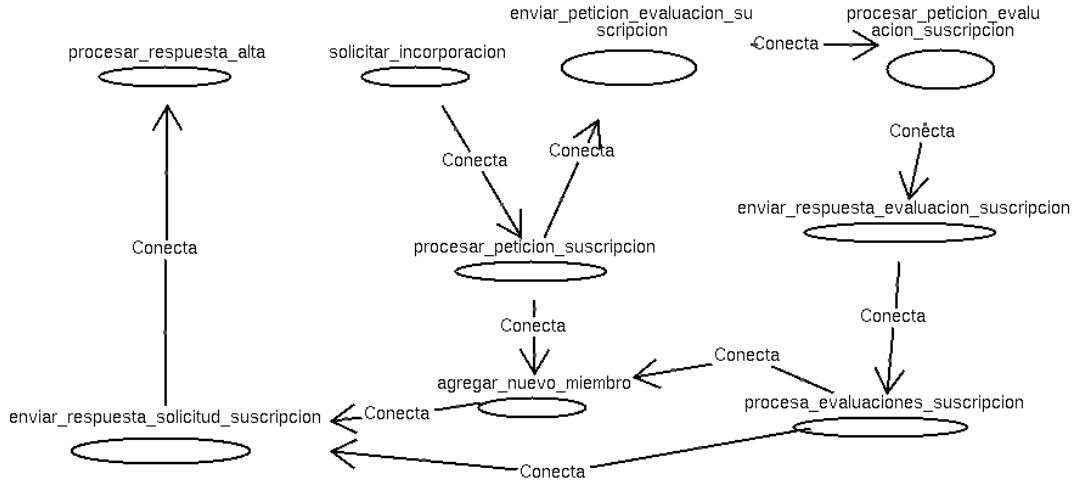


Ilustración 124. Dependencias entre las tareas del flujo *altas en comunidad*.

La tarea inicial (Ilustración 124) es la de solicitar la incorporación a la comunidad (*solicitar incorporación*). Esta petición es procesada por *procesar petición suscripción*, que puede continuarse con una evaluación de la idoneidad del nuevo miembro (*enviar petición envaluacion suscripción*) o con la agregación directa (*agregar nuevo miembro*), en el caso en que no haya otros usuarios registrados. Los evaluadores de la suscripción estudian la petición ejecutando *procesar evaluación suscripción* y devuelven la respuesta con *enviar respuesta evaluación*. Esta respuesta es procesada con *procesa evaluaciones suscripción* que

terminará con la agregación de un nuevo miembro o no. De cualquier forma, la respuesta se envía con *enviar respuesta solicitud suscripción* que será procesada por el solicitante con *procesar respuesta alta*.

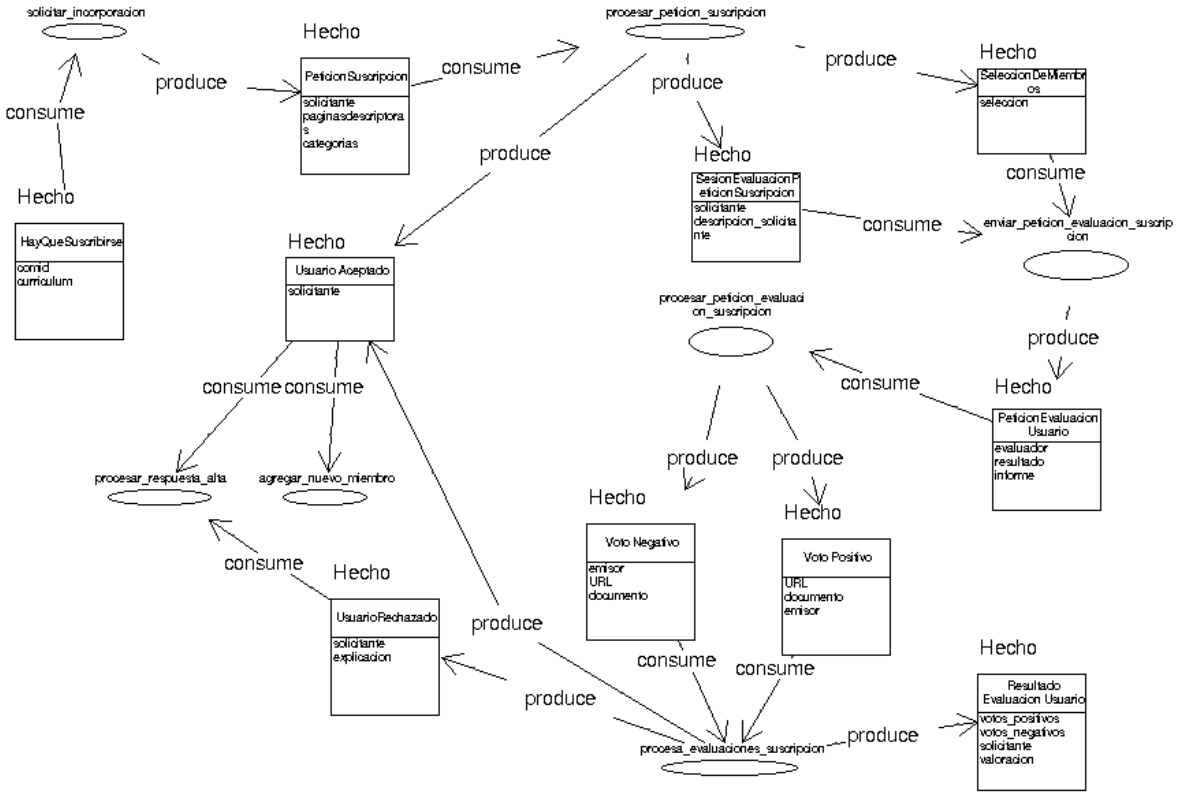


Ilustración 125. Descripción detallada del flujo *solicitar alta en comunidad*

La Ilustración 125 describe las entidades mentales intercambiadas en el flujo de trabajo *solicitar alta en comunidad*. Como en el flujo de trabajo *evaluar documento*, existen tareas cuya ejecución no dispara la ejecución de otras tareas. En este caso se trata de la tarea *agregar nuevo miembro*.

Las ejecución de las tareas del flujo *dar alta en comunidad* se definen dentro de un diagrama GRASIA asociado a la interacción *solicitar alta en comunidad*.

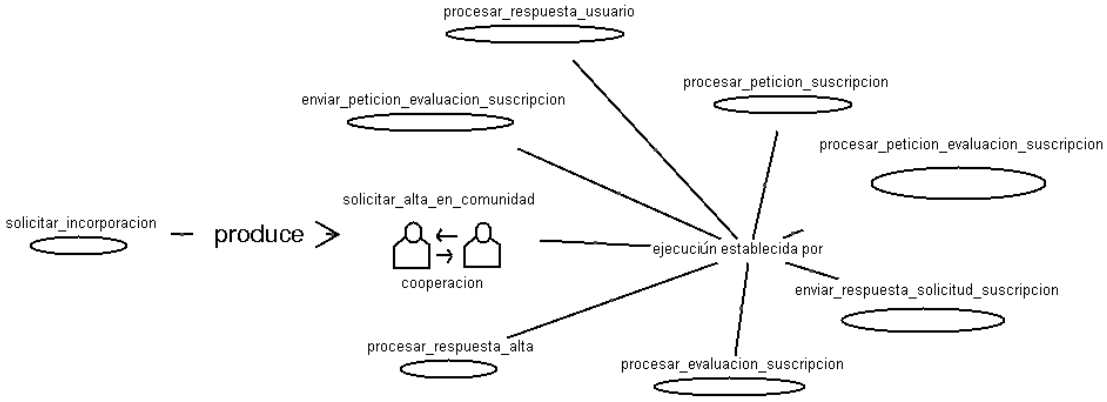


Ilustración 126. Relación de las tareas del flujo *dar alta en comunidad* con la interacción *solicitar alta en comunidad*

Para especificar esta ejecución se añade una nueva especificación de interacción a *solicitar alta en comunidad*. El tipo de la nueva especificación es un diagrama GRASIA (Ilustración 127)

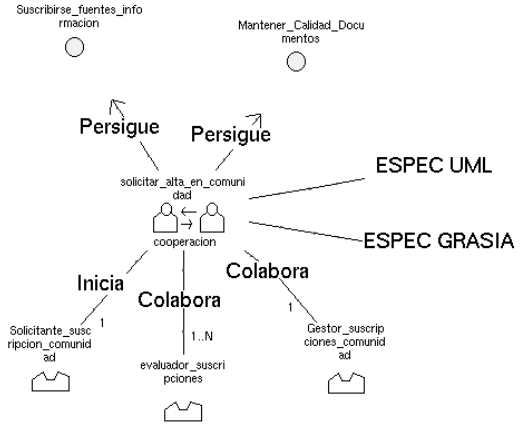


Ilustración 127. Refinamiento del modelo de interacción de *solicitar alta en comunidad*.

La Ilustración 127 muestra el modelo de interacción para *solicitar alta en comunidad* en el que se ha añadido una nueva especificación de tipo GRASIA.

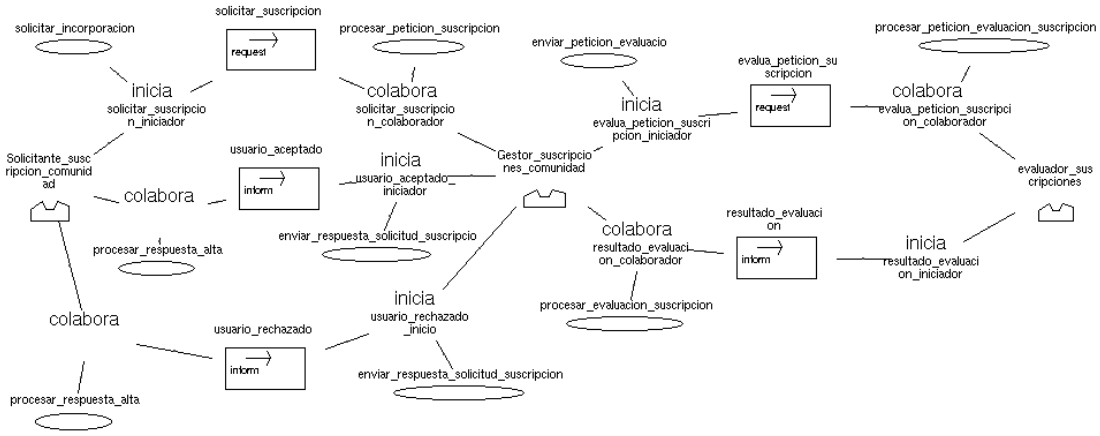


Ilustración 128. Unidades de interacción identificadas para la interacción *solicitar alta en comunidad.*

Las unidades de interacción de la Ilustración 128 se obtienen a partir del diagrama de colaboración de la Ilustración 104. Se ha intentado asimilar cada unidad de interacción con un paso de mensaje y cada tarea con las acciones esperadas por el emisor y el receptor.

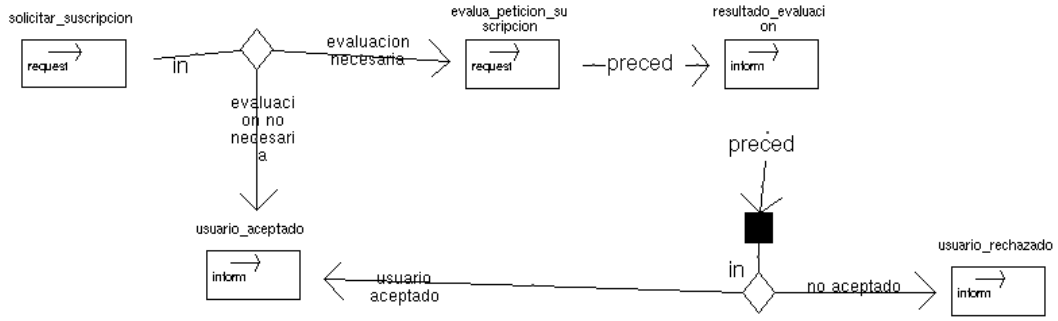


Ilustración 129. Ordenación de las unidades de interacción en *solicitar alta en comunidad*

Estas unidades de interacción se ordenan según indica la Ilustración 129. Respecto del flujo de trabajo inicial, sólo queda detallar quién es responsable de ejecutar la tarea, pero a partir de la ilustración anterior, es sencillo.

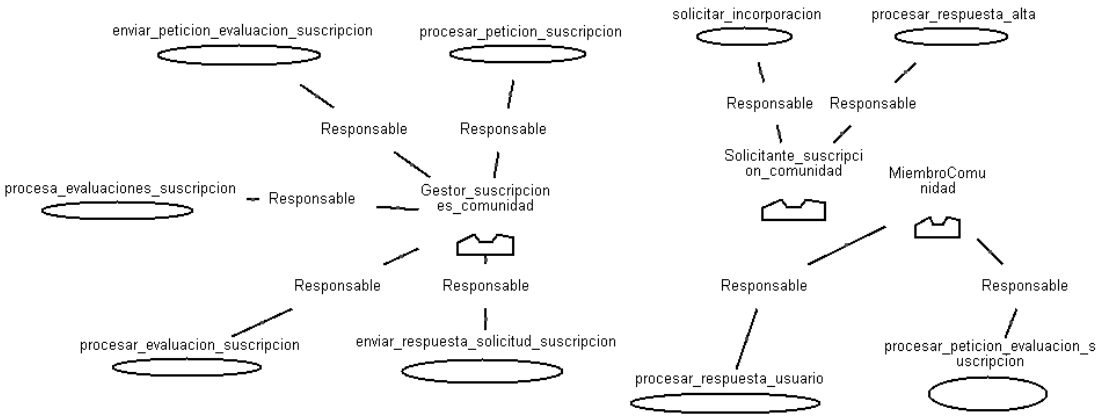


Ilustración 130. Responsables de la ejecución de tareas en el flujo *dar alta en comunidad*

La Ilustración 130 muestra la asociación de roles con tareas según lo indicado en la Ilustración 128.

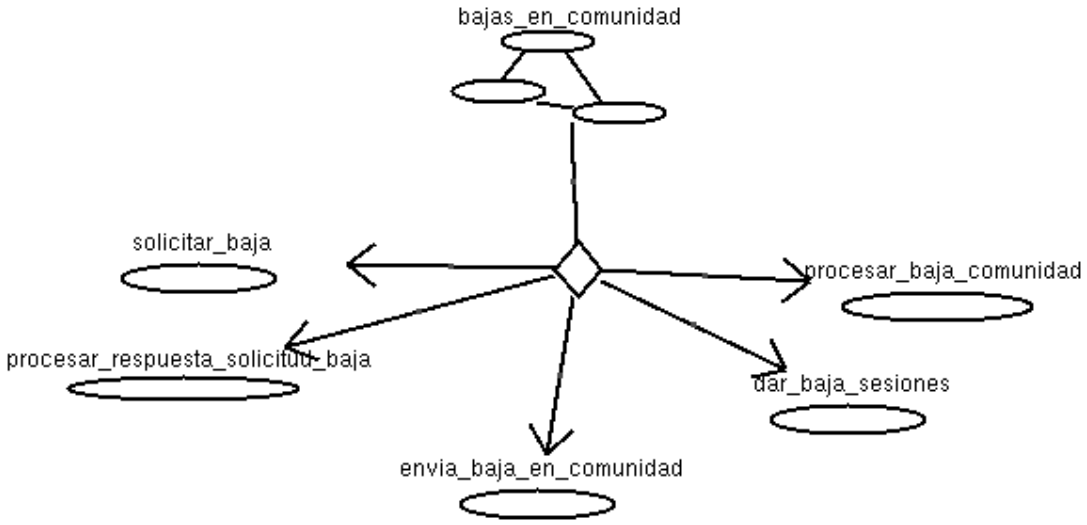


Ilustración 131. Tareas que componen el flujo de trabajo *bajas en comunidad*

Las *bajas en la comunidad* se rigen por el conjunto de tareas de la Ilustración 131. De estas tareas, llama la atención *dar de baja sesiones*. Esta tarea se ha incluido para dar una solución al problema de qué hacer con las evaluaciones en que el usuario está participando cuando éste se da de baja. En principio, esta tarea elimina al usuario de cuanto proceso de evaluación existe y de cualquier difusión de sugerencias planificada.

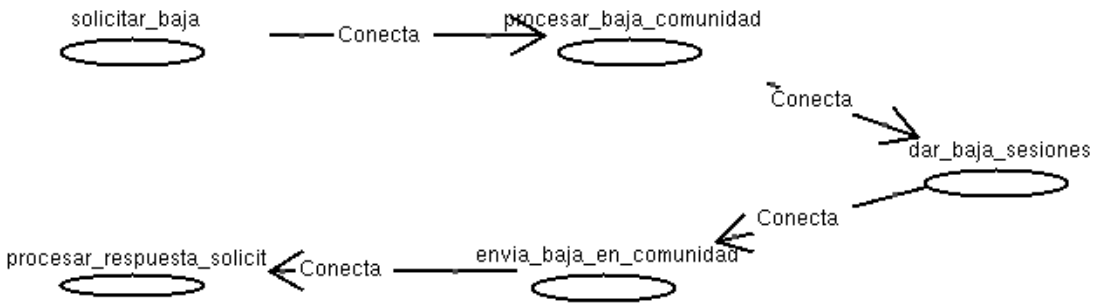


Ilustración 132. Dependencias entre las tareas del flujo *bajas en comunidad*

La tarea inicial de la Ilustración 132 es la ejecutada por el solicitante de baja (*solicitar baja*). Esta tarea es la que origina la petición en el Agente de Comunidad que se procesa con *procesa baja en comunidad*. Estas tareas identifica las sugerencias que el usuario está pendiente de recibir y se las proporciona a *dar baja sesiones*. El paso siguiente es informar al solicitante que ha sido dado de baja (*enviar baja en comunidad* y *procesar respuesta solicitud*).

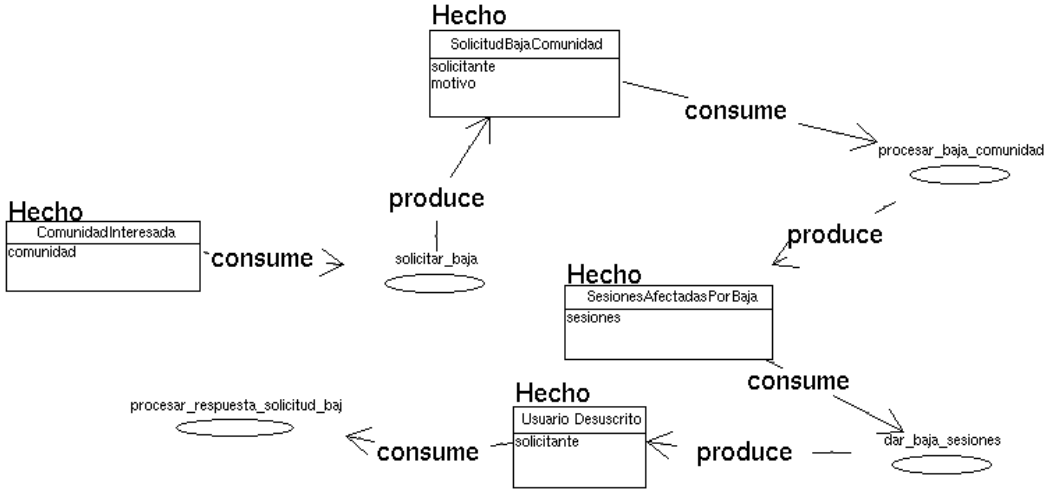


Ilustración 133. Descripción detallada del flujo *dar de baja en comunidad*

Las entidades mentales intercambiadas en el flujo de trabajo *dar de baja en comunidad* se presentan en la Ilustración 133. La tarea *dar baja sesiones* se encargaría de asegurar que el usuario ha sido de baja en las sesiones de evaluación de documentos o usuarios. Una vez conseguido, procedería a dar de baja al usuario en el registro de miembros. Cuando estas operaciones concluyen, produce un hecho *usuario desuscrito* que, aparte de informar al usuario de la ejecución de la baja, sirve para indicar la finalización de otro flujo de trabajo, *monitorizar acciones*.

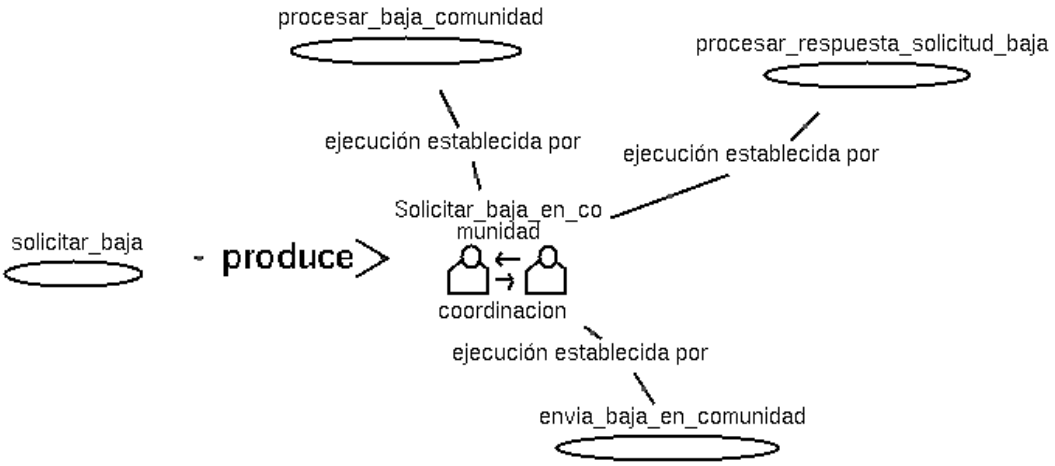


Ilustración 134. Relación de las tareas del flujo *dar baja en comunidad* con la interacción *solicitar baja en comunidad*

Las tareas de la Ilustración 132 se asocian con la interacción *solicitar baja en comunidad* en la Ilustración 134. Asociada a esta interacción se ha incluido un nuevo diagrama GRASIA para especificar la ejecución de tareas.

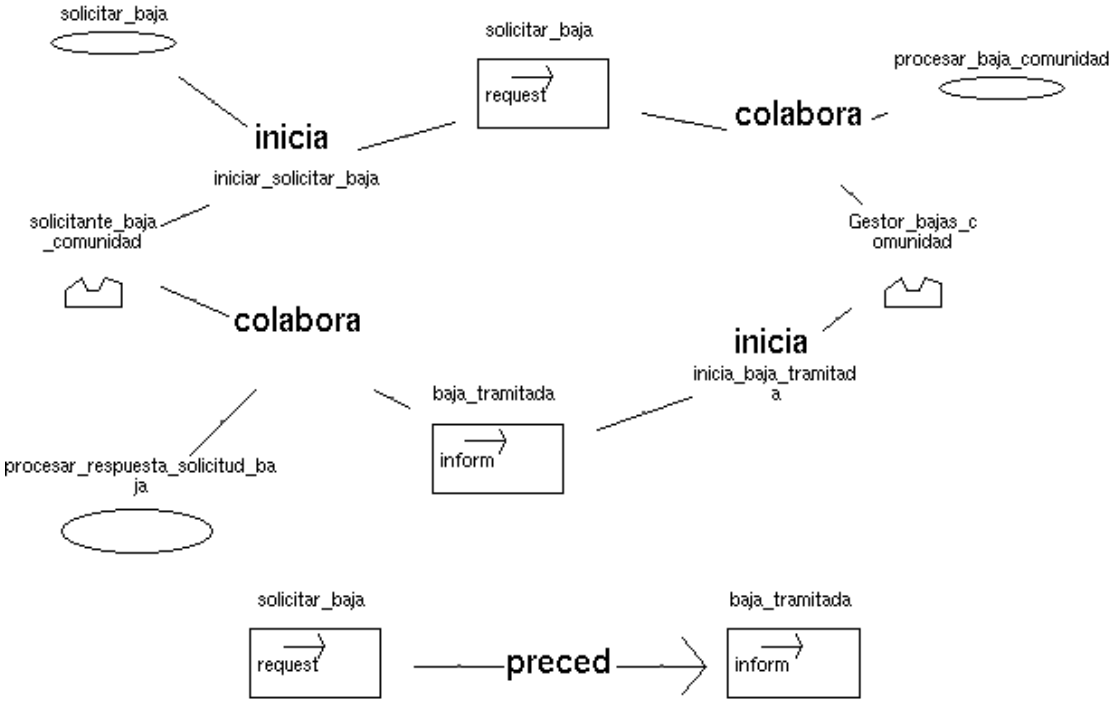


Ilustración 135. Diagrama GRASIA asociado a la interacción *solicitar baja en comunidad*.

Como en el caso anterior, las unidades de interacción provienen de los pasos de mensaje dentro del diagrama de colaboración de Ilustración 107 (B).

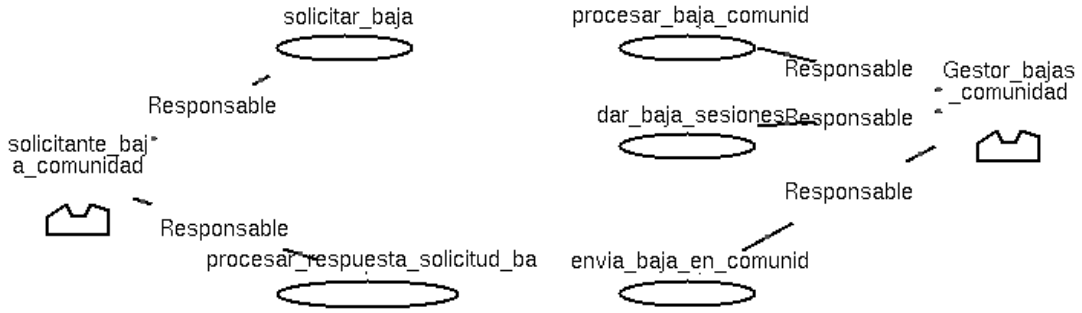


Ilustración 136. Responsables de la ejecución de tareas en el flujo *dar de baja en comunidad*

Como resultado del diagrama de la Ilustración 135 se obtienen las asociaciones de responsabilidad de tareas con respecto a los roles. Estas relaciones se ofrecen en la Ilustración 136.

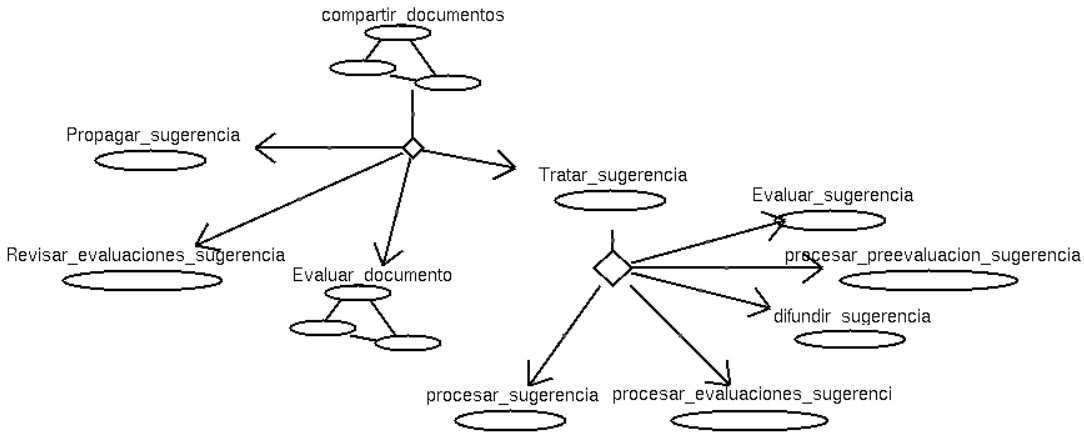


Ilustración 137. Refinamiento del flujo de trabajo para la compartición de documentos

El refinamiento del flujo de trabajo *compartir documentos* se hace de acuerdo con el diagrama de colaboración de la Ilustración 105. El planteamiento de este flujo de trabajo difiere de los anteriores en un detalle: existen flujos de trabajo dentro de un flujo de trabajo. Esta solución que podría haber sido aplicada a los otros flujos, se ha aplicado aquí para evaluar la importancia de la posibilidad de abstraer conjuntos de tareas e interacciones. En este caso, se trata de considerar el flujo *evaluar documento* como una unidad similar a la tarea, pero de orden superior, ya que la tarea no puede contener flujos de trabajo.

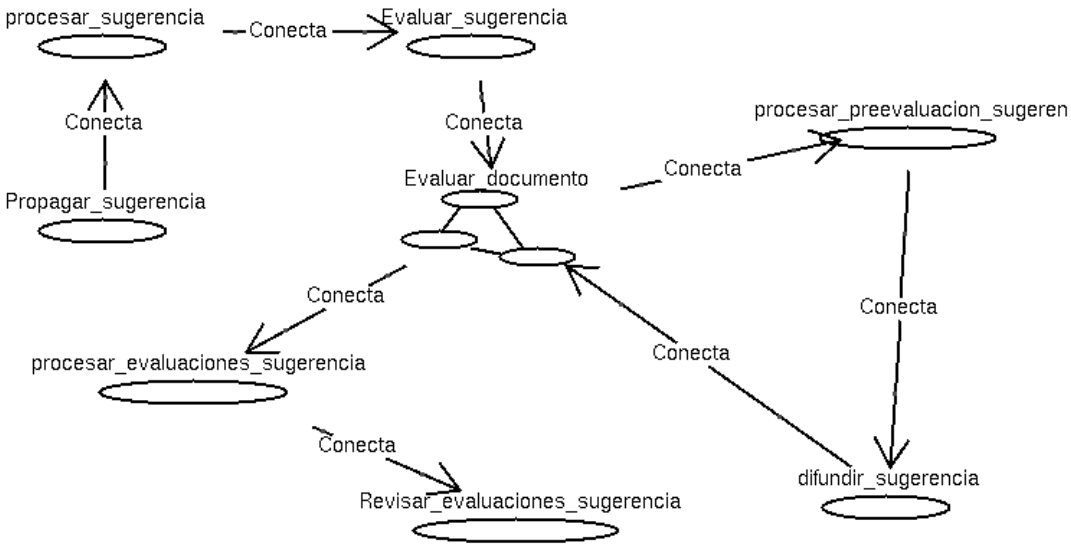


Ilustración 138. Relaciones entre las tareas del flujo de trabajo *compartir documentos*

Siguiendo este diagrama, primero el cliente ejecuta *propagar sugerencia*, en respuesta, la comunidad ejecuta *tratar sugerencia*. Esta tarea se descompone en el lanzamiento de las evaluaciones a realizar por los miembros de la comunidad (*evaluar sugerencia*), la evaluación de las respuestas (*procesar preevaluaciones sugerencias*), la difusión de la sugerencia a los otros miembros de la comunidad (*difundir sugerencia*), la recogida de las opiniones de los usuarios después de revisar el documento (*procesar evaluaciones sugerencias*) y la recepción de la respuesta por parte del usuario que sugirió el documento (*revisar evaluaciones sugerencia*).

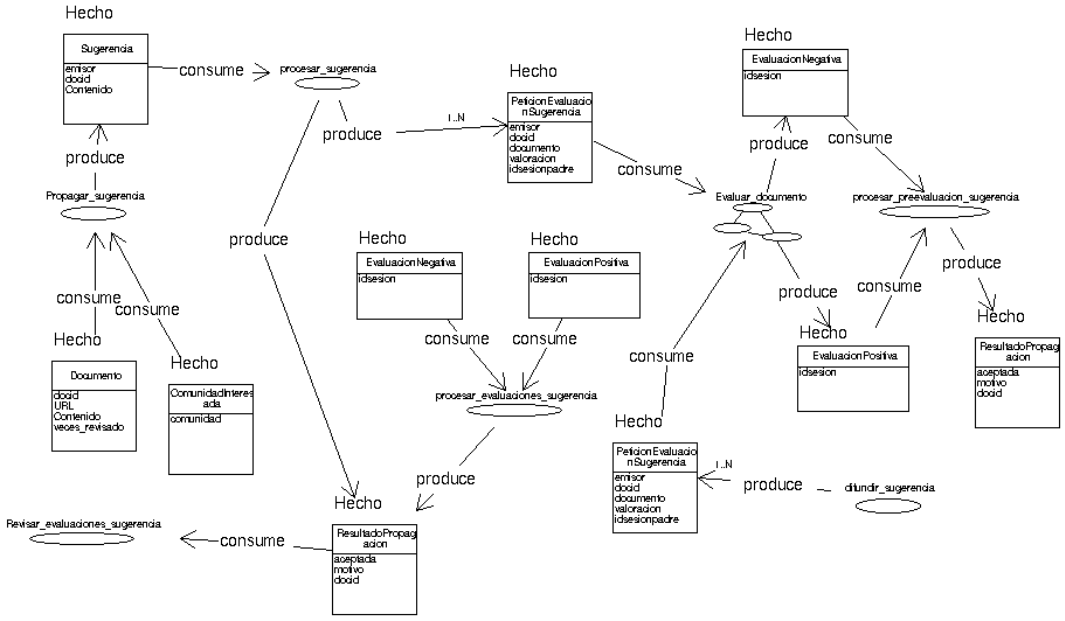


Ilustración 139. Descripción detallada del flujo de trabajo *compartir documentos*

La Ilustración 139 muestra cómo se ejecuta el flujo de trabajo de la Ilustración 138. De esta ilustración se comentará sólo cómo se consigue integrar el flujo de trabajo *evaluar documento* dentro del flujo *compartir documentos*. Las tareas *procesar sugerencia* y *difusión sugerencia en curso* producen un hecho, *sugerencia recibida*, que activa el flujo de trabajo *evaluar documento*. Este flujo produce a su término o bien un hecho *evaluación positiva* o bien *evaluación negativa*. Las distintas evaluaciones se recogen mediante las tareas *procesar preevaluación sugerencia* y *procesar evaluación sugerencia*. Cual de las dos debe ser, se discierne por el hecho *difusión sugerencia en curso*, producido por *difundir sugerencia*.

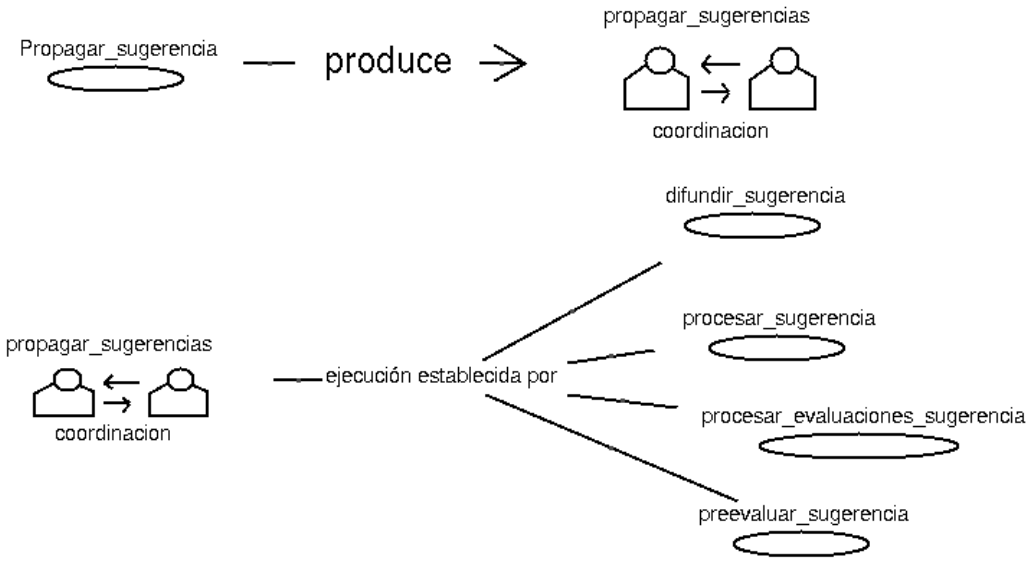


Ilustración 140. Relación de las tareas del flujo *compartir documentos* con la interacción *propagar sugerencias*

Las tareas de la Ilustración 138 se asocian con la interacción *propagar sugerencias* (ver Ilustración 140). A esta interacción se asocia un diagrama GRASIA para detallar la ejecución de tareas (Ilustración 141).

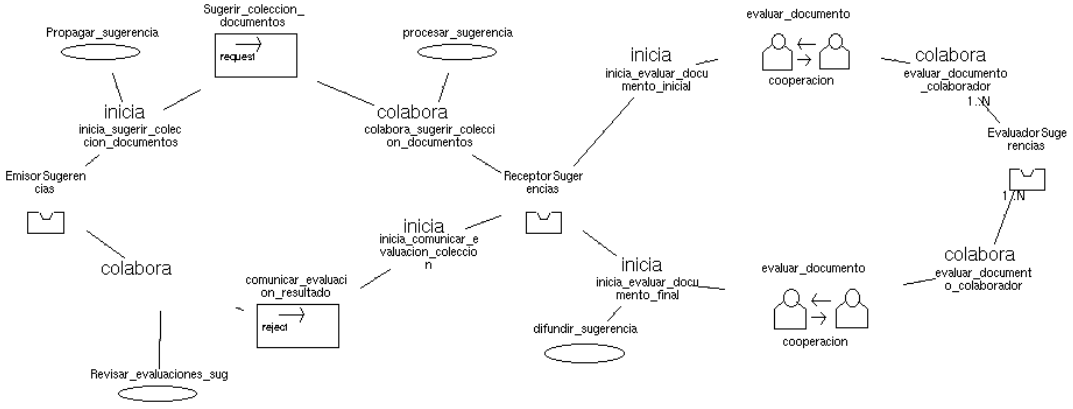


Ilustración 141. Diagrama GRASIA asociado a la interacción *compartir documentos.*

Dentro de este diagrama llama la atención que entre el *Receptor de Sugerencias* y el *Evaluador de sugerencias* no existen unidades de interacción, sino *interacciones.* Esto es consecuencia de haber definido el flujo de trabajo *compartir documentos* en términos del flujo *evaluar documentos.* Para iniciar esta interacción se ejecuta *procesar sugerencia* y para procesar los resultados, se ejecuta *procesar preevaluación sugerencia*, que no aparece en la

ilustración porque su ejecución es posterior al término de la interacción. Otro elemento que destaca es que se implican *N* evaluadores en el proceso. Por cada evaluador habrá una instancia en ejecución de la interacción *evaluar documento*.

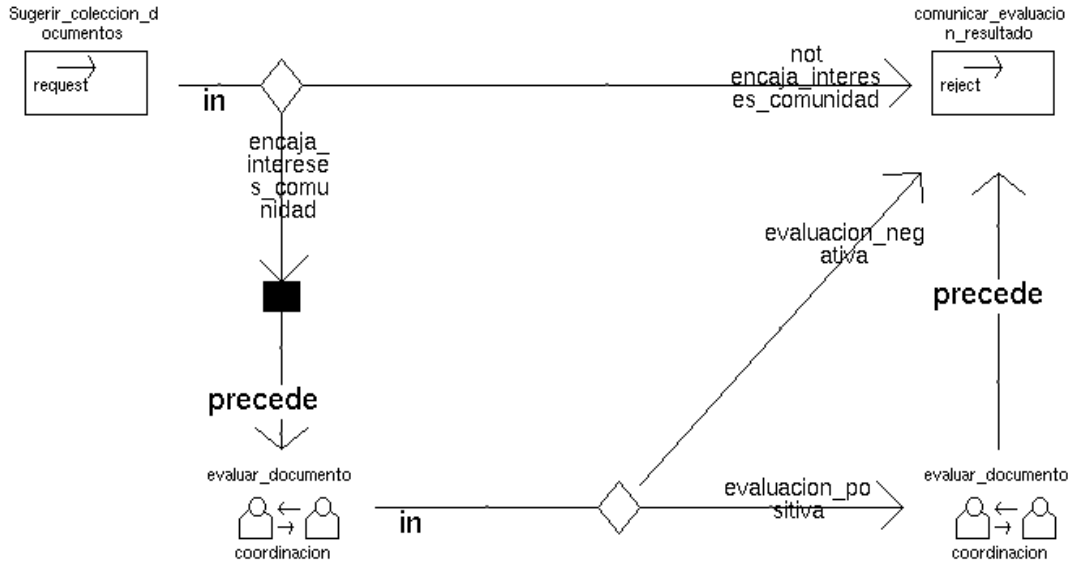


Ilustración 142. Ordenación de las unidades de interacción en la interacción *propagar sugerencias*

La ejecución de las unidades de interacción de la Ilustración 141 está sujeta a una bifurcación dependiendo de si la sugerencia encaja en los intereses de la comunidad o no (Ilustración 142). Si encaja, existe otra posible bifurcación dependiendo de si la evaluación de los usuarios ha sido positiva o negativa. De cualquier forma, la última unidad de interacción ejecutada será la de comunicación de resultados (*comunicar resultados evaluación*).

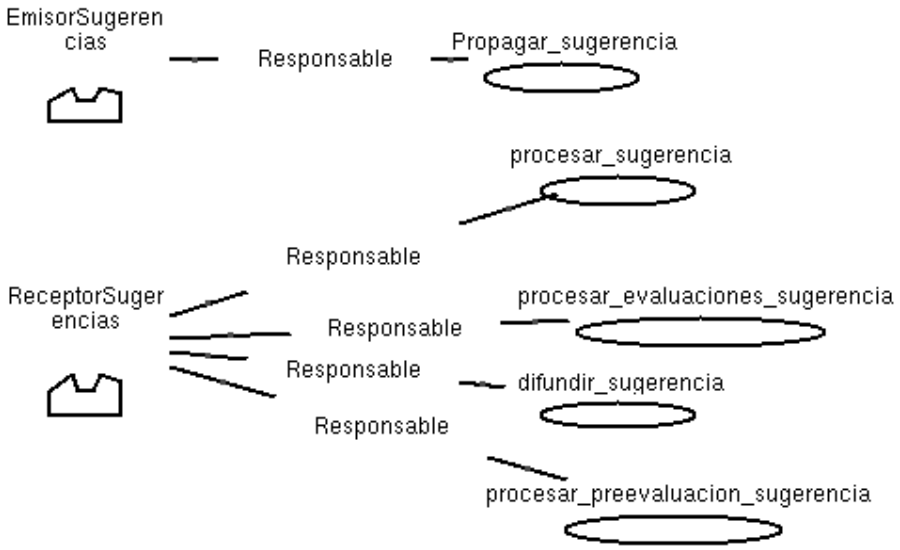


Ilustración 143. Responsables de la ejecución de tareas en el flujo *compartir documentos*

De la Ilustración 141 se obtienen los responsables de la ejecución de las tareas identificadas. Estos responsables se muestran en la Ilustración 143.

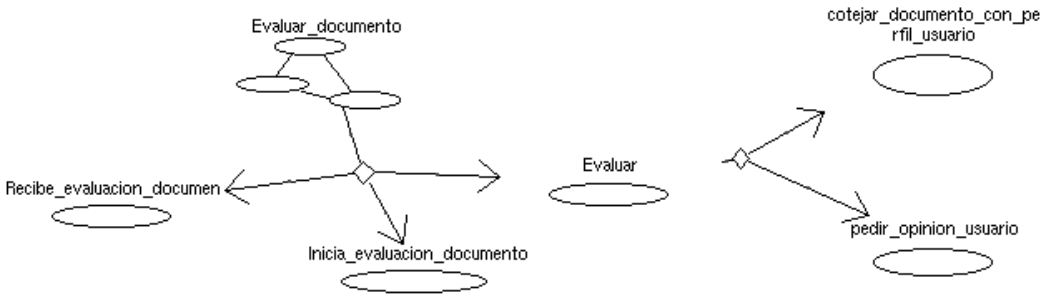


Ilustración 144. Tareas involucradas en el flujo de trabajo *Evaluar Documento*

El flujo de trabajo *evaluar documento* consiste en enviar un documento a un usuario y hacer que éste lo evalúe. Para ahorrar trabajo al usuario, se han dispuesto tareas que permiten desarrollar la tarea de evaluación de forma semiautomática. La tarea *cotejar documento con perfil usuario* se encarga de hacer un análisis estadístico del documento entrante contra la colección de documentos que definen al usuario. Si este análisis no es determinante, se acude al usuario.

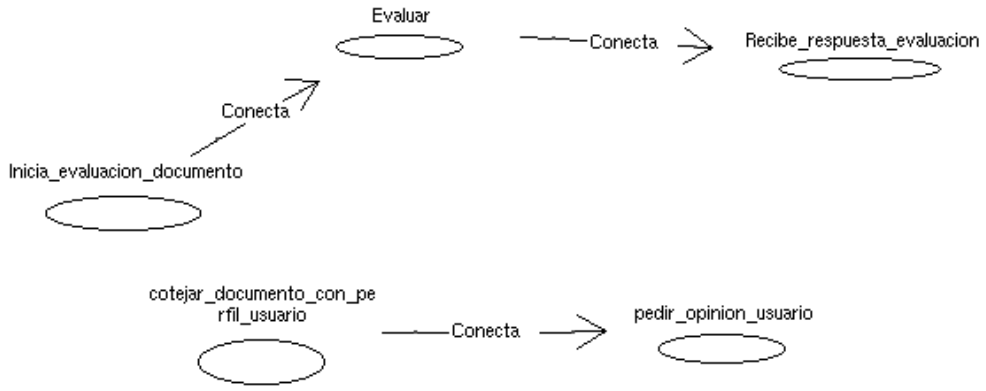


Ilustración 145. Relaciones entre las tareas del flujo *Evaluar Documento*

La primera tarea (Ilustración 145) es la que ejecuta la petición de evaluación (*inicia evaluación documento*). Ésta dispara el proceso de evaluación dentro del cual se ejecuta la evaluación semi-automática (*evaluar*) y después se envía la respuesta para que la reciba el iniciador (*recibe respuesta evaluación*).

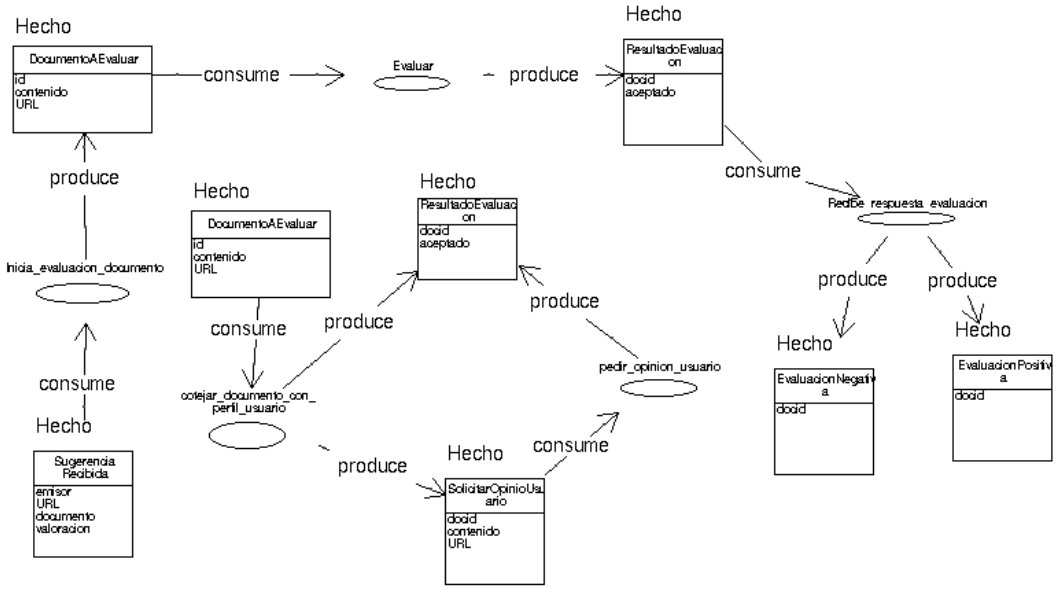


Ilustración 146. Descripción detallada del flujo *evaluar documento*

La Ilustración 147 muestra las entidades mentales intercambiadas según las instancias de *WFConecta* de la Ilustración 145. Se puede apreciar que el flujo de trabajo parece descompuesto en dos, la parte encargada de evaluar (tareas *cotejar documento con perfil usuario* y *solicitar opinion usuario*) y la encargada del proceso de petición y distribución de evaluaciones (el resto de tareas). El motivo es que la tarea *evaluar*, iniciada a petición de otro agente, se descompone en tareas cuya ejecución es local (no implica la ejecución de tareas en otros agentes). De hecho, estas tareas locales no aparecen el diagrama GRASIA de

la Ilustración 148. El resultado devuelto por *evaluar* es un hecho *resultado evaluación* que puede proceder o bien de *cotejar documento con perfil usuario* o bien *solicitar opinión al usuario.*

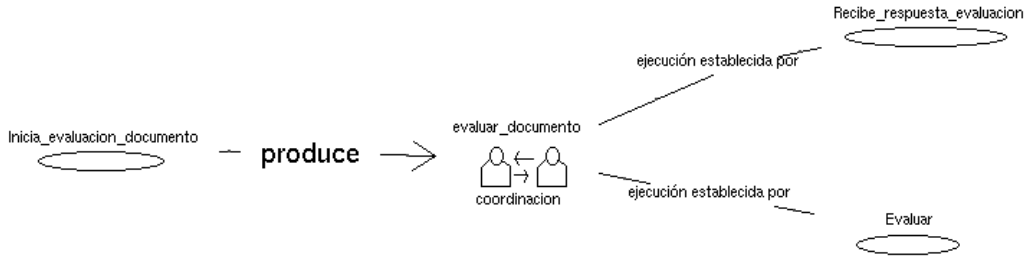


Ilustración 147. Asociación entre las tareas del flujo *Evaluar Documento* y la interacción *evaluar documento*

La interacción que define la ejecución de las tareas del flujo *evaluar documento* recibe el mismo nombre, *evaluar documento.* Nótese que la tarea *evaluar* no se ejecuta a primera vista dentro de la interacción. La ejecución de la tarea de evaluación se dispara no porque haya una interacción, sino porque de repente hay que satisfacer el objetivo *evaluar autónomamente,* como se verá más adelante.

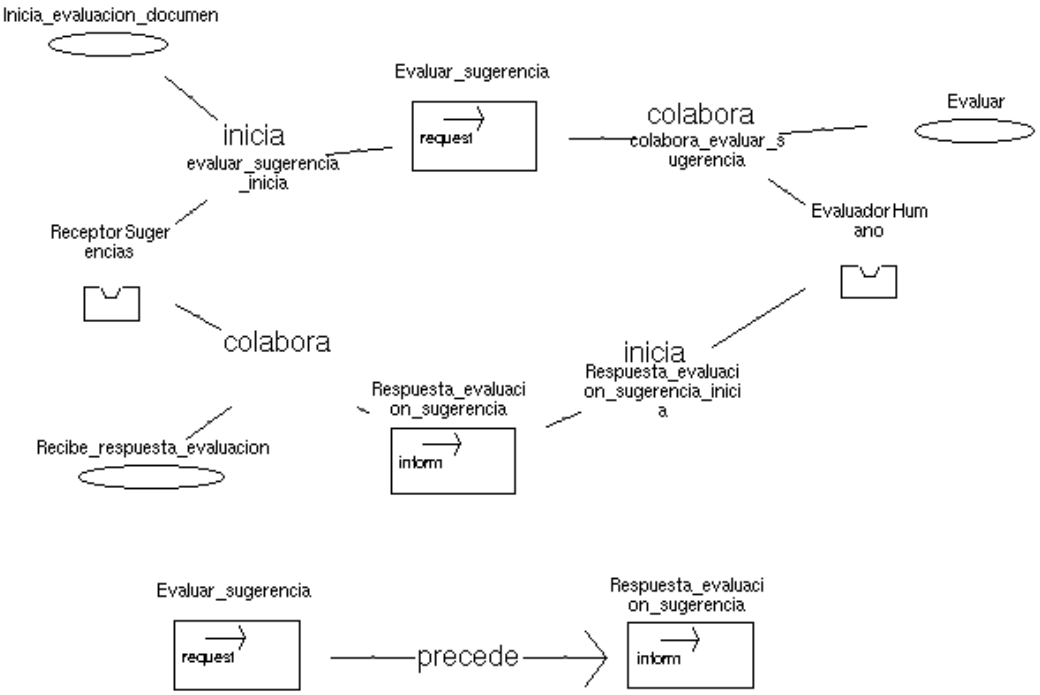


Ilustración 148. Diagrama GRASIA asociado a la interacción *Evaluar Documento.*

El diagrama GRASIA que especifica la interacción *evaluar documento* es uno de los más simples. Se trata de hacer una petición (*evaluar sugerencia*) y de esperar una respuesta (*respuesta evaluación sugerencia*).

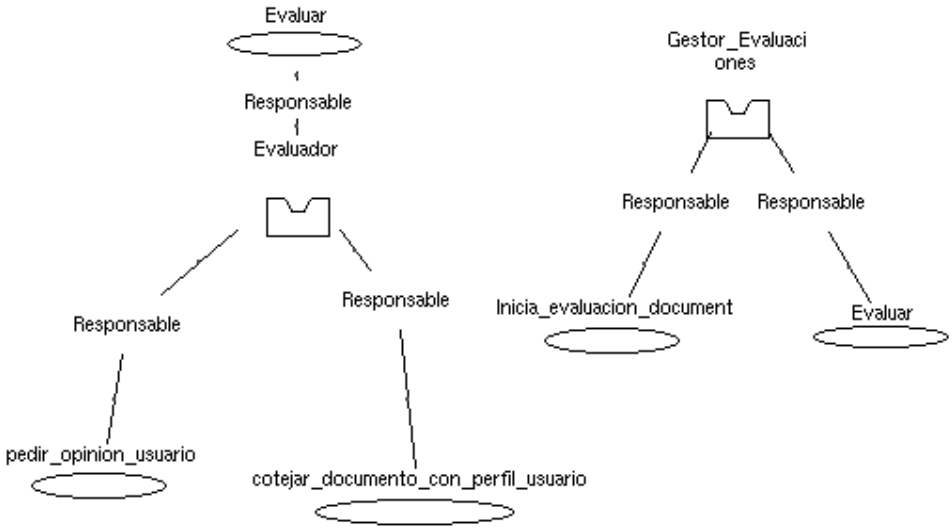


Ilustración 149. Responsables de la ejecución de las tareas del flujo *Evaluar Documento*

Los responsables de la ejecución de tareas, como en los casos anteriores, se obtienen directamente de la especificación de diagrama GRASIA de la Ilustración 148.

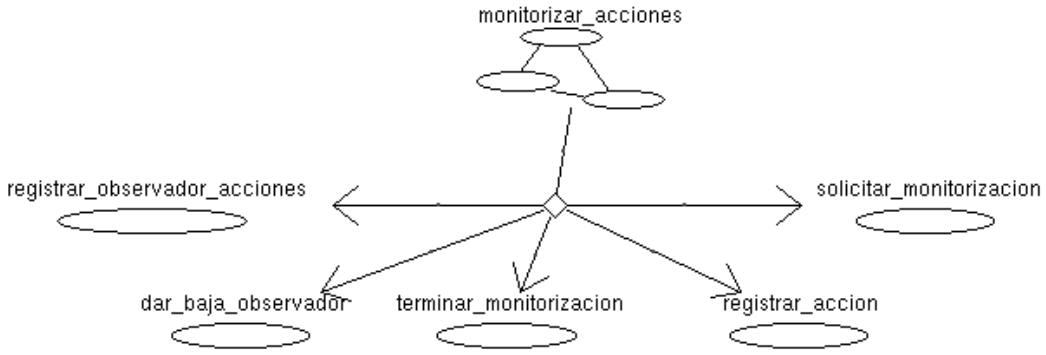


Ilustración 150. Tareas involucradas en el flujo de trabajo *monitorizar acciones*.

El refinamiento del flujo de trabajo *monitorizar acciones* se hace a partir del diagrama de colaboración de la Ilustración 110. Consiste en definir un patrón observador [Gamma et al. 95] utilizando los elementos de esta notación. Este flujo de trabajo permite elaborar estadísticas de las acciones de los usuarios y así saber si están satisfechos o no con los resultados proporcionados por sus agentes.

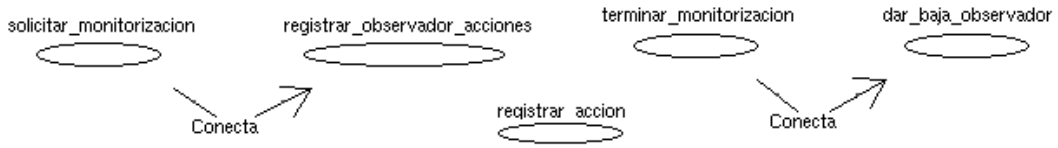


Ilustración 151. Relaciones entre las tareas del flujo de trabajo *monitorizar acciones*.

El flujo comienza con *solicitar_monitorización* que elabora una petición para el colaborador con el objetivo de que le informe de cuantas acciones se realicen (aquí interesan en concreto los votos positivos y negativos emitidos por los usuarios). La tarea *registrar_observador_acciones* registra al solicitante para informarle de futuras acciones. Se admite que varias entidades soliciten ser informadas de las acciones de los usuarios. Cada acción realizada se registra con la tarea *registrar_acción* que utilizará el *gestor de estadísticas* para actualizar los datos de cada usuario. Cuando la monitorización se dé por terminada, el monitor ejecutará *terminar monitorización*, tras lo cual el agente observado tendrá que deregistrar al monitor con la tarea *dar_baja_observador*.

Como en el flujo *evaluar documento* (Ilustración 146), se tienen flujos separados de ejecución. En este caso existen tres componentes, primero la encargada de solicitar la monitorización (tareas *solicitar monitorización* y *registrar observador acciones*), por otro el registro de las acciones (tarea *registrar acción*) y para terminar la finalización de la monitorización (tareas *terminar monitorización* y *dar baja observador*). Esta vez, el motivo de la separación es que aunque existen dependencias (e.g. no se pueden recibir las acciones de los usuarios si no se ha registrado antes el observador) estas se refieren únicamente al instante de ejecución y para especificar el instante de ejecución se usan diagramas GRASIA. Por ello, no se han interconectado las tareas correspondientes.

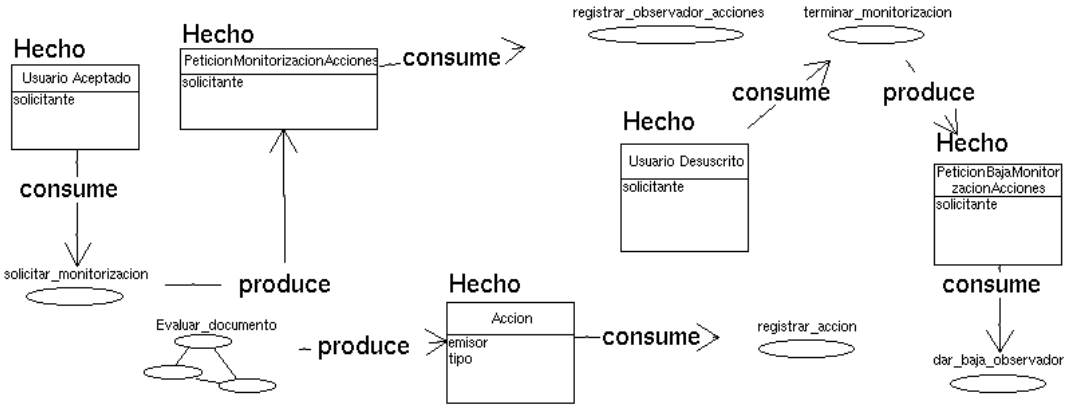


Ilustración 152. Descripción detallada del flujo de trabajo *monitorizar acciones*

Las entidades mentales intercambiadas en el proceso descrito en la Ilustración 151 se detallan en la Ilustración 152. La obtención de acciones se realiza obligando a las tareas a monitorizar a producir evidencias adicionales. Tal es el caso del flujo *evaluar documento* que generará evidencias de tipo *accion* para que estas se registren en el observador. Nótese que este flujo no aparece dentro del diagrama GRASIA de la Ilustración 153. Ello se debe a

que este flujo se ha introducido unicamente por motivos de claridad, y no porque pertenezca realmente al flujo *evaluar documento*.

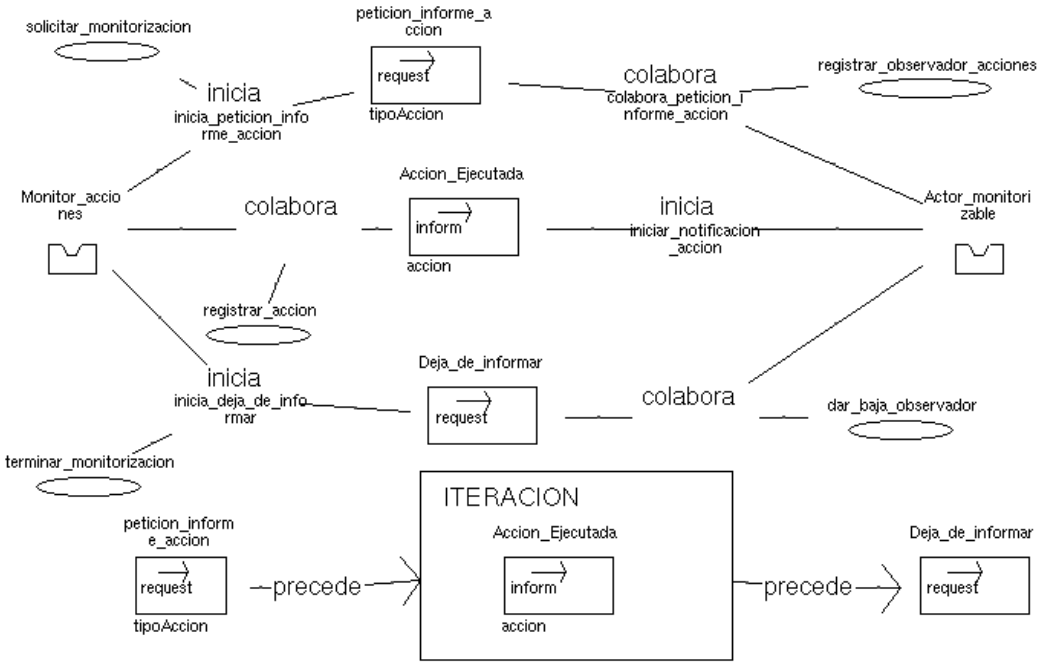


Ilustración 153. Diagrama GRASIA asociado a la interacción *monitorizar usuario*.

La especificación GRASIA de la interacción *monitorizar usuarios* (Ilustración 150) implementa un patrón observador. La unidad de interacción *accion_ejecutada* se repite hasta que el monitor decide que ya no necesita más información, esto es, hasta que se satisface el patrón de estado mental *inicia deja de informar*. Este patrón indica que debe existir un hecho *usuario desuscrito* para terminar la monitorización.

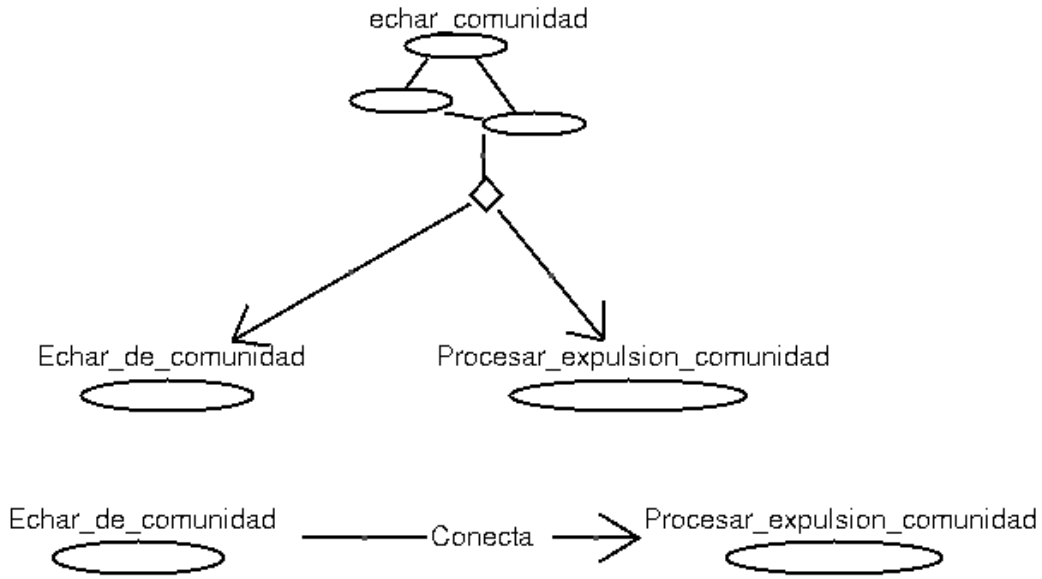


Ilustración 154. Tareas participantes en el flujo de trabajo *echar_comunidad*.

El flujo de trabajo *echar de comunidad* es el más simple de todos. Comienza cuando el *gestor de suscripciones de la comunidad* decide que un usuario no beneficia a la comunidad. Esta decisión se toma a partir de las estadísticas recolectadas sobre las acciones de los usuarios. La tarea *echar de comunidad* se encarga principalmente de eliminar de la comunidad al usuario y comunicarle el por qué del despido. El exmiembro de la comunidad, procesa la solicitud con *procesar_expulsión_comunidad*. Una decisión que se puede tomar a partir de la expulsión es volver a suscribirse.

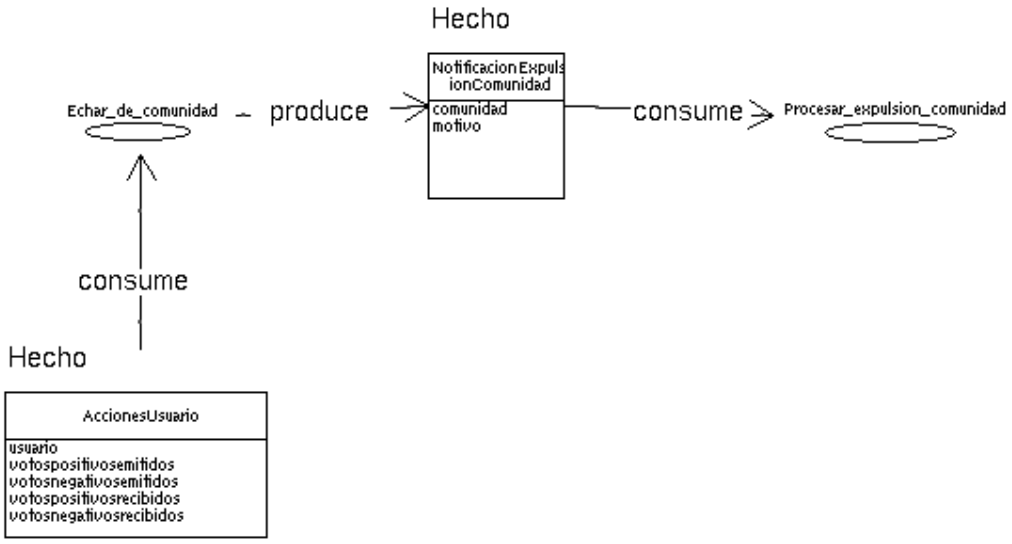


Ilustración 155. Descripción detallada del flujo de trabajo *echar de comunidad*

La Ilustración 155 muestra las entidades mentales intercambiadas en el flujo de trabajo *echar de comunidad*. La tarea *echar de comunidad* se activa cada vez que se modifican las estadísticas del usuario (hecho *acciones usuario*). Con esta información y la configuración del agente (no mostrada aquí pero sí en el patrón de estado mental *inicia echar comunidad* en http://grasia.fdi.ucm.es/metodologia) se procede a expulsar de la comunidad indicando el motivo (usuario inactivo, usuario insatisfecho con la información recibida o por proporcionar siempre información que no gusta a los otros miembros de la comunidad). Con esta información el *agente personal*, el que jugaba el rol *miembro de comunidad*, deberá decidir qué hacer. Una opción es hacer que la tarea produzca una entidad *hay que suscribirse a comunidad* para iniciar otra vez la suscripción automática a la comunidad (ver Ilustración 125).

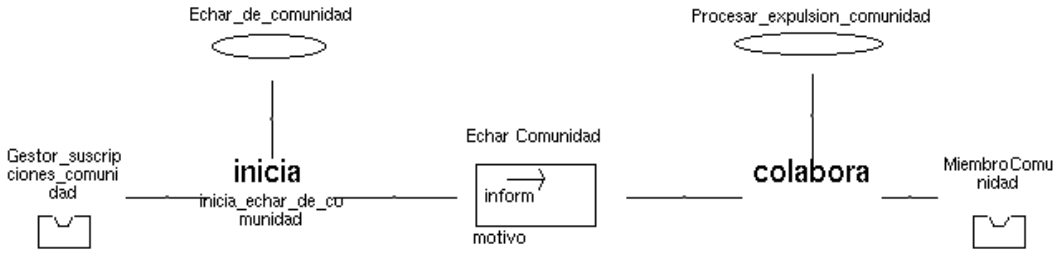


Ilustración 156. Diagrama GRASIA para representar la interacción *echar_usuarios*.

Las tareas de la Ilustración 167 se organizan según indica la Ilustración 156. No es necesario detallar ningun orden de ejecución porque sólo hay una unidad de interacción.

En cuanto a la satisfacción de objetivos, en esta sección se mostrará solamente como se satisfacen los objetivos asociados a las tareas del flujo de trabajo *compartir documento*s (ver Ilustración 137) asociadas al rol *emisor de sugerencias*. El resto de tareas se muestran en http://grasia.fdi.ucm.es/metodologia. Ya se ha estudiado cómo tiene lugar la ejecución de tareas dentro de flujos de trabajo, ahora se estudia cómo su ejecución consigue satisfacer objetivos. Para ello será relevante estudiar también las entradas, salidas, aplicaciones y recursos utilizados por las tareas.

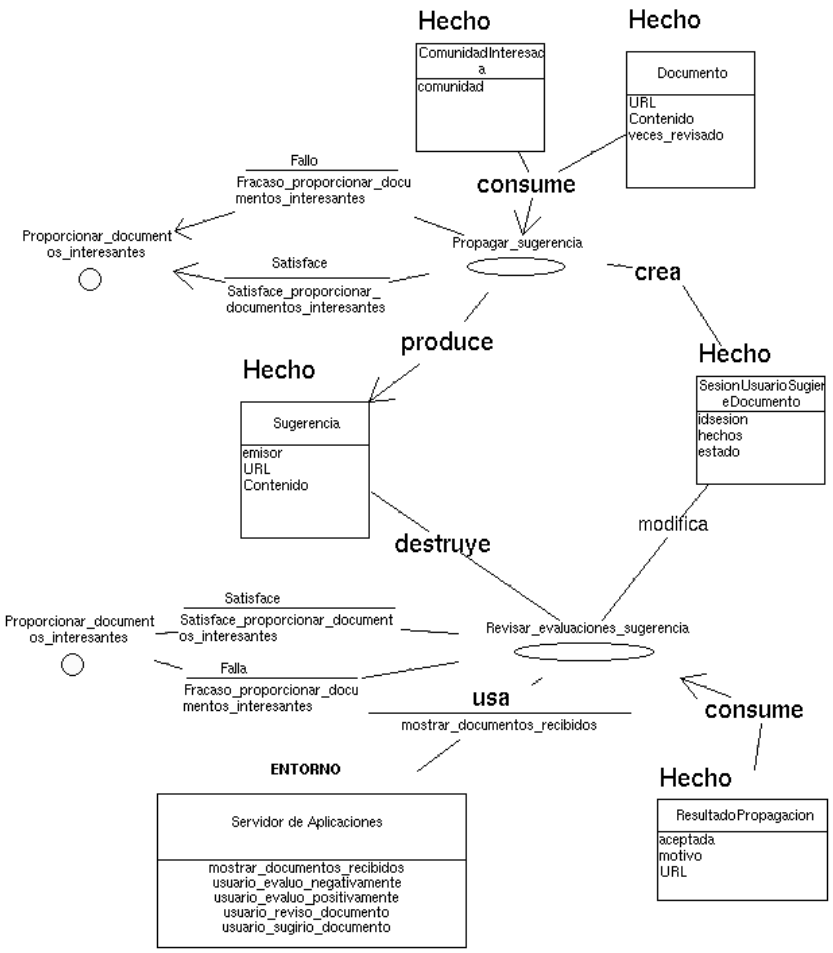


Ilustración 157. Satisfacción de los objetivos iniciales en base a la participación en flujos de trabajo (I)

La Ilustración 157 muestra cómo se satisface el objetivo *proporcionar documentos interesantes*. Este objetivo se satisface mediante dos tareas *propagar sugerencia* y *revisar evaluaciones sugerencia*. La que proporciona realmente las evidencias requeridas por el objetivo, que serán comentadas a continuación, es la segunda. Sin embargo, la segunda no puede suceder si no se ejecuta antes *propagar sugerencia*, de acuerdo con el flujo de trabajo. Así, se llega a la conclusión de que para alcanzar el objetivo *proporcionar documentos interesantes*, hay que ejecutar las dos tareas. Esto se indica en el diagrama asociando ambas tareas al mismo objetivo. Nótese que tras la ejecución de *propagar sugerencia*, no se satisface el objetivo asociado, pero su solución está más cercana ya que produce un hecho *sesion usuario sugiere documento* que se requiere en las condiciones de éxito y fracaso.

El hecho *sesion usuario sugiere documento* recoge el estado de la comunicación para poder detectar, por ejemplo, fallos en la comunicación y si la interacción ha terminado. En la Ilustración 157, las tareas pueden indicar estas situaciones durante la creación de este hecho fijando el valor “FALLO” (tarea *propagar sugerencia*) o bien modificando el slot *estado* de este hecho (tarea *revisar evaluaciones sugerencia*) cuando ya existe.

Al término del flujo de trabajo, la tarea *revisar evaluaciones sugerencia* utiliza la aplicación *servidor de aplicaciones* para informar al usuario de que se tiene el informe de aceptación de la sugerencia enviada.

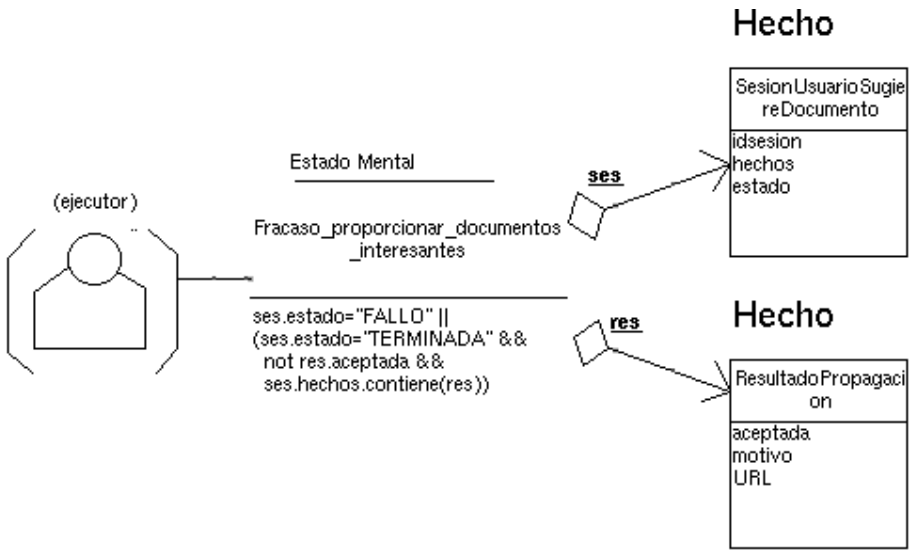


Ilustración 158. Condiciones para dar por fallido el objetivo *proporcionar documentos interesantes*

Para dar por fracasado el objetivo *proporcionar documentos interesantes*, se tiene que o bien ha ocurrido un fallo (el slot *estado* del hecho *sesion usuario sugiere documento* contiene el valor "FALLO"), o bien todo ha seguido los cauces normales, pero el documento no ha sido considerado interesante por la comunidad (el slot *aceptada* de *resultado propagación* está fijado a *false*).

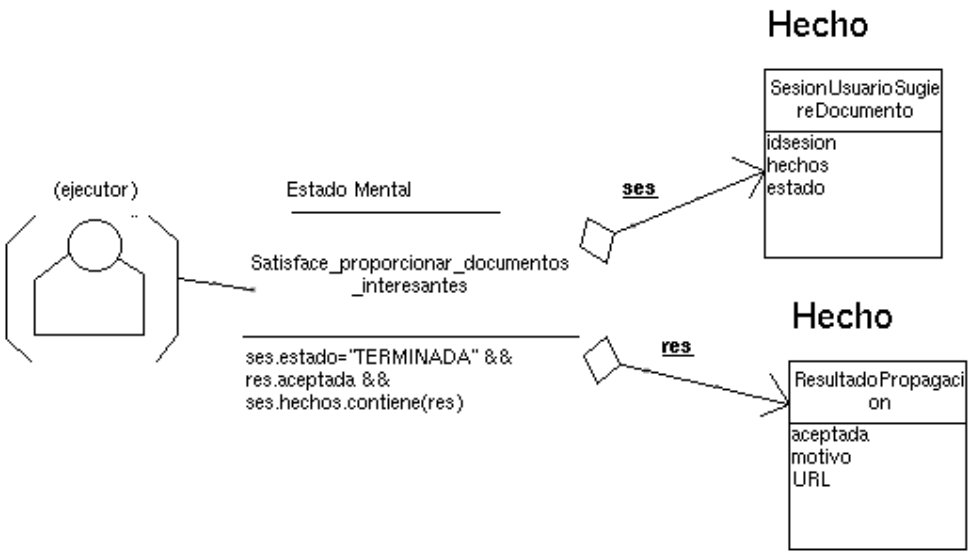


Ilustración 159. Condiciones para satisfacer el objetivo *proporcionar documentos interesantes*

Para dar por alcanzado el objetivo *proporcionar documentos interesantes*, el flujo de trabajo debe haber terminado, esto es, la tarea *revisar evaluaciones sugerencia* debe haberse ejecutado, y la sugerencia debe haber sido aceptada por la comunidad de usuarios (el slot *aceptada* de *resultado propagación* está fijado a *true*).

La satisfacción de los objetivos del sistema no tiene por qué depender exclusivamente de la existencia de entidades mentales. Así, existen objetivos que pueden alcanzarse o fracasar debido a las dependencias que exhiben respecto de otros objetivos.

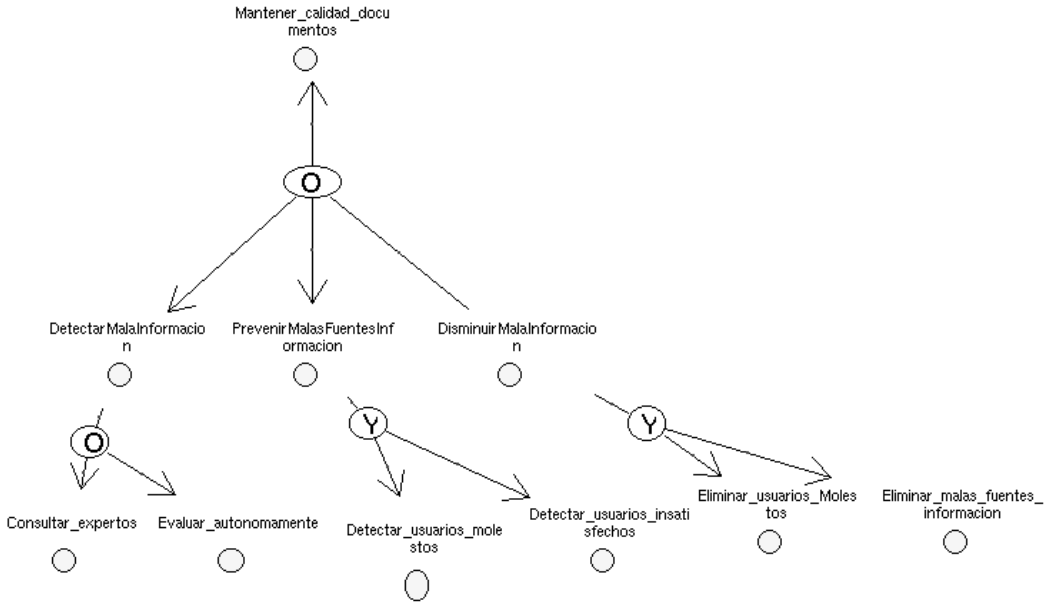


Ilustración 160. Dependencias entre objetivos (I)

Así, para satisfacer el objetivo *mantener la calidad de documentos*, se entiende que se pueden satisfacer cualquiera de sus subobjetivos (ver Ilustración 112). El motivo es que si se satisface alguno de ellos, se tendrán valoraciones de la calidad de la información, lo cual permitirá decidir con más acierto, o bien se evitará que aparezca información proveniente de fuentes que no han funcionado bien en el pasado. Por ello la calidad de la información que circule en las comunidades será, cuanto menos, mantenida.

El caso de sus sub-objetivos es diferente. *Prevenir malas fuentes de información* y *disminuir malas fuentes de información* necesitan que se satisfagan ambos sub-objetivos. Mientras que *detectar mala información* sólo necesita que se cumpla alguno. En los dos primeros, el motivo es que una identificación parcial o una eliminación parcial de malas fuentes, no es satisfactoria. Mientras que para detectar la mala información se puede confiar o bien en los expertos o bien en las herramientas de evaluación automática.

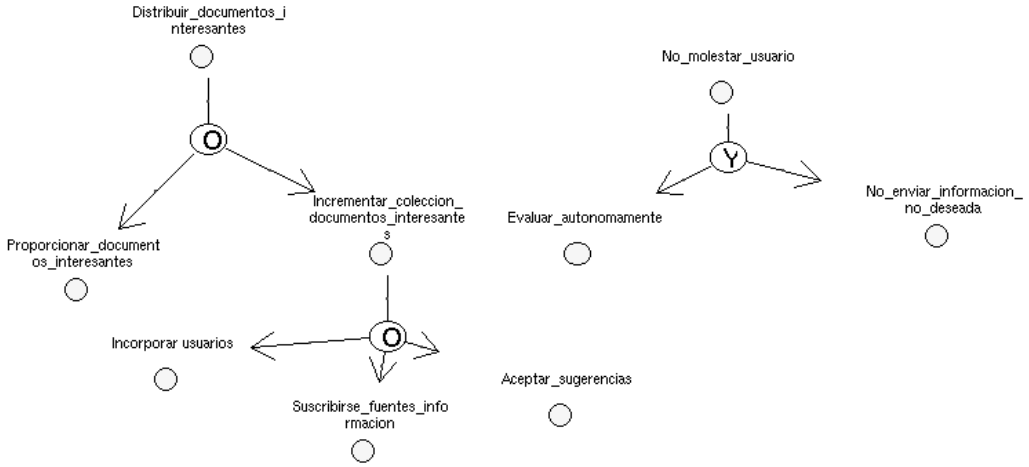


Ilustración 161. Dependencias entre objetivos (II)

Para no molestar al usuario se confía en la combinación de los mecanismos de evaluación automática junto con la consulta de expertos. El objetivo *evaluar autónomamente* se satiface invocando a los medios de evaluación automática, mientras que *no enviar información no deseada* utiliza mecanismos de filtrado colaborativo. Por ello, se requiere de la satisfacción de los dos objetivos.

La *distribución de documentos interesantes* es un objetivo compartido por *agentes personales* y por *agentes de comunidad*. Por ello, se entiende que se satisface cuando el agente sugiere documentos a la comunidad (objetivo *proporcionar documentos interesantes*) o cuando la comunidad incrementa su colección de documentos interesantes. Relacionado con *incrementar colección de documentos interesantes* están los objetivos encaminados a incrementar el número de usuarios en la comunidad (objetivos *suscribirse fuentes información* e *incorporar usuarios*), y a aceptar sugerencias de la comunidad (objetivo *aceptar sugerencias*), gracias a lo cual tienen lugar los procesos de filtrado colaborativo.

Las entidades mentales son importantes tanto para expresar la satisfacción o fracaso de los objetivos como para pasar información de una tarea a otra. Sin embargo, la actuación de las tareas, como se ha visto, crea muchos hechos que no son destruidos. Como se explicó en la sección 2.2, los modelos de objetivos y tareas pueden emplearse para especificar qué hacer con estas entidades mentales.

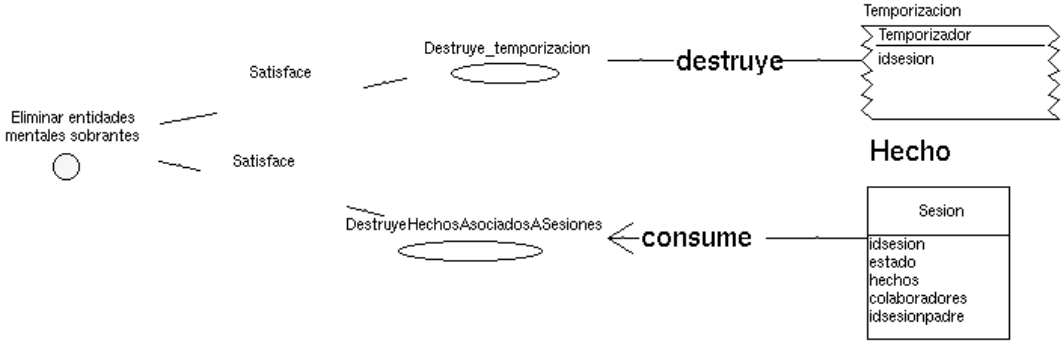


Ilustración 162. Descripción de la gestión del estado mental

La Ilustración 165 muestra un ejemplo de tareas que se encargan de destruir entidades mentales. La tarea *destruye temporizacion* elimina los eventos de temporización utilizados para detectar fallos en la comunicación o inactividad del usuario a la hora de evaluar documentos. Estos eventos han de ser eliminados si se producen cuando la sesión a la que están asociados ha terminado (ver la documentación completa). La tarea *destruye hechos asociados a sesiones* elimina la información relacionada con las distintas sesiones iniciadas por el agente una vez que estas han terminado.

Para terminar, queda hablar de la percepción de los agentes personales. El único agente que de momento necesita de modelos de entorno para definir su percepción es el *agente personal*. Hasta ahora la percepción era una asociación simple con el *servidor de aplicaciones*. Este agente está unido al servidor de aplicaciones ya que es a través de él que percibe las acciones de su usuario. Tras el estudio de tareas asociadas al agente, se está en condiciones de determinar exactamente cómo es esta percepción.

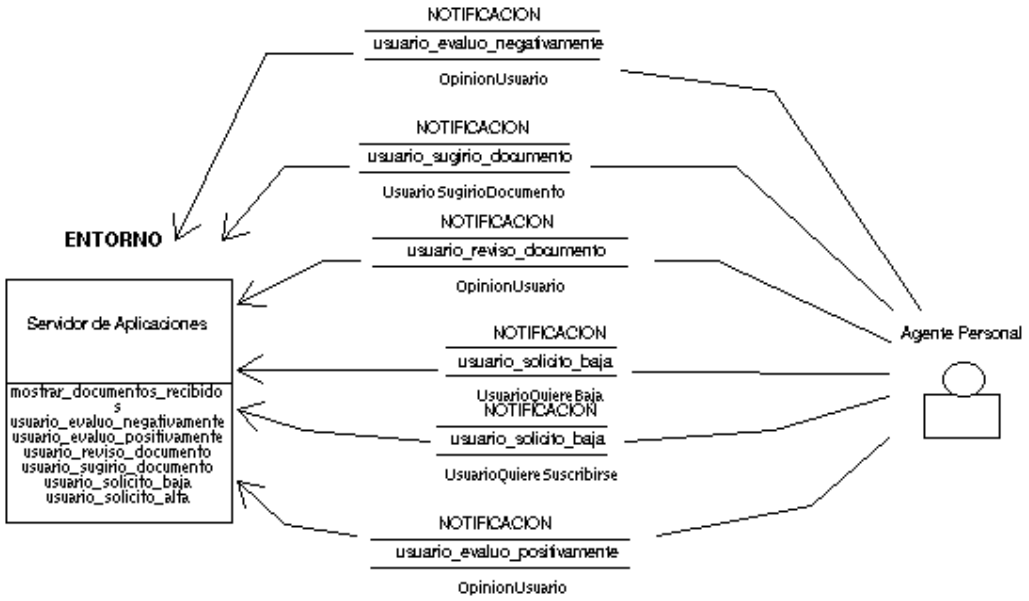


Ilustración 163. Percepción del agente personal

Según la Ilustración 166, cuando el usuario ejecuta una acción, se producen eventos. Los eventos que se tratan en el agente son *usuario sugirió documento*, que denota una sugerencia a una comunidad, *usuario quiere suscribirse*, que es interpretado por el agente como que hay que suscribirse a una comunidad, y *opinión usuario*, que hace referencia a la evaluación, positiva o negativa, de un documento o solicitud de evaluación de petición de suscripción.

4.6 Análisis-Construcción

Tras desarrollar los casos de uso necesarios para determinar la arquitectura del sistema, se estudian los restantes siguiendo las mismas actividades que en análisis-elaboración. Los modelos resultantes no deberían modifcar sustancialmente la visión que se tiene hasta ahora del sistema. En este caso de estudio, los nuevos casos de uso reutilizan los resultados de anteriores modelos para mostrar que en realidad sólo se están considerando situaciones especiales.

4.6.1 Estructuración de la etapa

Las actividades son las mismas que en la etapa anterior, solo que aplicadas a los casos de uso restantes. Sólo se mencionará el uso de actividades para la identificación de relaciones sociales, que no fueron consideradas en el apartado anterior.

Las dependencias sociales se establecen a tres niveles: organizaciones, grupos y agentes. Las dependencias entre organizaciones se generan con la actividad e*stabler relaciones a nivel de organización* (3.a). De forma similar, las dependencias entre grupos se generan con *establecer relaciones a nivel de grupo* **(**3.b) y las existentes entre agentes con *establecer relaciones a nivel de agente o rol* (3.c).

Actividad	**Tipo de resultado**	**Referencia**
Establer relaciones a nivel de organización (3.a).	Instancias de *AGORelacion* asociando *organizaciones*	Ilustración 168
Establecer relaciones a nivel de grupo **(**3.b)	Instancias de *AGORelacion* asociando *grupos*	
Establecer relaciones a nivel de agente o rol (3.c).	Instancias de *AGORelacion* asociando *agenes* o *roles*	

4.6.2 Resultados obtenidos

Tras construir la mayoría de casos de uso relevantes, quedan dos por considerar: *Intercambiar documentos i*nteresantes y *Extraer información de foros de noticias.*

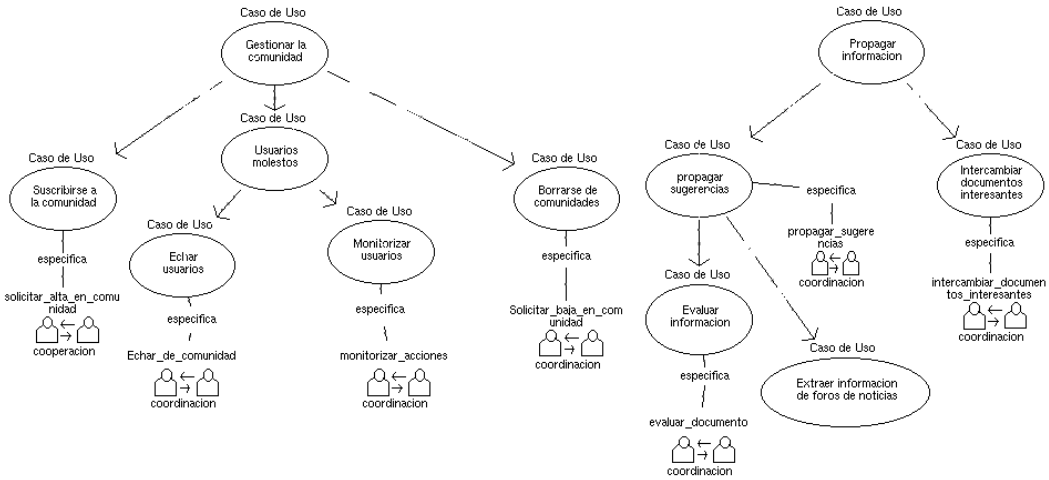


Ilustración 164. Nuevo caso de uso asociado a la propagación de información

Para extraer información de los foros de noticias no es necesario realizar ningún cambio en las interacciones ni añadir nuevos flujos de trabajo. Basta con definir un nuevo tipo de agente que herede de AgentePersonal y que tome anuncios publicados en el foro como sugerencias del usuario.

Intercambiar documentos interesantes plantea la necesidad de establecer dependencias sociales entre las distintas instancias de Empresa. Además, requiere que existan nuevos roles, flujos de trabajo e interacciones dedicados a tal efecto. El nuevo flujo de trabajo se denomina *Intercambiar documentos* (ver Ilustración 172). La nueva interacción es intercambiar documentos interesantes (ver Ilustración 165).

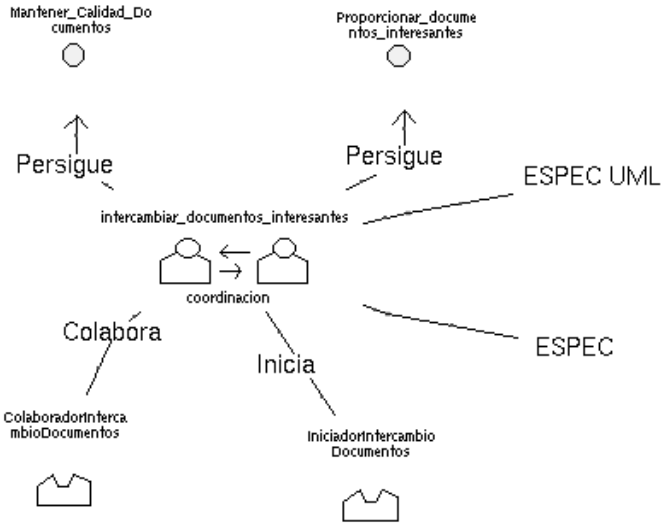


Ilustración 165. Interacción para el intercambio de documentos interesantes entre comunidades

En esta interacción, la organización designa un agente para que la represente frente a otras comunidades. Este agente desempeñará el rol *Iniciador Intercambio Documentos*. Por otro lado, otras organizaciones designarán otros agentes para que se encarguen de interactuar según el protocolo esbozado en la Ilustración 166. Estos agentes desempeñarán el rol *Colaborador de intercambio de documentos*.

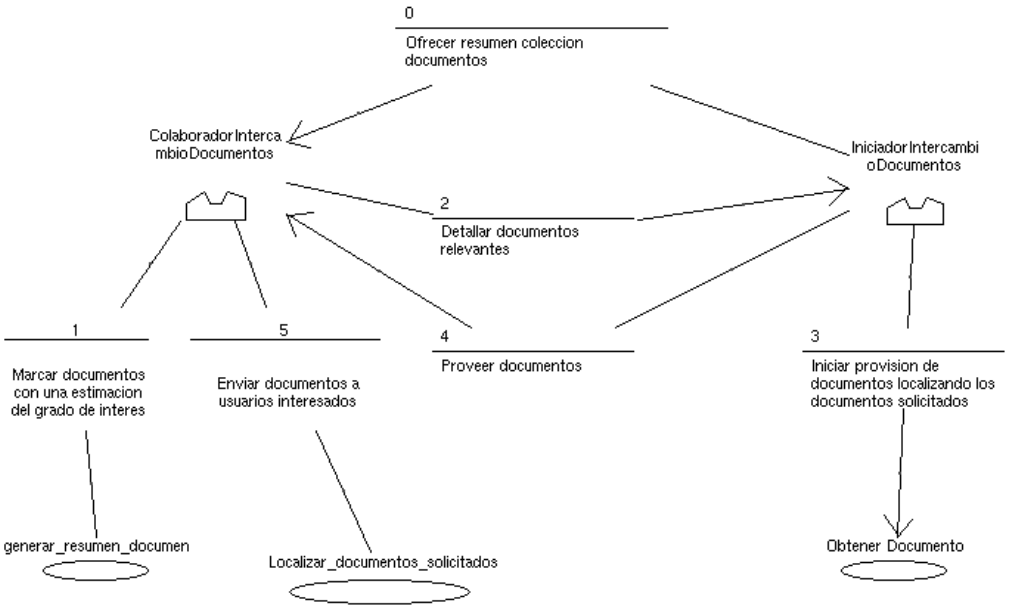


Ilustración 166. Diagrama de colaboración para la interacción *intercambio de documentos interesantes*

Los agentes encargados de representar a la organización frente a otras serán variantes de los *Agentes Personales*. Ello plantea interesantes posibilidades como que un agente representando una organización intervenga en el proceso de evaluación de documentos de otras comunidades de otras organizaciones. El nuevo agente se denomina *Agente Representante* (ver Ilustración 167).

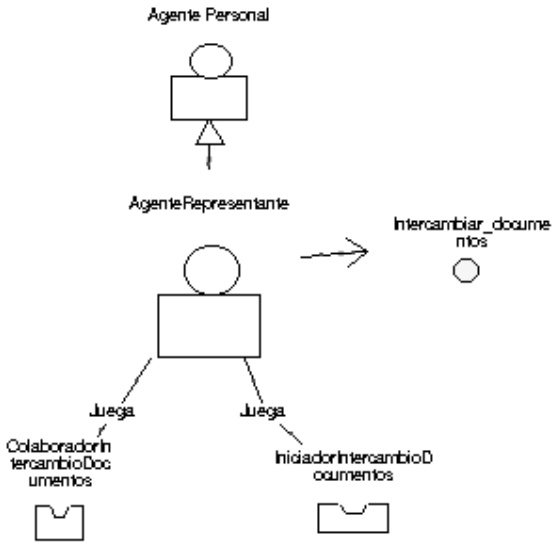


Ilustración 167. Agente Representante de la organización

La interactuación entre organizaciones obliga a plantear la existencia de dependencias entre organizaciones y entre grupos. Concretamente, se definen dependencias entre organizaciones *empresa*. Si estas dependencias se satisfacen, entonces se permitiría considerar las dependencias entre los agentes representantes.

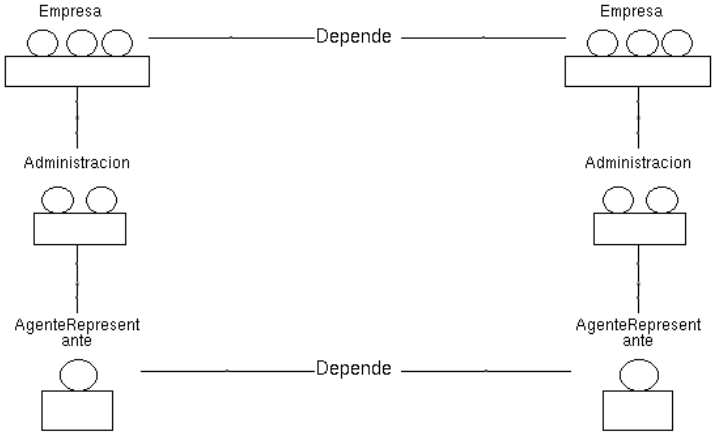


Ilustración 168. Dependencias sociales

Ilustración 168 muestra las dependencias antes mencionadas. Más adelante se concretará qué condiciones de actuación establecen estas dependencias. De momento, basta con saber que los *agentes representantes* no tienen libertad para contactar con otros *agentes*

representantes. El motivo es que no conviene dar información a otras organizaciones que no sean de confianza o que puedan hacer mal uso de la información suministrada.

Se identifica en el sistema un nuevo tipo de agente, el *agente extractor de noticias* y se hace que sea un caso especial de un *agente personal* (ver Ilustración 169).

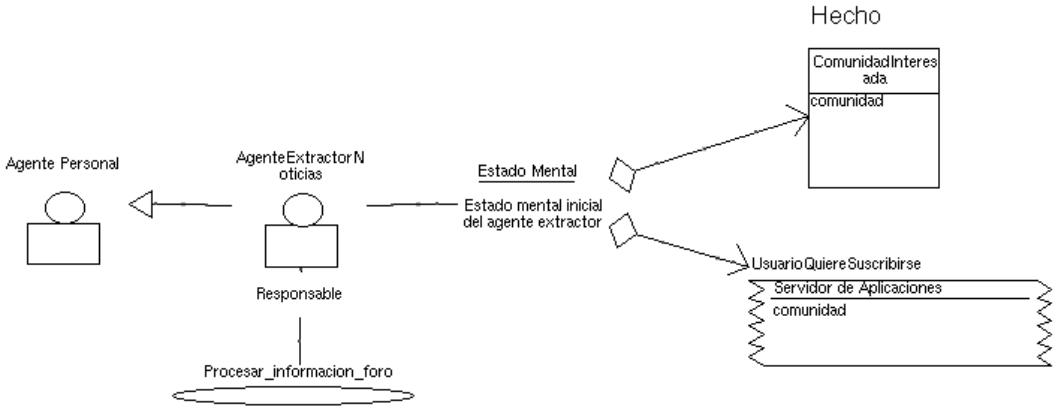


Ilustración 169. Agente extractor de noticias

Al igual que éste, se suscribe a comunidades en las que actúa como emisor de sugerencias extraídas de un foro de noticias. También puede actuar como evaluador, utilizando como criterio de evaluación la información obtenida del foro. El estado mental del agente se inicializa con las entidades mentales que disparan una suscripción en una comunidad.

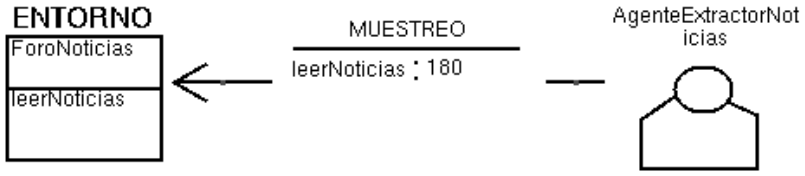


Ilustración 170. Percepción del *Agente Extractor de Noticias.*

La existencia del agente extractor de noticias también plantea la existencia de un nuevo tipo de aplicación, el *foro de noticias* (Ilustración 170). Este agente define su percepción en términos del foro de noticias, muestreándolo cada 180 minutos.

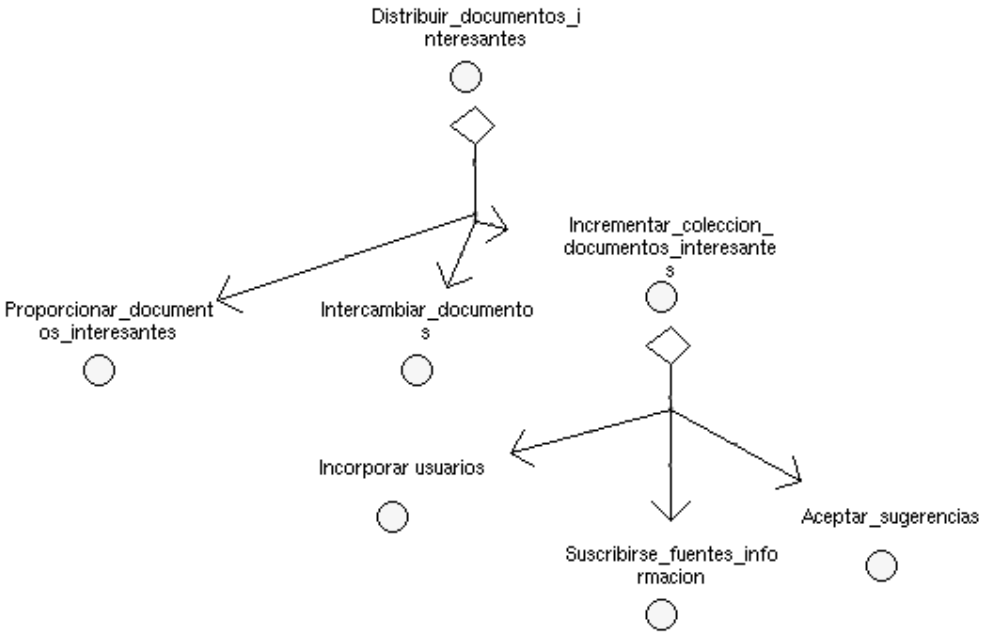


Ilustración 171. Inclusión del nuevo objetivo *intercambiar documentos*

Esta nueva funcionalidad, sugiere que ahora además de *proporcionar documentos interesantes* se puede *intercambiar documentos*. Como ello se puede entender como un caso especial de la *distribución de documentos interesantes*, *intercambiar documentos* se introduce como subobjetivo suyo.

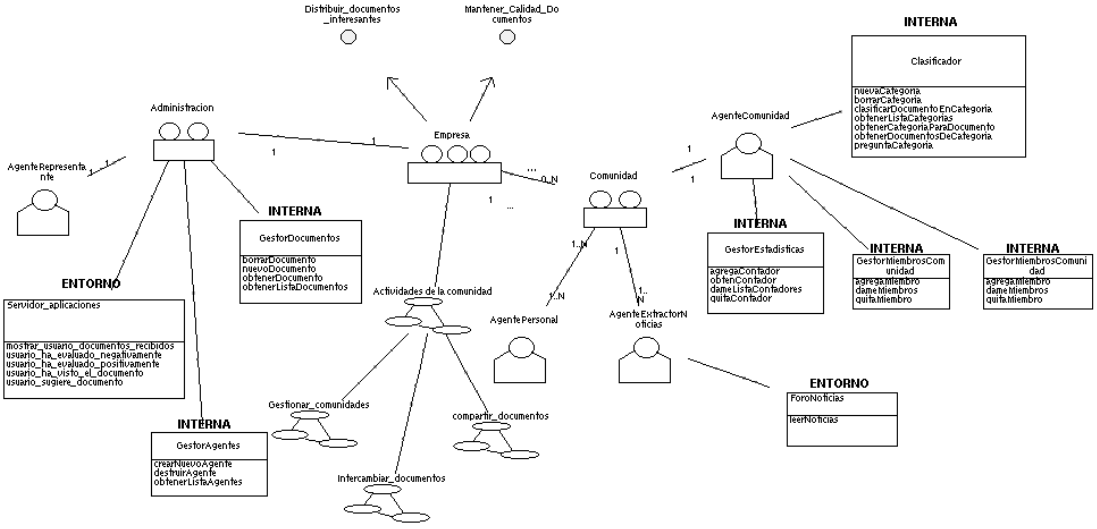


Ilustración 172. Estructura de la organización tras añadir el nuevo caso de uso y el nuevo tipo de agente

La organización queda como indica la Ilustración 172. La aplicación *ForoNoticias* se asocia al *Agente Extractor de Noticias* ya que este va a ser su único usuario. El nuevo flujo de trabajo *intercambiar documentos* se incorpora al flujo *actividades de la comunidad*.

4.7 Diseño-Construcción

Los nuevos casos de uso requieren que se tengan en cuenta las relaciones sociales. De hecho esta es la principal novedad respecto de lo que está hecho. Como resultado de esta etapa se obtendrán flujos de trabajo y diagramas GRASIA. El resto de la documentación se encuentra en http://grasia.fdi.ucm.es/metodologia.

4.7.1 Estructuración de la etapa

Como en la etapa anterior, no se repetirá el estudio detallado de cada actividad. Para complementar lo mostrado en secciones anteriores, se hará hincapié en el papel de las actividades orientadas a la identificación de dependencias sociales.

Las actividades encargadas de refinar las dependencias sociales son *relaciones de subordinación* (4.a) y *relaciones cliente-servidor* (4.b). En este caso, tiene más relevancia las relaciones de subordinación, por lo que la otra actividad se dejará pendiente. Ya no hace falta refinar a nivel de grupo, agente y organización, ya que se trabaja sobre dependencias ya existentes.

Actividad	Tipo de resultado	Referencia
relaciones de subordinación (4.a)	Instancias de *AGOSubordinacionCondicional y AGOSubordinacionIncondicional* . Modelos de agente para mostrar el estado mental requerido en los participantes.	Ilustración 173 Ilustración 174

4.7.2 Resultados obtenidos

La aparición del *Agente Representante* induce la existencia dependencias sociales entre organizaciones.

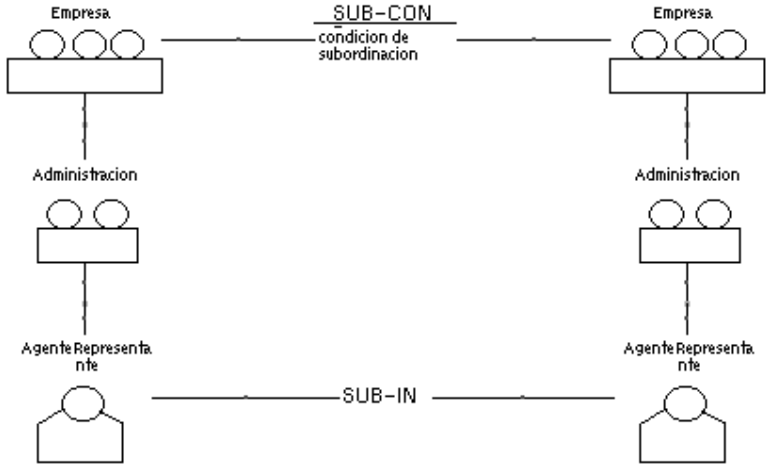


Ilustración 173. Relaciones sociales entre entidades *Empresa*

Las condiciones de subordinación se resumen en que la organización a la que pertenezca el solicitante, debe pertenecer a una lista de empresas admitidas que debe conocer previamente el agente y que se pueden modificar en tiempo de ejecución (ver Ilustración 174).

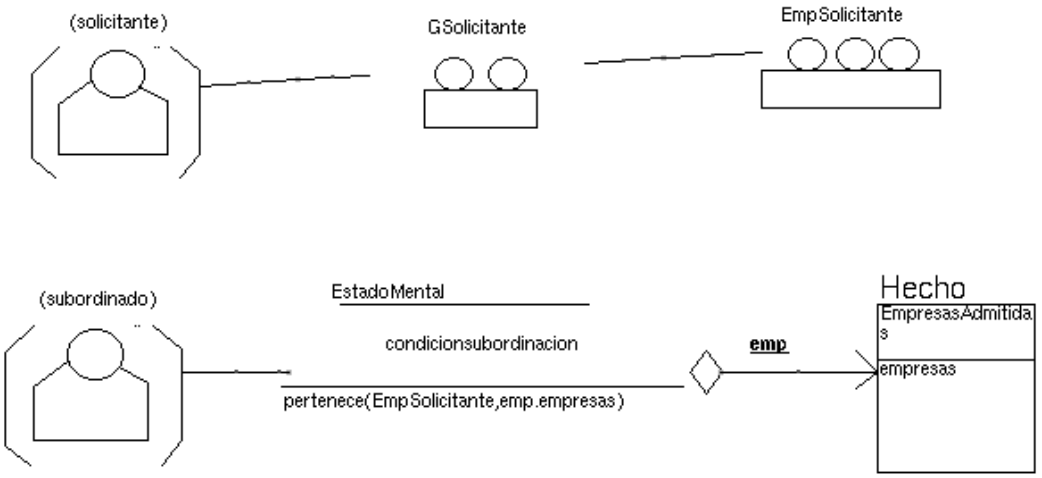


Ilustración 174. Condición de subordinación entre dos empresas para el intercambio de documentos

La introducción del nuevo flujo de trabajo, implica el definir tareas para el mismo así como las diferentes relaciones entre ellas. Por refinamientos del diagrama de colaboración inicial (ver Ilustración 166) se identifican las tareas de la Ilustración 175.

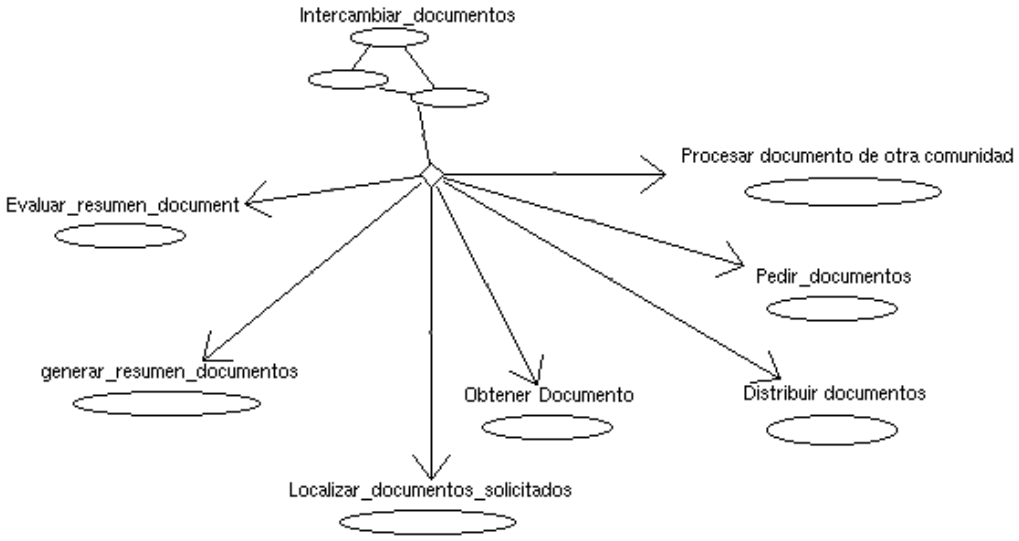


Ilustración 175. Tareas del flujo de trabajo *intercambiar documentos*

El *pedir documentos* significa solicitar a una organización un resumen de los documentos más relevantes que posea. El resumen se genera con la tarea *generar resumen de documentos* y se procesa con *evaluar resumen de documentos*. Este procesamiento consiste en cotejar los documentos de la otra organización con conjuntos de documentos de la actual. Si resulta interesante, se procede a generar una lista de cuáles son los

documentos a requerir (*solicitar documentos de interes*). Los documentos solicitados, se extraen mediante la tarea *obtener documento* y el nuevo documento se agrupa junto con otros recibidos en el solicitante mediante *procesar documento de otra comunidad*. Una vez obtenidos todos los documentos, se procede a *distribuir documentos*, tarea que hace llegar los documentos a las comunidades de la organización.

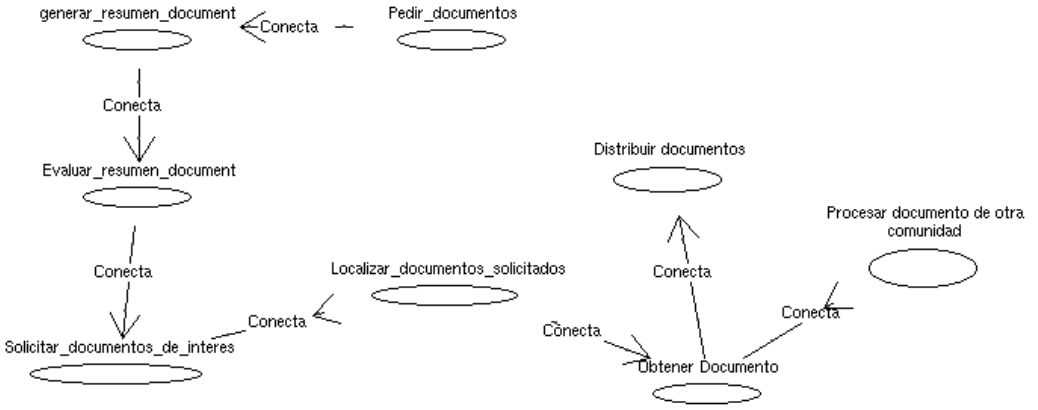


Ilustración 176. Relaciones entre las tareas del flujo de trabajo *intercambiar documentos*.

La secuencia de ejecución de tareas antes descrita se muestra en la Ilustración 176. Llama la atención la conexión entre *obtener documento, distribuir documentos* y *procesar documento de otra comunidad*. En este caso, la conexión de tareas es correcta, ya que la tarea *Obtener documento* produce dos tipos de entidades *DocumentoSolicitado* y *Provisión Finalizada*. Según cuál se genere, se activará una u otra. Esto se verá en la descripción de los estados mentales de los agentes durante la interacción.

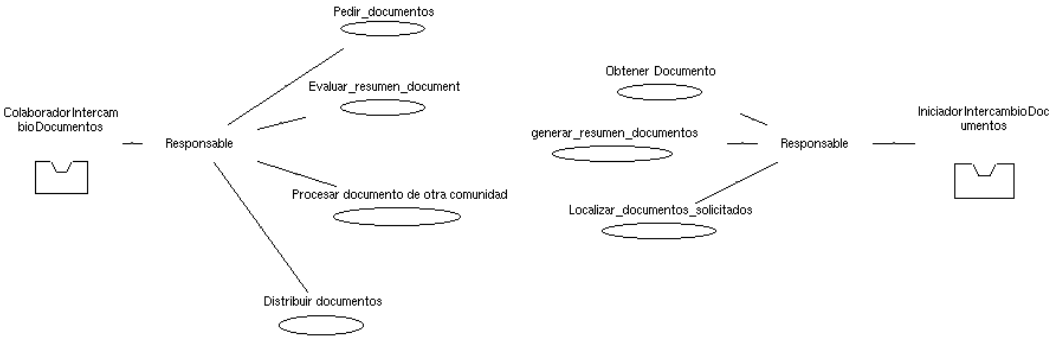


Ilustración 177. Asignación de responsabilidades en el flujo *intercambiar documentos*.

Las responsabilidades se asignan según la Ilustración 177. Esta asignación se obtiene a partir de los refinamientos del diagrama de colaboración que se muestran a continuación.

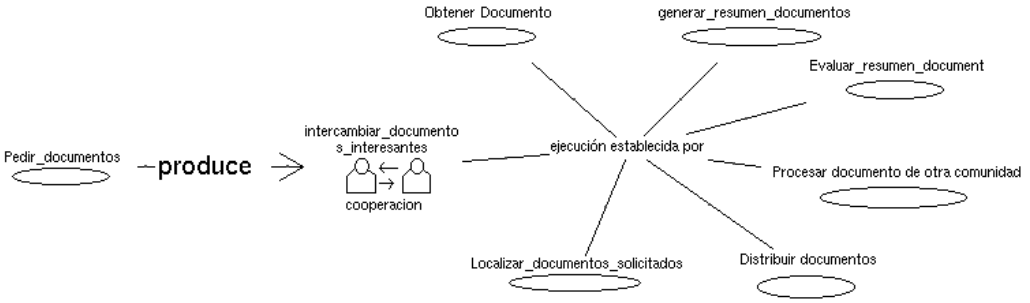


Ilustración 178. Participación de las tareas del flujo de trabajo *intercambiar documentos* en *intercambiar documentos interesantes*.

Las tareas del flujo de trabajo *intercambiar documentos* se enmarcan dentro de la interacción ya mostrada en la Ilustración 165. Como se puede deducir de las asociaciones *conecta* de la Ilustración 176, la tarea inicial es *Pedir documentos*. Es esta tarea la que produce la interacción.

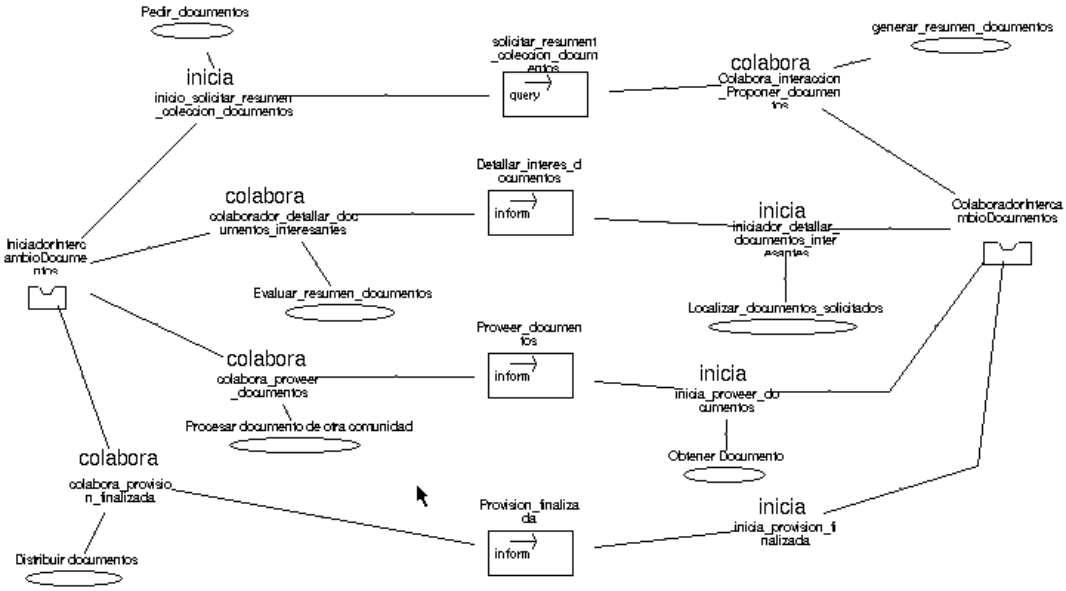


Ilustración 179. Unidades de interacción en la interacción *intercambiar documentos interesantes*

Las unidades de interacción de la Ilustración 179 identifican la ordenación de tareas de la Ilustración 175. Llama la atención que algunas instancias de *UIinicia* y *UIColabora* no se relacionan con ninguna tarea. Ello es porque no es necesario una tarea previa distinta de *generar resumen de documentos*. El resumen es directamente enviado al iniciador de la interacción. Una situación similar se encuentra en la unidad de interacción *Provisión finalizada*.

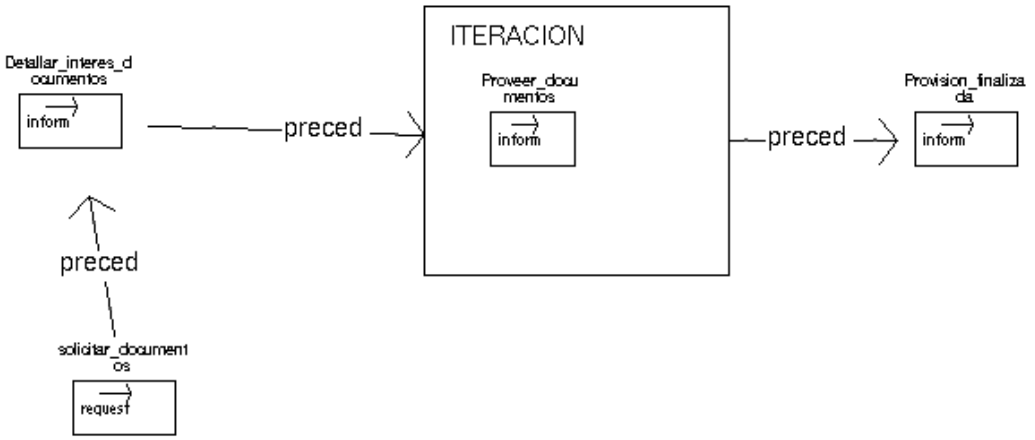


Ilustración 180. Organización de la unidades de interacción en la interacción *intercambiar documentos interesantes*.

Los documentos del colaborador no se proporcionan de una sóla vez, sino de manera escalonada, para evitar saturaciones de la red. Por ello, se plantea una iteración de la unidad de interacción *Proveer documentos*. La duración de la iteración depende de que se cumpla el estado mental que permite iniciar *Provisión finalizada*. La iteración termina porque el conjunto de documentos a suministrar es obtenido de la lista inicial de documentos acordada al principio de la interacción.

4.8 Conclusiones

A lo largo de este capítulo, se ha mostrado un desarrollo de un sistema de filtrado colaborativo de información basado en un caso del proyecto europeo PSI3 [PSI3 01]. El ejemplo muestra la naturaleza iterativa del proceso de generación de modelos y cómo las entidades que van apareciendo en análisis y diseño van afectando a la construcción de los distintos modelos, de acuerdo a las actividades definidas en el artículo tercero.

También se ha mostrado cómo aplicar las actividades identificadas en el capítulo tercero para generar de forma escalonada la especificación del sistema. En cada etapa se ha indicado qué activides se han ejecutado y qué resultados se han obtenido de cada una. El ejemplo completo, por su extensión, no ha sido incluido en su totalidad en este trabajo. La especificación detallada puede revisarse en http://grasia.fdi.ucm.es/metodologia. Esta documentación está pensada para facilitar su revisión y está estructurada siguiendo pautas mostradas en la sección 3.7.4 .

Aunque no se ha mostrado el aspecto de la implementación, con herramientas de generación automática de código, se soportaría la realización de prototipos en cada iteración para evaluar mejor las consecuencias de incorporar requisitos planteados por el cliente en la aplicación. Estas herramientas, como se indicó en el capítulo tercero, se basarían en la parametrización de armazones software. El concretar el armazón software y el proceso de parametrización son actividades paralelas al proceso de especficación del SMA. El nexo entre estas actividades y la generación de código se tiene al final de cada fase en cada

iteracción, dentro de la actividad *generar parametrización del armazón software*. Dentro de esta actividad se verificaría por un lado la corrección de la especificación y por otro la del proceso de parametrización.

Capítulo 5. CONCLUSIONES FINALES

Este último capítulo recoge las principales conclusiones de la memoria, describiendo las aportaciones realizadas, las próximas líneas de investigación y trabajo futuro.

5.1 Aportaciones

En esta tesis, las aportaciones principales son un lenguaje de especificación de SMA y su integración dentro de las prácticas de ingeniería.

El lenguaje de especificación de SMA parte del trabajo que hemos realizado en MESSAGE [Caire et al. 01], que extendemos al aportar una mayor cohesión de los modelos (de acuerdo al *principio de racionalidad*), al definir un nuevo modelo para definir el entorno y al considerar la integración de resultados de investigación en el área de agentes. Este lenguaje se ha construido con meta-modelos que definen cinco aspectos del SMA: agentes, interacciones, organización, entorno, tareas y objetivos. Para la elaboración de los meta-modelos presentados se ha realizado un estado del arte de cada área de investigación relacionada para estudiar qué elementos son importantes en el diseño de un SMA. Así, se han estudiado tecnologías de comunicación entre componentes, técnicas de planificación de tareas, métodos de estructuración y agrupación de agentes y teorías de agentes, entre otros. Dentro de este estado del arte se incluyen aportaciones originales del autor, concretamente técnicas para modelar control de agentes orientado por objetivos [Gomez-Sanz, Garijo y Pavon 00] y organización de los agentes [Garijo, Gomez-Sanz y Massonet 01].

La estructuración de estos elementos mediante meta-modelos ha sido de doble utilidad para el desarrollador ya que además de servir como notación, actúa como guía en la especificación del sistema. El desarrollador sabe desde el principio qué clases de elementos van a componer su SMA y cómo debe caracterizarlos. Ello ha facilitado el proceso de estudio del dominio del problema y de codificación de la solución en el SMA. Como ayuda en este

uso, cada elemento y asociación se ha acompañado de una semántica básica expresada en lenguaje natural e ilustrada con ejemplos de aplicación.

Adicionalmente, se han introducido mecanismos que permiten simplificar las especificaciones agrupando conjuntos de elementos. Tal es la función de las organizaciones, los flujos de trabajo y las interacciones. Gracias a estas abstracciones, es posible generar descripciones generales del sistema que más adelante se concretarán en agentes, tareas y unidades de interacción, respectivamente.

Los meta-modelos como estructuración de elementos relevantes en el desarrollo de SMA han sido publicados como aportación original en [Gomez-Sanz, Pavon y Garijo 02].

Para comprobar la aplicabilidad de los meta-modelos ha sido necesario personalizar la herramienta METAEDIT+. Esta herramienta se ha usado extensamente para desarrollar los casos de estudio presentados en esta tesis, definir procedimientos para generar documentación en HTML, y producir representaciones intermedias de los meta-modelos adecuadas para el proceso de generación de código. La generación de documentación también se ha tenido en cuenta. Concretamente, se ha invertido esfuerzo en definir un organización de contenidos en HTML para facilitar la comprensión del sistema. La organización final proporciona información referida a aspectos indicados en la sección 3.7.4 y está disponible en http://grasia.fdi.ucm.es/metodologia

La integración en las prácticas de ingeniería se ha descrito extensamente dentro del tercer capítulo. Esta integración es importante para conseguir asimilar la producción de SMA con la generación de software convencional. Mediante la integración los meta-modelos propuestos se deberían utilizar en el seno de un proceso de ingeniería del software. Es este proceso el que determina en cada momento qué nivel de detalle se requiere y qué debe especificarse, racionando el esfuerzo de generación de los modelos a lo largo del desarrollo para conseguir una representación coherente del sistema. Por ello, se ha invertido trabajo en determinar cuál es el papel de los meta-modelos en los procesos de ingeniería, cómo sería su utilización y cómo se llega a la implementación final del sistema.

En este sentido, la tesis aporta patrones arquitectónicos de diseño, un método genérico de generación de código, un conjunto de actividades para la generación de la especificación del sistema y su integración en el RUP.

Los patrones arquitectónicos se entienden como abstracciones de arquitecturas del sistema y sus componentes. Al identificar el conjunto de elementos necesarios para definir el agente y el SMA, se está aludiendo a qué elementos han de estar presentes en la arquitectura que soporte el SMA, cómo se relacionan y qué funcionalidad deben aportar. Los patrones identificados y componentes físicos que los pueden representar se han mostrado en la sección 3.7.4 .

El método de implementación adoptado ha sido el de tomar una plataforma de desarrollo o armazón software, marcar el código fuente para indicar cómo se usa la información de los modelos y aplicar el proceso de parametrización descrito en la sección 3.7. Este método es totalmente genérico ya que se basa en la instanciación de armazones concibiendo el código como un conjunto de documentos marcados.

Las actividades se han obtenido de la experimentación en diversos casos de estudio a lo largo de toda la tesis y según la experiencia en el diseño de SMA adquirida en los proyectos PSI3 [PSI3 01], P815 [Eurescom P815 99] y MESSAGE [Caire et al. 02]. La aplicación de estas actividades se ha ilustrado con detalle dentro del caso principal de estudio del capítulo

cuarto, mostrando para cada etapa qué actividades intervienen y qué resultados concretos se obtienen. También, se ha prestado atención al problema de combinar los diferentes modelos dedicando la etapa final de generación de los modelos a la comprobación de coherencia entre los modelos. Esta comprobación consiste en determinar si se cumplen las reglas presentadas en las secciones 2.2.7 , 2.3.4 ,2.4.5 , 2.5.4 y 2.6.4 .

La participación de estas actividades en el RUP ha sido expresada a través de los resultados a esperar en cada iteración y etapa. Las actividades se han organizado alrededor de las etapas de análisis y diseño mediante diagramas de actividades. El diagrama de actividades se sigue, dentro de cada iteración, desde el estado inicial al final para obtener los resultados indicados en la **Tabla 7**. Adicionalmente, y como aplicación práctica de estas ideas, en el capítulo cuarto se ha indicado con detalle cómo aparecen estas actividades en un desarrollo complejo y qué resultados se obtienen dentro del desarrollo.

Parte de estos resultados han sido publicados en varios congresos internacionales (*Modeling Autonomous Agents in a Multi-Agent World*, ACM, *Mexican International Conference of Artificial Intelligence*) y nacionales (Simposio Español de Informática Distribuida). Las publicaciones se han centrado en la definición de los meta-modelos [Gomez-Sanz, Pavon y Garijo 02] [Garijo, Gomez-Sanz y Massonet 01] y en conceptos básicos como construcción de agentes [Gomez-Sanz, Garijo y Pavon 00] y su definición [Gómez-Sanz 00]. Relacionado también con este trabajo, se tiene el trabajo realizado dentro MESSAGE [Caire et al. 01] publicado en el *Agent Oriented Software Engineering workshop* dentro de la conferencia *Autonomous Agents 2001*.

5.2 Limitaciones y líneas de investigación abiertas

Como limitación de la metodología hay que destacar la necesidad de una mayor validación con nuevos casos de estudio. A pesar de basarse en experiencia previa en dominios distintos (gestión de proyectos software [Eurescom P815 99], asistencia en viajes [Caire et al. 02]) y personalización [PSI3 01], sería conveniente tratar otros dominios con características particulares como son el diseño de sistemas en tiempo real o comercio electrónico.

Esta metodología permite un nivel de detalle que a veces puede ser excesivo. En defensa de la metodología, hay que aclarar que el nivel de detalle es proporcional al armazón donde se vaya a implantar el código de la aplicación y a la utilización o no de los medios de generación de código aquí comentados. Si el diseñador plantea únicamente documentar ligeramente el desarrollo, el nivel de detalle no tiene que ser tan elevado. Por otro lado, si desea utilizar los meta-modelos para parametrizar una aplicación ya desarrollada, los parámetros marcarán el nivel de detalle que el ingeniero debe proporcionar.

Una mejora observable y corregible a corto plazo es la incorporación a la metodología del concepto de trabajo en equipo. Si bien una persona puede completar la especificación de todo el sistema, cuando varias personas intentan completar diferentes aspectos del sistema, surgen problemas. Los meta-modelos presentados son compatibles con desarrollos en grupo siempre y cuando se establezcan parcelas de actuación centradas en flujos de trabajo donde los actores sean roles. La integración se realiza haciendo que los agentes finales se asocien a los roles mediante instancias de la meta-relación *WFJuega*. Las posibles inconsistencias originadas por el trabajo de distintas personas con distintos puntos de vistas quizás no sean tan sencillas de solucionar. Estas inconsistencias se darán sobre todo en diferentes

interpretaciones de los modelos. Para evitar estas inconsistencias, probablemente el camino sea desambiguar los modelos tomando como referencia el armazón software sobre el que se vaya a implementar. En este sentido, sería deseable dotar al sistema de medios para definir la semántica asociada al armazón software y para aplicarla al desarrollo actual.

En línea con el comentario anterior, hay que aclarar que no se ha proporcionado una semántica que describa cómo es el armazón y qué requisitos impone a la metodología. Este no es el objetivo del trabajo aunque se admite que es un requisito con vistas a la futura implantación de esta metodología. El lector convendrá en que la definición de la semántica de un armazón constituye por sí solo trabajo para otra tesis. Sin embargo, tampoco podía quedarse este trabajo sin decir nada al respecto. Por ello, se ha dedicado una sección del capítulo tercer a considerar aspectos de implementación y cómo participan en ellos los modelos propuestos.

Finalmente, se deja como aspecto abierto la aplicación de métodos formales a las especificaciones generadas. Dado el gran número de diagramas generados, es difícil asegurar que no se están cometiendo errores. En este sentido, los métodos formales, partiendo de una semántica como la descrita antes, podrían estudiar si existen inconsistencias entre los diferentes modelos o si faltan tareas que hagan posibles la satisfacción de un determinado patrón de estado mental.

5.3 Trabajo futuro

El trabajo futuro se enmarca dentro de tres líneas de desarrollo. Por un lado la mejora de la metodología, por otro la mejora de los procesos de generación de código y finalmente la inclusión de herramientas para soportar las actividades de generación de modelos.

Respecto de la metodología, los meta-modelos propuestos han sido diseñados para ampliarse con los resultados de otras áreas. En concreto, se espera fomentar la parte de control del agente incorporando soluciones concretas para razonamiento y aprendizaje. Éstas se integrarían dentro de los procesadores y gestores de estado mental, como ya se ha comentado con anterioridad. Otro aspecto en el que se quiere ahondar es la incorporación en la metodología de los aspectos de interacción en sistemas de tiempo real, levemente introducida aquí con las relaciones *EPercibe*.

En cuanto a la generación de código, ya se ha comentado que sería deseable estudiar cómo afecta la semántica operacional asociada con el armazón en los meta-modelos existentes. La viabilidad de este estudio es función directa de lo difícil que sea expresar la semántica operacional del armazón. En paralelo con este estudio, se prevé automatizar el proceso de parametrización del armazón. Aquí se ha mostrado cuáles son las herramientas mediante las que se espera resolver el problema (marcado de código con XML y sustitución de etiquetas por valores de los modelos que representan el SMA). Estas herramientas se pueden componer dentro de un entorno integrado de desarrollo para ayudar al ingeniero a utilizar la metodología con cualquier armazón.

Dado el alto número de actividades identificadas para generar los modelos (ver capítulo tercero), sería deseable la utilización de herramientas que automatizasen en parte la asignación de actividades a los trabajadores de un proyecto, el seguimiento del progreso de las actividades y la comunicación de los resultados de una actividad a otra. El problema es

similar al encontrado cuando se planifican las tareas en un proyecto software, para lo cual ya existen herramientas que lo solucionan.

Bibliografía

[1] Adaptative Ltd.DSTC Gazebo Software Solutions:*Common Enterprise Models Domain Task Force: Proposal on Organizational structure submitted*. Informe. OMG TC Work in Progress. 2000

[2] Ambvroszkiewizcz, S., K., C., O., M. y B., R.: *Modelling Virtual Enterprise: agent-based approach*. Actas de conferencia. Proceedings of II Iberoamerican Workshop on DAI and MAS. 1998.

[3] Barber, F., Botti, V., and Onaindia, E., *Temporal Data Representation and Reasoning in REAKT*, AI Communications, vol. 7, no. 3, pp. 175-202, 1994.

[4] Bauer, B., Müller, J. P., and Odell, J., *Agent UML:A Formalism for Specifying Multiagent Interaction,* International Journal of Software Engineering and Knowledge Engineering (IJSEKE), vol. 11, no. 3, 2001.

[5] Baümer, G., Breugst, M., Choy, S., and Magedanz, T.:*Grasshopper: a universal agent platform based on OMG MASIF and FIPA standards*. Informe. IKV ++ Technologies. 2000

[6] Bellifemine, F., Poggi, A. y Rimassa, G.: *JADE: a FIPA2000 compliant agent development environment*. Actas de conferencia. Proceedings of the fifth international conference on Autonomous agents, ACM. 2001.

[7] Bradshaw, J. M.: *KAoS: An Open Agent Architecture Supporting Reuse, Interoperabiliby, and Extensibility*. Actas de conferencia. Knowledge Acquisition Workshop (KAW). 1996.

[8] Bratman, M. E.: *Intentions, Plans, and Practical Reason*. Libro completo. Harvard University Press. 1987.

[9] Bratman, M. E., Israel, D., and Pollack, M., *Plans and Resource-bounded Practical Reasoning*, Journal of Computational Intelligence, vol. 4, no. 4, pp. 349-355, 1988.

[10] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., and Treur, J., *DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework*, International Journal of Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, 1997.

[11] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. y Mylopoulos, J.: *A Knowledge Level Software Engineering Methodology for Agent Oriented Programming*. Actas de conferencia. Proceedings of the fifth international conference on Autonomous agents, ACM. 2001.

[12] Breugst, M. y Magedanz, T.: *On the Usage of Standard Mobile Agent Platforms in Telecommunication Environments* . Actas de conferencia. 5th Int. Conference on Intelligence in Services and Networks, Springer Verlag. LNCS 1430. 275-286.1998.

[13] Brooks, R. A., *How to build complete creatures rather than isolated cognitive simulators*, en *Architectures for Intelligence***.** Lawrence Erlbaum Associates, 1991a, pp. 225-239.

[14] Brooks, R. A.:*Intelligence Without Reason*. Informe. Massachusets Institute of Technology, Artificial Ingelligence Laboratory. 1991b

[15] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J. y Massonet, P.: *Agent Oriented Analysis using MESSAGE/UML*. Actas de conferencia. Springer Verlag. LNCS 2222. 2001. pp. 119-135

[16] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J., and Massonet, P.:*Eurescom P907: MESSAGE - Methodology for Engineering Systems of Software Agents*. http://www.eurescom.de/~public-webspace/P900-series/P907/index.htm

[17] Carnegie Mellon:*Unicon*. http://www-2.cs.cmu.edu/People/UniCon/

[18] Carriero, N. and Gelernter, D., *Linda in Context*, Communications of the ACM, vol. Volume 32, no. Issue 4, pp. 444-458, 1989.

[19] Carver, N. and Lesser, V. R.:*The Evolution of Blackboard Control Architectures*. Informe. Department of Computer Science, University Massachusetts. 1992

[20] Chang, K.-H., Day, W. B. y Phiphobmongkol, S.: *An agent-oriented multiagent planning system*. Actas de conferencia. ACM conference on Computer Science, ACM Press New York, NY, USA. 107-114.1993.

[21] Coffman, E. G., Elphick, M. J., and Shoshani, A., *System deadlocks*, ACM Computing Surveys, vol. 3, no. 2, 1971.

[22] Collis, J. C. and Ndumu, D. T.:*The Role Modelling Guide*. Informe. Applied Research and Technology, BT Labs. 1999

[23] Corkill, D. D., *Blackboard Systems*, AI Expert, vol. 6, no. 9, pp. 40-47, 1991.

[24] Cost, R. S., Chen, Y., Finin, T., Labrou, Y., and Peng, Y., *Using Colored Petri Nets for Conversation Modeling*, en *Issues in Agent Communication***.** Springer Verlag, 2000.

[25] Davila Quintero, J. A.: *Agents in Logic Programming*. Tesis doctoral. University of London. 1997.

[26] De Giacomo, G., Lespérance, Y. y Levesque, H. J.: *ConGolog, a concurrent programming language based on the situation calculus*. Artificial Intelligence. Vol. 121 pp. 109-169.2000.

[27] Decker, K. S., *Task Environment Centered Simulation*, en *Simulating Organizations: Computational Models of Institutions and Groups*. AAAI Press/MIT Press, 1996.

[28] Decker, Keith S. and Lesser, Victor R.:*Designing a Family of Coordination Algorithms*. Informe. Department of Computer Science, University of Massachusetts. 1995

[29] Decker, S. Keith:*Environment Centered Analysis and Design of Coordination Mechanisms*. Informe. Department of Computer Science, University of Massachusetts. 1995

[30] DeLoach, S.: *Analysis and Design using MaSE and agentTool*. Actas de conferencia. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conferece (MAICS). 2001.

[31] DeLoach, S. A., Wood, M., and Sparkman, C. H., *Multiagent Systems Engineering*, The International Journal of Software Engineering and Knowledge Engineering, vol. 11, no. 3, June2001.

[32] Demazeau, Y.: *From cognitive interactions to collective behaviour in agent-based systems*. Actas de conferencia. European Conference on Cognitive Science. 1995.

[33] Departamento de Inteligencia Artificial (UPM):*PILLOW*. http://www.clip.dia.fi.upm.es/miscdocs/pillow/pillow.html

[34] Depke, R., Heckel, R. y Küster, J. M.: *Improving the Agent-Oriented Modeling Process by Roles*. Actas de conferencia. Proceedings of the fifth international conference on Autonomous agents, ACM. 2001.

[35] Durfee, E. H., Lesser, V. R., and Corkill, D., *Trends in Cooperative Distributed Problem Solving*, IEEE Transactions on Knowledge and Data Engineering, 1989.

[36] Eurescom P815:*Communications Management Process Integration Using Software Agents*. http://www.eurescom.de/public/projects/P800-series/p815/default.asp

[37] Ferber, J.: *Multi-Agent Systems*. Libro completo. Addison-Wesley. 1999.

[38] Ferber, J. y Gutknecht, O.: *A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems*. Actas de conferencia. Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), IEEE CS Press. 1998.

[39] Fikes, R. y Nilsson, J.: *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*. Artificial Intelligence. Vol. 2 no. 3/4,1971.

[40] Finin, T., Fritzson, R., McKay, D. y McEntire, R.: *KQML as an Agent Communication Language*. Actas de conferencia. 3rd International Conference on Information and Knowledge Management (CIKM94), ACM Press. 1994.

[41] FIPA:*Foundations for Intelligent Physical Agents*. http://www.fipa.org

[42] FIPA:*ACL Message Structure Specification*. http://www.fipa.org

[43] FIPA:*Agent Message Transport Service Specification*. http://www.fipa.org

[44] Fisher, M.: *Concurrent METATEM Processes -- A Language for Distributed AI*. Actas de conferencia. Proceedings of European Simulation Multiconference , SCS Press. 1991.

[45] Fisher, M., *Representing and Executing Agent-Based Systems*, en *Intelligent Agents*. Springer Verlag, 1995.

[46] Flake, S. y Geiger, C.: *CASA - Structured Design of a Specification Language for Intelligent Agents*. Actas de conferencia. Asian Computing Science Conference. 1999.

[47] Franklin, S. y Graesser, A.: *Is it an Agent, or Just a Program? A Taxonomy for Autonomous Agents*. Actas de conferencia. Agent Theories, Architectures, and Languages. 1996.

[48] Gamma, E., Helm, R., Johnson, R. y Vlissides , J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Libro completo. Addison Wesley Professional Computing Series. 1995.

[49] Garijo, F., Gomez-Sanz, J. J. y Massonet, P.: *Multi-Agent System Organization. An Engineering Perspective*. Actas de conferencia. MAAMAW 2001. Por publicar.

[50] Genesereth, M. R., *Software Agents*, Communications of the ACM, 1994.

[51] Gomez-Sanz, J. J., Garijo, F. y Pavon, J.: *Intelligent Interface Agents Behaviour Modelling*. Actas de conferencia. Mexican International Congress of Artificial Intelligence 2000, Springer Verlag. LNAI 1793. 2000. pp 598-609.

[52] Gomez-Sanz, J. J., Pavon, J. y Garijo, F.: *Meta-modelling of Multi-Agent Systems*. Actas de conferencia. ACM. SAC 2002 ACM. 2002. pp. 37 – 41.

[53] Gómez-Sanz, J. J.: *Termostatos y Agentes*. Actas de conferencia. Simposio Español de Informática Distribuida. 2000. pp. 21- 26.

[54] Hendler, J., Tate, A., and Drummong, M., *AI Planning: Systems and Techniques*, AI Magazine, vol. 11, no. 2, pp. 61-78, 1990.

[55] Huhns, M. S., *Multiagent Systems and Societies of Agents*, en *Multiagent Systems* . MIT Press, 2000.

[56] IBM:*Agent Building and Learning Environment (ABLE)*. http://www.alphaworks.ibm.com/tech/able

[57] Iglesias, C.: *Definicion de una metodología para el desarrollo de Sistemas Multi-Agente*. Tesis doctoral. Departamento de ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid. 1998.

[58] Iglesias, C., Mercedes Garijo, M., Gonzalez, J. C., and Velasco, J. R., *Analysis and design of multiagent systems using MAS-CommonKADS*, en *Intelligent Agents IV* . LNAI Volume 1365 ed. SpringerVerlag: Berlin, 1998.

[59] International Telecommunication Union:*ITU-120: Formal Description Techniques (FDT): Message Sequence Chart*. Informe. 99a

[60] International Telecommunication Union:*ITU 100:Formal Description Techniques (FDT)- Specification and Description Language (SDL)*. Informe. 99b

[61] Jacobson, I., Booch, G. y Rumbaugh, J.: *El Proceso Unificado de Desarrollo de Software*. Libro completo. Addison Wesley. 303-3792000.

[62] Jacobson, I., Rumbaugh, J. y Booch, G.: *The Unified Software Development Process*. Libro completo. Addison-Wesley. 1999.

[63] Jennings N. and J., C., *Towards a Social Level Characterisation of Socially Responsible Agents*, IEEE Proceedings on Software Engineering, vol. 144 (1) pp. 11-25, 1997.

[64] Kendall, E.: *A Methodology for Developing Agent Based Systems for Enterprise Integration*. Actas de conferencia. IFIP Working Conference of TC5 Special Interest Group on Architectures for Enterprise Integration. 1995.

[65] Kendall, E.: *Agent Roles and Role Models*. Actas de conferencia. Intelligent Agents for Information and Process Management. 1998.

[66] Kinny, D. and Georgeff, M., *Modelling and Design of Multi-Agent Systems*, Springer Verlag, 1997.

[67] Kinny, D., Georgeff, M., and Rao, A.:*A Methodology and Modelling Technique for Systems of BDI Agents*. Informe. 1997

[68] Laird, J. E., Bates Congdom, C., and Coulter, K. J.:*The SOAR's users manual v.8.2*. Informe. 1999

[69] Laird, J. E., Newell, A. y Rosenbloom, P. S.: *SOAR: an architecture for general intelligence*. Artificial Intelligence. Vol. 33 no. 1, pp. 1-64.1987.

[70] Langley, B., Paolucci, M. y Sycara, K.: *Discovery of Infrastructure in Multi-Agent Systems*. Actas de conferencia. Agents 2001 Workshop on Infrastructure for Agents, MAS, and Scalable MAS. 2001.

[71] Lespérance, Y., Levesque, H. J., Lin, F. y Marcu, D.: *Foundations of a Logical Approach to Agent Programmaing*. Actas de conferencia. Springer-Verlag. Lecture Notes in Artificial Intelligence. 1996.

[72] Lyytinen, K. S. y Rossi, M.: *METAEDIT+ --- A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment*. Actas de conferencia. Springer-Verlag. LGNS#1080.1999.

[73] MAGMA:*MAGMA Research Group*. http://www-leibniz.imag.fr/MAGMA/

[74] Malone, T. W. and Crowston, K., *The Interdisciplinary Study of Coordination*, ACM Computing Survey, vol. 26, no. 1, pp. 87-119, Mar.1994.

[75] Maturana, H.:*Autopoiesis, Structural Coupling and Cognition*. International Society for the System(s) Science(s) (ISSS). http://www.isss.org/maturana.htm

[76] McCabe, F. G. and Clark, K. L., *April - Agent PRocess Interaction Language*, en *Intelligent Agents*. Springer Verlag, 1995.

[77] Microsoft:*Distributed Component Object Model*. http://www.microsoft.com

[78] MultiAgent and Cooperative Robotics Lab:*AgentTool 1.8.3 User's manual*. http://www.cis.ksu.edu/~sdeloach/ai/mase.htm

[79] Myers, B. A. y Rosson, M. B.: *Survey on user interface programming*. Actas de conferencia. ACM. 195-202.1992.

[80] Newell, A., *The knowledge level*, Artificial Intelligence, vol. 18 pp. 87-127, 1982.

[81] Newell, A. and Simons, H. A., *GPS: A program that simulates Human Thought*, en *Computers and Thought*. Mc Graw Hill, 1963.

[82] Nowostawski, M., Purvis, M y Cranefield, S.: *Modelling and Visualizing Agent Conversations*. Actas de conferencia. 2001.

[83] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C., *ZEUS: A Toolkit for Building Distributed Multi-Agent Systems*, Applied Artificial Intelligence Journal, vol. 1, no. 13, pp. 129-185, 1999.

[84] OMG:*CORBA 2.4.2 Specification* . http://www.omg.org

[85] OMG:*MOF. Meta Object Facility (specification)*. Informe. 3-4-2000b

[86] OMG:*Task and Session CBOs Specification*. Informe. 1-4-2000c

[87] OMG:*Unified Modeling Language Specification. Version 1.3*. http://www.omg.org

[88] Papadopoulos, G. A. y Arbab, F.: *Coordination Models and Languages*. Actas de conferencia. Coordination Languages.1998.

[89] Pattison, H. e. a., *Instantiating Descriptions of Organizational Structures*, Pitman Publishing/Morgan Kaufman Publishers, 1987, pp. 59---96.

[90] Pressman, R. S.: *Software Engineering: A Practitioner's Approach*. Libro completo. McGraw-Hill Series in Software Engineering and Technology. McGraw-Hill, Inc. 1982.

[91] PSI3:*Personalized Service Integration Using Software Agents*. http://www.psi3.org/index.htm

[92] Rao, A., *AgentSpeak(L): {BDI} Agents Speak Out in a Logical Computable Language*, en *Agents Breaking Away*. Springer Verlag, 1996.

[93] Ribeiro, A. y Demazeau, Y.: *A Dynamic Interaction Model for Multi-Agent Systems*. Actas de conferencia. II Iberoamerican Workshop on D.A.I. and M.A.S. 1998.

[94] Rich, E. y Knight, K.: *Artificial Intelligence*. Libro completo. McGraw-Hill. 1990.

[95] Ricordel, P. M.: *Programmation Orientée Multi-Agents , Développement et Déploiement de Systèmes Multi-Agents Voyelles*. Tesis doctoral. INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE. 2001.

[96] Rosenschein, S. J. y Kaelbling, L. P.: *A Situated View of Representation and Control*. Artificial Intelligence. Vol. 73 no. 1/2, pp. 149-173.1995.

[97] Russell, S. y Norvig, P.: *Artificial Intelligence: a modern approach*. Libro completo. Prentice Hall. 1995.

[98] Sacerdoti, E.: *A Structure for Plans and Behaviours*. Artificial Intelligence. 1977.

[99] Shoham, Y., *Agent Oriented Programming*, Artificial Intelligence, vol. 60 pp. 51-92, 1993.

[100] Shoham, Y. and Tennenholtz, M., *On the emergence of social conventions: Modeling, analysis, and simulations*, Artificial Intelligence, vol. 94, no. 12, pp. 139-166, 1997.

[101] Sichman, J. S., Conte, R., Demazeau, Y. y Castelfranchi, C.: *A social reasoning mechanism based on dependence networks*. Actas de conferencia. 1994.

[102] Singh, M. P.: *Towards a formal theory of Communication for Multiagent Systems*. Actas de conferencia. International Joint Conference on Artificial Intelligence (IJCAI). 1991.

[103] Smith, R. G., *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*, IEEE Transactions on Computers, vol. C-29(12) pp. 1104-1113, 1980.

[104] Spivey, J. M.: *The Z Notation: a reference manual*. Libro completo. Prentice Hall. 1992.

[105] Sun Microsistems:*Remote Method Invocation Specification*. http://java.sun.com

[106] Sycara, K., Klusch, M., idof, S., and Lu, J., *Dynamic Service Matchmaking among Agents in Open Information Environments*, Journal ACM SIGMOD Record , Special Issue on Semantic Interoperability in Global Information Systems, 1999.

[107] Tanenbaum, A. S. y WoodHull, A. S.: *Sistemas Operativos: Diseño e implementación*. Libro completo. Prentice Hall. 1997.

[108] Tansley, D. S. W. y Hayball, C. C.: *Knowledge Based systems Analysis and Design a KADS developer's handbook*. Libro completo. Prentice Hall. 1993.

[109] Van Dyke Parunak, H., *Manufacturing experience with the contract net*, en *Distributed Artificial Intelligence***.** 1987, pp. 285-310.

[110] Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., and Blythe, J., *Integrating Planning and Learning: The PRODIGY Architecture*, Journal of Experimental and Theoretical Artificial Intelligence, 1995.

[111] Wagner, T. y Horling, B.: *The Struggle for Reuse and Domain Independence: Research with TAEMS, DTC and JAF*. Actas de conferencia. Proceedings of the 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS. 2001.

[112] Walker, A. y Wooldridge, M.: *Understanding the emergence of conventions in multi-agent systems*. Actas de conferencia. AAAI Press. 1995.

[113] Weld, D. S., *Recent Advances in AI Planning*, AI Magazine, vol. 20, no. 2, pp. 93-123, 1999.

[114] WHITAKER, R.:*Self-Organization, Autopoiesis, and Enterprises*. ACM SIGGROUP. http://www.acm.org/siggroup/auto/Main.html

[115] Wood, M. y DeLoach, S.: *Developing Multiagent Systems with agentTool*. Actas de conferencia. ATAL 2000. LNAI 1986. Castelfranchi, C. and Lespérance, Y.2000.

[116] Wooldridge M. y Jennings N.R.: *Pitfalls of Agent-Oriented Development*. Actas de conferencia. Proceedings of the Second International Conference on Autonomous Agents. 385-391.1998.

[117] Wooldridge, M., Jennings, N. R., and Kinny, D., *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems, vol. 15 2000.

[118] Wooldridge, M. J. y Jennings, N. R.: *Agent Theories, Architectures, and Languages: A Survey*. Actas de conferencia. Springer-Verlag. Wooldridge, M. and Jennings, N. R.1995.

[119] Wooldridge, M. J.: *The Logical Modelling of Computational Multi-Agent Systems*. Libro completo. 1992.

[120] Workflow Management Coalition:*The Workflow Management Coalition Specification: Workflow Management Coalition Terminology & Glossary*. Informe. 1999

[121] Zambonelly, F., Wooldridge M., and Jennings N.R., *Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems*, International Journal of Software Engineering and knowledge Engineering, 2000.

[122] Zilouchian, A., *Fundamentals of Neural Networks*, en *Intelligent Control Systems* . CRC Press, 2000, pp. 17-38.

Anexo I. PROCESO DE PARAMETRIZACIÓN DE PLANTILLAS

```
:-consult(pillow).
file_to_string(F,S):-
        read_file_to_codes(F,S,access(read)).

templateTerm(T,[]).
templateTerm(T,Vs):-
    nonvar(Vs),
        nonvar(T),
        Vs= [Key = Value|V1s],
        member(Key = Value,T),
        templateTerm(T,V1s).

templateTerm(T,[X|Vs]):-
    nonvar(X),
        templateTerm(T,Vs).
duplicated(P,D).
templateTermWithRow(X,[]).
templateTermWithRow(duplicated(P,D),[R|Rs]):-
        copy_term(duplicated(P,D),DP1),
        templateTerm(D,R),

        output_html(P),
        templateTermWithRow(DP1,Rs).
templateRepeatWithRows([],_,_,_).
templateRepeatWithRows([env(repeat,[id=X1],X2)|Rs],D,H,O):-
        template(env(repeat,[id=X1],X2),D,O),
        templateRepeatWithRows(Rs,D,H,O).

templateRepeatWithRows([X|Xs],D,Head,Others):-
        templateTermWithRow(duplicated(X,D),[Head]),
        templateRepeatWithRows(Xs,D,Head,Others).


template(env(repeat,[],X),D,Rows):-
        nonvar(X),
        templateTermWithRow(duplicated(X,D),Rows).

segment([],R,R).
segment([[X,Xs]|Rs],[[X,Ys]|Ls],R):-
        append(Xs,Ys,Zs),
        segment(Rs,[[X,Zs]|Ls],R).

segment([[X,Xs]|Rs],[[Y,Ys]|Ls],R):-
        X\=Y,
        append([[X,Xs]],[[Y,Ys]|Ls],Zs),
        segment(Rs,Zs,R).
```

```
loopRowData(Ls,Rs):-
        sort(Ls,L1s),
        maplist(transform(append([id=Y|_L1s],[id=Z|_L2s],L2s),L2s,[[id=Y|_L1s],[[id=Z|_L2s]]]),L1s,[
FRow|Rows]),
        segment(Rows,[FRow],Rs).

template(env(repeat,[id=Y],X),D,Rows):-
        nonvar(X),
    % Se utiliza esto en lugar del member para no ligar variables libres
    not(sublist(filter(X11,(nonvar(X11),X11=env(repeat,X13,X12))),X,[])),
        sublist(member(id=Y),Rows,Filtered),
        loopRowData(Filtered,Loops),
        forall(member([Head,Others],Loops),
        (copy_term(dup(X,D),dup(X1,D1)),templateTerm(D1,Head),templateRepeatWithRows(X1,D1,H
ead,Others))).
template(env(repeat,[id=Y],X),D,Rows):-
        nonvar(X),
        sublist(member(id=Y),Rows,Filtered),
        copy_term(dup(X,D),dup(X1,D1)),
        templateTermWithRow(duplicated(X1,D1),Filtered).

template(T,D,[R|Rows]):-
        copy_term(duplicated(T,D),duplicated(T1,D1)),
        templateTerm(D1,R),
        output_html(T1).



templateFile(F,Rows):-
        file_to_string(F,S),
        html_template(S,P,D),!,
        forall(member(E,P),
        (!,template(E,D,Rows))).
```

Anexo II. BNF DE AGENT0

<program>::=timegrain := <time>

CAPABILITIES := (<action> <mntlcond>)*

INITIAL BELIEFS:=<fact>*

COMMITMENT RULES := <commitrule>*

<commitrule>::=(COMMIT <msgcond> <mntlcond> (<agent> <action>)*)

<msgcond>::=<msgconj>|(OR <msgconj>*)

<msgconj> ::=<msgpattern> | (AND <msgpattern>*)

<msgpattern> ::=(B <fact> | ((CMT <agent>) <action>)|(NOT <mntlpattern>)

<mntlcond> ::= <mntlconj> | (OR <mntlconj>*)

<mntlconj> ::= <mntlpattern> | (AND <mntlpattern>*)

<mntlpattern> ::= (B <fact>) | ((CMT <agent> <action>) | (NOT <mntlpattern>)

<action>::=(DO <time> <privateaction>) | (INFORM <time> <agent> <fact>) | (REQUEST <time> <agent> <action>) |(UNREQUEST <time> <agent> <action>) | (REFRAIN <action>) | (IF <mntlconf> <action>)

<fact> ::= (<time> (<predicate> <arg>*))

<time>::=<integer> | now | <time-constant> | (+<time> <time>) | (- <time> <time>) | (x <integer> <time>)

<time-constant> ::= m|h|d|y

<agent> ::= <cadena_alfanumérica> | <variable>

<predicate> ::= <cadena_alfanumérica>

<arg> ::=<cadena_alfanumérica>|<variable>

<variable> ::= ? <cadena_alfanumérica> | ? ! <cadena_alfanumérica>

Anexo III. GLOSARIO

MOF. Meta Object Facilities.

SMA. Sistema Multi-Agente

MAS. Multi-Agent System

ISOA. Ingeniería del Software Orientada a Agentes

IA. Inteligencia Artificial

AI. Artificial Intelligence

DIA. Distributed Artificial Intelligence.

RUP. Rational Unified Process

UML. Unified Modelling Language

XML. eXtended Modelling Language

OMG. Object Management Group

FIPA. Foundation for Intelligent Physical Agents

CASE. Computer Aided Software Engineering.

POO. Programación Orientada a Objetos

OCL. *Object Constraint Language*

CSP. Communicating Sequential Processes

BDI. Believes Desires Intentiones

BNF. Backus-Naur Formalism

GRASIA GRupo de Agentes Software del departamento de sistemas Informáticos y progrAmación