Modelado Conceptual de Sistemas Cooperativos en base a Patrones en AMENITIES

Universidad de Granada

Tesis doctoral elaborada para optar al título de

Doctor en Informática

Presentada por:

José Luis Isla Montes

joseluis.isla@uca.es

Dirigida por:

Dr. D. Francisco Luis Gutiérrez Vela y Dr. D. José Luis Garrido Bullejos

Departamento de Lenguajes y Sistemas Informáticos

UNIVERSIDAD DE GRANADA

Granada, 2007

Editor: Editorial de la Universidad de Granada
Autor: José Luis Isla Montes
D.L.: Gr. 3027 - 2007
ISBN: 978-84-338-4725-6

A mi madre,
porque se lo debo todo

A mi mujer,
por ser tan maravillosa, paciente y hacerme tan afortunado

A mi hijo,
por su sonrisa

A la memoria de mi padre y de mi abuela,
porque este momento habría sido de enorme felicidad para ellos

A Dino, mi perro,
por sacarme a pasear diariamente

Índice General

Capítulo V Construcción del Modelo Cooperativo de AMENITIES en base a Patrones

Capítulo VI Casos de Estudio

Capítulo VII Conclusiones y Trabajos Futuros

Índice de Figuras y Tablas

Figuras

Tablas

Resumen

La demanda de sistemas para dar soporte a la comunicación, coordinación y/o colaboración efectiva de grupos de individuos u organizaciones durante la realización de tareas compartidas crece incesantemente. Sin embargo, por su propia naturaleza, esta clase de sistemas, los cuales conocemos como *Sistemas Cooperativos*, son bastante complejos. Su desarrollo requiere de métodos y técnicas de modelado que nos ayuden a especificar, analizar y, en definitiva, comprender cómo la gente trabaja en grupo.

Con este objetivo nuestro grupo de investigación ha creado la metodología AMENITIES, especializada en el análisis, diseño y desarrollo de sistemas cooperativos. El núcleo central de dicha metodología está constituido por el llamado *Modelo Cooperativo*, el cual es un modelo conceptual que nos ayuda a comprender el dominio del problema asociado con el sistema objeto y que está compuesto por un conjunto de modelos interrelacionados de comportamiento y de tareas. Sin embargo, a pesar del potencial que nos ofrece AMENITIES, cada vez que modelamos un nuevo sistema tenemos que partir prácticamente de cero.

A lo largo de nuestra experiencia en el modelado conceptual de tales sistemas hemos podido comprobar existencia de conceptos y escenarios comunes que deben ser especificados una y otra vez. Capturar dichas abstracciones, representarlas y documentarlas para facilitar su posterior reutilización en diversos proyectos es de un gran valor, tanto para el proceso de modelado como para la propia especificación.

En esta tesis proponemos el uso de patrones para el modelado conceptual (*patrones conceptuales*) de sistemas cooperativos en el marco de la metodología AMENITIES. Estos patrones nos facilitan la comunicación, especificación y reutilización de aquellas abstracciones clave que son recurrentes en el dominio del problema.

Tras analizar el estado del arte y destacar las limitaciones que para nuestro propósito presentan las distintas aproximaciones que se han realizado al modelado de patrones de software, proveemos un perfil UML, el

cual denominamos *PMP* (Pattern Modelling Profile o, en español, Perfil para Modelado de Patrones), que nos permite representar tanto la vista interna como externa de un patrón, independientemente de su tipo. De esta forma, a partir de un conjunto mínimo de elementos, los patrones se pueden definir de manera simple, intuitiva y fácil de comprender, como plantillas flexibles que representan familias de modelos semejantes (instancias del patrón), las cuales pueden usarse como guía para la creación y/o descripción de modelos, o partes de éstos, por medio de la correspondencia (ligadura) de los elementos del patrón con los elementos que forman sus instancias. Este perfil es usado por COMO-UML, notación propia de la metodología AMENITIES y que está basada en UML, para el modelado de los patrones conceptuales que usamos durante la construcción del Modelo Cooperativo de un sistema en el ámbito de dicha metodología, así como de cualquier otro tipo de patrón aplicable en fases posteriores.

En base a nuestra praxis recopilamos una colección de patrones útiles durante el modelado conceptual de los sistemas cooperativos. Estos patrones los modelamos con ayuda del perfil *PMP* y los describimos a partir de una plantilla uniforme que facilita su estudio, comparación y aplicación.

Establecemos un catálogo inicial que organiza y clasifica los patrones de la colección, a la vez que estructura el dominio del problema, favoreciendo la selección del patrón más adecuado en cada momento. Adicionalmente, creamos una red que interconecta los distintos patrones del catálogo en base a criterios que redundan en un enriquecimiento de nuestra capacidad de decisión y, por tanto, de selección. A partir de estos elementos determinamos un método flexible que marca las pautas generales para la selección y aplicación de los patrones de nuestro catálogo.

Con ánimo de allanar en la medida de lo posible el camino que va desde el dominio del problema al dominio de la solución, hacemos una primera aproximación a algunas de las potenciales conexiones que pueden establecerse entre los patrones conceptuales que hemos propuesto y algunos patrones de diseño que aparecen en la literatura. También ponemos de relieve los principales requisitos que debería satisfacer una herramienta para el modelado basado en patrones de acuerdo con nuestra propuesta.

Finalizamos el desarrollo de la tesis con la exposición de un par de casos de estudio a través de los cuales hemos podido comprobar cómo la

aplicación de patrones conceptuales agiliza y optimiza la construcción del Modelo Cooperativo de dos sistemas diferentes, facilitando la detección de las abstracciones clave recurrentes en el dominio del problema, permitiendo su reutilización en contextos concretos, empleando un vocabulario común que nos permite comunicar y razonar sobre dichas abstracciones, aumentando la comprensión, comunicación y mantenimiento de los modelos, así como de la documentación en general, y reduciendo considerablemente el tiempo de modelado.

Palabras clave

Palabras clave: *Sistemas Cooperativos, Modelado Conceptual, Patrones de Software, Especificación de Requisitos, Groupware, CSCW.*

Datos de la tesis

Título de la tesis:	*Modelado Conceptual de Sistemas Cooperativos en base a Patrones en AMENITIES*
Presentada por:	José Luis Isla Montes
Dirigida por:	Dr. D. Francisco Luis Gutiérrez Dr. D. José Luis Garrido Bullejos
Universidad:	Universidad de Granada
Departamento:	Lenguajes y Sistemas Informáticos
Programa de doctorado:	Especificación y Desarrollo de Software
Depósito:	28 de noviembre de 2007
Defensa:	Por determinar

Agradecimientos

En primer lugar quisiera agradecer a mis dos directores, y sobre todo amigos, Francis y José Luis, la encomiable labor que han desarrollado para que esta tesis llegue a buen puerto, pese a los baches encontrados en el camino. Les agradezco enormemente el esfuerzo que han realizado.

Igualmente quisiera reconocer el empuje que he recibido por parte de tantas y tantas personas, familiares y amigos, quienes sin su cariño, comprensión y ánimo habría sido imposible producir esta obra.

Especialmente, agradezco a Pepa, mi mujer, su amor, confianza, apoyo, estímulo, motivación y paciencia demostrada en todo momento.

Por último, y principalmente, le debo a Darío, mi recién nacido hijo, la finalización de esta tesis. ¡No estoy dispuesto a perderme ni una de sus sonrisas por este motivo!.

Capítulo I Introducción

Contenido

<<La ignorancia afirma o niega rotundamente; la ciencia duda>>

—Voltaire (1694-1778)

Filósofo y escritor francés

1. Motivación

Los seres humanos vivimos en sociedad y a lo largo de nuestra vida formamos parte de diferentes grupos sociales con los que nos relacionamos e identificamos. Ejemplos de algunos de éstos son la familia, los amigos, la empresa o departamento en el que trabajamos, nuestro grupo de investigación, etc.

Nos agrupamos, entre otros motivos, para poder afrontar cooperativamente muchos de los problemas que surgen a nuestro alrededor, los cuales serían intratables o se resolverían con mayor dificultad de no ser así. Sin embargo, para alcanzar con éxito dicho objetivo, es necesario

procurar las condiciones adecuadas que propicien una buena comunicación, coordinación y colaboración entre los miembros del grupo.

Desde hace un par de décadas, investigadores procedentes de diferentes ámbitos (informática, psicología, antropología, economía, sociología, etc.) estudian cómo las personas trabajan en grupo y la manera en que la tecnología puede ayudarles, lo que ha dado lugar a un prometedor campo de investigación conocido como *CSCW* (**C**omputer-**S**upported **C**ooperative **W**ork) o, en español, *Trabajo Cooperativo Soportado por Ordenador*. Sin embargo, la limitada accesibilidad y capacidad de las tecnologías de la comunicación e información, junto con la dificultad que encierra la comprensión y especificación de los requisitos reales que caracterizan a los llamados *sistemas cooperativos*[1], han supuesto un freno importante en su desarrollo.

No obstante, durante estos últimos años, estamos asistiendo a un espectacular avance y expansión de las telecomunicaciones[2] que está favoreciendo la implantación y demanda de este tipo de sistemas (videoconferencia, aplicaciones compartidas, sistemas de mensajería, soporte a la decisión de grupos, etc.). De este modo, la concepción tradicional del ordenador, como herramienta para el trabajo personal, se está transformando y dando paso a una visión mucho más amplia de éste, atribuyéndole nuevas capacidades, como la de facilitar la interacción[3] entre las personas y, en particular, el trabajo en grupo.

Ejemplo de esto son las numerosas aplicaciones colaborativas incluidas en la llamada Web 2.0 [O´Reilly, 2005]. Dentro de esta corriente se incluyen aplicaciones de carácter social en las que un grupo de personas se unen para la realización de actividades de carácter colaborativo (compartir y

[1] Sistemas informáticos que soportan el trabajo en grupo, con objeto de facilitar la comunicación, coordinación y compartición de recursos entre sus miembros. En este documento emplearemos los términos *Sistema Cooperativo* y *Sistema CSCW* indistintamente.

[2] La progresiva mejora de los servicios basados en internet y la telefonía móvil, así como los grandes intereses económicos que mueven a muchas compañías de este sector, están allanado enormemente el camino.

[3] Conocida como interacción persona-ordenador-persona (IPOP).

definir términos para una enciclopedia global, localizar fotografías en su posición geográfica, dar respuesta a preguntas complejas, etc.).

Sin embargo, para hacer frente a esta demanda tecnológica, es necesario disponer de los medios adecuados. Por una parte, los sistemas cooperativos son inherentemente complejos y su desarrollo requiere de métodos y técnicas de modelado con capacidad para especificar y analizar fielmente sus requisitos reales. En este sentido, como establece Grudin [1993], los protocolos sociales y las actividades de grupo deberían tenerse muy en cuenta para un diseño con éxito, de ahí la importancia que cobra una aproximación multidisciplinar para poder abordar convenientemente el desarrollo de este tipo de sistemas. Por otro lado, para atender las exigencias del mercado, hay que buscar mecanismos que faciliten su desarrollo rápido y fiable.

2. Planteamiento del problema

Con objeto de satisfacer la demanda que la sociedad reclama en torno al estudio y desarrollo de los sistemas CSCW, en el seno de nuestro grupo de investigación[4] se ha creado una metodología para el análisis, diseño y desarrollo de sistemas cooperativos denominada AMENITIES (**A ME**thodology for a**N**alysis and des**I**gn of coopera**TI**ve syst**E**m**S**) [Garrido, 2003; Garrido et al., 2002; Gutiérrez et al., 2002].

El núcleo central de dicha metodología está constituido por el llamado modelo cooperativo de AMENITIES, compuesto por un conjunto de modelos de comportamiento y de tareas. El resultado es un modelo conceptual cuyo propósito es la descripción del sistema cooperativo, independientemente de su implementación. De esta forma se proporciona una mejor comprensión del dominio del problema, dada la complejidad que caracteriza a esta clase de sistemas.

A pesar del potencial que nos brinda AMENITIES, cada vez que modelamos un nuevo sistema hay que partir prácticamente de cero. Sin embargo, en fases tempranas de modelado encontramos conceptos y

[4] Grupo de Especificación, Desarrollo y Evolución de Software (GEDES), TIC-164.

escenarios comunes que hemos de representar reiteradamente. Capturar dichas situaciones, representarlas y documentarlas para facilitar su posterior reutilización en otros proyectos sería de un gran valor, tanto desde el punto de vista del proceso como de la propia especificación.

Desde hace aproximadamente una década, los patrones de software se han erigido como un valioso instrumento para la descripción y reutilización del conocimiento experto empleado durante el proceso de ingeniería del software. Sin embargo, su explotación no ha sido la misma en todos los campos de aplicación y etapas de desarrollo. Al contrario de lo que sucede en otras fases, como puede ser la de diseño (*patrones de diseño*), hasta ahora han sido muy pocos los estudios relacionados con el uso de patrones durante la fase de modelado conceptual (*patrones conceptuales*) de sistemas software, siendo prácticamente inexistentes los dedicados a la especificación conceptual de sistemas cooperativos.

Teniendo en cuenta la complejidad inherente de estos sistemas, así como la relevancia que las decisiones tomadas en fases tempranas de desarrollo tienen sobre el producto final, formulamos como hipótesis de partida que el uso de patrones durante la etapa de modelado conceptual y su integración en una metodología orientada hacia la especificación y desarrollo de tales sistemas puede proporcionar claros beneficios. Asimismo, los patrones contribuyen en la creación de un vocabulario compartido que va a facilitar el entendimiento entre las personas involucradas en un proyecto de software (analistas, diseñadores, programadores, usuarios finales, clientes, etc.), actuando como una lengua franca [Erickson, 2000a] y favoreciendo el diseño participativo [Dearden et al., 2002].

La hipótesis de la que parte esta tesis es que si el proceso de especificación de sistemas cooperativos con AMENITIES se apoya en el uso de patrones conceptuales específicos del dominio del problema –el trabajo en grupo-, la construcción del modelo cooperativo se agiliza, se facilitan la detección y especificación de los requisitos reales del sistema, y los modelos generados son más fáciles de comprender, mantener y comunicar. Además se sientan las bases para facilitar el paso a una fase de diseño inicial, basada en patrones, mediante el establecimiento de las relaciones oportunas entre patrones conceptuales y patrones de diseño específicos.

El problema planteado en la presente tesis se encuadra dentro del ámbito de la ingeniería de requisitos y, en particular, del modelado conceptual de sistemas cooperativos en base a patrones dentro del marco de la metodología AMENITIES.

3. Metodología de trabajo

A partir de las labores de investigación realizadas dentro del programa de doctorado "Especificación y Desarrollo de Software" del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada, efectuamos un estudio completo sobre los patrones de diseño y su uso durante el desarrollo de software en general. Este estudio dio lugar a un análisis más detallado sobre la problemática de la especificación de patrones, derivando en la creación de un modelo de especificación estructural de patrones, basado en UML, en el contexto de una herramienta de diseño [Isla, 2001, 2002; Isla y Gutiérrez, 2003].

Desde el nacimiento de la metodología AMENITIES [Garrido, 2003], y como consecuencia de la actividad investigadora desarrollada en el marco del proyecto CICYT titulado *Análisis y Modelado de Sistemas Colaborativos (AMENITIES)*[5], nos planteamos la necesidad de integrar en dicha metodología patrones específicos que soporten el modelado conceptual de tales sistemas.

El plan de trabajo que hemos seguido para lograr nuestro objetivo ha consistido fundamentalmente en las siguientes tareas:

1) Reconocer la existencia de situaciones y conceptos que reiteradamente deben ser especificados durante el modelado conceptual de estos sistemas.

   Nuestra experiencia en la construcción del Modelo Cooperativo de diversos sistemas aplicando esta metodología, nos ha permitido constatar la existencia de tales escenarios comunes y demostrar que éstos pueden ser generados a partir de modelos genéricos

[5] Referencia TIN2004-08000-C03-02

reutilizables, los cuales pueden ser capturados y descritos en forma de patrones de software.

2) Buscar una notación adecuada para el modelado de los distintos tipos de patrones a utilizar.

   Una vez examinadas las notaciones existentes sobre modelado de patrones y puesto de relieve sus limitaciones para nuestro propósito, hemos visto conveniente crear una notación universal, sencilla, potente y fácil de aprender que facilite la definición y aplicación de patrones en general. Es universal porque es válida para la representación de cualquier tipo de patrón, independientemente de los elementos de modelado que intervengan. Es sencilla y fácil de aprender porque introduce un conjunto mínimo de elementos de modelado y está basada en un estándar, el lenguaje de modelado UML, sobre el que también está basado el lenguaje COMO-UML, propio de la metodología AMENITIES. Además de por todo lo anterior, es potente porque los patrones son modelados como plantillas flexibles, las cuales representan modelos genéricos totalmente parametrizables, posibilitando la generación de modelos semejantes (las distintas instancias del patrón) que conforman con las restricciones que impone el patrón sobre los elementos de modelado participantes.

   Esta notación nos permite modelar tanto los patrones conceptuales, necesarios para agilizar la construcción de los modelos propuestos por AMENITIES, como aquellos que serán usados, a posteriori, durante la etapa de diseño.

3) Validar la notación propuesta mediante la definición de un conjunto de patrones.

   Hemos representado una familia de patrones aplicables en la construcción de los distintos modelos que componen el Modelo Cooperativo de AMENITIES.

4) Elaborar un esquema para la descripción uniforme de patrones.

Hemos diseñado una plantilla específica para la descripción uniforme de los patrones sugeridos, la cual facilita su estudio, comparación y aplicación en el marco de la metodología AMENITIES.

5) Proyectar la estructura para un catálogo inicial de patrones.

   Sobre la base de este catálogo, hemos clasificado y relacionado nuestros patrones. Naturalmente, esperamos que esta organización vaya evolucionando a medida que aumente nuestra experiencia, así como la de otros autores, en el hallazgo y aplicación de nuevos patrones.

6) Realizar una propuesta metodológica para la construcción del Modelo Cooperativo de un sistema.

   Hemos formulado un método simple para facilitar la selección y aplicación de patrones del catálogo.

7) Construir el Modelo Cooperativo de un sistema concreto con AMENITIES en base a los patrones propuestos en nuestro catálogo.

   Para comprobar la validez de nuestra propuesta, así como su capacidad de integración con la metodología AMENITIES, presentamos un par de casos de estudio consistentes en la construcción en base a patrones de los Modelos Cooperativos correspondientes a un Sistema para Aprendizaje Cooperativo usando Jigsaw y un Sistema para la Gestión de la Cooperación dentro de un Proyecto de Investigación Coordinado.

No obstante, aunque esta fue nuestra planificación inicial, la cual mantiene una secuencia lógica de trabajo, muchas de estas tareas las emprendimos en paralelo debido a que, además de adelantar trabajo y marcar un camino con ciertas garantías de éxito, a menudo nos aportaban los resultados y experiencia necesarios para poder seguir indagando en la tarea que teníamos entre manos.

Podemos decir, por tanto, que el proceso que hemos seguido ha sido iterativo y retroalimentado.

4. Objetivos científicos

En esta tesis se estudia la aplicación de patrones en fases tempranas de modelado de sistemas cooperativos y cómo es posible integrar éstos dentro de la metodología AMENITIES para:

- Facilitar la toma de decisiones y acelerar la construcción del Modelo Cooperativo de AMENITIES en base a la reutilización de patrones conceptuales.
- Hacer más comprensible el modelo generado y mejorar la documentación asociada.

Los objetivos científicos que pretendemos conseguir se pueden concretar en:

- Revisar los trabajos existentes relacionados con la especificación de patrones en general.
- Analizar el estado del arte respecto al uso de patrones en el desarrollo de sistemas cooperativos.
- Extender la metodología AMENITIES para poder soportar la construcción del Modelo Cooperativo en base a patrones conceptuales específicos del dominio. Para ello es necesario:
  - Organizar los patrones con el fin de facilitar su aplicación y gestión.
  - Establecer un formato de descripción uniforme.
  - Ampliar la notación utilizada en AMENITIES, denominada COMO-UML, para poder especificar los patrones como elementos de modelado de primer orden.
- Detectar algunos de los patrones que pueden ser aplicados durante la construcción del Modelo Cooperativo, teniendo en cuenta las distintas vistas que lo componen.

- Sentar las bases para facilitar el paso a una fase de diseño inicial mediante el establecimiento de las relaciones oportunas entre patrones conceptuales y patrones de diseño específicos.

5. Aportaciones

Aunque de manera pormenorizada el capítulo VII, dedicado a las conclusiones, detalla todas las aportaciones que se derivan de esta tesis, en este punto podemos adelantar resumidamente algunas de sus contribuciones principales:

- Una notación que extiende COMO-UML para poder definir y representar adecuadamente los patrones aplicables en los diferentes modelos.

- Un modelo de proceso que guía la selección y aplicación de patrones en las distintas etapas de modelado.

- Un formato uniforme de descripción de patrones que facilita su estudio, comparación, aplicación y uso dentro del equipo de desarrollo.

- Un vocabulario común, compuesto por los nombres de los patrones, que favorece la comunicación y discusión de soluciones y conceptos asociados con los problemas que aparecen.

- La estructura básica de un catálogo de patrones que agiliza la selección del más adecuado en cada momento. Lógicamente, tanto su estructura como su contenido evolucionará a medida que vaya aumentando nuestro conocimiento del dominio.

- Un conjunto de patrones conceptuales detectados en base a nuestra propia experiencia durante el análisis de este tipo de sistemas. Servirá como punto de partida para la construcción de un catálogo que irá incorporando la experiencia de otros autores en ésta y otras fases de desarrollo, complementando así a otros catálogos ya existentes.

6. Estructura de la tesis

En los siguientes apartados hacemos una breve descripción de los capítulos y apéndices que componen esta tesis:

- El presente capítulo básicamente motiva y plantea el problema que abordamos en esta tesis y los objetivos que pretendemos cubrir, así como la metodología que seguimos para ello. También exponemos las convenciones tipográficas que hemos adoptado.

- El Capítulo II introduce los sistemas cooperativos y estudia la problemática de su desarrollo. A continuación, describe someramente la metodología AMENITIES, centrándose en las distintas vistas que componen el Modelo Cooperativo (modelo conceptual del sistema) y su método de construcción. Este método es ilustrado a través de la construcción de un Modelo Cooperativo concreto.

- El Capítulo III da un repaso por el mundo de los patrones de software y su integración en los procesos de software. Examina los beneficios que aporta la reutilización de patrones durante las etapas tempranas de modelado de sistemas software en general y revisa los trabajos que han tratado este asunto hasta la fecha. El capítulo concluye exponiendo la situación actual existente en relación con la aplicación de patrones durante las etapas tempranas de desarrollo de sistemas cooperativos.

- El Capítulo IV constituye el eje central de esta tesis. Proponemos un perfil (profile) UML para el modelado de patrones de software que nos permite modelar y aplicar, además de patrones de diseño, los patrones conceptuales que necesitamos integrar con la metodología AMENITIES para la construcción del Modelo Cooperativo. Si bien, antes hacemos algunas consideraciones sobre el tratamiento que da UML a los patrones y recorremos los principales trabajos existentes sobre modelado de patrones.

- El Capítulo V aborda la construcción del Modelo Cooperativo en base a patrones. Proponemos una estructura para nuestro catálogo y una plantilla específica para la descripción uniforme de los patrones que contiene. Luego, a partir del catálogo definido, planteamos el método

de construcción del Modelo Cooperativo en base a patrones. Especificamos también los requisitos más relevantes que debería satisfacer una herramienta de modelado basado en patrones de acuerdo con nuestra propuesta. Por último, vislumbramos algunas posibles conexiones entre los patrones conceptuales planteados y determinados patrones de diseño que aparecen en la literatura.

- El Capítulo VI muestra dos casos de estudio en los que aplicamos el perfil, el catálogo y el método de construcción del Modelo Cooperativo para dos sistemas particulares.

- Finalmente, el Capítulo VII resume las conclusiones y los trabajos futuros.

- Dentro de los apéndices incorporamos información complementaria, en concreto:

  - El Apéndice A describe someramente la notación COMO-UML utilizada para el modelado de las distintas vistas del Modelo Cooperativo.

  - El Apéndice B contiene algunos ejemplos de patrones que forman parte de nuestro catálogo.

  - El Apéndice C es un glosario de términos en el que definimos aquellos conceptos que consideramos imprescindibles para un correcto seguimiento de la tesis.

  - El Apéndice D muestra una introducción a la nueva sintaxis y semántica que propone UML 2 para los diagramas de actividad. Los diagramas de tareas y subactividades modelados en esta tesis, así como los patrones que facilitan su construcción, están basados en esta nueva notación.

- El último apartado de la tesis incluye la bibliografía utilizada.

7. Convenciones tipográficas y estilo de escritura

Con ánimo de mejorar, en la medida de lo posible, la legibilidad de esta tesis, se han adoptado una serie de convenciones tipográficas. Aunque muchas de ellas no difieren apenas de las que son utilizadas habitualmente en los textos científicos, recapitulamos y ejemplificamos algunas de éstas:

- Los símbolos matemáticos y términos específicos que se quieren resaltar se escriben en letra cursiva:
  - "Observamos que *n* disminuye cuando *m* aumenta [...]"
  - "Un *rol* es una abstracción de las actividades [...]"
- Las expresiones incluidas en el texto pertenecientes a la especificación de algún modelo, así como la URL de una website o documento electrónico, se muestran en letra `courier`:
  - "El rol `Socio` tiene que realizar al menos las tareas `CompartirRecurso` y `AlcanzarSubobjetivo` [...]"
  - ARONSON, Elliot (2000) "The Jigsaw Classroom: A Cooperative Learning Technique", `http://www.jigsaw.org`
- Las citas a otros trabajos publicados se hacen mediante el sistema autor-año o Harvard:
  - "Buschmann et al. [1996] presentan una interesante clasificación que atiende a [...]"
  - "Similar a los patrones de código o idioms [Coplien, 1992]"
- Los nombres de los patrones pertenecientes a nuestro catálogo aparecen en letra versalita:
  - "La ligadura del patrón JOINT VENTURE en dicho contexto [...]"
- Las transcripciones literales correspondientes a trabajos de otros autores aparecen en cursiva y entre comillas dobles:

> *"Una descripción de clases y objetos que se comunican entre sí dispuestos para resolver un problema de diseño general en un contexto particular."*
>
> —Gamma et al., 1995.

- Las notas a pie de página[6] son referenciadas usando números arábigos consecutivos y únicos dentro de cada capítulo. Cada vez que comienza un nuevo capítulo se reinicia su numeración.

- Las figuras y tablas son numeradas incorporando el número[7] del capítulo delante:

  - Tabla 2.3 (Tabla 3 del Capítulo 2)

  - Figura 3.1 (Figura 1 del Capítulo 3)

- Se usa una variante de la Backus-Naur Form (BNF) para expresar el formato permitido de las notaciones que son textuales. Las convenciones de esta notación son:

  - Los símbolos no terminales están en itálica y entre ángulos (p. ej., <*NoTerminal*>).

  - Los símbolos terminales (palabras reservadas, cadenas, etc.) aparecen entre comillas simples (p. ej., 'and').

  - Las definiciones de reglas de producción de símbolos no terminales se indican con el operador '::='

  - Las alternativas en una producción se separan por el símbolo '|' (p. ej., <*alternativa1*> | <*alternativa2*>).

  - Los ítems opcionales se encierran entre corchetes (p. ej., [<*item*>]).

  - La repetición de un ítem se denota por un asterisco '*' después de ese ítem.

---

[6] Las notas a pie de página están formateadas con fuente arial de 9 puntos.

[7] Letra en caso de apéndice (ej. Fig. A.1)

- Se encierran entre paréntesis los ítems que necesitan agruparse. P. ej., (<*item1*> | <*item2*>)* indica una secuencia de uno o más ítems que pueden ser <*item1*> o <*item2*>.

- Las abreviaturas más utilizadas en esta tesis son:

Tabla 1.1: Abreviaturas utilizadas

cap. (caps.)	Capítulo (capítulos)
fig. (figs.)	Figura (figuras)
ed. (eds.)	Edición o editor (ediciones o editores)
p. (pp.)	Página (páginas)
p. ej.	Por ejemplo
sec. (secs.)	Sección (secciones)
v.	Ver, véase
vol. (vols.)	Volumen (volúmenes)
vs.	Versus, en oposición a

En cuanto al estilo de escritura utilizado en esta tesis, hacemos las siguientes consideraciones:

- A menudo empleamos la primera persona del plural. En ciertas ocasiones lo hacemos con ánimo de incluir al lector en la discusión, en otras para hacer copartícipes a los directores de esta tesis de las reflexiones vertidas y las descripciones realizadas. Más que el plural mayestático, en esta tesis usamos lo que se conoce como el plural de modestia.

- Hemos hecho un esfuerzo por no usar un lenguaje que pueda parecer sexista. No obstante, queremos dejar claro que cuando utilizamos el género gramatical masculino en algunos de los identificadores que aparecen en los modelos (p. ej. identificadores de roles), hemos de entenderlo como neutro y que, desde luego, no hay intención de discriminar sexo alguno.

Capítulo II

Modelado Conceptual de Sistemas Cooperativos con AMENITIES

Contenido

<<La formulación de un problema es más importante que su solución>>

—Albert Einstein (1879-1955)

Científico estadounidense de origen alemán

1. Introducción

Hablar de sistemas cooperativos es hablar de sistemas con capacidad para la comunicación, coordinación y/o colaboración efectiva y eficiente de individuos (personas, agentes inteligentes, robots, etc.) u organizaciones (empresas, departamentos, equipos de trabajo, etc.) durante la realización de tareas compartidas y, a menudo, complejas.

Como podrá imaginar el lector, y como veremos en este mismo capítulo, la gama de sistemas encuadrados dentro de esta categoría es bastante amplia, estableciéndose diferentes criterios para su clasificación (espacio-tiempo, área de aplicación, tarea colaborativa predominante, etc.). Entre estos sistemas nos encontramos aplicaciones para la edición o administración cooperativa de documentos, sistemas de mensajería (correo electrónico, chat, foros de discusión, etc.), sistemas de video conferencia,

sistemas de soporte a la decisión de grupos, gestión de conocimiento compartido, agendas colaborativas, sistemas para la gestión de flujos de trabajo, etc.

Con frecuencia, abordar el desarrollo de este tipo de sistemas se convierte en una labor compleja. Dotar a un grupo de trabajo de la tecnología necesaria para permitir la comunicación, coordinación y cooperación entre sus miembros no es fácil; pero el mayor obstáculo reside en conseguir que la tecnología proporcionada se ajuste a las necesidades reales del grupo. Muchos sistemas cooperativos no han logrado el éxito esperado debido a una insuficiente comprensión de sus requisitos, más que por problemas tecnológicos. Obtener un mayor conocimiento sobre cómo la gente trabaja en grupo y cómo la tecnología afecta a su trabajo es primordial. Parafraseando a Ellis et al. [1991]: "*Los desarrolladores de groupware tienen que ser conscientes de los efectos potenciales de la tecnología sobre la gente, su trabajo y sus interacciones. Una sensibilidad a esta dimensión puede diferenciar entre un sistema groupware que es aceptado y usado regularmente dentro de una organización, y otro que es rechazado.*"

Estudiar y desarrollar exitosamente este tipo de sistemas requiere de un enfoque multidisciplinar que integre conocimientos procedentes de disciplinas de carácter tecnológico (sistemas distribuidos, telecomunicaciones, interacción persona-ordenador, inteligencia artificial, etc.) y social (sociología, psicología, etc.).

Con objeto de que grupos de trabajo multidisciplinares puedan abordar participativa y sistemáticamente el estudio de los sistemas cooperativos, así como su posterior diseño y desarrollo, en el seno de nuestro grupo de investigación (Grupo de Especificación, Desarrollo y Evolución de Software) se ha creado la metodología AMENITIES (**A ME**thodology for a**N**alysis and des**I**gn of coopera**TI**ve syst**E**m**S**) [Garrido, 2003; Garrido et al., 2002, 2005a; Gutiérrez et al., 2002]. Esta metodología facilita la construcción de un modelo conceptual que describe el sistema independientemente de su implementación, proporcionando así una mejor comprensión del dominio del problema, dada la complejidad que caracteriza a estos sistemas. Este modelo conceptual está formado por un conjunto de modelos relacionados de comportamiento y de tareas, lo que permite describir el complejo comportamiento de los grupos de trabajo [Garrido y Gea, 2001].

2. Los sistemas cooperativos

2.1. CSCW, groupware y otros conceptos

CSCW (*Computer-Supported Cooperative Work*, traducido al español como *Trabajo Cooperativo Asistido por Computadora*)[1] hace referencia a un área de investigación interdisciplinar que tiene por objeto estudiar cómo las personas trabajan en grupo y cómo la tecnología puede ayudarles [Greif, 1988]. Partiendo de la observación del trabajo cooperativo, se pretende generar la tecnología necesaria (*groupware*) para la construcción de sistemas que den soporte a la interacción del grupo durante sus actividades colaborativas. A esta clase de sistemas los conocemos como *sistemas cooperativos/colaborativos/CSCW o,* de manera general*, groupware*.

En la literatura existente sobre el tema se ofrecen varias definiciones para el término *groupware*, entre ellas:

- "*Sistemas basados en computadoras que soportan grupos de personas comprometidas en una tarea común (u objetivo) y que proporcionan una interfaz para un entorno compartido*" [Ellis et al., 1991]

- "*Uso de tecnologías de la información para ayudar a que las personas trabajen juntas de una forma más efectiva*" [Malone y Crowston, 1990]

- "*El software que acentúa el entorno multiusuario coordinando cosas, así que los usuarios vean a los demás y sin crear conflictos entre ellos*" [Lynch et al., 1990]

- "*Software y hardware para entornos interactivos compartidos*" [Wells y Kurien, 1996]. Aquí, el *entorno* hace referencia al software y hardware que establece el contexto de la interacción. El término *interactivo* significa que las restricciones de tiempo son administradas por el sistema. *Compartido* alude a que dos o más participantes interaccionan de forma que existe influencia entre ellos. En este sentido, cuando hablamos de *contexto compartido* (subconjunto del entorno) nos estamos refiriendo al conjunto de objetos y acciones que

[1] Término acuñado en 1984 por Irene Greif y Paul Cashman

se pueden realizar sobre éstos, que son visibles por un grupo determinado de usuarios.

Para soportar el nivel de interacción que requiere el groupware es necesario atender tres aspectos clave: la *comunicación,* la *coordinación* y la *colaboración* [Ellis et al., 1991].

La *comunicación* permite el intercambio de información. Este proceso se caracteriza por la identificación de tres elementos clave: los participantes, la información que se transmite y el medio utilizado. Ésta debería ser eficaz, de modo que quienes envíen y reciban una determinada información entiendan lo mismo, y eficiente, en cuanto a que el consumo de recursos sea el mínimo.

Podemos distinguir entre *comunicación sincrónica*[2] y *asíncrona*. Decimos que es sincrónica cuando la comunicación se establece en tiempo real, es decir, la interacción es simultánea (p. ej., comunicación por teléfono, videoconferencia, etc.). Cuando la interacción entre los participantes se realiza en distintos momentos decimos que la comunicación es *asíncrona* (p. ej., comunicación por e-mail, fax, carta, etc.).

La *colaboración* tiene que ver con la interacción entre varios participantes para la consecución de un objetivo común, generalmente la creación compartida de algún artefacto que representa el resultado.

Al igual que la comunicación, la colaboración puede ser sincrónica o asíncrona, dependiendo de si ésta se produce en tiempo real o no.

Hay autores [Schlichter et al., 1998; Bannon y Schmidt, 1989] que consideran sinónimos los términos colaboración y cooperación, sin embargo, hay otros [Dillenbourg et al., 1995; Tiessen y Ward, 1997] que destacan ciertas diferencias entre ellos. En el caso de la *cooperación,* la tarea es dividida en subtareas independientes asignadas a diferentes participantes, y la coordinación es necesaria sólo cuando hay que juntar los resultados parciales. En cambio, cuando se habla de *colaboración*, en un principio no está clara la responsabilidad de cada uno de los participantes, existiendo un

[2] Aunque es ampliamente usado, el término *síncrono* no está reconocido por la Real Academia Española

compromiso mutuo para la resolución del problema. Cada uno de los participantes interviene en todas y cada una de las partes en que se puede dividir el problema o proyecto común, teniendo que coordinarse con los demás para resolver cada parte.

Aunque reconocemos sus diferencias, en esta tesis usaremos indistintamente ambos términos, ya que la infraestructura necesaria y los modelos utilizados van a ser similares en ambas situaciones.

La *coordinación* permite que cada unidad o parte de un todo sepa cómo y cuándo actuar para conseguir un objetivo mayor. Según Malone y Crowston [1990], la coordinación es la actividad encaminada a gestionar las dependencias entre actividades realizadas en grupo para la consecución de un objetivo. Se identifican las dependencias entre actividades para su correcta sincronización y posterior planificación en tareas asignadas a participantes para llevarlas a cabo. La efectividad de la comunicación y la colaboración depende de la coordinación, esto es, de la organización, planificación y sincronización de las actividades del grupo.

A la vista de lo anterior, creemos que otra definición posible para el término *groupware* podría ser *el conjunto de aplicaciones encaminadas a facilitar la comunicación, coordinación y/o colaboración efectiva de grupos durante la realización de tareas compartidas*.

Existen muchos otros conceptos relacionados con los sistemas cooperativos (p. ej. *contexto compartido, ventana de grupo, telepuntero, vista, sesión, rol*, etc.) que, con objeto de no sobrecargar demasiado este apartado y agilizar su lectura, serán definidos conforme se vayan introduciendo. No obstante, en cualquier momento, el lector podrá consultar el glosario de términos que aparece en el Apéndice C de esta tesis.

2.2. Perspectivas de estudio

La integración de conocimientos procedentes de distintas disciplinas es decisiva para la investigación y desarrollo exitoso de los sistemas cooperativos, de ahí su carácter interdisciplinar.

Ellis et al. [1991] señalan que existen al menos cinco disciplinas o perspectivas para el estudio de los sistemas groupware:

- Sistemas distribuidos

  Los sistemas cooperativos son considerados habitualmente sistemas distribuidos ya que, a menudo, sus usuarios se encuentran en diferentes lugares, y tanto los datos como el control están descentralizados. Consecuentemente, algoritmos eficientes para este tipo de sistemas beneficiarían al groupware.

- Telecomunicaciones

  El incremento de la conectividad y el ancho de banda, junto al uso de protocolos eficientes para el intercambio de información con distinto formato (texto, gráfico, voz y video), mejorarían las posibilidades de comunicación del groupware. El gran reto es hacer que las interacciones distribuidas sean tan efectivas como las interacciones cara a cara.

- Interacción persona-ordenador[3] (IPO) [Dix et al., 1998; Lorés, 2001]

  El groupware invita a tratar aspectos relacionados con las interfaces mutiusuario o de grupo. Es vital que sociólogos, psicólogos, usuarios finales, etc., intervengan en el desarrollo de estas interfaces, ya que éstas son sensibles a factores como las dinámicas de grupo y las estructuras organizacionales.

- Inteligencia artificial

  Puede contribuir en la incorporación de agentes activos para la mejora de las interacciones entre personas y ordenadores. Las aplicaciones groupware deberían ser flexibles y adaptarse al comportamiento de los distintos equipos o métodos de trabajo.

- Ciencias sociales

[3] Según el SIGCHI (Special Interest Group in Computer Human Interaction) de ACM, la interacción persona-ordenador es "*la disciplina relacionada con el diseño, evaluación e implementación de sistemas informáticos interactivos para el uso por seres humanos, y con el estudio de los fenómenos más importantes con los que está relacionado*".

> Entender la naturaleza del trabajo en grupo y el impacto de la tecnología en los procesos de grupo son aspectos clave que deben abordarse desde el punto de vista de la sociología [Wilson, 1991]. Los sistemas groupware deberían estar desarrollados teniendo en cuenta los principios y recomendaciones procedentes de las ciencias sociales. Tal y como apuntan Ellis et al. [1991], "*creemos que en el diseño groupware es muy difícil separar los problemas técnicos de los intereses sociales, y que los métodos y teorías de las ciencias sociales serán críticas para el éxito del groupware*".

2.3. Principales taxonomías del groupware

Existen diferentes formas posibles de categorizar los sistemas groupware. La mayoría de éstas no tienen unas líneas divisorias rígidas, de manera que un determinado sistema podría pertenecer a más de una categoría.

En primer lugar, exponemos una de las taxonomías más extendidas de los sistemas cooperativos, debida inicialmente a DeSanctis y Gallupe [1987] y, posteriormente, refinada por Johansen [1989], la cual se basa en una clasificación espacio-temporal de éstos. A continuación, presentamos otra categorización posible que divide los sistemas en función del área de aplicación. Seguidamente, mostramos la que realizan Grudin y Poltrock [1997] en la que se tipifican los sistemas atendiendo a la tarea colaborativa predominante (comunicación, cooperación o coordinación). Por último, y de manera secundaria, describimos una forma particular de clasificar los sistemas debida a Ellis et al. [1991], basada en el grado de proximidad que guardan los sistemas con la definición que estos autores hacen del término *groupware*.

2.3.1. Según el espacio y el tiempo

Es posible clasificar los sistemas groupware en función del lugar en el que se encuentran los usuarios que interaccionan. Así, podemos hablar de sistemas que ayudan en la interacción cara a cara, cuando los participantes se encuentran en el mismo lugar, o sistemas que facilitan la interacción distribuida, cuando los participantes están localizados en diferentes lugares.

Por otro lado, dependiendo de si un sistema se concibe para posibilitar la interacción en tiempo real o no, podemos hablar, respectivamente, de sistemas de interacción sincrónica o asíncrona.

Según esta diferenciación, la Tabla 2.1 presenta las cuatro formas posibles de catalogar estos sistemas, a saber: de interacción cara a cara, de interacción asíncrona, de interacción sincrónica distribuida y de interacción asíncrona distribuida.

Tabla 2.1: Clasificación espacio-temporal

	Mismo Tiempo	**Distinto Tiempo**
Mismo Lugar	Interacción Cara a Cara	Interacción Asíncrona
Distinto Lugar	Interacción Sincrónica Distribuida	Interacción Asíncrona Distribuida

Algunos ejemplos de sistemas pertenecientes a cada uno de estos grupos son los siguientes:

- Interacción cara a cara
  - Pantalla compartida

    Este es el caso, por ejemplo, de la utilización de un cañón de video, una pantalla y un programa de presentaciones (Microsoft PowerPoint, Impress, KeyNote, etc.). La pantalla sirve como medio de interacción entre la audiencia y el/la presentador/a.

  - Cuadro de mandos de un avión

    Con este sistema, parte de la comunicación que existe entre piloto y copiloto se canaliza a través de señales acústicas o visuales que aparecen en el panel de control.

- Interacción asíncrona

- Tablón de anuncios

  Los participantes se comunican a través de notas en el tablón.

- Ordenador compartido

  Se puede utilizar, por ejemplo, para la edición de un documento en distintos momentos por varias personas.

- Interacción sincrónica distribuida

  - Editor de textos sincrónico distribuido

    Varios usuarios localizados en lugares diferentes pueden editar un mismo documento en tiempo real. Es necesario controlar la concurrencia (p. ej. dos o más usuarios pueden desear modificar una misma línea o carácter a la vez). Un ejemplo de esta clase de sistemas es GROVE [Ellis et al., 1991].

  - Espacio de trabajo compartido

    Posibilita la resolución de problemas en grupo de manera sincrónica y entre usuarios que se encuentran en distintos lugares.

    Un ejemplo de este tipo de espacios es BSCW (Basic Support for Cooperative Work) [Appelt, 1999]. Éste es un sistema independiente de la plataforma que soporta la cooperación sincrónica y asíncrona a través de internet con un navegador. Proporciona un espacio de trabajo compartido que los grupos utilizan para el almacenamiento, edición, compartición y manejo colectivo de documentos. A través de un sistema de notificación de eventos los usuarios son informados de las acciones relevantes que son realizadas por otros usuarios dentro del espacio de trabajo. También permite la planificación y realización de reuniones electrónicas.

  - Chat

    Facilita que varios usuarios mantengan una charla en tiempo real dentro de una sala virtual específica (chatroom) mediate la

difusión de mensajes de texto. Por defecto, estos mensajes irán a parar a todos los usuarios que estén en la sala, a no ser que se haya creado un chat privado incluyendo algunos usuarios seleccionados. En algunos casos puede haber alguien que actúe como moderador. Aunque actualmente el uso que se hace de esta tecnología es fundamentalmente lúdico, no hay que perder de vista el potencial que su aplicación puede tener en otros campos, como puede ser el educativo [Ortega et al., 1999, 2000].

- Videoconferencia

  Simula la experiencia del diálogo cara a cara permitiendo, en tiempo real, mantener una conversación entre usuarios que se están viendo. Además de estas prestaciones, la videoconferencia de escritorio o desktop videoconferencing (DTVC) añade la posibilidad de intercambiar ficheros, compartir aplicaciones, utilizar pizarras compartidas, etc. Ejemplos de este tipo de productos son Microsoft NetMeeting, Cuseeme, etc. Este medio de comunicación tiene un futuro muy prometedor. Actualmente la videoconferencia es aplicable en multitud de campos y situaciones. Por ejemplo, una de las áreas que más se está beneficiando de su aplicación es la educación a distancia [Isla y Ortega, 1999, 2001].

- Interacción asíncrona distribuida

  - Correo electrónico

    El producto groupware más conocido y de mayor éxito, probablemente debido su sencillez y facilidad de uso[4]. Las características de este tipo de aplicaciones son sobradamente conocidas.

  - Foros de discusión

    Básicamente consiste en un repositorio de mensajes, similares a los de correo electrónico, enviados por multitud de usuarios

[4] La utilización del correo postal como metáfora parece que ha contribuido en gran medida a su gran éxito.

localizados en diferentes lugares. Se basa en el sistema de discusión distribuida sobre internet llamado USENET. Los usuarios pueden leer y enviar estos mensajes, denominados artículos, a grupos especializados en temas concretos. De esta forma es posible solicitar información o responder a cuestiones formuladas previamente por otros usuarios sobre algún tema específico. Para facilitar la búsqueda de grupos éstos se estructuran jerárquicamente.

2.3.2. Según el área de aplicación

Es posible aplicar el groupware en multitud de áreas diferentes. Algunas de sus principales áreas de aplicación son:

- Sistemas de mensajería

  Soportan el intercambio asíncrono de mensajes. Son ejemplos el correo electrónico, los foros de discusión, etc.

- Editores multiusuario

  Permiten la edición y composición de un documento por varias personas. Cuando es posible editar un mismo objeto por varios usuarios a la vez decimos que son sincrónicos o de tiempo real, en caso contrario decimos que son asíncronos. En el primer caso, el editor maneja los bloqueos y la sincronización de forma transparente, de forma que los usuarios editan los objetos compartidos como si fueran objetos privados. Algunos editores proveen notificación explícita de las acciones de los demás usuarios.

- Sistemas de soporte a la decisión de grupos y salas de reuniones electrónicas

  Los sistemas de soporte a la decisión de grupos o GDSS (Group Decisión Support Systems) proporcionan facilidades para la exploración de problemas no estructurados en grupo. Se pretende aumentar la productividad durante las reuniones de toma de decisión, incrementando la rapidez para adoptarlas y mejorando la calidad de

éstas. Habitualmente incluyen herramientas para generar, organizar, categorizar y priorizar ideas; realizar votaciones, etc.

Estos sistemas se implementan a veces como salas de reuniones electrónicas físicas (interacción cara a cara), las cuales incluyen varios ordenadores, una gran pantalla compartida, equipamiento de audio y video, etc. También pueden implementarse como salas virtuales (interacción sincrónica distribuida) mediante la utilización de un entorno de trabajo compartido que incluya las herramientas apropiadas.

- Videoconferencia

  Comentada anteriormente.

- Agentes inteligentes

  En ocasiones, durante una sesión de trabajo colaborativo pueden intervenir participantes que no son humanos, sino software. Este tipo de participantes son denominados agentes inteligentes o agentes software. Al igual que el resto de participantes, desempeñan roles concretos dentro del grupo y, por consiguiente, son responsables de la realización de ciertas tareas.

- Sistemas de coordinación

  Este tipo de sistemas buscan la coordinación de los esfuerzos de trabajo individuales para conseguir un objetivo mayor. Para facilitar la coordinación, normalmente, estos sistemas favorecen la conciencia de grupo (group awareness), esto es, que cada miembro del grupo sea consciente de las acciones relevantes que realiza el resto. Esto es especialmente importante cuando se llevan a cabo tareas sin estructurar, o débilmente estructuradas, de forma distribuida. Estos sistemas también informan del momento en el que deben o pueden intervenir y de la acción a realizar dentro del proceso colaborativo.

- Sistemas para la gestión del conocimiento compartido

  Soportan los procesos de creación, transformación, organización, búsqueda y recuperación del conocimiento.

Un ejemplo de este tipo de sistemas es KnowCat (Knowledge Catalyser) [Cobos y Alamán, 2002]. Éste es un sistema distribuido para trabajo en la web que, sin necesidad de supervisión, permite la creación incremental de conocimiento estructurado de forma colaborativa, como resultado de la interacción de los usuarios con dicho conocimiento. El principal mecanismo que propone KnowCat para la gestión del conocimiento es el proceso de cristalización de conocimiento, el cual se basa en el trabajo colaborativo de comunidades virtuales de expertos [Hill et al., 1995].

2.3.3. Según la tarea colaborativa predominante

Grudin y Poltrock [1997] señalan otra tipología consistente en la agrupación de la tecnología groupware en base a la tres tareas típicas del groupware: comunicación, cooperación y coordinación.

Cualquier producto groupware contempla una o más de estas categorias, sin embargo, a menudo sucede que alguna de estas categorías predomina sobre las otras. De esta forma, es posible encuadrar algunos sistemas dentro de esta categorización, por ejemplo:

- Tecnologías para la comunicación
  - Correo electrónico
  - Videoconferencia
  - Difusión de video y audio

    Permite la transmisión de video, audio y datos simultáneamente a lugares indeterminados. Es parecido a la emisión de televisión, pero con la ventaja de que la comunicación se da en ambos sentidos. Por ejemplo, muchos grupos han usado tecnología Multicast Backbone (MBONE) para realizar conferencias online en internet.

- Tecnologías para la cooperación/colaboración (espacios compartidos de información)
  - Espacios compartidos de tiempo real

Permiten a la gente trabajar conjuntamente de forma sincrónica, favoreciendo la conciencia de grupo. Algunos ejemplos de este tipo de tecnología son:

- Pizarras y aplicaciones compartidas

  Estas aplicaciones a menudo aparecen integradas en los productos de videoconferencia de escritorio. Una pizarra compartida es un tipo de aplicación que permite dibujar, escribir, mover el cursor, borrar, etc., a múltiples usuarios de manera simultánea. La tecnología para compartir aplicaciones permite utilizar una aplicación monousuario por varios usuarios, transmitiendo las ventanas de la aplicación a cada uno de ellos e integrando todas sus entradas en un flujo único. Este tipo de tecnología está incluida, por ejemplo, en Microsoft NetMeeting.

- Sistemas para soporte a la decisión de grupos (GDSS) y salas de reuniones electrónicas

- Mundos virtuales

  Espacios generados por ordenador que incluyen distintas herramientas para el trabajo colaborativo.

– Espacios compartidos asíncronos

  El trabajo colaborativo no siempre requiere comunicación en tiempo real o interacción simultánea. A veces es posible estructurar el trabajo de manera que los usuarios pueden contribuir independientemente para un producto compartido. Para ello, simplemente es necesario un espacio compartido de información bien organizado donde puedan dejar sus contribuciones y recuperar la información creada por otros. Un ejemplo de este tipo de espacios son:

  - Foros de discusión

  - Sistemas de gestión de documentos

Facilitan el almacenamiento, organización, búsqueda, modificación, control de versiones y permisos de acceso a los documentos.

- Tecnologías para la coordinación
  - Agenda y planificación

    Además de ayudar a los equipos a coordinar su trabajo, sirven como sistemas de administración de información personal. Esta tecnología puede consultar las agendas personales de cada usuario con el propósito de encontrar el día y hora conveniente para la realización una reunión de grupo. Su integración con e-mail agiliza su convocatoria. También facilita la planificación de los recursos necesarios para esas reuniones, como puede ser la reserva de la sala para su celebración.

  - Gestión del flujo de trabajo (workflow management)

    Esta tecnología ayuda a las organizaciones a especificar, ejecutar, monitorizar y coordinar el flujo de trabajo dentro de un entorno distribuido. Permiten automatizar los procesos de trabajo, definiendo las políticas o procesos impuestos por la organización para la realización de un trabajo específico. Para ello utilizan dos componentes, uno para modelar el flujo de trabajo y otro para ejecutarlo. Un ejemplo típico es el enrutamiento de formularios. Con esta tecnología se facilita la creación de éstos, la especificación de la ruta que deben seguir dentro de la organización, personas que pueden acceder, notificación de fechas límite, crear alarmas que recuerden las tareas pendientes, etc.

2.3.4. Otras clasificaciones

Ellis et al. [1991] realizan una particular clasificación (Fig. 2.1) basada en el grado de acercamiento de los sistemas a la definición que ellos hacen del groupware (v. sec. 2.1.) y establecen que los sistemas tienen un nivel más alto en el espectro cuanto más se aproximan a dicha definición. Para ello utilizan dos dimensiones que son: la capacidad para dar soporte a la realización de una tarea común y la existencia de un entorno compartido.

Así, por ejemplo, mientras que en un sistema de correo electrónico los usuarios no disponen de un entorno compartido, en un sistema de aula virtual los usuarios disponen de un entorno común que dispone de los recursos necesarios para acceder a los diferentes temas propuestos por el profesor, realizar actividades de manera conjunta, hacer exámenes, coordinar actividades, establecer reuniones, etc.

Teniendo en cuenta la otra dimensión de la definición, el espectro oscila entre la inexistencia de tareas comunes, como puede ser el caso de un sistema de tiempo compartido convencional, en el que los usuarios ejecutan concurrentemente tareas separadas e independientes, hasta un sistema donde existe un alto nivel de acoplamiento debido a que los usuarios trabajan en tiempo real en alguna tarea común, por ejemplo, una sistema de brainstorming.

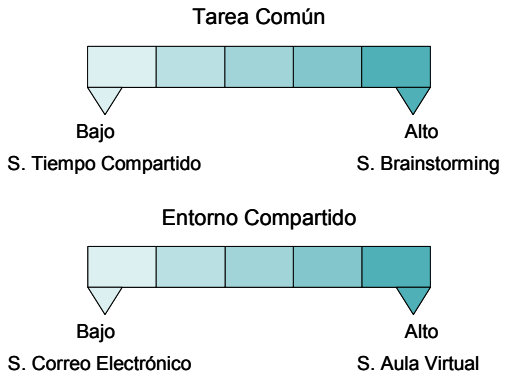


Fig. 2.1: Clasificación del groupware según las dos dimensiones establecidas por Ellis et al. [1991]

2.4. Desafíos

Hay barreras que es necesario superar para conseguir una mayor efectividad del groupware. Éstas no se deben únicamente a problemas de tipo tecnológico, sino también a cuestiones sociales, organizacionales y culturales.

Es necesario acortar la distancia existente entre [Baecker, 1993]:

- El trabajo y los procesos individuales y en grupo.
- El trabajo con software convencional y con groupware.
- El trabajo en la oficina y en un espacio virtual común.

- Reuniones locales y distribuidas.
- Trabajo en redes locales e internet.
- Trabajo sincrónico y asíncrono.
- Trabajo con computadora y sin computadora.
- Uso de CSCW *in vitro* (en laboratorio) e *in vivo* (en un marco real de trabajo).
- Beneficio y aceptación por parte del administrador y por parte del trabajador.

Por otro lado, existe una serie de problemas relacionados con el diseño de estas aplicaciones que están siendo abordados desde distintos frentes [Ellis et al., 1991]:

- Interfaces de grupo

  El diseño de este tipo de interfaces presenta nuevos problemas que difieren de aquellos que aparecen en las interfaces para un único usuario. Por ejemplo, uno de los retos consiste en cómo soportar el complejo comportamiento que supone la interacción de múltiples usuarios con un alto grado de actividad y concurrencia.

  Una buena interfaz de grupo debería presentar la actividad global del grupo pero sin perturbar el trabajo de los usuarios.

- Procesos de grupo

  Son tareas bien definidas que requieren la participación de un conjunto de usuarios. Mientras que los procesos de grupo favorecen la sinergia y el paralelismo, un exceso de coordinación puede sobrecargar al grupo y disminuir su efectividad.

  - Protocolos de grupo

    Son formas de interacción mutuamente acordadas. Se conocen como *protocolos tecnológicos* aquellos que son soportados por hardware/software, imponiendo normalmente el turno de participación (p. ej., el mecanismo de turno de participación

establecido en algunos sistemas de videoconferencia). Cuando el control se deja a los propios participantes, y no son forzados por el sistema, se conocen como *protocolos sociales*, los cuales se caracterizan por ser menos formales (p. ej., solicitud de turno de palabra a mano alzada).

Cada aproximación tiene sus ventajas e inconvenientes. Dejar los procesos de grupo a los protocolos sociales favorece la colaboración, pero puede dar lugar a procesos ineficientes, incompletos o que favorecen la distracción. Por otro lado, embeber un proceso de grupo como un protocolo tecnológico asegura que el proceso es seguido, dota de estructura a las tareas del grupo y guía a los usuarios menos experimentados. Sin embargo, los protocolos tecnológicos pueden llegar a ser demasiado restrictivos y limitar el estilo de trabajo del grupo.

- Factores sociales y organizacionales

  Una de las mayores dificultades a la hora de diseñar una herramienta radica en cómo conseguir que ésta sea útil en diferentes situaciones. Las necesidades de un grupo pueden depender de factores organizacionales (grupo estratificado, entre iguales, etc.) o sociales (abierto/cerrado, democrático, etc.). En general, especializar una herramienta para que se ajuste a las necesidades de un grupo particular requiere conocer el grupo y la organización a la que pertenece.

- Control de concurrencia

  Se refiere a los problemas relacionados con la resolución de los conflictos que a veces se producen por las operaciones simultáneas de varios participantes. Por ejemplo, el *tiempo de respuesta* (tiempo requerido para que las acciones del usuario se reflejen en su propia interfaz) y el *tiempo de notificación* (tiempo para que las acciones se propaguen al resto de interfaces) deben ser muy breves.

- Otras cuestiones

  Es fundamental contar con protocolos de comunicación que sean efectivos y que tengan en cuenta los diferentes requisitos de los

medios de comunicación, modelos que manejen la complejidad del control de acceso determinando en cada momento quién puede acceder y de qué manera, mecanismos de notificación que alerten y modifiquen la interfaz de un usuario rápidamente en respuesta a las acciones realizadas por alguien en otra interfaz, etc.

3. La metodología AMENITIES

3.1. Introducción

AMENITIES (**A ME**thodology for a**N**alysis and des**I**gn of coopera**TI**ve syst**E**m**S**) [Garrido, 2003; Garrido et al., 2002, 2005a; Gutiérrez et al., 2002] es una metodología basada en modelos de comportamiento y de tareas que permite abordar de manera sistemática el análisis y diseño de sistemas cooperativos, favoreciendo su posterior desarrollo.

Esta metodología facilita el modelado conceptual[5] de un sistema cooperativo tomando como referencia marcos teóricos cognitivos y metodológicos [Cañas y Waern, 2001; Gea et al., 2003]. Se centra en el concepto de grupo y cubre los aspectos relevantes de su comportamiento (cambios de rol, dinámicas de trabajo, etc.) y estructura (organización, leyes, etc.).

Al igual que cualquier otra metodología, consta de un conjunto de modelos y fases a seguir para alcanzar su objetivo, en este caso, la obtención de un sistema cooperativo correcto y completo.

Esta metodología ha sido aplicada con éxito en diferentes ámbitos, como por ejemplo, el modelado de la colaboración en un sistema de gestión de conocimiento compartido [Cobos et al., 2003] o la gestión de recursos en un sistema de control de emergencias [Garrido et al, 2002].

[5] Proceso durante el cual el analista construye uno o varios modelos, denominados modelos conceptuales, que ayudan a entender y simplificar el dominio del problema. Estos modelos describen, usualmente mediante alguna notación gráfica, el conocimiento necesario acerca del sistema para poder abordar su desarrollo. Representan la visión (modelo mental) que los usuarios y analistas deberían compartir sobre el sistema.

3.2. Esquema general de la metodología

La Figura 2.2 muestra el esquema general de la metodología AMENITIES. En ésta se representan los principales modelos utilizados y las fases implicadas.

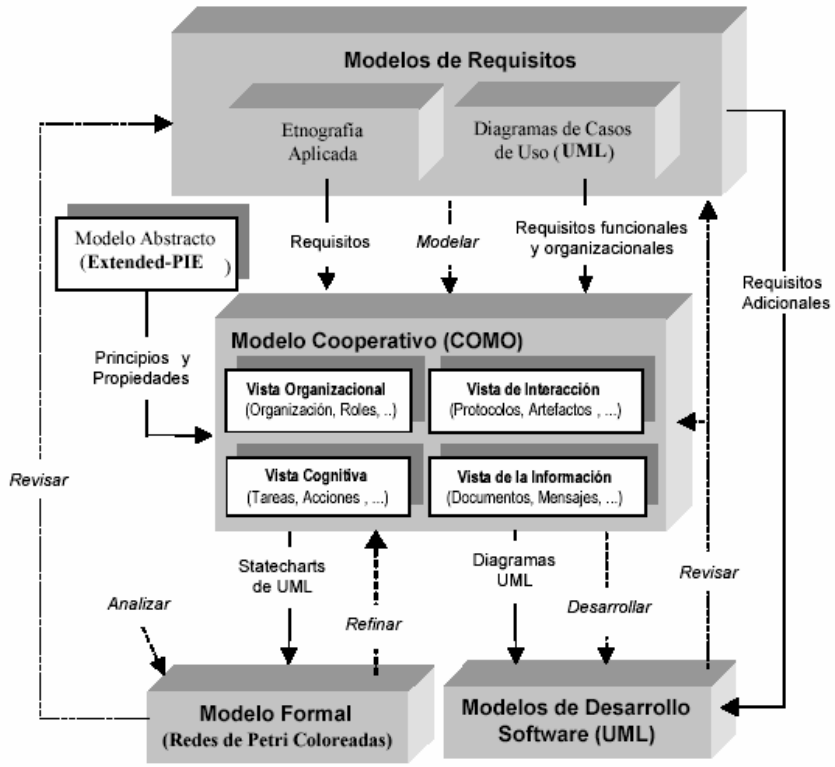


Fig. 2.2: Esquema general de la metodología AMENITIES [Garrido, 2003]

La metodología consta de las siguientes fases:

1) Análisis del sistema y obtención de requisitos.

2) Modelado del sistema cooperativo.

3) Análisis del modelo cooperativo.

4) Diseño del sistema por transformación del modelo cooperativo a partir, normalmente, del análisis realizado anteriormente.

5) Desarrollo del sistema software.

Se utiliza un proceso iterativo que refina el modelo como consecuencia del análisis y revisión de los requisitos de partida y del modelo cooperativo.

A continuación describimos cada uno de los modelos y su objetivo dentro de la metodología.

3.3. Modelos utilizados

3.3.1. Modelos de requisitos

La elicitación u obtención de requisitos se lleva a cabo, principalmente, usando la técnica de etnografía aplicada [Hughes et al., 2000; Jordan, 1996] y la de casos de uso de UML [Jacobson, 1992].

En primer lugar se utiliza la etnografía para descubrir y describir informalmente los requisitos reales del sistema. Después, a partir de esta información, se construyen los diagramas de casos de uso que estructuran y especifican los requisitos funcionales y los roles/actores del sistema.

La etnografía, bien por observación directa (mediante la inmersión en el lugar de trabajo) o indirecta (utilizando entornos de observación), facilita la obtención de aquellos requisitos implícitos, derivados de la forma en que la gente realmente trabaja y no de cómo debería trabajar según los procesos definidos formalmente. Además, facilita la identificación de las peculiaridades socio-culturales de los grupos. En definitiva, la etnografía proporciona una mejor comprensión de las prácticas de trabajo, y del comportamiento y organización de los grupos que las llevan a cabo. De este modo, es posible identificar la estructura de los grupos, aspectos socioculturales que podrían afectar a la interacción y comunicación entre los participantes, comportamiento de los miembros del grupo y su influencia en el resto de participantes, requisitos funcionales que emanan de las prácticas de trabajo y los recursos utilizados, etc.

Sin embargo, aparte de la dificultad que comporta la traducción de resultados empíricos en pautas concretas para el diseño de sistemas, entre otras cosas, motivado por la diferencia de vocabulario existente entre el

ámbito social y tecnológico[6,7], la etnografía no es un enfoque completo y tiene que apoyarse en otras técnicas, por ejemplo, la de casos de uso.

Los casos de uso se emplean para capturar el comportamiento deseado del sistema en desarrollo (requisitos funcionales) y representar de forma abstracta los objetivos de los usuarios.

Según UML [OMG, 2003], los casos de uso facilitan la especificación funcional de un sistema, o parte de éste, sin hacer referencia a su estructura interna. Éstos describen la secuencia de acciones, incluyendo las variantes, que un sistema (u otra entidad) puede realizar interactuando con los actores (usuarios, otros sistemas o máquinas), para producir un resultado observable de valor para uno o más actores.

UML proporciona una representación gráfica (diagramas de casos de uso) para representar la vista externa de éstos, sus relaciones (inclusión, extensión o generalización) y los actores que intervienen, o más bien, los roles que éstos desempeñan junto a su posible jerarquía. Complementariamente, las secuencias de interacción concretas podrían mostrarse usando, por ejemplo, las plantillas para la especificación de casos de uso que propone Durán [2000].

De manera particular, AMENITIES adopta una visión global del sistema, incluyendo también dentro de éste a los propios usuarios, otros subsistemas previamente existentes, etc. Por tanto, la funcionalidad de un caso de uso puede estar asociada también a los propios usuarios y no solamente al sistema, permitiendo describir interacciones funcionales entre usuarios.

---

[6] Precisamente, uno de los objetivos que se pretenden con la aplicación de patrones durante el modelado conceptual de sistemas cooperativos, es tender un puente para salvar, o al menos acortar, la distancia entre ambos dominios. Un trabajo destacable en este sentido es el de Martin et al. [2001, 2002], el cual centra su interés en las cualidades de los patrones como medio para capturar, organizar y hacer más accesible a los diseñadores el corpus de conocimiento procedente de décadas de estudios de etnografía.

[7] Plowman et al. [1995] utiliza los términos “tecno-habla” y “etno-habla” para distinguir entre ambos dominios conceptuales.

En definitiva, parte de la información recopilada a través de la etnografía aplicada podrá ser utilizada para construir los diagramas de casos de uso, los cuales van a facilitar la representación de la funcionalidad del sistema desde el punto de vista de los usuarios, junto a la identificación y clasificación de los roles que participan en dicha funcionalidad. Como se verá más adelante, el conocimiento de estos roles va a ser imprescindible para poder especificar la estructura del grupo y su comportamiento.

3.3.2. Modelo cooperativo

Es el núcleo central de la metodología. Consiste en un modelo conceptual que describe el sistema independientemente de su implementación, proporcionando así una mejor comprensión del dominio del problema, dada la complejidad que caracteriza a estos sistemas. Está formado por un conjunto de modelos interrelacionados de comportamiento y de tareas, lo que permite describir el complejo comportamiento de los grupos de trabajo [Garrido y Gea, 2001].

Para especificar el modelo cooperativo se utiliza una notación denominada COMO-UML (v. Apéndice A), la cual está basada en UML y es propia de la metodología.

Se utiliza un modelo abstracto de sistema, denominado Extended-PIE [Padilla, 2002] que permite describir y analizar actividades de cooperación y propiedades abstractas relacionadas con aspectos de diseño [Garrido et al., 2000; Gea y Gutiérrez, 2001]. Estas propiedades proporcionan información útil para comprender y satisfacer requisitos generales (p. ej. usabilidad, consistencia, etc.), sirviendo de guía para la construcción del modelo cooperativo.

Como se puede ver en la Figura 2.2, el modelo cooperativo se describe a partir de cuatro vistas complementarias (*organizacional, cognitiva, de interacción* y *de información*) que modelan diferentes aspectos del dominio del problema. Describimos cada una de estas vistas a continuación, pero antes presentamos el marco conceptual de trabajo que va a permitir construir este modelo.

3.3.2.1. Marco conceptual de trabajo

Para estudiar los aspectos más relevantes de un sistema y construir su modelo cooperativo, AMENITIES utiliza un marco de trabajo que abarca los conceptos principales (Tabla 2.2), comunes a esta clase de sistemas, así como sus relaciones (Fig. 2.3). El conjunto de todos estos elementos forma la base para la concepción abstracta del grupo utilizada en esta metodología.

Tabla 2.2: Definición de conceptos en los que se basa el modelo cooperativo de AMENITIES [Garrido, 2003]

Concepto	Definición
Evento	*Ocurrencia de algún hecho que tiene una localización tanto en el espacio como en el tiempo.*
Acción	*Unidad básica de trabajo ejecutable atómicamente.*
Artefacto	*Dispositivo (hardware y/o aplicación software) utilizado para llevar a cabo ciertas acciones.* Se puede considerar como la unidad tecnológica básica que sirve de soporte a un sistema cooperativo.
Objeto de información	*Entidad que contiene la información requerida para llevar a cabo acciones, o que se genera como resultado de éstas.* Son objetos pasivos que no poseen comportamiento.
Subactividad	*Unidad de trabajo formada por un conjunto de acciones y otras subactividades que permite estructurar tareas.*
Tarea	*Conjunto de subactividades/acciones cuya realización permite alcanzar objetivos.* Permiten estructurar y describir el trabajo a llevar a cabo por el grupo en el sistema cooperativo. Poseen un alto nivel de abstracción y se relacionan, o incluso se identifican, directamente con objetivos de los usuarios o del grupo.
Actor	*Usuario, programa o entidad que puede desempeñar roles.* Los programas que pueden desempeñar roles dentro de un grupo de trabajo, emulando el comportamiento humano, son conocidos como agentes inteligentes.

Rol	*Comportamiento esperado de un actor en base a un conjunto identificable de tareas a realizar.* Permite resolver la asociación dinámica que existe entre actores y tareas. Puede verse como el estado que posee un actor en un momento dado dentro del grupo. Esto va a servir como punto de partida para describir la evolución del comportamiento de los actores dentro del sistema.
Capacidad	*Habilidad o responsabilidad asociada a un actor que le permite desempeñar roles y llevar a cabo tareas, subactividades o acciones.* A menudo las capacidades restringen el comportamiento de los actores, ya que las leyes de la organización pueden demandar ciertas capacidades para poder desempeñar determinados roles.
Tarea cooperativa	*Tarea en la que interviene más de un actor, desempeñando el mismo o distinto rol.*
Organización	*Conjunto de roles, y relaciones entre ellos, que se dan en un lugar de trabajo.*
Ley	*Norma impuesta por una organización que restringe su funcionamiento en base a reglas sociales, culturales, capacidades de los actores, etc.* Se refiere a las obligaciones, permisos, prohibiciones, etc., que forman parte de la política que gobierna el comportamiento del grupo.
Grupo	*Conjunto de actores desempeñando roles que pertenecen a una misma organización o que participan en la realización de tareas cooperativas.* Puede ser establecido formalmente por la organización (p. ej. atendiendo a la estructura jerárquica existente) o informalmente, a iniciativa de los propios participantes, por deseos o intereses particulares.
Protocolo de interacción	*Conjunto de reglas de comportamiento que utilizan los miembros de un grupo para llevar a cabo subactividades.* Especifican cómo deben coordinarse sus participantes. Pueden ser tecnológicos (soportados por hardware/software) o sociales (controlados por los propios participantes).

En la Figura 2.3, usando un diagrama de clases de UML, se exponen las relaciones existentes entre los conceptos contemplados anteriormente.

Según este metamodelo, podemos destacar brevemente que:

- Un sistema cooperativo está formado por:
  - Una o más organizaciones, compuestas por uno o más roles interconectados, las cuales, a su vez, pueden incluir otras organizaciones.
  - Uno o más grupos de actores.
  - Un conjunto de leyes, eventos, artefactos y objetos de información.
- Los actores pueden desempeñar uno o más roles dentro de una organización y pueden poseer un cierto número de capacidades.
- Los roles incorporan una o más tareas compuestas por subactividades y/o acciones. Al mimo tiempo, las subactividades pueden componerse de otras subactividades y/o acciones.
- Los actores realizan acciones, si tienen capacidad para ello, tras la ocurrencia de eventos y utilizando artefactos.
- Las leyes interrogan las capacidades de los actores para poder realizar acciones y desempeñar roles.
- Los actores realizan las acciones en la medida en que desempeñan roles que tienen asignadas tareas.
- Una acción puede ser de envío o recepción de señales (un tipo de evento). Las acciones pueden requerir o generar determinados objetos de información (p. ej. documentos).
- Los grupos pueden realizar subactividades usando protocolos de interacción.
- Hay tareas que pueden interrumpir a otras.

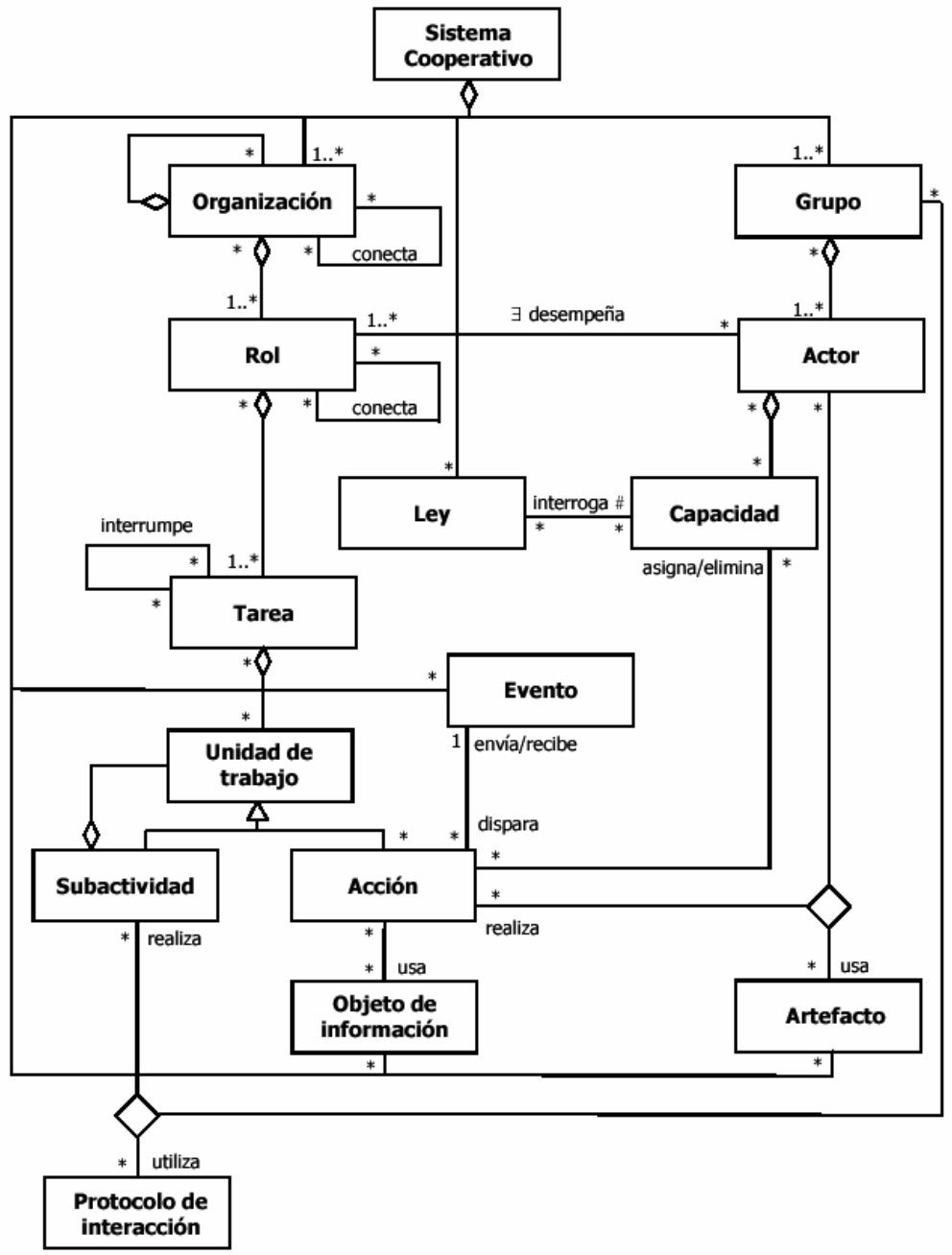


Fig. 2.3: Relaciones entre los conceptos del marco conceptual de trabajo de AMENITIES [Garrido, 2003]

3.3.2.2. Vista organizacional

Según el marco conceptual de trabajo anterior, las organizaciones se articulan a partir del concepto de rol. Este concepto es clave en la descripción del comportamiento de un grupo de trabajo, tanto desde el punto de vista de

sus actividades, como de las dinámicas que la organización y el propio grupo imponen.

Investigaciones relacionadas con la teoría de grupos y CSCW proporcionan dos claras corrientes:

- Concepción del trabajo cooperativo en base a una *organización* permanente y formal del trabajo mediante estructuras preestablecidas.

- Concepción de la diversidad del trabajo cooperativo, en el que múltiples personas designadas informalmente trabajan en *equipo* para fines concretos.

La metodología permite especificar estas dos visiones de grupos de trabajo. En cuanto a la primera, la especificación de una organización implica la identificación de los roles que intervienen en el sistema, así como las restricciones y capacidades que afectan a la evolución del comportamiento desde el punto de vista de los cambios de rol. La segunda visión prevé la posibilidad de llevar a cabo una especificación de grupos más relajada, sin restricciones o capacidades impuestas. Los equipos de trabajo se asocian a tareas cooperativas, o actividades dentro de éstas, y sus miembros se deciden dinámicamente a partir de responsabilidades concretas.

3.3.2.3. Vista cognitiva

Esta vista representa el conocimiento que cada miembro del grupo tiene o adquiere en el contexto organizacional. Este conocimiento se refleja especificando las tareas que se realizan. Normalmente se identifica el modelado del usuario con el modelado de tareas.

Para la especificación de esta vista se define cada rol mediante la identificación de las tareas y sus atributos, los eventos que las habilitan y las capacidades/leyes necesarias para poder participar en cada tarea.

3.3.2.4. Vista de interacción

La interacción entre los miembros de un grupo de trabajo puede adoptar formas diversas. Se puede decir que varios usuarios están interaccionando cuando realizan actividades distintas dentro de una misma tarea cooperativa, o cuando las acciones de un miembro afectan directa o indirectamente al comportamiento de otros miembros o grupos.

Esta vista contempla la interacción entre participantes cuando las tareas no están estructuradas o lo están débilmente. Esto ocurre, por ejemplo, cuando el siguiente paso en una actividad depende de resultados anteriores y éstos no son conocidos de antemano, o cuando el proceso depende de condiciones externas a la tarea.

Para abordar este tipo de interacción, AMENITIES proporciona lo que se denominan protocolos de interacción. Éstos son conjuntos de reglas que especifican los pasos generales a seguir para realizar la actividad en base a protocolos sociales, los cuales implican sincronización y comunicación de cualquier clase de información (gestos, mensajes, documentos, etc.) [Garrido et al., 2002]. Ejemplos de éstos pueden ser petición-respuesta, mensajes encolados, etc.

Esta vista identifica estos protocolos, sus participantes (humanos, agentes, etc.), atributos (punto a punto, difusión, sincrónico/asíncrono, cara a cara, etc.) y tipo de medio para soportarlo (videoconferencia, correo, chat, etc.), el cual viene determinado por sus atributos.

3.3.2.5. Vista de información

De acuerdo con el nivel de abstracción de AMENITIES, la información puede aparecer implícita en subactividades/acciones y también, si se cree conveniente, de manera explícita como objetos de información dentro del flujo de control entre subactividades/acciones.

Puesto que las entidades que representa están relacionadas con subactividades/acciones, la vista de información está conectada con la vista de interacción y cognitiva.

3.3.3. Modelo formal

Partiendo del modelo cooperativo y aplicando esquemas de traducción a los elementos de la notación COMO-UML, es posible obtener automáticamente una Red de Petri Coloreada (CPN) [Garrido y Gea, 2002]. El resultado es un modelo formal que permite realizar un análisis automatizado del sistema.

Mediante la ejecución de la red resultante, es posible simular el sistema, validando requisitos generales (consistencia, completitud, etc.) y evaluando la usabilidad del sistema en términos de tareas. También es posible verificar propiedades (interbloqueos, alcanzabilidad, etc.) aplicando técnicas específicas de análisis.

3.3.4. Modelos de desarrollo de software

Para construir estos modelos, la metodología propone el uso de las distintas notaciones que provee UML.

La conexión de AMENITIES con el desarrollo del software va ser el diseño arquitectónico, el cual va a servir como punto de partida para el equipo de desarrolladores, ofreciendo una visión común del sistema. El diseño arquitectónico del software va a proporcionar un conjunto de componentes y relaciones entre ellos. Implementar un sistema cooperativo a partir de un conjunto de subsistemas que se comunican mediante interfaces bien definidas es vital, ya que este tipo de sistemas son por naturaleza distribuidos [Garrido et al., 2005b, 2006, 2007].

Del modelo cooperativo pueden derivarse los componentes en que se va a dividir el sistema, la funcionalidad de éstos, metainformación (roles sociales, restricciones impuestas por la organización, etc.), así como su comportamiento y despliegue [Garrido et al., 2004].

Al igual que los modelos de tareas se han utilizado tradicionalmente como guía para el diseño de una aplicación interactiva, derivando su interfaz de usuario [Paternò, 1999], la idea es que el modelo cooperativo sirva para la generación, al menos semiautomática, de la interfaz de usuario de las aplicaciones groupware.

3.4. Método de modelado

Se utiliza un método sistemático, compuesto por una serie de etapas, para construir el modelo cooperativo. El método, fundamentalmente basado en los conceptos de rol y tarea, mantiene las relaciones semánticas entre las distintas representaciones de cada una de las etapas, lo que resulta en un único modelo construido y organizado jerárquicamente.

El método consta de las siguientes etapas:

1) Representación de la organización mediante un conjunto de roles conectados por transiciones.

2) Definición de cada uno de los roles en base a conjuntos de tareas.

3) Descripción de las tareas mediante conjuntos de subactividades y acciones.

4) Especificación de protocolos de interacción para aquellas subactividades que lo requieran.

3.4.1. Especificación de la organización

Esta etapa se centra en la representación de *grupos de la organización*, como composición formal y perdurable en el tiempo que es impuesta por la propia organización, y no de *equipos de trabajo*, los cuales son una composición informal y temporal de participantes con objetivos concretos. Éstos últimos, se abordarán en la tercera etapa, centrada en la definición de tareas.

La especificación de la organización consiste, por un lado, en la identificación de los roles que intervienen y sus relaciones, y por otro, en la especificación de las leyes de la organización que restringen la evolución del comportamiento en base a cambios de rol.

AMENITIES utiliza lo que denomina *diagramas de organización*, expresados en notación COMO-UML, para realizar dicha especificación.

Un diagrama de organización representa la estructura organizativa de un conjunto de actores, en base a los roles que pueden desempeñar dentro de

un sistema, y su evolución a través de dicha estructura, en virtud de las restricciones (leyes o capacidades) que la organización impone.

Así, mediante el concepto de rol, se describe la estructura estática y dinámica de una organización, independientemente de los actores concretos que participen.

Estos diagramas están basados en los diagramas de estados de UML. Los estados representan los diferentes roles que los actores pueden asumir en la organización. Las transiciones reflejan los posibles cambios de rol, en función del cumplimiento de ciertas leyes o capacidades que etiquetan las transiciones.

Por lo tanto, el comportamiento de un actor viene dado por el rol que desempeña, por las capacidades adquiridas y por las leyes que la organización impone para poder jugar otros roles.

Para ilustrar este apartado, proponemos la representación de la organización de una comunidad de vecinos (Fig. 2.4). Para su especificación (roles, transiciones, leyes y capacidades) se han tenido en cuenta las reglas de comportamiento y estructura que marca la *Ley de Propiedad Horizontal (LPH)*[8]. A continuación, explicamos someramente este diagrama[9].

Cada uno de los roles que aparecen (`Comunero`, `Presidente`, `Vicepresidente`, `Secretario` y `Administrador`) se corresponden con cargos específicos contemplados en la *LPH*. Éstos se representan mediante estados dentro de un estado compuesto, el cual representa la organización `ComunidadVecinos`. Además, por medio de expresiones de multiplicidad, se representa dentro de cada rol u organización la cantidad de actores que pueden desempeñarlo.

[8] Ley 8/1999, de 6 de abril, de reforma de la Ley 49/1960, de 21 de Julio, sobre Propiedad Horizontal (B.O.E. de 23 de Julio) (`http://www.todalaley.com/mostrarLey527p1tn.htm`)

[9] Ante cualquier duda, y para mayor información acerca de la notación utilizada, se recomienda al lector la consulta del Apéndice A, el cual contiene una exposición detallada de la sintaxis y semántica de COMO-UML

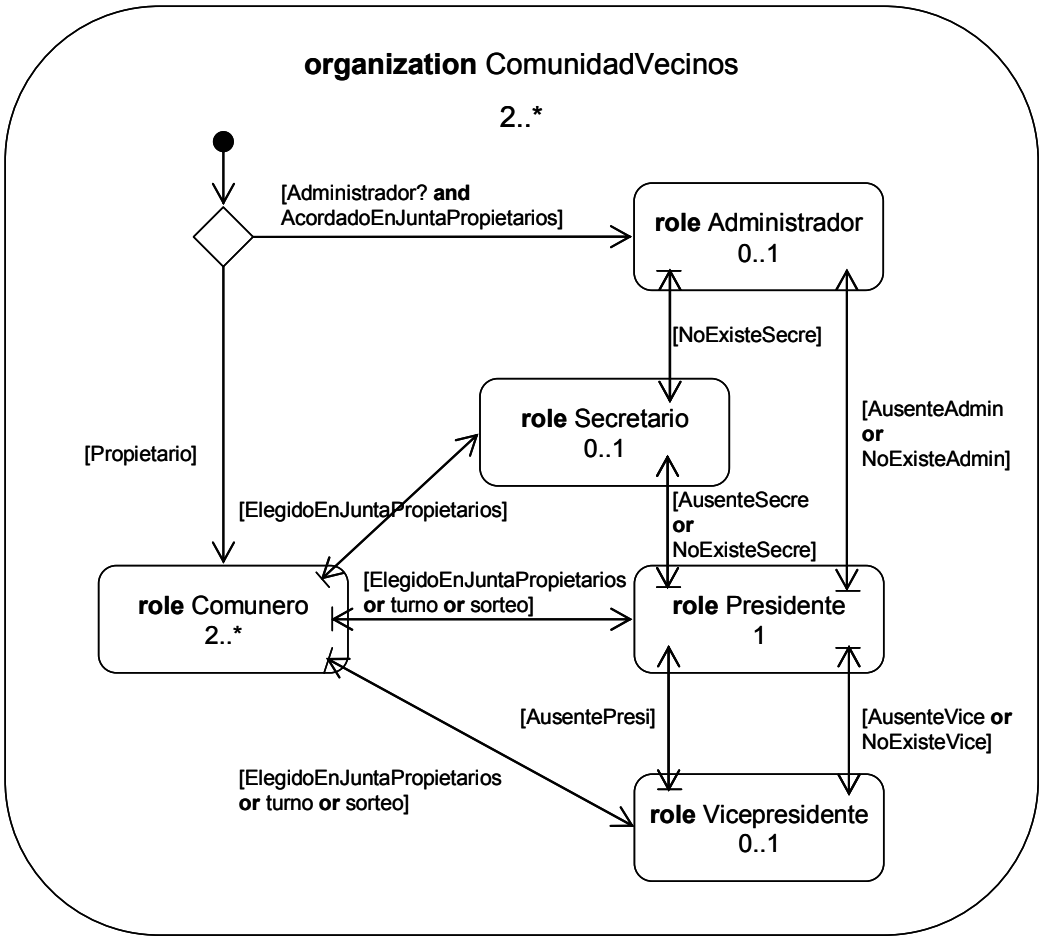


Fig. 2.4: Especificación de la organización correspondiente a una comunidad de vecinos

Los posibles cambios de rol que los actores pueden experimentar dentro de la organización se reflejan por medio de transiciones que conectan los estados. Las transiciones pueden estar etiquetadas con *leyes* de la organización (expresadas como guardas, es decir, entre corchetes) o con *capacidades* requeridas para los actores (guardas cuyos identificadores terminan con el símbolo ?), las cuales deberán cumplirse para que la transición pueda dispararse, o lo que es lo mismo, para que pueda producirse el cambio de rol. De esta forma, la evolución de los actores en la estructura organizativa se regula a través de las leyes y capacidades exigidas por la organización.

Las transiciones pueden ser de dos tipos:

- *De cambio.*- El actor que está desempeñando el rol inicial abandona éste para jugar el rol final. Se representa como una flecha que conecta

el rol inicial con el final. Uno de los extremos posee una línea transversal para indicar cuál es el estado inicial.

- *Aditivas*.- El actor que está desempeñando el rol inicial no abandona éste, asumiendo también el rol final. Si deja de cumplirse la condición que dispara dicha transición, el actor abandona el rol final, permaneciendo en su rol inicial. Se representa con una doble flecha que conecta ambos roles. Uno de los extremos posee una línea transversal para indicar cuál es el estado inicial.

Pueden aparecer cajas de decisión representadas como rombos, las cuales disponen de transiciones de salida etiquetadas con condiciones que determinan las diferentes alternativas con respecto a los roles que pueden ser desempeñados.

Por tanto, observando el diagrama, podemos decir que:

- Los actores de la organización `ComunidadVecinos`, entre los cuales tiene que haber al menos dos (2..*), son fundamentalmente los `Comuneros` y el `Administrador`. Según la *LPH*, un actor adquiere la condición de `Comunero` (como mínimo tienen que existir dos, ya que la multiplicidad es `(2..*)`) por el hecho de ser dueño (v. ley `[Propietario]`) de cualquiera de las dependencias privativas que hay en el edificio: locales, viviendas, trasteros, plazas de garaje, etc. Por otro lado, no es obligatorio dotar un cargo de `Administrador` (v. multiplicidad 0..1), pues sus funciones pueden ser ejercidas por el `Presidente`, pero en caso de contratar alguno, éste debe estar capacitado para ello y su contratación debe ser acordada en Junta de Propietarios (v. capacidad y ley `[Administrador? and AcordadoEnJuntaPropietarios]`). A su vez, es el único rol que puede ser ejercido por una persona que no tenga la condición de `Comunero`, pudiendo también asumir las funciones del cargo de `Secretario`, si éste no existe (v. ley `[NoExisteSecre]`).

- Para poder desempeñar los cargos de `Presidente` o `Vicepresidente`, la persona tiene que ser obligatoriamente `Comunero` (v. transiciones aditivas desde `Comunero` a `Presidente` y `Vicepresidente`) y se debe cumplir alguna de las condiciones siguientes: ha sido elegido en Junta de Propietarios, le ha llegado su turno o ha sido seleccionado por

sorteo (v. ley `[ElegidoEnJuntaPropietarios or turno or sorteo]`). En cambio, para poder actuar como `Secretario`, tan sólo es necesario ser elegido en Junta de Propietarios (v. ley `[ElegidoEnJuntaPropietarios]`) o que el puesto sea desempeñado por el `Administrador` cuando no existe.

- En cuanto al rol de `Presidente`, el papel más relevante en la organización, es el único cargo que debe cubrirse obligatoriamente (v. multiplicidad = 1), pudiendo asumir adicionalmente los roles de `Administrador`, `Secretario` y `Vicepresidente`, si éstos están ausentes o no existen (v. condiciones `[AusenteAdmin or NoExisteAdmin]`, `[AusenteSecre or NoExisteSecre]` y `[AusenteVice or NoExisteVice]`).

- El `Vicepresidente`, si existe, podrá sustituir al `Presidente` cuando éste último se encuentre ausente (v. condición `[AusentePresi]`).

Como se verá en el siguiente apartado, los actores también pueden modificar su comportamiento sin realizar un cambio de rol. Además, un actor, desempeñando un rol determinado, puede interrumpir su trabajo actual y emprender uno nuevo, con o sin necesidad de cambio de rol.

3.4.2. Especificación de roles

La definición de roles facilita la conexión entre la especificación de la organización y las actividades individuales/colaborativas a realizar por sus miembros.

Durante esta etapa se identifican y asocian las tareas a roles, partiendo normalmente de la información que nos ofrecen los modelos de requisitos (etnografía aplicada y diagramas de casos de uso). Adicionalmente, para cada tarea asociada a cada uno de los roles se pueden especificar:

- Tareas, pertenecientes al mismo rol o a otros, que pueden interrumpir su realización. Por defecto, una tarea no puede ser interrumpida.

- Eventos que disparan la ejecución de la tarea.

- Leyes que permiten su realización.

- Acciones que pueden aparecer en las transiciones de entrada y salida de la tarea llevando a cabo cambios controlados en el estado global del sistema.

- Objetos de información involucrados. Éstos van a formar parte de la vista de información.

- Tipo de tarea (individual o cooperativa) en función del número de actores necesarios para efectuarla.

- La multiplicidad, o número de actores que han de intervenir bajo cada uno de los roles involucrados, exclusivamente cuando la tarea es cooperativa.

Los diagramas COMO-UML usados para realizar esta especificación se denominan *diagramas de rol*.

En la Figura 2.5 se reflejan los diagramas de rol correspondientes a tres de los roles (`Comunero`, `Secretario` y `Presidente`) de la organización `ComunidadVecinos`, incluyendo como muestra tan sólo algunas de sus tareas más relevantes.

Como podemos comprobar, la tarea `RealizarJuntaOrdinaria` es común a los tres roles, indicando que todos cooperan en su realización (v. cláusula `cooperative-task`). Aparte de esta tarea, el rol `Comunero` tiene la obligación de pagar a la comunidad una cuota mensual (tarea `PagarCuotaComunidad`). Por otro lado, la persona que actúa como `Secretario` de la comunidad tiene también la responsabilidad de custodiar las actas correspondientes a las Juntas de Propietarios (tarea `CustodiarActas`) y publicar los avisos que sean de interés para la comunidad (tarea `PublicarAviso`). En cuanto al `Presidente`, si fuese necesario, éste debe participar además en los procedimientos judiciales en representación de la comunidad de vecinos (tarea `RepresentarComunidadJuicio`).

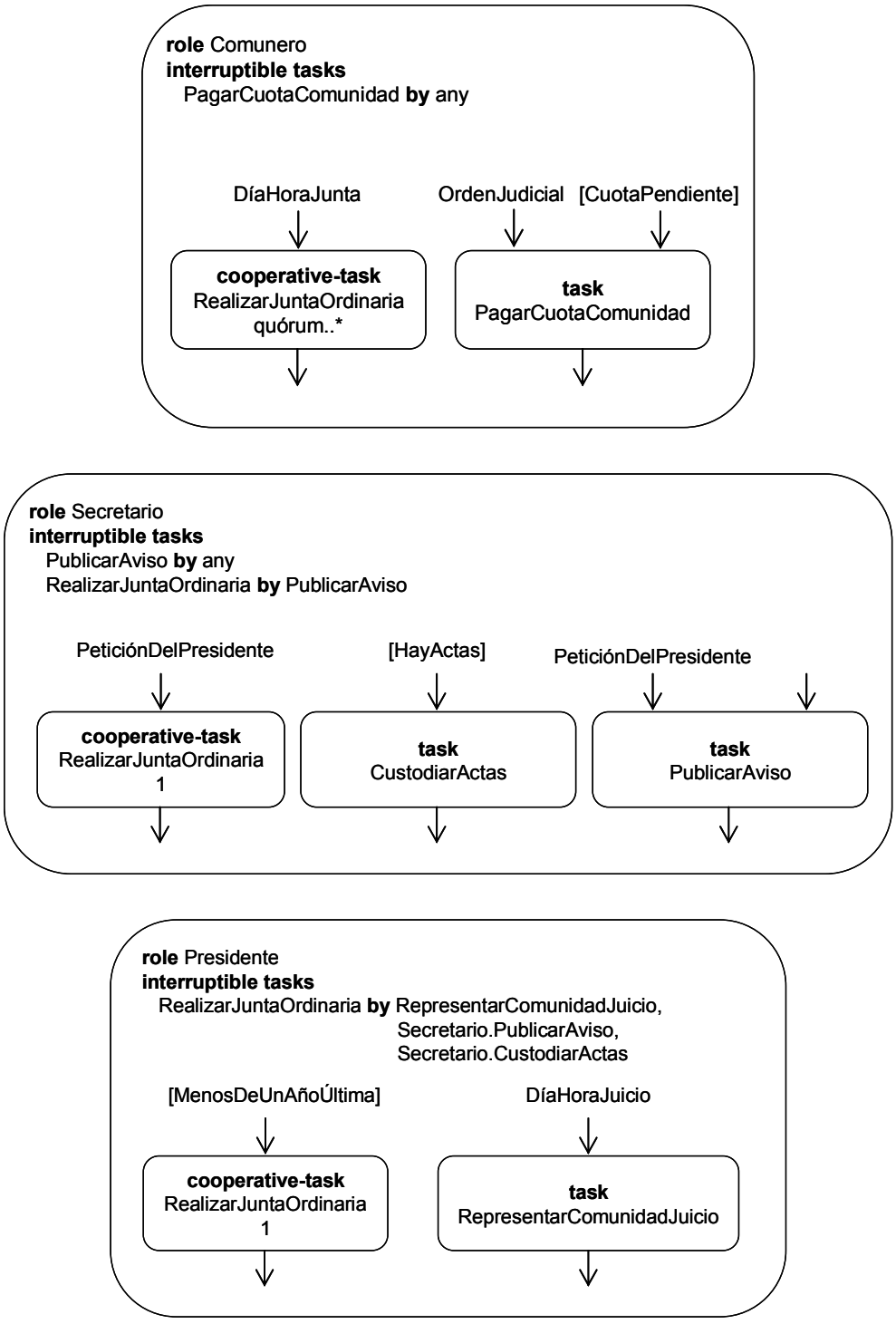


Fig. 2.5: Diagramas de rol (Comunero, Secretario y Presidente)

En la sección `interruptible tasks` se indican aquellas tareas que pueden ser interrumpidas por otras, lo que permite establecer prioridades entre ellas. Si no se indica nada acerca de una determinada tarea significa que ésta no se puede interrumpir, ya que, por defecto, una tarea no puede ser

interrumpida por otra. Una tarea puede ser interrumpida por otra tarea del mismo rol u otro distinto, siempre y cuando pueda también ser desempeñado por ese actor en un momento dado.

En nuestro ejemplo podemos observar cómo la tarea `RealizarJuntaOrdinaria` tiene la mayor prioridad en el caso del rol `Comunero`. Al no aparecer en la sección `interruptible tasks`, ésta no puede ser interrumpida. Sin embargo, la tarea `PagarCuotaComunidad` puede ser interrumpida por cualquier otra (v. cláusula `any`). Respecto al rol `Secretario`, la tarea que no puede dejar de lado es `CustodiarActas`, pudiendo interrumpir la tarea `PublicarAviso` por cualquier otra, y la tarea `RealizarJuntaOrdinaria` por `PublicarAviso` cuando es necesario publicar algún aviso importante. El `Presidente` no podrá interrumpir la tarea `RepresentarComunidadJuicio`. Sin embargo, podrá interrumpir `RealizarJuntaOrdinaria`, cuando tenga que desempeñar la tarea `RepresentarComunidadJuicio` como `Presidente`, o cuando tenga que realizar las tareas `PublicarAviso` y `CustodiarActas` como `Secretario`, llegado el caso de que el mismo actor tuviera que desempeñar también este cargo[10]. No obstante, como veremos más adelante en la Figura 2.6, la tarea `RealizarJuntaOrdinaria` no podrá interrumpirse en cualquier momento, debido a que hay restricciones que lo impiden para alguna de sus subactividades (v. la restricción `non-interruptible` aplicada a la subactividad `ReunióonJuntaOrdinaria)`.

Cada tarea puede tener determinados eventos que disparan su realización, y si no los tiene, es decisión propia del actor el emprenderla o no. Por ejemplo, la tarea `PagarCuotaComunidad`, responsabilidad de todo `Comunero`, muestra estas dos circunstancias. Por una parte, esta tarea puede realizarse cuando el actor lo desee, siempre y cuando tenga alguna cuota pendiente de pago (v. condición `[CuotaPendiente]`). Por otra parte, un requerimiento judicial (v. evento `OrdenJudicial`) podría obligar a que se ejecute esta tarea. En cuanto a la tarea `RealizarJuntaOrdinaria`, su realización requiere que llegue el día y hora especificados en la convocatoria

[10] Según la especificación de la organización realizada en la Figura 2.4, un actor con el rol de `Presidente` podría llegar a desempeñar el rol de `Secretario` si éste está ausente o no existe.

de la reunión (v. evento `DíaHoraJunta`). La intervención del `Secretario` en la tarea `RealizarJuntaOrdinaria` puede ser requerida cuando lo solicite el `Presidente` (v. evento `PeticiónDelPresidente`), por ejemplo para realizar la convocatoria de la Junta Ordinaria. Las tareas `CustodiarActas` y `PublicarAviso` las puede efectuar a iniciativa propia. Para la primera es necesario que exista algún acta que custodiar (v. condición `[HayActas]`). En el caso de la segunda, ésta puede también ser provocada a petición del `Presidente` (v. evento `PeticiónDelPresidente`). En relación con el `Presidente`, la tarea `RealizarJuntaOrdinaria` la iniciará cuando lo estime conveniente, con la condición de que no haya transcurrido más de un año desde la última Junta Ordinaria (v. condición `MenosDeUnAñoÚltima`). La tarea `RepresentarComunidadJuicio` la efectuará a partir del momento en el que el juicio comience (v. evento `DíaHoraJuicio`).

Hemos de resaltar que la especificación de las tareas cooperativas, en nuestro caso únicamente `RealizarJuntaOrdinaria`, incluye una expresión de multiplicidad que indica la cantidad de actores con la que contribuye cada rol a la realización de dicha tarea. Así, con el rol `Comunero` interviene una cantidad mínima de actores equivalente al quórum necesario para llevarla a cabo (v. multiplicidad `quórum..*`). Sin embargo, con el rol de `Secretario` o `Presidente` tan solo participa un actor (v. multiplicidad `1`).

3.4.3. Especificación de tareas

En esta fase se describen las tareas que, junto con la definición de roles realizada en la etapa anterior, constituyen la vista cognitiva del modelo cooperativo.

Los diagramas COMO-UML utilizados para esta especificación se denominan *diagramas de tareas y subactividades*, los cuales son una adaptación de los diagramas de actividad de UML.

Los diagramas de actividad han experimentado cambios importantes desde la versión UML 1.4, en la cual se basa la propuesta inicial de COMO-UML, hasta la introducción de UML 2.0 y posteriores versiones.

Hasta el momento, la representación de los diagramas de tareas y subactividades encontrados en las diversas publicaciones relacionadas con la

metodología AMENITIES [Garrido, 2003] ha estado basada en los diagramas de actividad de UML 1.4. Según esta versión y la 1.5, los diagramas de actividad son considerados como un caso especial de un diagrama de estados en el que sus estados son acciones o actividades. Las acciones no se pueden descomponer y su ejecución es atómica. Las actividades están compuestas a su vez por acciones/actividades. Cuando una acción/actividad termina, automáticamente se dispara cualquier transición que exista a su salida, pasando el flujo de control inmediatamente a la siguiente actividad/acción, salvo que exista alguna condición, la cual debería cumplirse. Aunque no es lo habitual, una transición de salida puede aparecer etiquetada con el nombre de un evento, disparando la transición cuando éste sucede y, por consiguiente, interrumpiendo y forzando la salida de la actividad anterior. Esto no puede ocurrir en el caso de las acciones, puesto que éstas no pueden ser interrumpidas por definición.

Desde UML 2.0 los diagramas de actividad poseen una nueva y rica sintaxis, así como una semántica inspirada en las Redes de Petri[11]. Esto le confiere más poder, flexibilidad y un tratamiento diferenciado con respecto a los diagramas de estado, los cuales son usados principalmente para modelar el comportamiento de los objetos de un sistema desde el punto de vista de sus cambios de estado.

No obstante, muchos de los diagramas pueden ser modelados haciendo uso de tan sólo de una pequeña parte de los elementos de modelado disponibles en la nueva notación. De hecho, a menudo no hay apenas diferencia entre modelos representados en distintas versiones, salvo su interpretación, ya que muchos de sus elementos básicos de modelado son comunes.

Aunque los modelos de tareas y subactividades que se presentan a modo de ejemplo en este capítulo respetan la concepción inicial de la metodología AMENITIES (diagramas de estado), en esta tesis hemos hecho un esfuerzo de adaptación hacia el nuevo enfoque propuesto en la versión oficial actual del estándar [OMG, 2007a, 2007b]. En consecuencia, todos los patrones y ejemplos de aplicación que mostraremos en posteriores capítulos

[11] Si desea saber más acerca de este formalismo puede consultar http://www.informatik.uni-hamburg.de/TGI/PetriNets/

serán modelados e interpretados de acuerdo con este nuevo tratamiento. En el Apéndice D, se dan algunas nociones básicas sobre los diagramas de actividad en UML 2. Asimismo, si el lector está interesado en obtener más detalles sobre este tema puede consultar la especificación oficial del estándar [OMG, 2007b, pp. 295-417].

La descripción de tareas facilita la especificación de la coordinación entre actores ya que, a través de las subactividades incluidas en cada tarea cooperativa, se establece cómo y cuándo interviene cada uno de los actores participantes. Como hemos visto, también dentro de los propios diagramas de rol es posible reflejar distintas formas de coordinación mediante la especificación de eventos que activan las tareas en momentos determinados.

Las tareas pueden ser, como ya hemos visto, individuales (interviene sólo un actor) o cooperativas (intervienen varios actores con el mismo o distintos roles). Como es natural, centraremos nuestro interés en la representación y descripción de la única tarea cooperativa que hemos contemplado, la tarea `RealizarJuntaOrdinaria` (Fig. 2.6).

La Figura 2.6 representa la secuencia de subactividades[12] completa (no aparecen acciones) relacionada con el proceso a seguir para realizar una Junta Ordinaria de propietarios, desde la preparación de la convocatoria (tarea `PrepararConvocatoriaJuntaOrdinaria`) hasta el envío de las actas de la reunión a cada uno de los propietarios implicados (tarea `EnviarActaJuntaOrdinaria`). Este diagrama se corresponde con el nivel más alto de descripción del proceso. A partir de éste, es posible ir detallando su descripción mediante la representación de diagramas que concretan cada una de las subactividades incluidas. Más adelante especificaremos algunas de estas subactividades (Fig. 2.7 y Fig. 2.8).

El flujo de control de esta tarea es secuencial, no existiendo tareas paralelas en este nivel de especificación. Aparece, además, una condición que permite restringir el momento en el que una determinada transición puede dispararse. En este caso, indicamos que la celebración de la Junta Ordinaria

[12] Las subactividades se diferencian gráficamente de las acciones en que las primeras incluyen en su esquina inferior derecha un pequeño símbolo que representa a dos estados conectados por una transición.

(subactividad `Reunión JuntaOrdinaria`) debe realizarse el día y a la hora que indica la convocatoria (v. condición `[DíaHoraJunta]`).

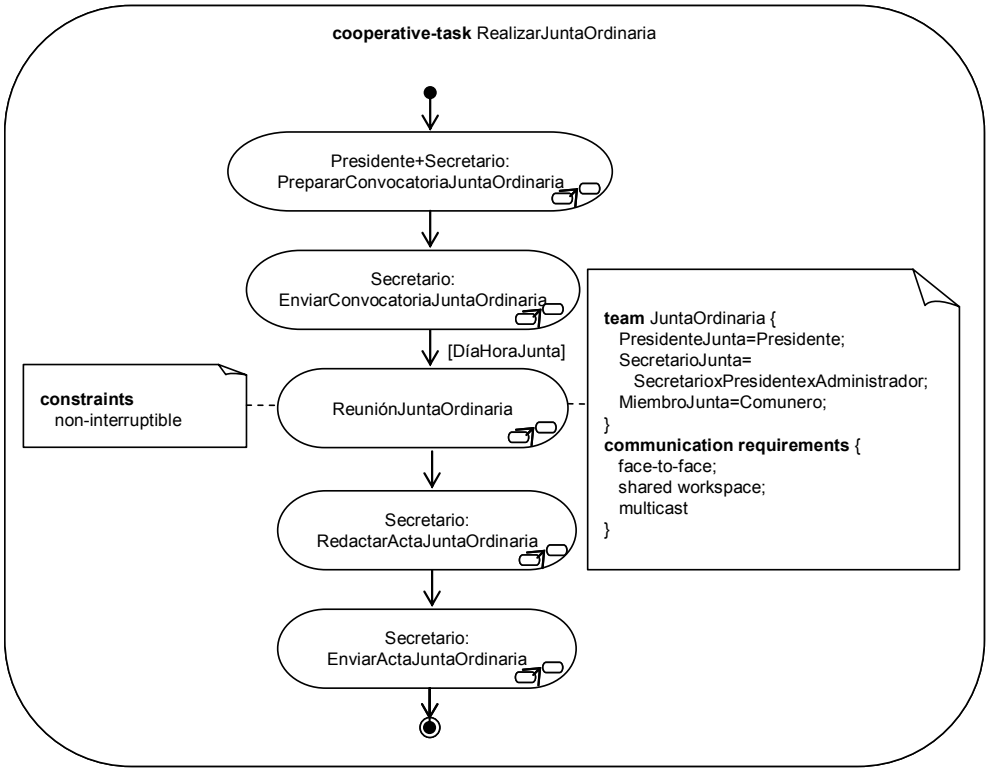


Fig. 2.6: Diagrama de la tarea `RealizarJuntaOrdinaria`

Las tareas, subactividades o acciones pueden llevar asociados comentarios para facilitar la comprensión de los modelos que, de acuerdo con UML, se representan como rectángulos con una de sus esquinas doblada. En el ejemplo, la subactividad cooperativa `ReunionJuntaOrdinaria` está unida a dos comentarios. Uno de ellos está relacionado con la especificación de *equipos de trabajo* y el otro con la aplicación de una restricción para indicar que dicha subactividad no puede ser interrumpida[13].

Como ya se ha comentado en este capítulo, los *equipos de trabajo* hacen alusión a grupos de actores de una o varias organizaciones que se reúnen para realizar una tarea concreta. En contraposición, los *grupos de*

[13] Obsérvese que mientras para los roles Secretario y Presidente la tarea `RealizarJuntaOrdinaria` podía ser interrumpida por otras tareas, ésta no podrá ser interrumpida cuando se esté realizando la subactividad `ReunionJuntaOrdinaria`.

organización, definidos a través de los llamados diagramas de organización, definen la organización desde el punto de vista de la estructura organizativa general y su evolución. La asignación de subactividades/acciones a roles o equipos de trabajo es de un gran interés, puesto que permite describir dinámicas de grupo con fines concretos y a corto plazo. La especificación de los grupos de organización, junto a la de los equipos de trabajo, compone la vista de la organización.

Básicamente, la especificación de un equipo de trabajo consiste en una redefinición de los roles que se desempeñan "oficialmente" dentro de la organización origen, lo que permite redistribuir sus miembros dependiendo de los requisitos concretos de trabajo.

Para especificar un equipo de trabajo se utiliza la cláusula `team` dentro de un comentario que deberá estar asociado a la tarea cooperativa en cuestión. En la figura 2.6 hemos indicado que el equipo encargado de llevar a cabo la tarea `Reunión JuntaOrdinaria` se denomina `JuntaOrdinaria` y que los roles específicos que intervienen son:

- `PresidenteJunta`.- Desempeñado por el `Presidente` de la organización `ComunidadVecinos`.
- `SecretarioJunta`.- Puede ser ocupado por el `Secretario`, `Presidente` o `Administrador` de la comunidad[14] (v. expresión de roles `SecretarioxPresidentexAdministrador` para indicar que tan sólo uno de ellos puede desempeñar tal cargo)[15].
- `MiembroJunta`.- Representado por todos aquellos actores que ostentan el papel de `Comunero`.

Dentro de este mismo comentario también aparece una sección titulada `communication requirements` para especificar los requisitos de

[14] Es necesario recordar que, según el diagrama de organización de la comunidad (Fig. 2.4), si el cargo de `Secretario` no existe éste puede ser ocupado por el `Administrador` o, en última instancia, por el propio `Presidente`.

[15] Para más información acerca de la construcción de expresiones de rol consúltese el Apéndice A.

comunicación necesarios para el desarrollo de esta actividad. Según éstos, se debe garantizar la existencia de un espacio de trabajo compartido (`shared workspace`), que cada participante pueda ver al resto (`face-to-face`) y que los mensajes enviados por cada participante lleguen al resto de participantes (`multicast`). Esta información es importante puesto que permite determinar el tipo de artefacto necesario para poder soportar la interacción. En este caso podría ser, por ejemplo, un sistema de videoconferencia.

Cada subactividad/acción muestra los roles de la organización, o del equipo concreto asignado, que desempeñan los actores encargados de ejecutar dicha actividad. Si una subactividad de nivel superior ya ha especificado el rol responsable, éste podría obviarse, simplificándose su especificación cuando todas las subactividades/acciones pertenecen al mismo tipo de participante. Si hay que especificar varios roles, COMO-UML proporciona una sintaxis específica que permite expresar las distintas combinaciones posibles (v. Apéndice A). Por ejemplo, en la subactividad `PrepararConvocatoriaJuntaOrdinaria` se indica que intervienen tanto el `Presidente` como el `Secretario` de la comunidad de vecinos (`Presidente+Secretario`).

Como hemos mencionado anteriormente, para obtener más detalles de este proceso es posible desplegar cada una de las subactividades que comprende. Como muestra, en las Figuras 2.7 y 2.8 detallamos, respectivamente, la subactividad `Reunión JuntaOrdinaria` y una de las subactividades de ésta, en concreto `RealizarVotaciónPunto`.

Básicamente, la subactividad `Reunión JuntaOrdinaria` consta de dos flujos de trabajo concurrentes. Por una parte, el actor que desempeña el papel de Secretario de la Junta Ordinaria (`SecretarioJunta`) toma nota (subactividad `Tomarnotas`) de los hechos relevantes que acontecen a lo largo de toda la reunión para su inclusión en el acta[16] de la Junta Ordinaria. Por otra parte, y de manera paralela, se ejecuta una secuencia de subactividades/acciones que consiste esencialmente en: la apertura de la reunión (acción `IniciarReunión JuntaOrdinaria`) por parte del Presidente

[16] Obsérvese el flujo de objetos existente en el que se recoge el objeto (`BorradorActa`) y su estado a la entrada (`NoModificado`) y salida (`Modificado`) de la actividad `TomarNotas`

de la Junta Ordinaria (`PresidenteJunta`); la lectura, también por parte del Presidente, de cada punto según el orden del día (acciones `LeerPrimerPunto` y `LeerSiguientePunto`); el debate del punto en cuestión (subactividad `DebatirPunto`) y, si procede, su votación (subactividad `RealizarVotaciónPunto`) por todos los participantes (v. la palabra reservada `all` en ambos casos para indicar tal circunstancia); y por último, el cierre de la reunión por parte del Presidente de la Junta Ordinaria (acción `FinalizarReuniónJuntaOrdinaria`).

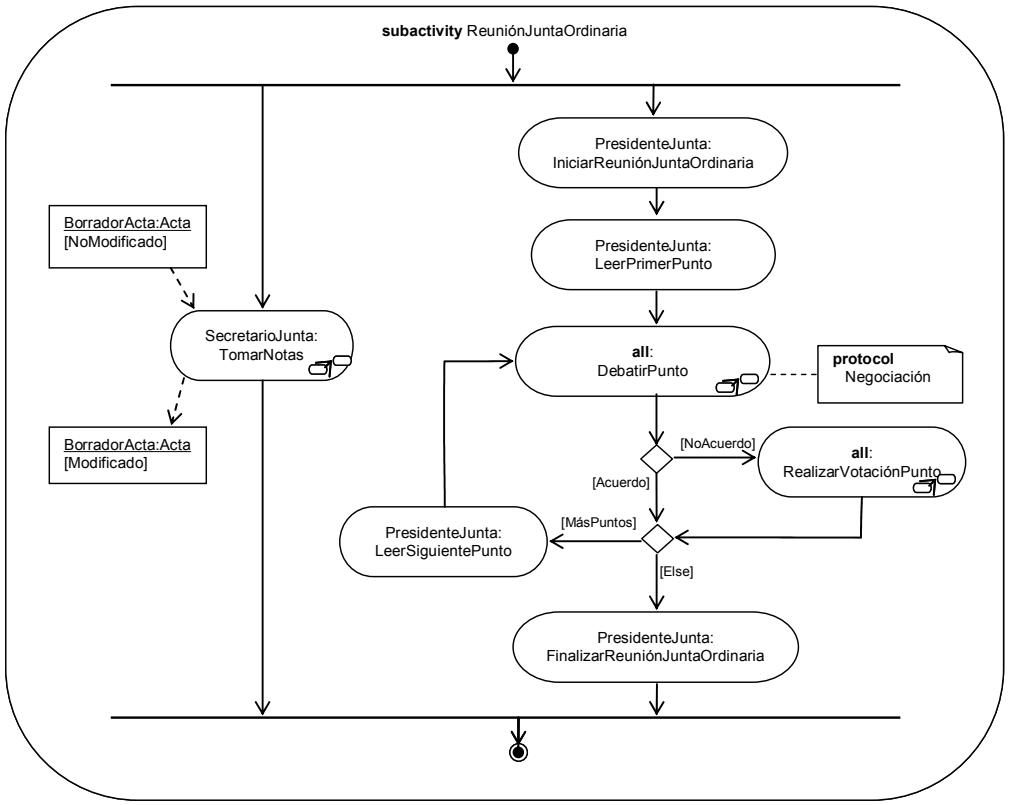


Fig. 2.7: Diagrama de la subactividad `ReuniónJuntaOrdinaria`

En cuanto a la subactividad `RealizarVotaciónPunto`, desplegada en el diagrama de la Figura 2.8, lo primero que hace el `PresidenteJunta` es iniciar la sesión de votación (acción `IniciarVotación`). A continuación, el `SecretarioJunta` informa de las distintas opciones (subactividad `InformarOpciones`) a votar (p. ej., abrir piscina/no abrir, pintar de blanco/negro/azul, etc.). Inmediatamente el `PresidenteJunta` solicita el voto (subactividad `VotarOpción`) de los propietarios (`MiembroJunta`) para cada una de las opciones posibles, a la vez que el `SecretarioJunta` va registrando los votos emitidos (subactividad `RegistrarVotos`). Una vez emitidos y

registrados todos los votos, el `SecretarioJunta` informa del resultado de la votación (subactividad `InformarResultado`). Por último, el `PresidenteJunta` da por finalizada la sesión (acción `FinalizarVotación`).

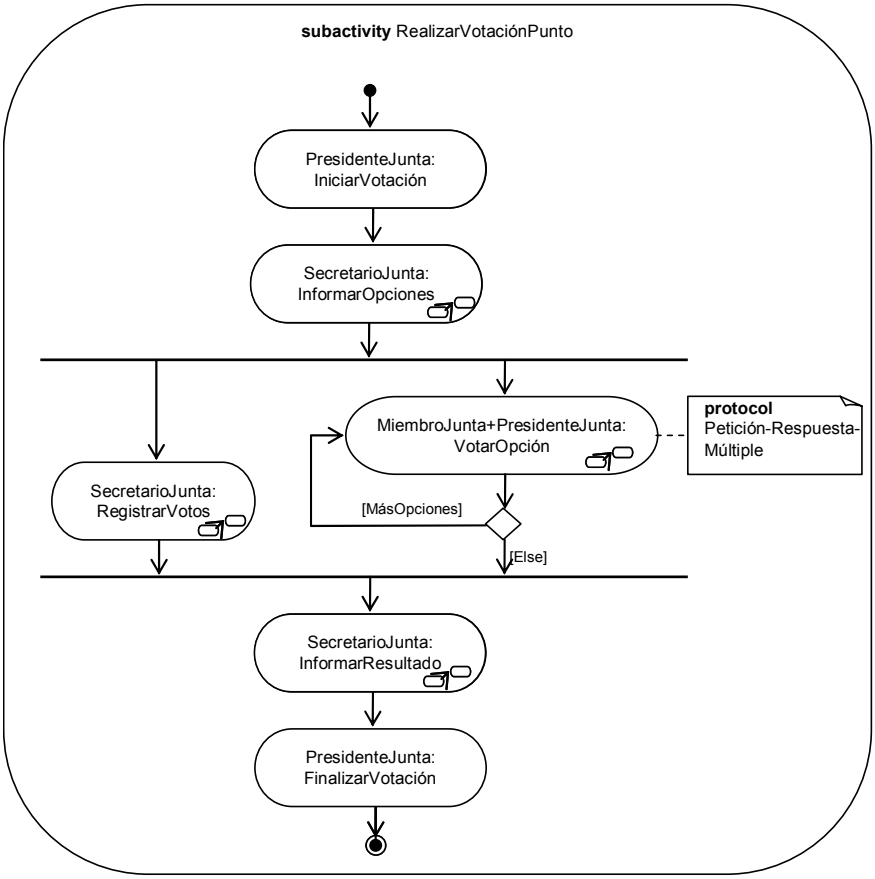


Fig. 2.8: Diagrama de la subactividad `RealizarVotaciónPunto`

Existen dos cajas de decisión en el diagrama de la Figura 2.7, las cuales derivan el flujo de control dependiendo, en un caso, de si se ha llegado a un acuerdo tras debatir el punto que se está tratando (condiciones `[Acuerdo]` y `[NoAcuerdo]`) o, en otro caso, de la existencia de más puntos que tratar (condiciones `[MásPuntos]` y `[Else]`). Así, si no se ha alcanzado un acuerdo, el punto se somete a votación (subactividad `RealizarVotaciónPunto`). A continuación, si existen más puntos que tratar, se lee el siguiente punto (acción `LeerSiguientePunto`) comenzando de nuevo el ciclo con su debate (subactividad `DebatirPunto`). En caso de que este punto sea el último, se da por finalizada la reunión (acción `FinalizarReuniónJuntaOrdinaria`).

En este tipo de diagramas aparecen a menudo objetos de información involucrados en el flujo de control, los cuales van a formar parte de la vista de información. En el ejemplo de la Figura 2.7 aparece el borrador del acta (`BorradorActa`) que es utilizado por el Secretario de la Junta Ordinaria para ir anotando cualquier hecho que sea necesario constatar. También es posible representar los diferentes estados por los que un objeto va pasando. Como se refleja en el diagrama, el estado `[NoModificado]` que posee el objeto `BorradorActa` antes de entrar en la subactividad `TomarNotas` cambia, una vez concluida ésta, al estado `[Modificado]`.

Por último, en ambos diagramas aparecen sendos comentarios sobre los protocolos de interacción utilizados en la coordinación de ciertas subactividades. Particularmente, en la Figura 2.7, la subactividad `DebatirPunto` está regida por el uso de un protocolo de negociación (v. cláusula `protocol Negociación`) y, en la Figura 2.8, la subactividad `VotarOpción` por el protocolo petición/respuesta-múltiple (v. cláusula `protocol Petición-Respuesta-Múltiple`). Este extremo será convenientemente detallado en la siguiente sección.

3.4.4. Especificación de protocolos de interacción

La realización de esta especificación proporciona la vista de interacción del modelo cooperativo. Aunque, como hemos visto, los actores pueden interaccionar entre ellos de manera indirecta[17], otras veces éstos interaccionan directamente durante la realización de alguna subactividad. Los protocolos de interacción están relacionados precisamente con la interacción directa entre participantes, la cual puede adoptar diversas formas, dependiendo de la actividad concreta a realizar y de los roles desempeñados por los actores.

Los protocolos de interacción van a servir para especificar las propiedades (protocolos sociales, patrones de comportamiento, tecnología utilizada, etc.) que caracterizan la interacción directa entre participantes. El

[17] Por ejemplo, el hecho de que un determinado actor esté ausente puede provocar que otro, con un rol diferente, tenga que sustituirlo y desempeñar el rol del primero en base a la aplicación de alguna ley de la organización.

objetivo es poder usar protocolos tecnológicos implícitos en artefactos (p. ej., videoconferencia, correo electrónico, chat, etc.), de acuerdo con los protocolos sociales necesarios para la coordinación de las acciones que forman parte de una subactividad y, por lo tanto, de los participantes que en ella intervienen.

Algunos ejemplos de protocolos de interacción básicos identificados son:

- Negociación (*Negotiation*).- Los participantes implicados pueden intervenir en la actividad sin un orden preestablecido y con el mismo nivel de responsabilidad para llegar a un acuerdo.

- Petición-Respuesta-Simple (*Simple-Request-Reply*).- Forma de interacción sincrónica donde un participante formula una petición para, a continuación, recibir una respuesta, la cual es esperada por el solicitante antes de continuar.

- Petición-Respuesta-Múltiple (*Multiple-Request-Reply*).- Es un tipo de protocolo petición-respuesta-simple en el que la respuesta recibida no es única.

- Mensajes Encolados (*Queued-Messages*).- Sería la versión asíncrona de un protocolo petición-respuesta. En este caso, el participante que hace la petición no tiene que esperar la respuesta del otro (o de los otros) para poder continuar.

Por ejemplo, como ya se mencionó en la sección anterior, la Figura 2.8 tiene un comentario asociado a la subactividad `VotarOpción` indicando que ésta se rige por un protocolo *Petición-Respuesta-Múltiple*. Esto es debido a que el sistema de votación se organiza como sigue: el `PresidenteJunta` solicita al conjunto de propietarios (rol `MiembroJunta`) el voto para una opción determinada (p. ej. votos en contra de la apertura de la piscina) y espera una respuesta múltiple (el voto de varios propietarios con derecho a voto).

La descripción de esta subactividad, según el protocolo de interacción asociado, la hacemos en el diagrama que aparece en la Figura 2.9.

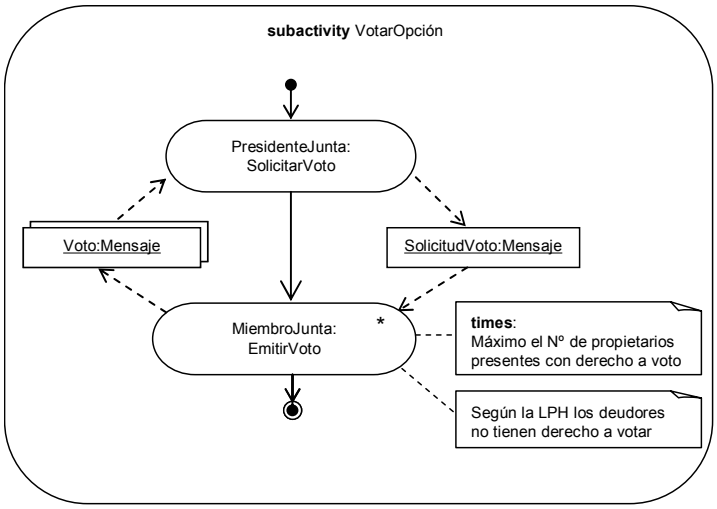


Fig. 2.9: Diagrama de la subactividad `VotarOpción`

El flujo de información que hay entre el `PresidenteJunta` y los propietarios (rol `MiembroJunta`) está representado por el flujo asociado al objeto `SolicitudVoto` y al conjunto de objetos `Voto`, ambos de tipo `Mensaje`. La petición (objeto `SolicitudVoto`) realizada por el `PresidenteJunta` es recibida por todos los propietarios presentes[18], respondiendo cada uno de éstos con su voto (conjunto de objetos `Voto`). El número de votos emitidos, a favor o en contra, siempre será menor o igual que el número de propietarios no deudores presentes, ya que según la Ley de Propiedad Horizontal éstos no tienen derecho a voto (v. comentarios anexos a la acción `EmitirVoto`). La inclusión de un asterisco dentro del estado de acción `EmitirVoto` significa que dicha acción se realiza de manera concurrente (p.ej. alzan la mano a la vez).

Los protocolos de interacción son un claro ejemplo de conocimiento reutilizable relacionado con modelos de comportamiento típicos que facilitan la coordinación entre las acciones/subactividades y, por lo tanto, entre los participantes que en ella intervienen.

Por ejemplo, el modelo correspondiente al protocolo *Petición-Respuesta-Múltiple* podría generalizarse como se indica en la Figura 2.10., lo

[18] No hay que olvidar que, según los requisitos de comunicación heredados y expresados en el diagrama de la Figura 2.6, cada petición y respuesta debe ser recibida por todos los participantes.

que podría ser la base para la especificación de un patrón[19] que nos ayude a describir y construir los modelos.

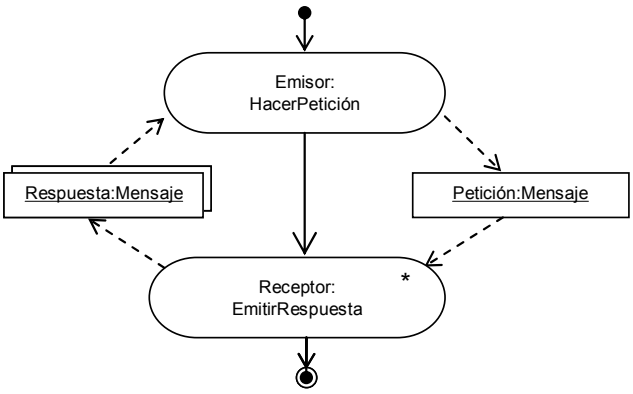


Fig. 2.10: Modelo genérico correspondiente al protocolo Petición-Respuesta-Múltiple

4. Conclusiones del capítulo

En el presente capítulo hemos tratado dos de los tres pilares sobre los que se fundamenta nuestro estudio: los *Sistemas Cooperativo*s y la *Metodología AMENITIES*. El tercer pilar necesario, los *Patrones de Software*, lo tratamos en el siguiente capítulo.

A la vista de las definiciones que algunos autores destacados en la materia han vertido en este sentido, y una vez repasados algunos de los conceptos relacionados, podemos convenir en que *un sistema cooperativo es aquél que está pensado para facilitar a un grupo de individuos/organizaciones la realización de tareas compartidas a través de la comunicación, coordinación y/o colaboración efectiva de sus miembros*.

La aplicabilidad de los sistemas cooperativos es bastante amplia. Un examen de las principales taxonomías propuestas que permiten clasificar la tecnología para trabajo en grupo, nos ha reportado una idea de su alto potencial.

[19] Lógicamente incluyendo otros elementos de modelado necesarios para realizar una definición más precisa.

Hemos puesto de manifiesto las dificultades que existen para afrontar con éxito el estudio y desarrollo de estos sistemas, principalmente debidas a la insuficiente comprensión de sus requisitos, más que por problemas de tipo tecnológico. Obtener un mayor conocimiento sobre cómo la gente trabaja en grupo y cómo la tecnología puede ayudar es primordial. Para ello, hemos visto que es importante utilizar un enfoque interdisciplinar que integre conocimientos procedentes tanto del ámbito tecnológico (sistemas distribuidos, telecomunicaciones, interacción persona-ordenador, inteligencia artificial, etc.) como del ámbito social (sociología, psicología, organización del trabajo, etc.). Justamente, dentro del campo de investigación conocido como *CSCW* (*Computer-Supported Cooperative Work*) se intenta acometer esta problemática desde un punto de vista interdisciplinar.

Con el propósito de facilitar el estudio y análisis sistemático de estos sistemas, así como favorecer su posterior diseño y desarrollo, nuestro grupo de investigación ha propuesto la metodología AMENITIES.

Esta metodología, la cual intentamos sintetizar en la segunda parte de este capítulo, facilita el modelado conceptual de un sistema cooperativo desde un punto de vista del grupo, teniendo muy en cuenta el modelo cognitivo que reside en el grupo y las características del modelo social en el que está basado.

Hemos definido y relacionado cada uno de los conceptos que forman parte del marco conceptual de trabajo utilizado por dicha metodología. A su vez, hemos explicado el conjunto de modelos y fases que la componen, haciendo especial hincapié en el llamado *Modelo Cooperativo*, núcleo central de esta metodología.

El Modelo Cooperativo es un modelo conceptual que describe el sistema independientemente de su implementación, proporcionando así una mejor comprensión del dominio del problema, dada la complejidad que caracteriza a este tipo de sistemas. Incluye un conjunto de modelos interrelacionados de comportamiento y de tareas utilizados para describir la compleja forma de proceder de los grupos de trabajo. Hemos explicado cada una de las cuatro vistas complementarias (organizacional, cognitiva, de interacción y de información) que es posible distinguir dentro de este modelo, las cuales proyectan diferentes aspectos de la descripción del sistema.

A través de un sencillo ejemplo, hemos ilustrado el método de construcción del Modelo Cooperativo. En concreto, hemos especificado la organización de una comunidad de vecinos y, a partir de ésta, hemos modelado algunos de sus roles y tareas cooperativas más relevantes.

Sin embargo, hasta ahora, y a pesar de la capacidad de AMENITIES, cada vez que hemos construido el Modelo Cooperativo de un nuevo sistema hemos tenido que partir prácticamente desde cero. Como veremos en el siguiente capítulo, los patrones de software son un valioso instrumento que nos va a permitir describir y reutilizar modelos que capturan conceptos y escenarios de trabajo cooperativo que comúnmente especificamos durante esta etapa inicial.

-Esta página se encuentra deliberadamente en blanco-

Capítulo III

Modelado Conceptual de Sistemas Cooperativos en base a Patrones

Contenido

<<Un experto es un hombre que ha dejado de pensar: sabe>>

—F. Lloyd Wright (1867-1959)

Arquitecto estadounidense

1. Introducción

Cuando un experto se enfrenta a un problema que aparece una y otra vez en su dominio de trabajo, normalmente lo resuelve con más pericia que una persona no avezada en la materia. La razón es que el experto posee un amplio conocimiento, basado en su experiencia, que facilita la detección y resolución de dichos problemas. Sin embargo, a menudo este conocimiento permanece oculto en la mente del experto y no es compartido con otras personas. Para el experto no siempre es sencillo transmitir cuándo, cómo, dónde y porqué usa determinadas soluciones, simplemente las aplica.

En ciertos ámbitos, donde la toma de decisiones es una labor intensiva y compleja, a menudo surgen problemas que deben ser tratados reiteradamente. En estos casos, contar con algún instrumento que ayude a detectar, describir y compartir las soluciones más apropiadas para estos problemas sería de una gran utilidad. Los *patrones de software* desempeñan dicha misión en el marco de la ingeniería del software.

Desarrollar software no es fácil, es una labor que se caracteriza precisamente por la necesidad de tomar multitud de decisiones,

frecuentemente en grupo, para resolver los problemas que se presentan en las distintas fases que componen el ciclo de vida de un producto software. No obstante, se ha podido comprobar que muchos de estos problemas aparecen, una y otra vez, en determinadas fases de desarrollo y para dominios de aplicación específicos. Los patrones de software facilitan la descripción exhaustiva y razonada de tales problemas y de su solución, para que dicho conocimiento pueda ser reutilizado en diferentes ocasiones.

La aparición de los patrones en el ámbito del desarrollo de software ha supuesto un salto significativo en lo que a reutilización se refiere. Hasta entonces, ésta se había centrado fundamentalmente en la reutilización del código de programa. Los patrones han abierto el camino hacia la reutilización del conocimiento empírico que ha demostrado su validez en la resolución de los distintos problemas que podemos encontrar en cualquier etapa del desarrollo de software. En este sentido, los patrones se pueden ver como componentes reutilizables de conocimiento.

Los patrones de software no suponen una ventaja únicamente por permitir la creación de un corpus de conocimiento sobre buenas prácticas de diseño u otro tipo de experiencias útiles durante el desarrollo de software. Como ocurre con cualquier otra disciplina, para facilitar el entendimiento entre las personas, es fundamental establecer un vocabulario común que permita identificar y transmitir los conceptos del dominio y sus relaciones. Justamente, otra de las cualidades esenciales de los patrones es su capacidad para construir un lenguaje de comunicación que permita compartir y discutir los problemas, así como sus posibles soluciones.

Para obtener todas estas ventajas, es primordial la integración de los patrones en los procesos y métodos de desarrollo de software. Para ello, es necesario contemplar los patrones como elementos de modelado de primer orden, representarlos y describirlos de manera adecuada, entrenar e instruir al equipo de desarrollo en una filosofía de trabajo orientada a patrones y contar con una colección estructurada de patrones que abarque todo el proceso y facilite la selección del más apropiado en cada momento.

Uno de los periodos más complejos y decisivos dentro de este proceso es el que abarca la obtención, análisis y especificación de requisitos de un sistema. Durante estas etapas tempranas, el ingeniero de requisitos o analista desarrolla modelos conceptuales que ayudan a entender y simplificar

los problemas que aparecen en el dominio de aplicación. Para agilizar y mejorar la construcción de tales modelos, es posible aplicar un tipo de patrones, conocidos como *patrones conceptuales* [Riehle y Züllighoven, 1996] *o de análisis* [Fowler, 1997], cuyo objetivo es facilitar la comprensión, comunicación y reutilización de aquellas abstracciones claves recurrentes en un determinado dominio del problema. Sin embargo, pese a su importancia, hay muy pocos trabajos publicados en relación con la aplicación de este tipo de patrones en dominios concretos y bajo una metodología específica. En esta tesis pretendemos cubrir este objetivo en el ámbito particular del modelado conceptual de sistemas cooperativos aplicando la metodología AMENITIES [Isla et al., 2006a, 2007].

En la primera parte de este capítulo se presenta una panorámica general sobre los patrones de software. Se repasa un poco su historia y se examinan las definiciones más interesantes que se han dado sobre éstos y otros conceptos relacionados. Se distinguen las distintas propuestas de clasificación y descripción de patrones. Se señalan los beneficios que se pueden obtener con la integración de éstos en los procesos de software y se consideran las pautas que deben tenerse en cuenta para que su implantación sea efectiva. Esta primera parte termina, dando a conocer algunos de los foros más importantes especializados en el tema.

En el siguiente bloque se aborda la aplicación de patrones durante las fases tempranas de modelado del software. Se introduce el concepto de patrón conceptual y se expone la importancia que puede tener su aplicación en la construcción ágil de modelos conceptuales de calidad y, como consecuencia, en la obtención del producto que el cliente realmente desea. Se plantean los beneficios que para el proceso, sus participantes y los artefactos generados supondría la reutilización de patrones durante estas fases iniciales. Por último, para finalizar este bloque, se señala la necesidad de aumentar y aunar los esfuerzos en esta materia, haciéndose un recorrido por los escasos trabajos existentes.

En la última parte del capítulo, teniendo en cuenta distintos enfoques de trabajo, se muestra la situación actual del desarrollo de sistemas cooperativos en base a patrones y, en particular, la aplicación de éstos en etapas tempranas de modelado.

2. Los patrones de software

2.1. Origen

Los trabajos realizados por el arquitecto Christopher Alexander [Alexander et al., 1977; Alexander, 1979] han sido considerados por muchos autores como la principal fuente de inspiración para los patrones de software[1]. Aunque sus trabajos versan sobre la descripción y uso de patrones en el ámbito del diseño arquitectónico y la planificación urbana, muchas de sus ideas han sido trasladadas al campo de la ingeniería del software.

En realidad, el concepto de patrón, tal y como lo percibe Alexander, puede extrapolarse prácticamente a cualquier otra disciplina:

> *"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y también la solución para ese problema, de forma que esta solución pueda utilizarse un millón de veces sin tener que hacer dos veces lo mismo"*
>
> —Alexander et al., 1977.

Alexander buscaba estructuras arquitectónicas que tuvieran una influencia positiva sobre la gente y proporcionaran un mayor confort y calidad de vida. Estas estructuras podrían ser reutilizadas en distintos contextos, siempre y cuando se cumpliesen las condiciones del problema especificado. Una descripción detallada del problema y su solución, desglosada en distintas secciones para facilitar su comprensión y aplicación, define cada uno de sus patrones.

Los primeros indicios que se conocen sobre la aplicación del concepto de patrón en el desarrollo de software datan de finales de los 80's, cuando Ward Cunningham y Kent Beck construyen un conjunto de patrones para el desarrollo de interfaces de usuario en Smalltalk [Beck, 1988].

[1] Existen otros autores que no comparten esta tesis (véase p. ej. [Fowler, 1997, p. 6]).

A principios de los 90´s Jim Coplien elabora un catálogo de patrones de programación, también llamados *idioms* (v. sec. 2.3), específicos del lenguaje C++ [Coplien, 1992]. Por aquel tiempo, Erich Gamma elabora una tesis sobre desarrollo de software orientado a objetos [Gamma, 1991] y señala la importancia que tiene la documentación y reutilización de ciertas estructuras de diseño recurrentes. Un par de años más tarde, Gamma realiza un borrador de la primera versión de un catálogo de patrones de diseño [Gamma, 1993], el cual es la base para la publicación del primer libro [Gamma et al., 1995] que existe sobre patrones de software y en el que se presenta un conocido catálogo de patrones para el diseño orientado a objetos.

Al año siguiente se publica otra importante obra[2] [Buschmann et al., 1996] que utiliza una perspectiva diferente a su predecesor. Mientras que en Gamma et al. [1995] se utilizan los patrones para dar solución a problemas típicos de diseño orientado a objetos, los patrones proporcionados por éste último se aplican en una fase de diseño arquitectónico.

Para Gamma et al., un patrón consiste en la especificación abstracta de la estructura y/o comportamiento de un conjunto de clases/objetos que se relacionan. La aplicación de un patrón a un problema de diseño orientado a objetos consiste, básicamente, en la definición de aquellos elementos concretos del sistema (clases, objetos, relaciones, etc.) que satisfacen la especificación abstracta del patrón. Estos elementos formarían una instancia del patrón en el diseño del sistema que se está desarrollando. A diferencia de este tipo de patrones, los que se proponen en Buschmann et al. [1996] son de grano más grueso. En lugar de formar “microarquitecturas” de objetos/clases, éstos son descritos como “macroarquitecturas” de subsistemas predefinidos que se relacionan.

Debido al interés que han despertado en la comunidad informática, en general, y dentro de la ingeniería del software, en particular, han aparecido infinidad de patrones y tipologías (v. sec. 2.3) cuyo radio de acción se extiende por todo el ciclo de vida del software y hacia multitud de dominios de aplicación. Igualmente, desde principios de los 90´s, se vienen organizando

[2] Conocido como libro POSA (**P**attern-**O**riented **S**oftware **A**rchitecture: A System of Patterns).

diferentes encuentros que permiten el intercambio y discusión de ideas y experiencias en este campo (v. sec. 2.6).

2.2. Definición y conceptos relacionados

Aunque el concepto de *patrón*[3] puede ser más o menos intuitivo, ciertamente es difícil encontrar una definición comúnmente aceptada para el término *patrón de software*. Esto es así debido a que existen muchas comunidades interesadas en su estudio y con perspectivas muy diferentes, las cuales dependen fundamentalmente de su ámbito de aplicación. En cualquier caso, vamos a hacer un recorrido por algunas de las definiciones más notables que aparecen en la literatura.

Una de las definiciones más famosas es la que realiza Gamma et al. [1995]:

> *"Una descripción de clases y objetos que se comunican entre sí dispuestos para resolver un problema de diseño general en un contexto particular."*
>
> —Gamma et al., 1995

Como se puede comprobar, esta definición está claramente enfocada hacia un determinado tipo de patrones, en concreto, patrones para el diseño orientado a objetos.

Una definición mucho más universal y abstracta es la que proponen Riehle y Züllighoven [1996]:

> *"Un patrón es la abstracción de una forma concreta que se repite en contextos específicos no arbitrarios."*
>
> —Riehle y Züllighoven, 1996

[3] Según el Diccionario de la Real Academia de la Lengua Española [Real Academia Española, 2003], un *patrón* se define como *"modelo que sirve de muestra para sacar otra cosa igual"*.

La definición, en este caso, no tiene porqué estar vinculada exclusivamente al ámbito del desarrollo de software, pudiendo ser igualmente válida en muchos otros ámbitos.

De manera semejante, otra definición bastante genérica es la que realiza Martin Fowler [1997, p. 8] sobre este mismo término:

> *"Un patrón es una idea que ha sido útil en un contexto práctico y probablemente será útil en otros."*
>
> —Fowler, 1997.

Alexander [1979, p. 247] destaca varias particularidades que merece la pena comentar:

> *"Cada patrón es una regla compuesta por tres partes, la cual expresa una relación entre un cierto contexto, un problema y una solución.*
>
> *Como un elemento en el mundo, cada patrón es una relación entre un cierto contexto, un determinado sistema de fuerzas que ocurren repetidamente en ese contexto, y una cierta configuración espacial que permite que las fuerzas se resuelvan.*
>
> *Como un elemento del lenguaje, un patrón es una instrucción que muestra como esa configuración espacial puede ser usada, una y otra vez, para resolver el sistema de fuerzas dado, siempre que el contexto sea relevante.*
>
> *El patrón es, en resumen, al mismo tiempo una cosa, que ocurre en el mundo, y una regla que nos dice cómo crear esa cosa, y cuándo debemos crearla. Es a la vez un proceso y una cosa. La descripción de una cosa que está viva, y una descripción del proceso que genera esa cosa."*
>
> —Christopher Alexander, 1979.

Según Alexander, el problema se da en un contexto determinado y en presencia de fuerzas o intereses que se dan reiteradamente y que compiten entre sí. La solución que el patrón propone debería encargarse de balancear o resolver dicho sistema de fuerzas de la manera más apropiada para ese contexto.

Un patrón puede formar parte de un lenguaje como una instrucción que sintetiza su esencia y facilita la transmisión del conocimiento y el entendimiento entre las personas. Mediante la combinación de varios de estos patrones se podrían componer sentencias más complejas con las que poder hacer frente a problemas de distinta envergadura.

En el último párrafo se subrayan dos propiedades muy importantes, el carácter instructivo y constructivo de todo patrón. También se puede observar cómo para Alexander un patrón y su instancia son la misma cosa. En relación con esto último, Coplien [1996] dice:

> *"Podría decirte cómo hacer un vestido especificando la ruta de unas tijeras sobre la tela en términos de ángulos y longitudes de corte. O bien, podría darte un patrón. Leyendo la especificación, podrías no tener idea de lo que estabas construyendo o si lo has construido correctamente al terminar. El patrón presagia el producto: es la regla para hacer la cosa, pero también es, en muchos aspectos, la propia cosa."*
>
> —James O. Coplien, 1996.

Soluciones generales, algoritmos, heurísticas, etc., se presentan a veces como si fueran patrones. La polémica sobre lo que un patrón debería ser, o no ser, está servida. En este sentido, Winn y Calder [2002] realizan una serie de consideraciones sobre los requisitos que debería cumplir todo patrón[4]. Según estos autores, un patrón de diseño debería:

- Implicar un artefacto.

[4] En principio, el artículo se centra en los patrones de diseño, aunque muchas de las cuestiones tratadas en este artículo podrían extenderse a otros tipos de patrones.

- Enlazar distintos niveles de abstracción ayudando a los diseñadores a hacer conexiones entre los diferentes niveles.

- Ser funcional y no funcional. Debería ser inherentemente funcional, ya que documenta una solución para un problema, y no funcional, ya que debería discutir la viabilidad de la solución para adaptarla efectivamente a otro contexto.

- Ser reconocible en la solución desarrollada.

- Capturar los "hot spots" [Pree, 1995] de un sistema, es decir, los patrones deberían capturar las partes de la solución que son adaptables conforme evoluciona el sistema.

- Formar parte de un lenguaje. Un patrón puede conectarse y estar compuesto por otros patrones. Así pues, debería formar parte de un conjunto de patrones interrelacionados. El todo es más que la suma de las partes.

- Validarse por el uso. La reiterada presencia en sistemas reales confirmaría su utilidad. Por ejemplo el catálogo de patrones de Gamma et al. [1995] presenta una sección de usos conocidos.

- Establecerse en un área o campo de aplicación determinado.

- Capturar una gran idea.

Como hemos visto, un patrón es aplicable a un problema individual. Sin embargo, son muchos los problemas que suelen aparecer, a menudo relacionados entre sí, durante las distintas fases de desarrollo de un producto de software. Por esta razón, y para facilitar su aplicación, éstos se suelen compilar, organizar y relacionar.

Un *catálogo de patrones* es una colección de patrones dentro de un dominio específico. El conjunto de patrones se dota de alguna estructura u organización para facilitar la selección de éstos durante el desarrollo de software. Es posible consultar un listado de enlaces a catálogos de patrones en Hillside Group [2007e].

A diferencia de un catálogo, un *lenguaje de patrones* [Alexander et al., 1977] incluye, además, una red de conexiones entre los patrones.

Normalmente dispone de información acerca de cuándo y cómo pueden combinarse varios patrones, con objeto de resolver problemas más grandes y complejos que aquellos que son tratados por cada patrón individualmente. Se podría ver como un vocabulario, formado por los nombres de los patrones, junto a una gramática que indica cómo combinar éstos para construir sentencias más complejas. Una relación de enlaces que se corresponden con trabajos sobre lenguajes de patrones puede consultarse en Hillside Group [2007b].

No obstante, muchos autores emplean indistintamente un término u otro para referirse a una colección estructurada de patrones interrelacionados dentro de un dominio específico. Otros optan por usar el término *sistema de patrones* [Buschmann et al., 1996].

2.3. Clasificación

Hay muchas formas posibles de catalogar los patrones, prácticamente tantas como tipos diferentes de problemas podemos encontrar a lo largo del ciclo de vida del software. No obstante, existen algunas taxonomías ampliamente conocidas que merece la pena comentar.

En primer lugar podemos hablar de la clasificación realizada por Buschmann et al. [1996]. En ésta se diferencian tres tipos relacionados de patrones:

- *Patrones de arquitectura*. Expresan un esquema u organización básica de la estructura de un sistema software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y pautas para organizar las relaciones entre ellos.

- *Patrones de diseño*. Proporcionan un esquema para refinar los subsistemas o componentes de un sistema software, o las relaciones entre ellos. Describen estructuras recurrentes de componentes que se comunican y resuelven un problema de diseño dado en un contexto particular.

- *Patrones de código (Idioms)*. Son los patrones de más bajo nivel, específicos para un lenguaje de programación determinado. Un "idiom" describe cómo implementar aspectos de componentes o de

relaciones entre ellos usando las características de un lenguaje dado [Coplien, 1992].

Otra importante categorización en la que, hasta donde nosotros sabemos, se introduce por primera vez el término patrón conceptual, foco de atención en esta tesis, es la que realizan Riehle y Züllighoven [1996]. Aquí los autores distinguen entre:

- *Patrones conceptuales*. Usados para facilitar la construcción del modelo conceptual de un sistema. Se describen por medio de términos y conceptos de un dominio de aplicación particular (dominio del problema). Guían la percepción que se tiene de un dominio de aplicación y ayudan tanto a comprenderlo como a describirlo. Según los autores, éstos deberían apoyarse en metáforas del dominio de aplicación.

- *Patrones de diseño*. Usados en la construcción del modelo de diseño. Emplean conceptos pertenecientes al dominio de la solución. Varían su escala desde "macro-arquitecturas", como pueden ser los patrones de arquitectura según Buschmann et al. [1996], hasta "micro-arquitecturas", como los de Gamma et al. [1995]. Los autores sostienen que el paso del modelo conceptual al modelo de diseño se puede facilitar si es posible establecer una relación clara entre patrones de diseño y patrones conceptuales.

- *Patrones de programación*. Descrito por medio de constructores pertenecientes a algún lenguaje de programación concreto. Similar a los patrones de código o idioms [Coplien, 1992]. Asimismo, si éstos se relacionan con los de diseño, la etapa de implementación se podría agilizar.

En consonancia con la idea de patrón conceptual, Fowler [1997] acuña el término *patrón de análisis* para referirse a aquellos patrones que son aplicables durante la fase de análisis de un sistema. Los patrones de análisis capturan modelos conceptuales, vinculados a un dominio de aplicación concreto, que pueden ser reutilizados en distintas aplicaciones (reutilización del conocimiento del dominio). Su utilización tiene un objetivo doble, por una parte, acelerar el desarrollo de los modelos de análisis abstractos que capturan los principales requisitos del problema, y por otra, facilitar la

transformación del modelo de análisis en el modelo de diseño [Geyer-Schulz y Hahsler, 2002].

No obstante, los patrones también pueden ser clasificados en virtud de su dominio de aplicación, dando lugar, como es lógico, a una clasificación mucho más extensa. Aunque hay patrones que son de carácter general, esto es, independientes del dominio, muchos otros son creados para resolver problemas en ámbitos concretos de aplicación. Por ejemplo, podemos hablar de patrones especializados en: sistemas distribuidos [DeBruler, 1995], sistemas reactivos [Aarsten et al., 1996; Meszaros, 1996], sistemas cooperativos [Lukosch y Schümmer, 2006; Schümmer, 2002; Martin et al., 2001], educación [Anthony, 1996], etc.

Otra interesante tipificación es la que atiende a la etapa o aspecto que estamos considerando dentro del proceso de desarrollo. La estructuración de un catálogo a partir de este criterio podría facilitar en gran medida la selección y aplicación del patrón más adecuado en cada instante del proceso. Así, por ejemplo, podemos distinguir entre patrones de análisis [Fowler, 1997; Geyer-Schulz y Hahsler, 2002], arquitectura [Buschmann et al., 1996; Shaw, 1995; Meunier, 1995], diseño [Gamma et al., 1995], programación [Coplien, 1992, 1999; Cargill, 1996], proceso [Coplien, 1995], organización [Fowler, 1996; Harrison, 1996; Weir, 1998], tiempo [Carlson et al., 1999], interfaz de usuario [Molina et al., 2002a, 2002b; Bradac y Fletcher, 1998; Granlund et al., 2001], hipermedia [Rossi et al., 1996a; Bernstein, 1998; Nanard y Nanard, 1999; German y Cowan, 2000], interacción persona-ordenador [Borchers, 2001; Bayle et al., 1998; Rossi et al., 1999; Tidwell, 1998], seguridad [Yoder y Barcalow, 1999], etc.

No obstante, dentro de cada catálogo, también es posible usar otros niveles de agrupación, facilitando la selección del patrón más adecuado en cada momento. Por ejemplo, Gamma et al. [1995, p. 10] clasifica los patrones según su propósito (creación, estructura o comportamiento) y ámbito de aplicación (clase u objeto), estableciendo una serie de relaciones interesantes entre ellos, como por ejemplo:

- Patrones que se usan frecuentemente juntos (ej. *Composite-Iterator* o *Composite-Visitor*).

- Patrones que son a menudo alternativos (ej. *Prototype* alternativo con *Abstract Factory*).

- Patrones con estructuras similares (ej. *Composite* y *Decorator*).
- Patrones relacionados [Gamma et al., 1995, p. 12].

2.4. Descripción

Los patrones van más allá de la mera exposición de una solución para un problema conocido. Un patrón debe reflejar convenientemente las razones por las cuales nos puede interesar su uso. Además, para facilitar su aplicación y comparación, la descripción debería ser realizada de una manera comprensible, clara y estructurada.

Meszaros y Doble [1998] presentaron un original e interesante trabajo sobre las recomendaciones que se deben seguir para la descripción de patrones. A través de un lenguaje de patrones, los autores describen y ayudan a aplicar las mejores prácticas utilizadas para su escritura.

Según estos autores, la descripción de un patrón está formada por una serie de elementos obligatorios, junto con otros opcionales. Éstos se materializan en un conjunto de secciones que dotan de estructura la descripción. Así, los aspectos esenciales que deberían contemplarse son:

- *Nombre*: Debe ser representativo y reflejar sucintamente su esencia. Va a formar parte del vocabulario que facilita la comunicación de las abstracciones.
- *Contexto*: Circunstancias bajo las cuales se da el problema. Impone una serie de restricciones a la solución. Responde a preguntas tales como, ¿en qué situaciones puede ser aplicado?, ¿cómo reconocer esas situaciones?. Muestra las precondiciones bajo las cuales el problema y su solución pueden ocurrir y para las que la solución es deseable.
- *Problema*: Muestra qué es lo que resuelve la aplicación del patrón.
- *Fuerzas*: Consideraciones, a menudo contradictorias, que se tienen en cuenta cuando se elige una solución para el problema.
- *Solución*: Describe cómo resolver el problema equilibrando las fuerzas. Debe expresarse genéricamente para que pueda ser aplicada en distintas situaciones. Se suelen detallar los elementos participantes y

las relaciones, tanto estáticas como dinámicas, que producen el resultado deseado. Puede incluir variantes que difieren según el contexto en el que se aplican. Normalmente se especifica mediante diagramas y prosa.

Entre los elementos considerados opcionales, los cuales pueden ser más o menos útiles dependiendo del patrón concreto que estamos describiendo, están:

- *Indicios*: Síntomas que pueden indicar que el problema existe.
- *Contexto resultante*: Consecuencias (buenas y malas) de la aplicación del patrón. Problemas nuevos que podrían surgir y cómo estos podrían resolverse con otros patrones.
- *Patrones relacionados*: Otros patrones que forman parte del mismo lenguaje de patrones y con los cuales se relaciona. Por ejemplo, otras soluciones para el mismo problema, patrones que han podido aplicarse antes o que podrían aplicarse después, aquellos que son alternativos, los que son aplicados simultáneamente, etc.
- *Ejemplos*: Ilustran la aplicación del patrón.
- *Código ejemplo:* Muestra cómo implementar el patrón.
- *Razón:* Cuenta detalladamente porqué funciona y porqué es apropiado el patrón para este problema.
- *Alias:* Otros nombres por los que es conocido el patrón.
- *Agradecimientos:* A quienes hayan contribuido significativamente en el desarrollo del patrón o de las técnicas descritas en éste.

Coincidiendo con Riehle y Züllighoven [1996], pensamos que la descripción de un patrón debería adaptarse a su tipo y dominio de aplicación específico, presentando de forma explícita sólo aquellas secciones que nos ayuden realmente a usar y entender mejor el patrón, y eliminando aquellas que se consideren superfluas o redundantes.

Uno de los formatos de descripción más famosos, conocido como formato del GoF[5], es el que proporciona Gamma et al. [1995]. Su estructura cuenta con las siguientes secciones: *nombre, clasificación, intención, alias, motivación, aplicabilidad, estructura, participantes, colaboraciones, consecuencias, implementación, código ejemplo, usos conocidos y patrones relacionados.*

Al igual que Gamma et al., la mayoría de los autores basan sus formatos en el llamado "formato canónico o Alejandrino[6]". Las secciones que componen dicho formato las listamos a continuación y, para resaltar su similitud, las comparamos con el formato del GoF:

- *Nombre*. Puede además incluir un *alias,* si el patrón es conocido por otro nombre, y una *clasificación*. Se corresponde con las secciones *nombre, clasificación* y *alias* del formato utilizado por el GoF.

- *Problema*. Equivale a la sección *intención* del GoF.

- *Contexto*. Semejante a las secciones *escenario* y *aplicabilidad* del GoF.

- *Fuerzas*. Coincide con la sección *motivación* del GoF.

- *Solución*. En el formato del GoF esta sección se divide en las secciones de *estructura* (diagramas de clases y diagramas de interacción que ilustran las secuencias de peticiones y colaboraciones entre objetos), *participantes* (clases y/o objetos que participan en el patrón y sus responsabilidades) y *colaboraciones* (cómo colaboran los participantes para llevar a cabo sus responsabilidades).

- Ejemplos. En el formato del GoF usos conocidos en sistemas reales, código ejemplo e implementación.

- Contexto resultante. Las secciones consecuencias y patrones relacionados del GoF desempeñan este papel.

---

[5] **G**roup **o**f **F**our (la Banda de los Cuatro), haciendo clara alusión a los cuatro autores (E. Gamma, R. Helm, R. Johnson y J. Vlissides) del famoso libro [Gamma et al., 1995].

[6] Empleado por C. Alexander (citado en la sección 2.1. de este mismo capítulo) para la descripción de sus patrones.

- *Razón*. Principalmente, la sección *motivación* del GoF contempla este tipo de cuestiones, aunque en realidad está prácticamente dispersa por casi todas las secciones.

- *Patrones relacionados.* Esta sección coincide con su homónima en el GoF.

- *Usos conocidos.* Equivalente en el formato del GoF a su homónima.

2.4.1. Especificación de la solución

Como veremos en detalle dentro del siguiente capítulo (v. cap. IV, sec. 3), el grado de formalización con el que un patrón debería ser descrito, y en particular la solución que éste propone, ha dado lugar a diversos trabajos y posiciones divergentes.

Si bien una descripción puramente narrativa puede dar lugar a interpretaciones imprecisas, en el otro extremo, una especificación demasiado formal puede ser difícil de comprender, obstaculizando a menudo la aplicación del patrón. Esto es especialmente crítico en el caso de los equipos de desarrollo multidisciplinares, donde algunos de sus miembros podrían carecer de los conocimientos mínimos necesarios para poder entender ciertas notaciones formales.

La utilización de algún tipo de formalismo que permita la descripción precisa y sin ambigüedades de un patrón es la base para la construcción de herramientas que ayuden en la definición, aplicación y validación de patrones en contextos determinados. Sin embargo, la utilización de patrones durante el modelado de un sistema requiere de cierta flexibilidad, así como de altas dosis de intuición y experiencia por parte del diseñador, lo que supone un obstáculo para que este proceso sea totalmente automático. No obstante, se han construido diversas herramientas [Hillside, 2007c] que pueden ayudar al diseñador a aplicar patrones "tipo" bajo unas condiciones "ideales". La búsqueda de una notación adecuada para la especificación precisa de patrones de diseño orientado a objetos y su integración dentro de una herramienta para soportar su aplicación es una labor que ha sido emprendida en diversos trabajos [Eden, 2000; Meijers, 1996; Lauder y Kent, 1998; Riehle, 1996; Le Guennec et al., 2000; Isla, 2001, 2002; Isla y Gutiérrez, 2003].

Bajo nuestra perspectiva, y subscribiendo la cita de Coplien [1996] que hemos incluido en este mismo capítulo (v. sec. 2.2), creemos que todo patrón de software debería presagiar el resultado de su aplicación, de forma que permita al usuario de un patrón, casi de un simple "vistazo", comprender su esencia. En consecuencia, el modelo que un patrón propone como solución debería servir de muestra para construir de manera ágil y flexible el modelo que estamos buscando. Con este propósito, en el capítulo IV, definimos un profile [Isla et al., 2005a] que extiende el lenguaje UML [OMG, 2003] a través de un conjunto mínimo de elementos, permitiéndonos representar de manera simple, intuitiva y fácil de comprender, tanto la cara externa como interna de un patrón de software. De acuerdo con este profile, consideramos los patrones como modelos genéricos parametrizados (de cualquier tipo) que actúan como plantillas flexibles para la construcción y/o descripción de familias de modelos semejantes (las instancias del patrón). Para conseguir el grado de flexibilidad necesario, aplicamos restricciones de multiplicidad, optatividad, agrupamiento o incertidumbre sobre ciertos elementos del patrón, limitando así las posibles variantes de sus instancias. Las instancias se crean/describen mediante la ligadura (uno a uno, uno a varios) de cada uno de sus parámetros con cada uno de los elementos de estas instancias (argumentos), siempre y cuando éstos cumplan con las restricciones que el patrón impone.

2.5. Integración en los procesos de software

2.5.1. Introducción

Desarrollar software es un proceso indudablemente complejo. Esto es así por varias razones:

- El software es complejo por naturaleza.
- Normalmente es necesario trabajar en grupo.
- Suelen existir problemas de comunicación con los usuarios, incluso entre los propios miembros del equipo de desarrollo.
- El software evoluciona y se debe facilitar su mantenimiento.

- Los proyectos pueden tener un tamaño considerable.

Al mismo tiempo hay un creciente interés por construir software fiable, gastando la menor cantidad de tiempo y dinero posible. La ingeniería del software pretende precisamente satisfacer dicha demanda, estableciendo y aplicando principios y métodos propios de la ingeniería. La reutilización de componentes de software es una de las actividades encaminadas a la consecución de dicho objetivo.

Tradicionalmente esta reutilización ha estado basada fundamentalmente en el código. El uso de la tecnología orientada a objetos ha potenciado y favorecido decisivamente este propósito. Sin embargo, otros tipos de artefactos (especificaciones y modelos de requisitos, diseños, documentación, etc.), aunque sí de una manera informal, no han sido reutilizados sistemáticamente; máxime sabiendo que la construcción de éstos repercute enormemente sobre la calidad del producto final y el tiempo de desarrollo. La intención de los patrones es dar un paso más allá a través de la reutilización sistemática del conocimiento experto empleado en la creación de todos estos artefactos.

La complejidad inherente al desarrollo de software hace de éste un caldo de cultivo idóneo para la identificación y aplicación de numerosos patrones en todos los dominios existentes.

Aparte de la ventaja que supone la reutilización del conocimiento que el experto emplea para afrontar más ágilmente los problemas que se encuentra durante la construcción de todos estos artefactos, el equipo de desarrollo se beneficia, por otro lado, de la posibilidad de compartir un vocabulario común, compuesto por los nombres de los patrones, que favorece el entendimiento entre ellos y con los usuarios finales. Adicionalmente, la inclusión de patrones mejora la documentación y, por ende, los modelos que se generan, desde el punto de vista de su legibilidad, comprensión y mantenimiento.

En definitiva, el desarrollo de software en base a patrones permite aliviar la complejidad intrínseca de su desarrollo y contribuye en la obtención de software más rentable y de mejor calidad.

2.5.2. Beneficios

La integración de patrones en los procesos y métodos de desarrollo de software tiene una serie de ventajas, las cuales han sido evidenciadas y documentadas en diversas ocasiones dentro de la industria del software [Schmidt y Stephenson, 1995; Schmidt, 1995a; Brown, 1996; Beck et al., 1996]:

- Se reutiliza el conocimiento experto.

  El equipo de desarrollo va a disponer de una especie de memoria corporativa, o base de conocimiento, que incorpora la experiencia previa en otros proyectos, mediante la inclusión de una selección de las mejores decisiones tomadas durante las distintas fases de desarrollo. Este conocimiento podrá ser reutilizado en posteriores proyectos, ayudando a no cometer los mismos errores y facilitando la formación e integración de nuevos participantes en el equipo.

- Se facilita la detección de problemas y su modelado.

  El interés de los patrones no reside únicamente en la posibilidad de proporcionar una solución a un problema concreto. A menudo, lo que resulta realmente complicado es reconocer dichos problemas en determinados contextos. Los patrones no sólo procuran la especificación de una solución, también proveen la descripción del problema que ayudan a resolver y las condiciones bajo las cuales éste sucede, posibilitando así su modelado e identificación en diferentes situaciones.

  El uso de catálogos de patrones ayuda en la detección de estos problemas, ya que éstos clasifican e interrelacionan patrones aplicables en situaciones específicas.

- Se provee un vocabulario común.

  Compartimos un mismo vocabulario que nos permite comunicar y razonar sobre conceptos complejos relacionados con las soluciones y problemas que aparecen.

  Cuando se trabaja en equipo, es esencial que todos sus integrantes compartan un fondo común de conocimientos mínimos, compuesto

por todos aquellos conceptos y experiencias que hacen falta para poder alcanzar las metas que el grupo se propone.

Aún cuando el grupo posea cierta homogeneidad desde un punto de vista de la formación de sus componentes, esta necesidad se mantiene. La misma formación no implica necesariamente tener la misma experiencia, ni incluso, a veces, compartir exactamente el mismo vocabulario. Sin embargo, cuando se trata de equipos de desarrollo heterogéneos, formados por profesionales procedentes de distintas disciplinas, como puede ocurrir en el caso del desarrollo de software según una filosofía centrada en el usuario o basada en un diseño participativo [Schuler y Namioka, 1993], la necesidad de establecer este fondo común es vital. Esto sucede, por ejemplo, durante el desarrollo de los sistemas cooperativos, donde la colaboración entre usuarios y expertos en informática, sociología, psicología, organización del trabajo, etc., es imprescindible para obtener un producto que recoja las necesidades reales del cliente.

Compartir un conjunto de patrones definidos sobre un dominio determinado puede servir como nexo de unión o fondo de conocimientos entre las personas involucradas. Desde esta perspectiva, los patrones pueden verse como una lengua franca [Erickson, 2000a] que facilita el entendimiento entre los distintos participantes durante el proceso de desarrollo. Entre especialistas de un mismo campo, el hecho de poder compartir soluciones a problemas comunes supone un valor añadido; pero entre especialistas procedentes de distintas disciplinas, el papel de los patrones como lengua común cobra un protagonismo determinante.

- Se mejora la documentación.

  La propia descripción de los patrones usados durante el desarrollo de un sistema software forma parte de la documentación asociada a un proyecto. Esto permite que durante la generación de la documentación se puedan referenciar, mediante el nombre del patrón, las soluciones empleadas sin tener que entrar en el detalle, lo que simplifica y mejora la comprensión.

  La notación utilizada en la creación de los modelos correspondientes debería contemplar los patrones como un elemento de modelado de

primer orden. Su inclusión permitiría la elevación del nivel de abstracción de los modelos generados.

La mejora de la legibilidad, comprensión y mantenimiento de la documentación en general y, por lo tanto, de los modelos que se generan, es una consecuencia directa del uso de patrones.

2.5.3. Consideraciones para su implantación

Para poder obtener los beneficios descritos anteriormente entendemos que es necesario:

- Disponer de una amplia colección de patrones que abarque todo el proceso.

  El equipo de desarrollo debería estar en posesión de patrones especializados en el dominio de aplicación del producto que se va a construir. Idealmente, la colección debería ser lo suficientemente extensa como para abarcar las distintas etapas del ciclo de vida. Así, estableciendo las conexiones adecuadas entre patrones aplicables en etapas contiguas, se podría agilizar la transición entre ellas.

- Organizarlos y estructurarlos para favorecer su gestión.

  Para ello, se pueden seguir las siguientes pautas:

  - Mantenimiento de una estructura y formato de descripción uniformes.

    Todos los patrones deberían presentar las mismas secciones y reflejar los aspectos esenciales que ayuden en la identificación y aplicación del patrón. Para su especificación puede ayudar el uso de una estructura tabular.

    Por otro lado, el lenguaje de descripción del patrón debería ser sencillo y homogéneo, empleando un vocabulario próximo a todos los posibles usuarios. La creación de un diccionario de términos, la definición de las convenciones tipográficas adoptadas y la aclaración de las notaciones utilizadas van a favorecer esta circunstancia.

- Clasificación de los patrones en función de criterios que faciliten su selección.

  Es necesario organizar los patrones de manera que cuando reconozcamos un problema, sea rápida la localización del patrón que ayude a resolverlo. Además, esta clasificación da lugar a una ordenación del espacio del problema, lo que facilita su reconocimiento.

  Puede ser útil agruparlos según la fase del ciclo de vida donde se aplican, el tipo de problema que intentan resolver, el nivel de abstracción, etc.

- Establecimiento de relaciones de colaboración entre ellos.

  Es deseable instaurar las interconexiones oportunas de cara a poder afrontar determinados problemas mediante la combinación de patrones. Estas relaciones pueden ser muy útiles para la transición de un nivel de abstracción a otro, refinando unos patrones mediante otros más detallados. Esto puede desembocar en la creación de un catálogo o, más apropiadamente, un lenguaje de patrones, que facilite la combinación de distintos patrones que colaboran en la resolución de problemas más complejos.

- Utilización de algún tipo de herramienta que ayude en la gestión de los patrones.

  Una herramienta que implemente todas estas cuestiones de formato que han sido tratadas, así como las distintas relaciones y clasificaciones que se pueden dar entre los patrones existentes, podría agilizar su consulta, mantenimiento y aplicación.

  La consulta debería estar accesible en todo momento y posibilitar el empleo de diferentes criterios de búsqueda. Por ejemplo, sería interesante ver los patrones relacionados con algún tipo de problema o etapa de desarrollo concreta, los patrones que de alguna forma se relacionan con otro patrón (bien por refinamiento, generalización, composición, proximidad, similitud, etc.), los que fueron aplicados en algún proyecto anterior, enlaces de un patrón con casos reales de aplicación en proyectos pasados, etc.

Es indispensable que las altas, bajas y modificaciones de patrones estén controladas para asegurar la calidad de la base de conocimiento.

Una herramienta colaborativa podría facilitar la edición compartida de patrones, la realización de reuniones electrónicas para discutir y refrendar las solicitudes de creación o cambio, la habilitación de algún tipo de servicio (síncrono o asíncrono) para el intercambio de impresiones con otras personas del equipo que ya tengan experiencia con el uso de algún patrón concreto, etc.

- Integrarlos con los métodos de desarrollo utilizados.

  Dicha integración conlleva básicamente el establecimiento de:

  - Una representación adecuada para éstos dentro de los modelos que se generan.

    Los patrones deberían formar parte de la notación como elementos de modelado de primer orden. Dentro de los modelos deberían ser percibidos e interpretados correctamente, tanto si son usados como caja negra o como caja blanca. En ambos casos debería quedar patente el papel que desempeña cada uno de los elementos del contexto dentro del patrón. Los patrones pueden ser usados como simples guías de modelado con capacidad para orientar al diseñador en la elección de los distintos elementos que deberían participar en el modelo, tanto desde el punto de vista de su estructura como de su comportamiento. Esto se podría facilitar a través de una herramienta que ayude en la aplicación automática o semiautomática de patrones en contextos específicos.

    Su integración dentro del lenguaje de modelado propicia la elevación del nivel de abstracción de los modelos desarrollados, mejorando la legibilidad, comprensión y mantenimiento de éstos.

  - La conexión de los distintos problemas que podemos encontrar en cada una de las etapas del método de desarrollo con aquellos patrones que pueden ser útiles para su resolución.

- Formar al equipo de desarrollo en la utilización de esta técnica y tomar conciencia de sus ventajas.

  Es imprescindible que el equipo experimente y tome conciencia de los efectos positivos que sobre los productos construidos y el proceso de desarrollo proporciona la utilización de patrones.

2.6. Foros especializados

Desde la organización del primer taller de trabajo sobre patrones de software [Anderson y Coad, 1994] hasta el presente, se han sucedido una serie de encuentros en el marco de las prestigiosas conferencias *OOPSLA* (*Object-Oriented Programming, Systems, Languages and Applications*) que han servido de soporte para la discusión e intercambio de ideas y experiencias sobre este tema.

Adicionalmente, numerosos congresos sobre informática incluyen entre sus tópicos, la aplicación y búsqueda de patrones relacionados con alguno de los temas de interés del congreso.

Asimismo, existen varias conferencias internacionales cuyo tema central son los patrones de software. Las conferencias *PLoP™* (*Pattern Languages of Programs*)[7] que patrocina *The Hillside Group*[8] [Hillside, 2007d] se encuentran entre las más famosas.

Algunos de los artículos presentados en dichas conferencias (*PLoP, EuroPLoP, Koala PLoP, Mensore PLoP, SugarLoafPLoP, UP, ChiliPLoP, Viking PLoP*) se recopilan en la serie *Pattern Languages of Program Design* [Coplien y Schmidt, 1995; Vlissides et al., 1996; Martin et al., 1998; Harrison et al., 1999; Manolescu et al., 2006], cuyas obras son un referente para la comunidad interesada en los patrones de software.

[7] PLoP es una marca registrada de The Hillside Group, Inc.

[8] Corporación sin ánimo de lucro interesada en la difusión de los patrones de software entre la comunidad informática. Este grupo mantiene uno de los principales portales especializados que existen sobre patrones de software [Hillside, 2007a].

3. La aplicación de patrones durante las etapas tempranas de modelado de sistemas software

3.1. Introducción

Entre los problemas más complejos y decisivos que es necesario abordar durante el desarrollo de un sistema software están la obtención, análisis y especificación de sus requisitos. Gran parte de esta dificultad reside en que:

- El analista o ingeniero de requisitos[9] debe comprender el dominio de aplicación[10]. Sin embargo, esta persona puede no tener el conocimiento y la experiencia suficiente como para entenderlo convenientemente.

- Para llegar a comprender el dominio y descubrir los requisitos reales del sistema, el analista debe interactuar con los llamados stakeholders[11] (usuarios finales, directivos, personal técnico de los sistemas relacionados, proveedores, etc.). Sin embargo, estas personas suelen expresar los requisitos mediante sus propios términos, usando un conocimiento implícito de su propio trabajo.

- Diferentes stakeholders pueden tener requisitos distintos y expresarlos de manera diferente.

A lo largo de este proceso el analista desarrolla uno o varios modelos, denominados *modelos de análisis o conceptuales*, que ayudan a entender y simplificar los problemas que aparecen en el dominio de aplicación. Éstos complementan la especificación de requisitos realizada en lenguaje natural, representando de una forma más técnica y detallada, el contexto, estructura y comportamiento del sistema.

---

[9] Denominación comúnmente usada para designar al Ingeniero de software que desempeña las tareas de obtención, análisis y especificación de requisitos.

[10] También conocido como dominio del problema o del negocio.

[11] "*Cualquier persona que tiene influencia directa o indirecta sobre los requerimientos del sistema*" [Sommerville, 2002].

Estos modelos tienen un gran valor, principalmente debido a que, por un lado, son una abstracción que representa la visión (modelo mental) que tiene el analista sobre el dominio de aplicación, y por otro, son usados como puente entre el análisis y el diseño del sistema a desarrollar. Por ello, es de máxima importancia que dichos modelos estén bien construidos y sean válidos, es decir, que definan correctamente el sistema que desea el cliente.

En un ciclo de vida clásico, los errores cometidos durante estas etapas tempranas de modelado suelen tener un coste de reparación muy alto, por encima de aquellos cuyo origen está en el diseño o la implementación, ya que, por lo general, implica tener que rediseñar e implementar nuevamente las partes del sistema afectadas por los cambios. Por lo tanto, cuanto antes se detecten estos errores, menor será el coste de reparación.

Una prometedora técnica que permite acelerar y mejorar el desarrollo de estos modelos, a la vez que ayuda a reducir el número de errores cometidos, consiste en la aplicación de *patrones de análisis.*

Según Martin Fowler, "los *patrones de análisis* son grupos de conceptos que representan una construcción común en el modelado del negocio. Éstos pueden ser relevantes para un sólo dominio, o pueden abarcar muchos dominios" [Fowler, 1997, p. 8].

Como ya se ha comentado en este capítulo, los patrones de análisis son también ampliamente conocidos como *patrones conceptuales*[12]. Riehle y Züllighoven [1996] introdujeron este término para referirse a aquellos patrones aplicables en la construcción de un modelo conceptual para un dominio de aplicación concreto.

Claramente, los patrones conceptuales están centrados en el *qué* (dominio del problema) y no en el *cómo* (dominio de la solución). Representan

[12] A lo largo de esta tesis usaremos preferentemente este término. No obstante, es necesario señalar que, aunque la mayoría de los autores no hacemos distinción alguna entre ambas denominaciones, hay quienes prefieren diferenciarlos. Por ejemplo, Cabot y Raventós [2004] usan el término *patrón conceptual* para referirse a aquellos patrones que representan estructuras específicas de conocimiento encontradas en diferentes dominios, mientras que usan el término patrón de análisis para aquellos que especifican un conocimiento genérico y dependiente del dominio.

y documentan escenarios comunes que aparecen en el dominio que comparten analistas y stakeholders para hablar sobre los requisitos del sistema que se va a construir, aislándose de aquellos conceptos propios del diseño o la implementación.

El uso de una aproximación al modelado en base a estas "*convenciones de pensamiento*" [Hay, 1996] va a mejorar y facilitar la comunicación y análisis del dominio del problema.

El objetivo, por tanto, de los patrones conceptuales/análisis es la comunicación, el entendimiento y la reutilización durante el proceso de análisis y modelado conceptual, del conocimiento relacionado con aquellas abstracciones clave que son recurrentes en ciertos dominios.

3.2. Beneficios que aporta la reutilización de patrones conceptuales

Las ventajas que supone la aplicación de patrones conceptuales durante las etapas tempranas de modelado de un sistema software se pueden resumir en las siguientes:

- Guían la percepción que se tiene de un dominio de aplicación y ayudan a encontrar, comprender y describir los problemas comunes que aparecen dentro de éste.

- Proporcionan un lenguaje de comunicación o *lengua franca* [Erickson, 2000a] que facilita el entendimiento entre el personal técnico y las personas relacionadas con el dominio del problema. Para ello, tal y como apuntan Riehle y Züllighoven [1996], es conveniente que los patrones sean descritos por medio de términos y conceptos propios del dominio del problema y apoyarse en metáforas de éste.

- Optimizan el análisis por medio de la reutilización de modelos conceptuales que ya han demostrado su validez en dominios semejantes, mejorando la calidad del software producido. Los modeladores sólo tienen que modificar y combinar estructuras existentes, más que crear nuevas desde cero [Hay, 1996].

- Simplifican la construcción de los modelos conceptuales y facilitan su validación. Los patrones ayudan a reducir o eliminar los errores de modelado más gruesos, puesto que los elementos básicos del modelo están ya definidos [Hay, 1996].

- Facilitan la comprensión, comunicación, documentación y mantenimiento de los modelos generados cuando los patrones utilizados se identifican claramente dentro de éstos.

- Reducen el tiempo de desarrollo significativamente y mejoran la interfaz entre la etapa de análisis y de diseño [Geyer-Schulz y Hahsler, 2002].

A pesar de todo, los patrones no están todavía ampliamente usados durante las etapas tempranas de modelado, a diferencia de lo que ocurre en etapas posteriores. Entre las posibles causas de este retraso creemos que las más importantes son:

- La ausencia de colecciones de patrones para dominios específicos.

- La carencia de patrones realmente útiles y con el nivel de abstracción adecuado que ayuden a construir este tipo de modelos.

- La falta de integración con metodologías concretas que faciliten la definición de los patrones y ordenen su aplicación sistemática.

Justamente, esta tesis pretende dar solución a estas necesidades, mediante la integración de patrones conceptuales en la metodología AMENITIES.

3.3. Una panorámica de los principales trabajos realizados

Es necesario señalar que el número de publicaciones relacionadas con este tema es de momento escaso, sobre todo en comparación con la atención que se ha prestado a otros tipos de patrones, como por ejemplo los de diseño. Además, los trabajos existentes están muy disgregados, abordando la aplicación temprana de patrones desde muy diversas perspectivas. Aumentar y aunar los esfuerzos en este terreno es una tarea que no podemos eludir. No

obstante, se han desarrollado algunos trabajos significativos en este terreno que, a continuación, vamos a presentar y analizar.

Así, por ejemplo, hay autores que han concentrado sus energías en la exploración de métodos y herramientas para dar soporte al proceso de modelado conceptual en base a patrones.

Este es el caso de Petia Wohed [2000a], quien sugiere un método basado en una estructura de navegación guiada por preguntas que proporciona los patrones conceptuales más adecuados durante el proceso de análisis de sistemas de información. Desarrollan un prototipo de herramienta dependiente del dominio para automatizar la construcción y refinamiento de esquemas conceptuales a partir de las respuestas dadas por el usuario a las preguntas previamente diseñadas. Esta aproximación es aplicable únicamente a dominios muy concretos y simples, lo que permite limitar el número de preguntas sobre el dominio y recolectar la información necesaria para la construcción de los esquemas conceptuales. En [Wohed, 2000b] se traslada esta aproximación a otro dominio, el de la especificación de transacciones, basándose en los patrones de transacción de Coad et al. [1995].

En Fernández y Yuan [2000] se describen conjuntos mínimos de casos de uso que especifican mini-aplicaciones genéricas básicas. Los casos de uso son seleccionados de manera que la aplicación se pueda usar en diferentes situaciones. Los autores consideran que éste es un "nuevo" tipo de patrón de análisis, al cual denominan SAP (Semantic Analysis Pattern). Describen un método basado en SAPs para construir, total o parcialmente, modelos conceptuales de forma sistemática. El método parte de la premisa de que ya disponen de un completo catálogo con los patrones de análisis más interesantes de la literatura. Básicamente consiste en buscar aquellos SAPs que cubran algunos de los requerimientos de la aplicación, después buscar otros patrones de análisis más pequeños que se puedan utilizar, a continuación comprobar si se pueden aplicar patrones de diseño o arquitectura[13] y, por último, aplicar patrones similares a los de Fowler [1996] para añadir flexibilidad y extensibilidad a los modelos. Como resultado se

[13] Los autores sostienen que algunos de estos patrones pueden ser usados durante el proceso de análisis.

obtendría un esqueleto inicial del modelo conceptual que habría que completarlo ad hoc. En otros trabajos [Fernández y Yuan, 1999; Fernández, 2000; Fernández et al., 2000] han propuesto algunos patrones de análisis generales.

Purao et al. [2003] plantea una aproximación que usa mecanismos de aprendizaje para el diseño conceptual semi-automático con reutilización de patrones de análisis. El objetivo es utilizar un asistente inteligente que implemente dichos mecanismos de aprendizaje y facilite la creación de un diseño conceptual preliminar que deberá ser depurado más tarde.

Con el propósito de facilitar la especificación y uso de los patrones durante el análisis, Pantoquilho et al. [2003] proponen una plantilla especializada en la descripción de este tipo de patrones que complementa a otras ya existentes.

Una visión muy diferente a las anteriormente expuestas es la que se proporciona en Bergholtz y Johannesson [2000]. Aquí, los patrones no se utilizan para ayudar en la generación de los modelos conceptuales, al contrario, éstos son utilizados como un instrumento de validación de estos modelos, generando explicaciones en lenguaje natural acerca de las relaciones que existen entre los elementos estructurales del modelo.

Otra interesante línea de investigación emprendida es la que se relaciona con la búsqueda de *patrones de análisis estables*. Hamza [2002] suscribe que los patrones de análisis no han alcanzado su verdadero potencial debido a que carecen de estabilidad. Cuando pretendemos aplicar un patrón de análisis en contextos diferentes, el cual modela un problema específico, muchas veces los desarrolladores se ven forzados a analizar ese mismo problema desde cero, mermando así su reusabilidad. Para solventar este inconveniente proponen el desarrollo de lo que ellos llaman *patrones de análisis estables* [Hamza y Fayad, 2002a], en base a conceptos de estabilidad del software [Fayad y Altman, 2001]. Además, proponen un lenguaje de patrones [Hamza y Fayad, 2002b] que muestra los pasos a seguir para la construcción de este tipo de patrones.

Otras aproximaciones, con una fuerte repercusión práctica en el modelado conceptual, son aquellas que están encaminadas hacia la recopilación y aplicación de patrones de análisis para diferentes dominios.

Dejando de lado el ámbito concreto de los sistemas cooperativos, el cual abordaremos en la sección siguiente, en este punto podemos hablar de algunos de los dominios para los que se han propuesto y aplicado patrones específicos.

Este es el caso de Fowler [1997], quizás la obra más relevante e influyente dentro del movimiento que está relacionado con el estudio de los patrones de análisis. Este libro presenta una colección de 85 modelos de objetos del negocio reutilizables en diferentes ámbitos (relaciones organizacionales, cantidades, medidas, responsabilidades, análisis financiero, inventarios, contabilidad, planificación, comercio, etc.). No obstante, gran parte de estos patrones no son exclusivos de un sólo dominio, pudiendo ser aplicados en varios de ellos, lo que eleva aún más su capacidad de reutilización.

En la línea del libro anterior, Hay [1996] describe estructuras comunes que aparecen durante el modelado de datos (o modelado entidad/relación) dentro de muchas empresas, las cuales guardan relación con áreas tales como contabilidad, contratos, planificación de materiales, etc.

De igual forma, Coad et al. [1995] propone un conjunto de 31 patrones conceptuales de uso general, compuestos por no más de dos o tres clases, y numerosas estrategias que guían el proceso de análisis.

Otro de los ámbitos beneficiados por el uso de patrones conceptuales ha sido el de las interfaces de usuario. Desde esta perspectiva, Granlund et al. [2001] describe la aproximación PSA (Pattern-Supported Approach) para el diseño en base a patrones de interfaces de usuario en el contexto de la visualización de información. PSA sugiere la aplicación de una cadena de patrones pertenecientes a diferentes niveles de análisis y diseño. Comienzan con la especificación del sistema mediante la aplicación de patrones del dominio de aplicación, los cuales recomiendan el uso de patrones que describen procesos típicos del negocio. A continuación, éstos últimos apuntan a patrones concretos de tareas, los cuales pueden estar compuestos a su vez por patrones de subtareas, que facilitan el análisis de tareas/usuario. La aproximación considera estos patrones iniciales como *patrones de información* (específicos del dominio de aplicación). A continuación comienza la fase de diseño propiamente dicha en la que se utilizan patrones independientes del dominio de aplicación. Se aplican patrones de estructura y navegación de la

información en base al contenido de los patrones de tareas, para después utilizar patrones de diseño de la interfaz de usuario en base al trabajo de Tidwell [1998].

Molina et al. [2002a, 2003] muestra otro interesante trabajo aplicado al desarrollo de interfaces de usuario en base a patrones conceptuales. Los autores proponen un modelo, denominado JUST-UI, para la especificación de interfaces de usuario abstractas soportada por patrones conceptuales. JUST-UI identifica patrones típicos que aparecen en dichas interfaces y los abstrae para trabajar en términos del dominio del problema. En base a su experiencia investigadora, tanto en el ámbito académico como industrial, los autores desarrollan un lenguaje o colección de patrones conceptuales [Molina et al., 2002b] útiles en la descripción de interfaces de usuario abstractas, los cuales guían la fase de diseño y proveen estrategias de generación automática de código para prototipado rápido. Siguiendo una aproximación similar a PSA [Granlund et al., 2001], utilizan patrones en fases tempranas para propagarlos a fases posteriores. El lenguaje de patrones propuesto permite detectar necesidades comunes o requisitos de los usuarios, guiando la información que el analista debe obtener y cómo debe ser organizada. Estos patrones han sido usados como conceptos de especificación en la herramienta de modelado conceptual *Oliva Nova Modeller* ® desarrollada por CARE Technologies.

Otros ámbitos para los que se han recopilado y aplicado patrones de análisis/conceptuales específicos son: refinerías de petróleo [Zhen y Shao, 2002], sistemas empotrados [Konrad et al., 2004], sistemas de información geográfica [Lisboa et al., 1998], etc.

4. Situación actual de la aplicación de patrones en etapas tempranas de desarrollo de sistemas cooperativos

4.1. Introducción

La estrecha relación que existe entre la organización, los usuarios, la tecnología y las tareas a realizar, hace que los sistemas colaborativos, a menudo conocidos por esta razón como sistemas socio-técnicos [Mumford, 2000], sean especialmente difíciles de analizar y diseñar. Sin embargo, a

medida que la experiencia ha ido creciendo, investigadores y profesionales se han percatado de la existencia de patrones que pueden ser aplicados, una y otra vez, durante el desarrollo de estos sistemas.

Mientras que muchos de estos patrones son aplicados en las etapas avanzadas del desarrollo, otros están pensados para las etapas tempranas. En el primer caso, suelen consistir en soluciones de diseño detallado, o de grano fino, dependientes de alguna tecnología concreta (p. ej., orientación a objetos), que proceden del conocimiento y la experiencia existente en distintas áreas de la informática como son: sistemas distribuidos, interacción persona-ordenador, bases de datos, ingeniería del software, etc. En el segundo caso, habría que hacer una distinción. Por un lado, aquellos patrones útiles en un diseño inicial del sistema, los cuales proporcionan soluciones de alto nivel, o de grano grueso, a problemas típicos que aparecen en el dominio de aplicación. Éstos suelen ser independientes de una tecnología de diseño concreta. Por otro lado, aquellos patrones que describen conceptos y escenarios comunes en el dominio del problema, los cuales, además de ayudar a comprender el dominio, a veces forman modelos reutilizables durante el modelado conceptual. Éstos derivan comúnmente de la experiencia y conocimiento existente acerca de cómo las personas trabajan en grupo, el cual emana de disciplinas de índole social tales como la etnografía aplicada, la sociología, la psicología, las ciencias del trabajo, etc.

Se han publicado numerosos patrones aplicables en una etapa de diseño detallado. Algunos de estos patrones de diseño forman parte de catálogos de propósito general, como por ejemplo el de Gamma et al. [1995]. Otros abordan problemas de diseño comunes en esta clase de sistemas, como por ejemplo: manejo de eventos [Schmidt, 1995b], concurrencia [Sane y Campbell, 1996; Schmidt et al., 2000], colaboración entre objetos [Guerrero y Fuller, 1999], sistemas distribuidos [Das Neves y Garrido, 1998], etc. Sin embargo, como veremos a continuación, se han publicado pocos trabajos relacionados con la aplicación de éstos en una etapa de diseño temprano o de alto nivel, pero menos aún, son los que versan sobre la descripción de conceptos o escenarios de trabajo colaborativo mediante patrones. En general, se usa exclusivamente la prosa como la forma natural para la descripción de estos patrones. Casi todos estos trabajos se centran meramente en destacar la utilidad de los patrones como instrumento para la comunicación, mientras que son prácticamente inexistentes los que enfatizan

el importante papel de éstos como guías para la construcción de los modelos de análisis.

4.2. Perspectivas de aplicación

4.2.1. Los patrones como instrumentos para la comunicación interdisciplinar

Como ya hemos reseñado en este capítulo, una de las principales ventajas de usar patrones es la posibilidad de compartir un mismo lenguaje que nos permita comunicar y razonar sobre las soluciones y problemas que aparecen comúnmente en un determinado dominio. Mientras que unos patrones están exclusivamente ligados al personal técnico[14], otros pretenden servir como *lengua franca* [Erickson, 2000a] entre éstos y las personas relacionadas con el dominio del problema.

De acuerdo con Ellis et al. [1991], los sistemas colaborativos deberían facilitar la interacción entre los miembros del grupo y soportar las prácticas y actividades habituales en el lugar de trabajo. Por tanto, para poder abordar su diseño con éxito, es preciso conocer las interacciones y procesos de trabajo que se dan en el dominio del problema. Sin embargo, hacer explícito este conocimiento no es fácil para los técnicos, ni a veces para los propios usuarios, recurriendo a menudo a expertos en el dominio o a técnicas de elicitación de requisitos como la etnografía aplicada.

4.2.1.1 Patrones de interacción

Los sistemas colaborativos son por naturaleza interactivos, por lo que los resultados obtenidos en el campo de la interacción persona-ordenador (IPO) son muy valiosos para el desarrollo de estos sistemas.

[14] Por ejemplo, el catálogo de Gamma et al. [1995], cuyos patrones capturan soluciones exitosas en el dominio del diseño orientado a objetos, forma un vocabulario que puede ser compartido sólo por diseñadores.

Avalada por múltiples trabajos[15], entre otros [Bayle et al., 1998; Tidwell, 1998; Granlund et al., 2001; Montero et al., 2002, 2005b], la IPO tiene ya un importante bagaje en el uso de patrones para el diseño de la interacción persona-ordenador, donde el diseño de interfaces de usuario es simplemente un caso particular de ésta. Por ejemplo, aunque no está específicamente centrado en los sistemas cooperativos, uno de los trabajos más interesantes sobre el desarrollo de sistemas interactivos basado en lenguajes de patrones es el de Borchers [2001]. Según el autor, como el desarrollo de interfaces requiere de la colaboración de ingenieros de software, de especialistas en IPO, así como de usuarios y/o expertos en el dominio, es necesario construir un marco de trabajo interdisciplinar basado en lenguajes de patrones. Jan Borchers propone tres lenguajes de patrones interdependientes (lenguaje de patrones del dominio de la aplicación, de interacción persona-ordenador y de ingeniería del software), aplicables en etapas distintas del proceso de desarrollo y capaces de conectar los distintos puntos de vista de los participantes.

Sin embargo, cuando hablamos de sistemas cooperativos, nos interesa especialmente conocer cómo los componentes de un grupo de trabajo interaccionan entre ellos.

El estudio de la interacción entre las personas por medio de ordenadores, ha dado lugar a nueva área de investigación denominada IPOP (Interacción Persona-Ordenador-Persona), en inglés HCHI (Human-Computer-Human Interaction). A diferencia de la IPO, la corta trayectoria de la IPOP, enfrentada a problemas emergentes relacionados con la forma en que los sistemas pueden contribuir a facilitar o inhibir la interacción social entre sus usuarios, no ha dado lugar a muchos trabajos[16].

[15] Para ver una muestra de los trabajos más interesantes y algunos de los lenguajes de patrones existentes, el lector puede consultar tres famosas websites [Erickson, 2004; Borchers, 2005; Fincher, 2000] especializadas en el tema.

[16] Tan solo conocemos la celebración de dos reuniones importantes que han tratado este asunto específicamente. Una es el *Workshop on Human-Computer-Human-Interaction Patterns* celebrado dentro del congreso CHI2004, en Viena (Austria), del 25 al 26 de abril de 2004. La otra es el *Focus Group at EuroPLoP04: HCHI-Patterns*, celebrado en Irsee (Alemania), del 9 al 10 de julio de 2004.

Los estudios etnográficos [Hughes et al., 2000] son una valiosa fuente para la obtención de patrones dentro de la IPOP. Uno de los trabajos más notables en este sentido es el de Erickson [2000b], precursor en la descripción de escenarios de trabajo cooperativo a través de patrones. En su trabajo discute y describe narrativamente varios patrones que han sido derivados de estudios sobre etnografía aplicada.

Partiendo de las ideas de Erickson [2000b], el trabajo de Martin et al. [2001, 2002] centra su interés en la utilización de los patrones como medio para capturar, organizar y hacer más accesible a los diseñadores el corpus de conocimiento procedente de décadas de estudios de etnografía realizados en entornos de trabajo cooperativo. Los patrones sirven como medio para documentar y describir las interacciones comunes que se producen en dichos entornos. La intención es que éstos sean utilizados como recursos útiles durante el análisis y diseño de sistemas CSCW.

Estos autores no proponen patrones prescriptivos, consistentes en soluciones-plantilla para la resolución de determinados problemas, sino patrones descriptivos. En sintonía con Erickson [2000b], buscan patrones orientados a la descripción de fenómenos en el lugar de trabajo, más que en proporcionar soluciones de diseño: "[…] *nos hemos centrado en el desarrollo de patrones descriptivos que transmiten la naturaleza de escenarios para aquellos que buscan el desarrollo de tecnologías que son sensibles a la naturaleza de los entornos de trabajo* […]" [Martin et al., 2001].

En Martin et al. [2001] dieron a conocer su experiencia en la creación de un lenguaje preliminar de patrones de interacción cooperativa y presentaron un par de ejemplos de patrones para este lenguaje. Desafortunadamente, tras un seguimiento a lo largo de todo este periodo, hemos podido comprobar que este lenguaje apenas ha evolucionado desde entonces. Para obtener información sobre su estado actual puede consultarse la web [Martin et al., 2005] del proyecto *PoinTer* (Patterns of Interaction: a Pattern Language for CSCW) en el que participan los autores.

4.2.1.2 Patrones hipermedia

El éxito que está teniendo Internet en nuestra sociedad está influyendo en la forma de interaccionar con las personas y los ordenadores.

La demanda de aplicaciones colaborativas que utilizan la Web como plataforma tecnológica para la interacción y presentación de contenidos hipermedia va en aumento. La familiaridad de los usuarios con esta tecnología y la necesidad, cada vez mayor, de entornos que presten soporte a la comunicación, coordinación y colaboración entre usuarios, está favoreciendo enormemente este proceso migratorio. Por esta razón, la obtención y aplicación de patrones durante el desarrollo de sistemas hipermedia cooperativos [Groenbaek et al., 1994] es una tarea que no podemos eludir. Pero, de momento, han sido muy pocos los trabajos realizados en este sentido.

Paradójicamente, el campo de la hipermedia ha sido uno de los que más aportaciones han realizado al mundo de los patrones de software. Sin embargo, casi todas han estado enfocadas hacia el diseño de aplicaciones hipermedia "no colaborativas". Los primeros estudios realizados sobre este tipo de patrones aparecieron en el seno del grupo de Rossi et al., quienes dieron a conocer algunos patrones para la creación de aplicaciones orientadas a objetos con funcionalidad hipermedia [Rossi et al., 1996a] e hicieron una primera aproximación para la creación de un lenguaje de patrones aplicable en este dominio [Rossi et al., 1996b]. Más tarde, en [Rossi et al., 1997] pusieron de manifiesto la necesidad de motivar a la comunidad hipertexto para que este asunto sea discutido y para promover la construcción de un catálogo de patrones relacionados, como ya sucedió en Gamma et al. [1995]. Desde entonces han aparecido diversos trabajos que han tratado este tema desde diversos puntos de vista [Lyardet et al., 1999; Garzoto et al., 1999; Paolini y Garzoto, 1999] proporcionando algunos de ellos interesantes colecciones de patrones [German y Cowan, 2000; Nanard y Nanard, 1999; Discenza y Garzotto, 1999; Bernstein, 1998; etc.].

Por desgracia, la "hipermedia colaborativa", a diferencia de la "no colaborativa", ha dado lugar a muy pocos trabajos de momento. Entre los nuevos tópicos a explorar en este campo se encuentran: el mantenimiento de la conciencia de grupo, el modelado de usuarios y roles en situaciones de cooperación o la provisión de canales de comunicación adicionales. Justamente, Schümmer et al. [1999] propone un conjunto de patrones que atiende este tipo de problemas. Éstos fueron clasificados en patrones de interacción (*Communication Channel* y *Session*), patrones de colaboración (*User Role* y *Group Location Awareness*) y patrones de interface de usuario (*Avatar* y *Virtual Room*).

Las metodologías para el diseño de sistemas hipermedia deberían extenderse para cubrir aspectos relacionados con la hipermedia colaborativa (comunicación, cooperación, coordinación, grupos de usuarios, etc.) y beneficiarse del uso de un lenguaje específico de patrones que ayude al diseñador en la resolución de los problemas típicos que aparecen dentro de este dominio. En esta línea, Schümmer et al. [1999] expone brevemente cómo se podría extender el método de diseño hipermedia orientado a objetos OOHDM [Schwabe et al., 1996] para este fin. En Isla y Gutiérrez [2002] defendemos la integración de patrones dentro de una metodología específica para su aplicación sistemática durante el análisis, diseño y desarrollo de sistemas colaborativos hipermedia. Destacamos la necesidad de explorar nuevos mecanismos que permitan la selección, comparación y composición de patrones a partir del reconocimiento de problemas concretos. La unificación de un catálogo y la utilización de criterios de clasificación adecuados podrían ayudar en esta dirección. Dicho catálogo debería cubrir el mayor rango posible de problemas. Asimismo, defendemos la utilización de una notación con capacidad expresiva suficiente como para definir de forma clara, sencilla y precisa los diferentes tipos de patrones.

4.2.1.3 Patrones socio-técnicos

Un grupo de investigadores interesados en lo que han venido a denominar *patrones socio-técnicos*, han elaborado unos cuantos trabajos que son de gran interés para el tema que nos ocupa. Los *patrones socio-técnicos* tratan sobre las relaciones existentes entre la tecnología y los factores sociales. Su objetivo es proporcionar soluciones a problemas frecuentes que aparecen en contextos mediados y no mediados por ordenador, combinando productivamente relaciones sociales con tecnologías de la información y la comunicación.

Como parte del congreso *CSCW 2002*, tuvo lugar en la ciudad de Nueva Orleans un interesante encuentro centrado en este tema, denominado *Workshop on Socio-Technical Pattern Languages.* Entre los trabajos presentados en este taller y relacionados con el diseño de sistemas CSCW está el de Thomas et al. [2002]. En este trabajo los autores recalcan la necesidad de crear lenguajes de patrones socio-técnicos, junto a herramientas y metodologías que soporten su uso durante todo el proceso de desarrollo de software. Señalan, además, que se deberían implementar

componentes de software basados en dichos lenguajes para desarrollar las aplicaciones. Para ello es necesario expresar, en un formato útil y usable por los desarrolladores, el conocimiento relevante que existe sobre el dominio socio-técnico. De esta forma, los patrones podrían ayudar a comprender y analizar los problemas que aparecen en dicho dominio y crear los modelos necesarios. Los autores disponen de una página web (`www.truthtable.com/websitewelcome_page_index.html`) en la que se incluye información sobre este tipo de patrones.

En consonancia con Martin et al. [2001, 2002], Crabtree y Rodden [2002] consideran los patrones como un medio adecuado para estructurar y presentar material etnográfico. Sugieren que una aproximación descriptiva de los patrones para el campo del CSCW complementaría una perspectiva prescriptiva. Más que en codificar problemas y soluciones, se centran en las propiedades comunicativas de tales lenguajes [Erickson, 2000a, 2000b]. Defienden que los patrones socio-técnicos deberían verse como el medio para soportar el problema de la comunicación interdisciplinar (etnógrafos, desarrolladores, usuarios, etc.).

Herrmann et al. [2002] destaca las características y requisitos que deberían poseer los patrones socio-técnicos para CSCW. Sostienen que su descripción debería poder expresar la eventualidad y la vaguedad [Herrmann y Loser, 1999], junto al determinismo y la formalidad. El motivo es que las interacciones sociales son a menudo informales y no pueden estar determinadas a priori, con lo que la descripción de los patrones debería ser lo suficientemente flexible como para respetar la potencial libertad de decisión y la incertidumbre. Debido a estas limitaciones, sugieren que sería mejor usar los patrones como reflexión y punto de partida para la ingeniería de requisitos [Herrmann et al., 2003] y el diseño participativo [Dearden et al., 2002], más que como guías precisas del comportamiento humano.

Uno de los autores más prolíficos en el terreno de la aplicación de patrones como soporte para el desarrollo de sistemas groupware es Till Schümmer. Dentro del foro sobre patrones socio-técnicos anteriormente citado, éste propuso la construcción de un lenguaje de patrones groupware [Schümmer, 2002] para la orientación en el campo del CSCW y facilitar la comunicación dentro del equipo de desarrollo, entre el equipo y los usuarios finales, así como entre los propios usuarios finales. En palabras de este autor:

> *"Habiendo trabajado en un equipo interdisciplinar que desarrolla y evalúa groupware, encontrábamos problemas cuando un nuevo miembro se unía a la división o un investigador invitado participaba: el lenguaje que es usado por la comunidad CSCW parece contener contradicciones o definiciones diferentes de los mismos términos. Vemos la necesidad de un lenguaje común entre todas las partes participantes [...]"*
>
> —Till Schümmer, 2002.

Junto a Robert Slagter, este autor ha propuesto un proceso de desarrollo (*OSDP*) participativo [Schümmer y Slagter, 2004] basado en la utilización de dos niveles de patrones según el tipo de participante al que va destinado: de *alto nivel* (dirigidos a los usuarios finales) y de *bajo nivel* (destinados a los desarrolladores de software). Los patrones de alto nivel están centrados en el comportamiento del sistema, tal y como es percibido por el usuario final. Éstos refuerzan a los usuarios para encontrar sus requisitos y dar forma a la aplicación que desean. Por supuesto, para que puedan ser entendidos por los usuarios, su descripción tiene que ser más descriptiva y prosaica que los patrones de bajo nivel, los cuales incluyen detalles más técnicos y especifican cómo implementar ese comportamiento. Esta separación de niveles de abstracción favorece la participación de los usuarios finales en el proceso de desarrollo y facilita la interacción con los desarrolladores. Durante la ingeniería de requisitos, los usuarios finales esbozan escenarios de uso con ayuda de los patrones de alto nivel que describen cómo la interacción del grupo puede ser soportada mediante groupware. En la fase de desarrollo, los desarrolladores implementan estos escenarios usando los patrones de bajo nivel como guías de desarrollo. De esta forma, los usuarios finales interactúan estrechamente con los desarrolladores, proponiendo la aplicación de patrones e identificando los conflictos que aparecen cuando usan los prototipos. En Lukosch y Schümmer [2006], se presenta un lenguaje de patrones centrado en aspectos técnicos que surgen durante el desarrollo de groupware. Narran su experiencia en el uso de este lenguaje, tanto en el ámbito académico como en un proyecto real, demostrando que los patrones sirven como vehículo educacional y comunicativo. Educa a los desarrolladores en el diseño e implementación de aplicaciones groupware y facilita la comunicación entre desarrolladores, clientes y usuarios finales para una mejor comprensión de los requisitos. Este

lenguaje de patrones es sólo parte de una colección más amplia de lenguajes centrada sobre diversos aspectos del desarrollo de groupware, en concreto, comunidades virtuales [Schümmer, 2004a], privacidad [Schümmer, 2004b], formación espontánea de grupos [Schümmer, 2004c] y administración de objetos compartidos [Lukosch y Schümmer, 2004].

4.2.2. Los patrones como guías para la construcción de modelos

Los métodos de ingeniería de software se basan en modelos gráficos para la especificación y diseño de sistemas. Durante las etapas tempranas de desarrollo construimos modelos conceptuales o de análisis que nos ayudan a reflexionar y establecer un modelo mental del sistema que vamos a construir. Estos modelos constituyen la primera representación del sistema. Son la base para la construcción de los modelos de diseño, los cuales especifican el sistema desde un punto de vista técnico y con el suficiente nivel de detalle como para que pueda ser implementado.

Como ya sabemos, muchos patrones proporcionan modelos reutilizables que actúan como bloques o, más apropiadamente, como guías[17] para la construcción de todos estos modelos. Sin embargo, la mayor parte de ellos son específicos de otros dominios o son aplicables en la fase de diseño.

Hemos conocido tan solo un par de trabajos que, aunque de soslayo, tocan el tema de la aplicación de patrones durante el modelado temprano de sistemas cooperativos.

David et al. [2003] es uno de estos trabajos. En éste se hace un recorrido por un proceso para el diseño, desarrollo y uso de sistemas cooperativos. Se explica el conjunto de modelos utilizados y se hace referencia a los posibles tipos de patrones que podrían aplicarse.

[17] Preferimos usar el término “guía” ya que, desde nuestro punto de vista, los patrones no deberían actuar como bloques rígidos, sino como plantillas flexibles que marcan las directrices para facilitar la selección de los elementos concretos que participan y sus relaciones, otorgándole al diseñador un cierto margen de libertad durante la creación de las instancias del patrón.

Los autores promueven la utilización de patrones para la construcción y transformación de unos modelos en otros. Hablan de *patrones de escenarios,* para facilitar el análisis, y de *patrones de transformación,* para pasar de los escenarios al modelo CAB-M (Collaborative Application Behavior Model). El objetivo de este modelo sería describir la estructura de los actores, artefactos, contextos y tareas que caracterizan el comportamiento de la aplicación colaborativa. A su vez, el modelo CAB-M estaría basado en patrones (de organización de actores, de tareas, de workflow, de contexto, de validación, etc.).

Con respecto a la infraestructura utilizada para el software, identifican tres capas funcionales: nivel de aplicación colaborativa (CUO-M), nivel de infraestructura groupware (CSA-M) y nivel de sistema distribuido (DSI-M). El primer nivel esta orientado al usuario. Controla la interacción y propone interfaces para el control de acceso y notificación de eventos. El segundo nivel actúa como un sistema operativo para grupos, proporcionando los servicios que necesita el primero. Soporta el manejo de sesiones y usuarios, proporciona herramientas cooperativas genéricas (ej. telepuntero) y es responsable del control de la concurrencia. El último nivel provee mecanismos de comunicación y sincronización de componentes distribuidos. Los autores proponen la utilización de patrones de proyección que ayuden en la transformación del modelo CAB en la arquitectura software basada en los modelos CUO-M, CSA-M y DSI-M.

Con todo, presentan tan sólo una panorámica muy general de este proceso. Prácticamente, no ofrecen información sobre cómo se define el modelo CAB-M (teóricamente el modelo conceptual del sistema colaborativo que van a desarrollar), ni dicen nada acerca de cómo se integran los patrones dentro del proceso propuesto. Asimismo, apenas proporcionan información sobre cómo se definen los diferentes tipos de patrones. Únicamente muestran un par de ejemplos de patrones de interacción basándose en una notación gráfica específica para el modelado del control de la interacción en base a la representación de agentes. Esta notación está muy ligada a la implementación y es incluida en AMF-C [Tarpin-Bernard et al., 1998] un modelo multiagente para sistemas cooperativos. Tampoco indican cómo se relacionan unos patrones con otros, cómo se usan para transformar o proyectar unos modelos en otros, cuáles son los catálogos de patrones que utilizan, etc.

El otro trabajo que conocemos relacionado con este tema es el de Geyer-Schulz y Hahsler [2002], pero en él también echamos de menos un acercamiento a las cuestiones que acabamos de plantear. Este trabajo promueve la reutilización del conocimiento del dominio mediante la introducción de patrones en la fase de análisis. No está centrado en el dominio de los sistemas colaborativos, pero presentan una familia de patrones de análisis para el tratamiento de un problema específico dentro de este tipo de sistemas, en concreto, el filtrado y compartición de información colaborativa. Estudian el potencial de reutilización que presentan estos patrones durante el análisis de varios de los componentes de un sistema de información y concluyen que éstos ayudan a reducir su tiempo de desarrollo significativamente.

5. Conclusiones del capítulo

En la primera parte de este capítulo hemos introducido el tercer pilar sobre el que se asienta esta tesis: los *Patrones de Software*.

Desde hace más de una década la ingeniería del software viene utilizando los patrones como un instrumento válido para la identificación, descripción y distribución de soluciones probadas empíricamente que pueden ser reutilizadas ante problemas concretos que aparecen comúnmente en determinadas fases de desarrollo y, a menudo, dentro de dominios de aplicación específicos.

Como hemos podido comprobar, a lo largo de esta andadura la comunidad ha generado una gran cantidad de patrones que pueden clasificarse de múltiples modos. Al mismo tiempo, se han sucedido y consolidado una serie de encuentros que están sirviendo de soporte para la discusión e intercambio de ideas y experiencias sobre este tópico. El estudio de los patrones de software desde diferentes perspectivas también ha dado lugar a diversas formas de interpretar, definir y describir un mismo concepto.

Hemos puesto de manifiesto que la integración de patrones en los procesos y métodos de desarrollo de software facilita:

- La reutilización del conocimiento que ha demostrado su validez en proyectos anteriores.

- La identificación de los problemas dentro del espacio.
- El entendimiento entre las distintas personas involucradas, ya que los patrones establecen un vocabulario común o lengua franca que permite compartir y discutir los problemas, así como sus soluciones.
- La legibilidad, comprensión y mantenimiento de la documentación en general y, por ende, de los modelos que se generan.

Sin embargo, para poder obtener estos beneficios y llevar a cabo adecuadamente dicha integración es conveniente:

- Disponer de una colección estructurada de patrones que se extienda, a ser posible, a todas las etapas del proceso y que permita dar solución al máximo número de problemas en cada una de éstas. Para ello es necesario que dicha colección esté sometida a un continuo proceso de crecimiento y mejora.
- Mantener un formato de descripción uniforme.
- Clasificar los patrones en función de criterios que faciliten su selección.
- Establecer relaciones de colaboración entre ellos.
- Definirlos y representarlos adecuadamente dentro de los modelos.
- Utilizar algún tipo de herramienta que soporte su gestión.
- Entrenar e instruir al equipo de desarrollo en una filosofía de trabajo basada en patrones.

Uno de los periodos más complejos y críticos dentro del proceso de desarrollo de software es el que está relacionado con la obtención, análisis y especificación de requisitos. Durante estas etapas tempranas se suelen construir modelos conceptuales con el objetivo de facilitar el análisis y la comprensión del dominio del problema. Para agilizar y mejorar la construcción de tales modelos, es posible aplicar un tipo de patrones que han sido catalogados como *patrones conceptuales* o *patrones de análisis*. El propósito de estos patrones es *la comunicación, el entendimiento y la reutilización durante el proceso de análisis y modelado, del conocimiento*

relacionado con aquellas abstracciones clave que son recurrentes en dominios concretos.

Una vez examinados en la segunda parte de este capítulo los trabajos existentes relacionados con la aplicación de patrones en este estadio inicial del desarrollo, hemos comprobado que éstos son escasos y están muy disgregados, abordando el tema desde muy diversas perspectivas. Observamos también que, a pesar de los beneficios que implica la reutilización de este tipo de patrones durante el modelado temprano, su aplicación todavía no está lo suficientemente extendida, a diferencia de lo que ocurre, por ejemplo, en la etapa de diseño. Entre las posibles causas de esto pueden estar:

- La ausencia de colecciones de patrones para dominios específicos.
- La carencia de patrones realmente útiles y con el nivel de abstracción adecuado que ayuden a construir este tipo de modelos.
- La falta de integración con metodologías concretas que faciliten la definición de los patrones y ordenen su aplicación sistemática.

Pero, ¿en qué situación se encuentra actualmente el modelado conceptual en base a patrones de los sistemas cooperativos en particular?. Para responder a esta cuestión hemos sacado a la luz los distintos trabajos que existen en torno a la aplicación de patrones durante las etapas tempranas de desarrollo de esta clase de sistemas. Nuestra conclusión es que son escasos los trabajos relacionados con la aplicación de éstos en una etapa de diseño temprano o de alto nivel, pero menos aún, son los trabajos que versan sobre la descripción de conceptos o escenarios de trabajo comunes en dicho dominio del problema mediante patrones. En general, se usa exclusivamente la prosa como la forma natural para la descripción de patrones. Casi todos estos trabajos se centran meramente en destacar la utilidad de los patrones como instrumento para la comunicación entre desarrolladores, expertos en el dominio y usuarios. Sin embargo, aunque en algunos casos se reconoce esta necesidad, son prácticamente inexistentes los trabajos que, a través de una notación específica, representan patrones concretos reutilizables durante el modelado conceptual de un sistema cooperativo. Tan sólo hemos conocido un par de trabajos que resaltan el papel de los patrones como guías para la construcción de modelos durante la

etapa de análisis de estos sistemas. Sin embargo, desafortunadamente, tocan el tema de soslayo y muy superficialmente.

Desde nuestro punto de vista, muchos de los conceptos y escenarios comunes que aparecen en el dominio del problema deberían formar parte del vocabulario utilizado en la construcción de los modelos conceptuales que definen estos sistemas. Su reutilización en forma de patrones facilitaría la creación, interpretación y comunicación de estos modelos. Esto es un desafío pendiente y en esta tesis abordamos este asunto. Por lo pronto, como se muestra en el capítulo siguiente, vamos a establecer una notación que facilite la definición de estos patrones, así como su aplicación durante el modelado.

-Esta página se encuentra deliberadamente en blanco-

Capítulo IV

Un Perfil UML para el Modelado de Patrones de Software

Contenido

<<Lo último que uno sabe es por dónde empezar>>

—Blaise Pascal (1623-1662)

Científico, filósofo y escritor francés

1. Introducción

Una de las aplicaciones más interesantes de los patrones es facilitar el modelado del software. Una gran parte de los patrones pueden verse como modelos genéricos que describen situaciones comunes de modelado. De este modo, los patrones pueden ser usados como guías o bloques de construcción reutilizables durante la creación de los modelos de software, tanto a un nivel de diseño como de análisis. Además, la identificación de los patrones en los modelos y, en general, en la documentación que se genera, favorece la comprensión, comunicación y mantenimiento de éstos.

La extensión de un lenguaje de modelado para la representación de patrones requiere de una notación, sintaxis y semántica específica. Para ello es necesario tener en cuenta que:

- Los modelos que definen los patrones (vista interna del patrón) deben ser lo suficientemente genéricos y flexibles como para que puedan ser adecuadamente instanciados.

- Los patrones, una vez aplicados, tienen que ser fácilmente identificables dentro de los modelos generados, así como los elementos de modelado con los que están relacionados (vista externa del patrón).

Aunque la descripción de un patrón incluye muchos otros aspectos (problema, contexto, relaciones con otros patrones, ejemplos de aplicación, etc.), los cuales pueden especificarse narrativamente, entendemos que son el modelado de su vista interna y externa los que requieren de una notación concreta y precisa.

En este capítulo definimos un *profile UML* o, en español, *perfil UML* para el modelado, tanto de la vista externa como interna, de patrones de software en general, los cuales consisten en diagramas genéricos de cualquier tipo que pueden reutilizarse para construcción de los modelos que se crean durante las distintas fases de desarrollo, incluyendo, por supuesto, los modelos conceptuales relacionados con los sistemas cooperativos que analizamos. Pero antes, para justificar la introducción de este profile, vamos a analizar el tratamiento que hace el estándar sobre este tema, así como las diferentes notaciones y profiles que otros autores han sugerido.

2. Los patrones desde la perspectiva de UML

2.1. Un tópico de especial interés a lo largo de sus diferentes versiones

UML (Unified Modeling Language) [OMG, 2003] es un lenguaje de modelado de propósito general que incluye una notación gráfica estándar que puede ser usada para el modelado de las diferentes visiones de un sistema de software.

Aunque los primeros indicios sobre la construcción de este lenguaje datan de octubre de 1994, su estandarización[1] fue llevada a cabo en 1997 por el OMG (Object Management Group)[2]. Desde entonces, y con la participación de numerosas e importantes empresas pertenecientes al sector de la informática, este consorcio se ha hecho responsable de su mantenimiento,

[1] En concreto su versión UML 1.1

[2] Consorcio sin ánimo de lucro que mantiene y produce especificaciones de tecnologías orientadas a objetos (`http://www.omg.org`)

dando lugar a sucesivas revisiones[3] que han ido aumentando la capacidad de expresión de este lenguaje. Actualmente, la versión oficial del estándar es la UML 2.1.1 [OMG, 2007a, 2007b], cuya versión oficial antecesora fue UML 1.5.

A lo largo de su historia, uno de los objetivos principales que este lenguaje ha perseguido es dar un soporte adecuado a conceptos de alto nivel tales como el de componente, colaboración, framework y, en particular, el concepto de patrón[4]. Así, influenciado por libros de gran calado que surgieron durante las primeras etapas de este lenguaje, como es el caso de Gamma et al. [1995], donde los patrones son vistos como estructuras formadas por clases/objetos reutilizables en una etapa de diseño orientado a objetos, UML trata los patrones como sinónimos de *colaboraciones parametrizadas* o, tal y como se denominan a partir de la versión 2.0, *colaboraciones plantilla*.

Básicamente, éste ha sido el enfoque utilizado desde el principio. Sin embargo, como hemos podido constatar[5], éste se ha tenido que ir depurando versión tras versión. Los continuos cambios introducidos obedecen claramente a un intento de dar solución a algunos de los problemas que este tratamiento presenta, los cuales hemos venido advirtiendo desde nuestros primeros trabajos sobre modelado de patrones [Isla, 2001, 2002; Isla y Gutiérrez, 2003]. Éstos se comentan en el punto siguiente (v. sec. 2.2).

Hemos podido observar cómo, después de ir eliminando algunas de las limitaciones que esta visión estaba imponiendo al modelado de patrones, el estándar se ha vuelto más abierto respecto a esta cuestión. Es más, en su versión actual (UML 2.1.1), la especificación [OMG, 2007a, 2007b] no dedica ninguna sección a los patrones de diseño, tal y como ha venido haciendo en sus versiones anteriores. De hecho, apenas aparece el término "patrón de diseño". Tampoco se defiende el uso de colaboraciones plantilla en ningún momento como el mecanismo más indicado para el modelado de patrones de diseño. No obstante, como no se dice nada en contra, podemos entender que

[3] UML 1.3, UML 1.5 y UML 2.0 están entre sus mayores revisiones

[4] Se puede comprobar cómo aparece esta necesidad explícita y reiteradamente en las distintas especificaciones de UML (p. ej. véase la sección "Objetivos de UML" perteneciente a la especificación de la versión UML1.5 [OMG, 2003, p.1-4]).

[5] Para conocer detalladamente algunos de los cambios experimentados a partir de la versión 1.3 el lector puede consultar Isla [2001, pp. 61-68].

las colaboraciones plantilla siguen siendo para el estándar un mecanismo válido para la representación de patrones de diseño, aunque tan sólo sea para modelar su vista externa, tal y como puede observarse en un ejemplo[6] concreto que aparece en la especificación del estándar [OMG, 2007b, p.630].

UML define una colaboración como "*la especificación de cómo una operación o clasificador, como un caso de uso, es realizado por un conjunto de clasificadores y asociaciones que juegan roles específicos usados de una manera específica. La colaboración define una interacción*" [OMG, 2003, p. Glossary-4].

Una colaboración describe, por tanto, un conjunto de roles que van a ser jugados por instancias de elementos clasificadores y sus enlaces, así como un conjunto de interacciones que definen la comunicación entre las instancias cuando juegan esos roles.

Cada rol describe el tipo de objeto que puede jugarlo, indicando por ejemplo las operaciones y los atributos requeridos para ese objeto. Las relaciones con otros roles son definidas por los roles de asociación, los cuales describen los enlaces necesarios entre los objetos de entre todos los posibles.

Como consta en su definición, UML utiliza normalmente las colaboraciones para especificar la realización de casos de uso y operaciones, mostrando el contexto en el que su comportamiento sucede, sin embargo, éstas son también utilizadas para modelar y dar un nombre a mecanismos interesantes desde un punto de vista arquitectónico como pueden ser los *patrones de diseño*.

La relación entre un caso de uso, o una operación, y la colaboración que lo realiza se modela como una relación de realización entre ambos, en cambio, una colaboración que modele un patrón de diseño puede estar sola y su contexto es el sistema global.

Una colaboración se representa en UML mediante una elipse punteada (v. Fig. 4.1). Esto permite ver la colaboración como un único bloque desde el exterior. El interior de la colaboración deberá contener los aspectos

[6] Se muestra una representación de la vista externa del patrón de diseño *Observer,* perteneciente al catálogo de Gamma et al. [1995].

estructurales y de comportamiento propios de ésta. La especificación estructural se hace mediante la descripción de las relaciones requeridas entre las instancias que conforman con los roles participantes y de las características estructurales que deben tener sus clasificadores base. Su comportamiento será expresado a través de diagramas de interacción (diagramas de colaboración y/o secuencia). Los diagramas de colaboración se utilizarán cuando sea necesario resaltar las relaciones estructurales entre los objetos que colaboran, mientras que los diagramas de secuencia destacan la ordenación temporal de los mensajes.

Una colaboración no contiene físicamente a ninguno de sus elementos estructurales, como es el caso de los paquetes o los subsistemas. Estos elementos deben haber sido declarados previamente, limitándose la colaboración a reflejarlos dentro de una interacción concreta. Así pues, una colaboración hace referencia a un bloque arquitectónico conceptual, más que a un bloque físico, pudiendo atravesar distintos niveles en un sistema.

Este mecanismo lo único que permite es dar un nombre a una sociedad de clases. Ahora bien, si lo que se pretende es modelar un patrón de diseño, dicho mecanismo debe actuar como una plantilla formada por un conjunto de abstracciones que colaboran entre sí para llevar a cabo algún comportamiento común e interesante. En tal caso, habrá que identificar claramente los elementos variables que deben ser ligados a elementos concretos cuando se aplique en un contexto particular. Por esta razón, la estructura de un patrón de diseño es capturada mediante una colaboración parametrizada (v. Fig. 4.1).

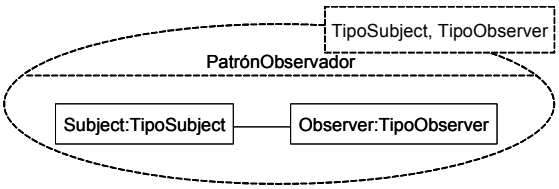


Fig. 4.1: Colaboración plantilla `PatrónObservador` con sus dos parámetros formales `TipoSubject` y `TipoObserver`

La reusabilidad de un patrón es obtenida utilizando cada rol representado en la colaboración (en realidad su clasificador base) como parámetro de la colaboración genérica. Los parámetros son ligados a

elementos concretos del modelo en cada instanciación de la colaboración parametrizada (v. Fig. 4.2).

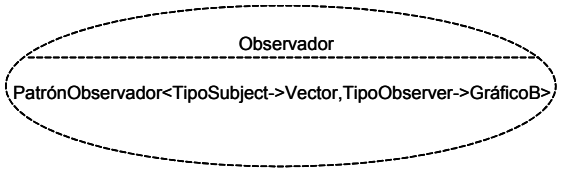


Fig. 4.2: Instanciación de `PatrónObservador` como consecuencia de una ligadura en la que se sustituyen `TipoSubject` por `Vector` y `TipoObserver` por `GráficoB`

Mientras que la mayoría de las colaboraciones pueden ser anónimas, al estar anexadas a un elemento del modelo que ya tiene un nombre, los patrones de diseño deben tener obligatoriamente un nombre que los identifique.

Es posible ver un patrón como una entidad simple desde fuera, pudiendo usarse para identificar la presencia de patrones como colaboraciones parametrizadas dentro del diseño de un sistema. En la aplicación o ligadura del patrón, una línea punteada es dibujada desde el símbolo de la colaboración hasta cada uno de los elementos que participan en ella. Cada línea es etiquetada con el nombre del rol (debe coincidir con el interno) que va a desempeñar el elemento participante dentro de la colaboración (v. Fig. 4.3).

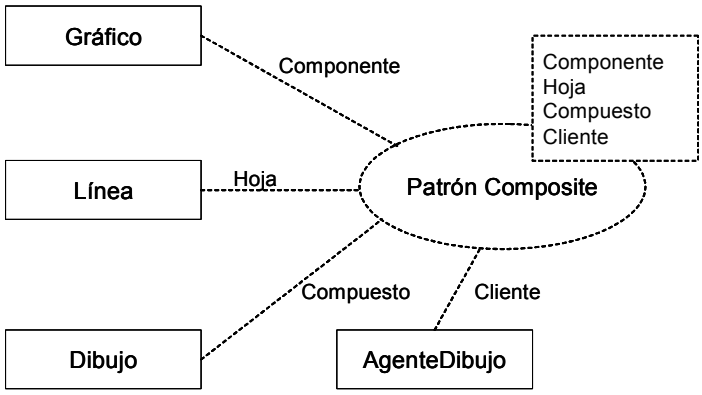


Fig. 4.3: Vista externa y aplicación del patrón Composite [Gamma et al., 1995] como una colaboración parametrizada

2.2. Algunas consideraciones sobre su tratamiento

El principal obstáculo que encontramos en el tratamiento que UML da a los patrones es su limitación a un sólo tipo de éstos, en concreto, a los *patrones de diseño orientado a objetos*. Para UML no existen más patrones que aquellos formados por estructuras de clases u objetos que dan solución a problemas típicos de diseño en contextos determinados. Sin embargo, en etapas de desarrollo anteriores al diseño, es habitual modelar aspectos de más alto nivel (p. ej., en el caso de los sistemas cooperativos modelamos los flujos de trabajo, la estructura social de los usuarios, la dinámica de la organización, los protocolos sociales, la cooperación entre actores, etc.), los cuales permiten mejorar la comprensión del dominio del problema. En consecuencia, muchos de los patrones aplicables durante las fases previas al diseño (patrones de análisis o conceptuales) no van a estar compuestos exclusivamente por clases/objetos relacionados, sino por elementos muy diversos (estados, transiciones, actividades, paquetes, eventos, señales, actores, roles, etc.) que van a facilitar el modelado conceptual de un sistema utilizando distintas perspectivas. Por tanto, parece razonable pensar que la noción de colaboración parametrizada no reúne los requisitos necesarios para poder modelar este tipo de patrones. Como veremos más adelante, en este mismo capítulo (v. sección 4) proponemos un profile o perfil UML que permite modelar patrones, independientemente del tipo de modelo que representan. Partiendo de este profile vamos a poder representar patrones que nos van a ayudar durante el modelado conceptual de sistemas software, y particularmente de sistemas cooperativos, tema central de esta tesis.

En cualquier caso, con independencia de lo expresado en el párrafo anterior y pese a que la versión actual de UML es más laxa en cuanto a la justificación de este tratamiento, creemos oportuno realizar una serie de consideraciones sobre la aproximación que ha venido utilizando UML para el modelado de patrones, en este caso, de diseño [Isla, 2001; Isla y Gutiérrez, 2003]:

- El propio OMG ha admitido en diversas ocasiones limitaciones en la definición que se hace del término *patrón* en UML, con respecto al uso que se hace del mismo en otros contextos.

  "*En UML, el término Patrón es un sinónimo de colaboración plantilla que describe la estructura de un patrón de diseño. Esta definición no es tan*

poderosa como la del término usado en otros contextos." [OMG, 2003, p. 2-129].

- Los elementos participantes en un patrón no pueden ser enlazados adecuadamente con los parámetros de la colaboración que lo representa.

  Habitualmente, los patrones de diseño tienen un número variable de elementos participantes. Una colaboración parametrizada tiene una cantidad fija de parámetros y un parámetro sólo puede ser sustituido por un argumento como máximo. Esto hace que muchos de los patrones de diseño existentes en la literatura (p. ej., *abstract factory, chain of responsability, composite*, etc.) no puedan ser instanciados adecuadamente.

  Es necesario que la especificación de los parámetros sea más precisa, indicando por ejemplo el tipo de los participantes, su cardinalidad, etc.

- Existen restricciones estructurales que no pueden ser expresadas mediante colaboraciones.

  Por ejemplo, restricciones para comprobar que "la existencia de cierta cantidad de elementos (p. ej. métodos) obliga la existencia de la misma cantidad de elementos de otro tipo (p. ej. clases)" no pueden ser representadas mediante colaboraciones.

  Esto es debido a que la especificación realizada de un patrón mediante una colaboración parametrizada no es lo suficientemente abstracta y está muy ligada al caso concreto, como ocurre con los diagramas que presenta Gamma et al. [1995].

- La capacidad para expresar ciertas restricciones de comportamiento está limitada.

  Por ejemplo, restricciones del tipo "un método m1 tiene que invocar a otro método m2" no pueden ser expresadas, si bien es cierto que podrían ser especificadas mediante notas o comentarios informales.

- Ausencia de especificación para el chequeo de conformidad de los elementos participantes con respecto a las restricciones impuestas por el patrón.

UML usa las colaboraciones parametrizadas solamente para enlazar y, supuestamente, chequear la conformidad de los actuales elementos de modelado con las restricciones estructurales que impone el patrón. Pero, en la documentación de UML no se especifica nada acerca de las reglas de conformidad que deberían cumplirse.

La única restricción que se exige es que el tipo de cada argumento debe ser el mismo, o un descendiente del tipo, que el del parámetro correspondiente.

- Los roles dependen de clasificadores o asociaciones base preexistentes definidos fuera del patrón.

  No es el rol el que hace de parámetro sino un clasificador o asociación base cuyo rol es una vista de éste. Conceptualmente hablando, esto carece de sentido si tenemos en cuenta que es el patrón de diseño el que debería ayudar a encontrar estos clasificadores. No se debería dar por hecho que estos clasificadores ya existen.

  Los patrones de diseño ayudan a encontrar nuevas abstracciones que no han sido contempladas en el diseño, sin embargo las colaboraciones obligan a que "todos" los elementos deban estar declarados previamente en algún lugar del diseño.

- Elementos que juegan un papel importante dentro de un patrón no pueden ser utilizados como parámetros.

  Por ejemplo, atributos y métodos deberían poder ser sustituidos por elementos concretos del contexto, sin embargo, éstos no pueden ser parámetros ya que no pueden ser roles. Esto supone una merma para la instanciación completa de un patrón.

- Elementos indispensables para la definición de un patrón de diseño son considerados elementos extra para una colaboración.

  Según UML, una colaboración puede también contener un conjunto de elementos de modelado limitadores, tales como restricciones y generalizaciones, quizá junto con algunos clasificadores extra. Estos elementos limitadores no participan en la propia colaboración, pero son usados para expresar restricciones extra sobre los elementos

participantes en la colaboración que no pueden ser cubiertos por los propios roles participantes. Por ejemplo, se puede requerir que un rol deba ser subclase de otro. Este tipo de requerimientos no puede ser expresado con roles de asociación y, por tanto, hay que incluir elementos extra.

- UML no define cómo los diagramas de interacción intervienen en el proceso de instanciación de un patrón de diseño

  Los patrones de diseño prescriben un comportamiento determinado para conseguir el propósito perseguido. Los diagramas de interacción pueden ser anexados a una colaboración con este objetivo. UML no define claramente cómo intervienen éstos en el proceso de "binding" ("ligadura" en español), sin embargo estas interacciones deberían verse como restricciones de comportamiento que deberían cumplir los elementos participantes para que conformen con la definición del patrón. Es necesario estudiar cómo expresar este tipo de restricciones. Sería interesante investigar cómo elevar el nivel de abstracción de los diagramas de interacción (colaboración/secuencia) para poder expresar adecuadamente el comportamiento de un patrón.

- Los propósitos para una colaboración y para un patrón de diseño son, en principio, diferentes.

  Según la especificación de UML 1.5 [OMG, 2003], la intención de una colaboración consiste en especificar cómo una operación o un clasificador, por ejemplo un caso de uso, es realizado por un conjunto de clasificadores y asociaciones, mostrando el contexto en el que su comportamiento sucede. Si además esta colaboración se parametriza el propósito de una colaboración es ampliado para que puedan ser utilizadas con el fin de modelar y dar un nombre a mecanismos interesantes desde un punto de vista arquitectónico como pueden ser los patrones de diseño.

  Existe cierta diferencia entre la razón de ser de una colaboración y la de un patrón de diseño. Un patrón de diseño indica cómo debería ser un diseño, desde el punto de vista de su estructura y de su comportamiento, con el objetivo de dar una solución a un problema frecuente de diseño que derive en una mayor flexibilidad, reusabilidad,

etc. Sin embargo, una colaboración pretende simplemente modelar la interacción que puede existir entre un conjunto de instancias (objetos) para lograr una tarea específica.

De hecho, la instanciación de una colaboración parametrizada da lugar a otra colaboración, sin embargo, el resultado de la instanciación de un patrón sobre una serie de elementos participantes da como resultado una porción de diseño que resuelve el problema de diseño planteado y que podría no incluir ninguna interacción.

En resumen, podemos concluir que UML presenta básicamente los siguientes inconvenientes para la realización de un tratamiento adecuado de los patrones:

- Se centra en un solo tipo de patrones, en concreto, los patrones de diseño orientados a objetos. No se tiene en cuenta la existencia de otros tipos de patrones, por ejemplo, los utilizados en fases previas al diseño (patrones conceptuales o de análisis) cuyos modelos no tienen porqué estar formados por estructuras de clases u objetos que interaccionan.

- La representación de patrones se basa exclusivamente en la utilización de colaboraciones parametrizadas, lo que limita enormemente los tipos de modelos que se pueden instanciar.

- Dicha representación es demasiado rígida, ya que está basada en el caso concreto, no permitiendo la utilización de parámetros opcionales o variables en número para posibilitar la ligadura de diferentes tipos de instancias cuyas estructuras son compatibles con el patrón.

- Lo único que se puede parametrizar son los roles, no pudiendo usar otros elementos (atributos, métodos, estados, acciones, condiciones, etc.) como parámetros. Además, los roles dependen de clasificadores o asociaciones base predefinidos, lo que carece de sentido si tenemos en cuenta que es el patrón el que nos debe ayudar a definir estos elementos y no al revés.

Centrados exclusivamente en el ámbito de los patrones de diseño orientados a objetos, y en un intento por ofrecer alguna solución a la mayor parte de los inconvenientes comentados con anterioridad, en trabajos previos

[Isla, 2001, 2002; Isla y Gutiérrez, 2003] hemos propuesto una aproximación alternativa para el modelado estructural de patrones de diseño con UML, basada en lo que hemos denominado *Diagramas-REP*.

Estos trabajos han servido de base y punto de partida para la definición del profile UML [Isla et al., 2005a] que proponemos en el punto 4 de este capítulo. Como ya veremos, este profile extiende UML para el modelado y aplicación de patrones de software en general, independientemente del tipo de modelo que estemos construyendo, superando los inconvenientes anteriormente comentados.

3. Un recorrido por los principales trabajos existentes sobre modelado de patrones

La búsqueda de una notación que permita representar adecuadamente la esencia de un patrón ha sido un tema candente desde la aparición de éstos en el campo de la ingeniería del software. Ante tal desafío, varios autores han proporcionado notaciones que varían, básicamente, en función del grado de formalidad empleado y el tipo de representación utilizada (textual o visual).

A menudo un patrón es visto como una serie de restricciones sobre un grupo de elementos participantes. Desde esta perspectiva, lenguajes formales basados en la lógica podrían ser buenos candidatos para realizar una especificación precisa de estas restricciones. Los lenguajes formales tienen sintaxis, semántica y reglas de inferencia precisas que posibilitan el razonamiento y la manipulación automática, lo cual es muy interesante desde el punto de vista del desarrollo de herramientas de diseño basadas en patrones. Como contrapartida, desde el punto de vista del usuario, la usabilidad de estas notaciones es menor, ya que generalmente suelen existir mayores dificultades para su comprensión y aplicación.

En cuanto a la decisión entre utilizar símbolos gráficos o textuales, excepto unos cuantos, los cuales emplean exclusivamente notación matemática, la mayor parte de los autores se muestra más proclive a usar

una notación visual[7], a veces complementada con expresiones textuales (formales o no).

Entre las notaciones formales textuales podemos señalar el trabajo de Lano et al. [1996], cuyas especificaciones están basadas en una versión de Object Calculus, y el de Mikkonen [1998], que tiene su base en la lógica temporal de acciones.

Respecto a las notaciones que incorporan elementos gráficos en su representación, podemos destacar el lenguaje "LePUS" (Language of Patterns Uniform Specification) [Eden, 2000]. Éste es un lenguaje formal que sirve para describir y razonar sobre diseños orientados a objetos y, en particular, sobre patrones de diseño. La especificación se puede realizar mediante expresiones lógicas o mediante diagramas, ambos semánticamente equivalentes. La intención de LePUS es comunicar explícita y precisamente aspectos tanto estructurales como de comportamiento.

Mientras que la notación empleada en UML sirve para describir un sistema software en un nivel tal que el paso a la implementación es casi directo[8], LePUS opera de forma más abstracta utilizando un universo de discurso simple, donde las clases y las funciones son entidades atómicas. LePUS permite definir relaciones entre conjuntos de estas entidades (herencia, referencia, etc.), las cuales deben existir en un programa para poder conformar con la especificación realizada.

Este lenguaje, en su forma textual, emplea secuencias de proposiciones para construir fórmulas bien formadas. Dichas proposiciones están constituidas por la combinación de variables, predicados y operadores. El dominio de los valores que pueden tomar las variables (funciones, clases, conjuntos uniformes), el tipo de predicados (*hierarchy (C)*, *Clan (F,C)*, *Tribe (F,C)*, …) y los operadores (s⊗c, …) caracterizan al lenguaje. Todos estos

[7] En contraposición a la utilización de notaciones textuales formales, pensamos que la adopción de notaciones visuales agiliza la interpretación y comunicación de los modelos, especialmente cuando se basan en notaciones gráficas ampliamente conocidas, como puede ser UML.

[8] Se usan símbolos que tienen una correspondencia directa con elementos empleados en la implementación (objetos, clases, atributos, etc.)

elementos tienen un símbolo gráfico que lo representa. Un diagrama realizado con LePUS es un predicado establecido sobre el universo del discurso, dicho de otra forma, un diagrama LePUS limita los modelos o programas que satisfacen el predicado especificado.

Otro interesante trabajo es el de Lauder y Kent [1998], donde una notación visual, denominada *Diagramas de Restricción* [Kent, 1997], combinada con los diagramas de clases de UML, es usada para el modelado de la estructura estática y dinámica de los patrones. De acuerdo con los autores, la especificación de un patrón debe ser dividida en tres niveles de modelado (roles, tipos y clases) ordenados de mayor a menor nivel de abstracción. El primero captura la esencia del patrón obviando detalles específicos del dominio de la aplicación, el segundo refina el anterior añadiendo normalmente restricciones específicas del dominio y el último representa una implementación concreta.

Le Guennec et al. [2000] ponen de relieve algunas de las limitaciones que poseen las colaboraciones parametrizadas para el modelado de patrones de diseño con UML y proponen un conjunto de modificaciones para el metamodelo de UML. La finalidad es modelar los patrones de diseño y representar sus ocurrencias con una mayor precisión, abriendo así una nueva vía para mejorar el tratamiento automático o semiautomático de éstos mediante el uso de herramientas.

Para ello, los autores definen las propiedades estructurales de un patrón como un conjunto de restricciones en el nivel M2 (meta-nivel) de UML. Así, para especificar un patrón de diseño de manera precisa usan colaboraciones en el meta-nivel con restricciones OCL explícitas.

Meijers [1996] define un modelo conceptual para representación y tratamiento de patrones de diseño denominado *Fragment Model*, el cual es llevado a la práctica con el desarrollo de un prototipo de herramienta capaz de definir e instanciar patrones.

Para Meijers, los elementos relevantes de un patrón no se reducen únicamente a las clases y asociaciones que lo componen, sino que también son elementos esenciales los atributos, las relaciones de herencia y los métodos. Cada uno de estos componentes relevantes para un patrón son capturados mediante lo que denomina un *fragmento*.

Los fragmentos están enlazados por los llamados *roles*. Un rol es propiedad de un fragmento y conecta con los fragmentos que son actores de ese rol. El rol determina el tipo de fragmento que puede jugarlo e impone una restricción sobre los actores. También es posible asociar una cardinalidad al rol para limitar el número de actores enlazados a éste. Cada patrón tiene un fragmento raíz al cual se conectan todos los fragmentos relevantes mediante su rol particular. El fragmento raíz es un fragmento más y puede incluso ser actor de otros roles.

Otro trabajo en la línea de las notaciones visuales es el que realizan Mapelsden et al. [2002]. Ellos proponen el lenguaje DPML (Design Pattern Modelling Language) compuesto por un conjunto elementos constructores propios para el modelado de patrones de diseño y sus instancias dentro de modelos de diseño UML. El modelo correspondiente a la solución del patrón se enlaza con elementos de modelado de UML que conforman con dicho modelo.

France et al. [2004] describen una técnica de metamodelado para especificar patrones de diseño mediante el lenguaje RBML [Kim et al., 2003], el cual es un lenguaje visual basado en UML y restricciones OCL, el cual ayuda a caracterizar familias de modelos UML en términos de roles asociados con metaclases de UML como base.

Otros trabajos [Albin-Amiot y Gueheneuc, 2001; Mak et al., 2004] presentan también técnicas basadas en el metamodelado para representar patrones.

Analizando los trabajos existentes podemos llegar a la conclusión de que cada una de las notaciones propuestas presenta alguno/s de los siguientes inconvenientes:

- Complejidad.

  A medida que la notación empleada es más formal, por lo general, suele ser más difícil de entender y, por tanto, de aplicar. Esto va en contra del objetivo que persiguen los patrones, que es facilitar el modelado. Además, los modelos se complican enormemente conforme que el patrón se hace más complejo.

Aunque una herramienta para la aplicación/validación de patrones se podría ver beneficiada con el uso de un enfoque formal, a nivel de usuario es necesario establecer un equilibrio entre formalidad y usabilidad de la notación.

- Poca difusión.

  Para facilitar su uso, ésta debería estar basada en alguna notación ampliamente conocida.

- Limitación a ciertos tipos de patrones.

  La notación debería permitir modelar cualquier tipo de patrón, independientemente de los elementos de modelado que intervengan.

  Prácticamente todas están enfocadas hacia la especificación de estructuras formadas por clases/objetos (patrones de diseño). Sin embargo, cuando modelamos un sistema, además de objetos y clases, usamos otros tipos de elementos (estados, actividades, paquetes, eventos, actores, roles, etc.) que nos permiten reflejar muchos otros aspectos relevantes (flujos de trabajo, estructura social de los usuarios, dinámica de la organización, protocolos sociales, cooperación entre actores, etc.) los cuales se recogen principalmente en fases tempranas de modelado. Los patrones conceptuales o de análisis [Fowler, 1997; Riehle y Züllighoven, 1996], aplicables en estas etapas iniciales, también deberían poder representar este tipo de elementos.

- Centrada exclusivamente en la vista interna del patrón.

  La notación debería ser capaz de modelar también su vista externa.

- Modifica directamente el metamodelo de UML.

  Antes de realizar este tipo de modificación (extensión dura), es necesario contemplar los propios mecanismos de extensión que incluye el estándar y que respetan el metamodelo original (extensión suave).

Para reducir estas limitaciones, en este capítulo presentamos un profile que realiza una extensión respetuosa de UML (*Unified Modeling*

Language) [OMG, 2003], a través de un pequeño conjunto de elementos. Esta extensión facilita la representación simple, intuitiva y fácil de comprender, tanto de la vista interna como externa de un patrón de software en general, los cuales consisten en diagramas genéricos de cualquier tipo que pueden reutilizarse para construcción de los modelos que se crean durante las distintas fases de desarrollo, incluyendo, por supuesto, los modelos conceptuales que necesitamos para el modelado inicial de los sistemas cooperativos. De acuerdo con la visión de Coplien [1996], nuestra intención es que la representación de un patrón de software permita presagiar el resultado de su aplicación, de forma que las personas que lo usen puedan comprender su esencia sin demasiado esfuerzo, casi de un simple "vistazo".

Otros trabajos han usado mecanismos de extensión suave para modelar patrones con UML, pero con enfoques muy diferentes al que presentamos en este capítulo.

Probablemente, el más destacable entre estos trabajos sea el que ha sido elaborado en el propio marco del OMG [OMG, 2004]. En este se presenta un profile para la especificación EDOC (Enterprise Distributed Object Computing) que proporciona medios estándar (Business Function Object Patterns – BFOP[9]) para expresar modelos de objetos. El objetivo es reutilizar modelos estándar para construir buenos modelos de objetos (entidades, procesos, eventos y reglas del negocio) para sistemas EDOC.

En nuestra opinión, este profile presenta algunos problemas. Por ejemplo, el cuerpo del patrón se define como un diagrama de colaboración. Los patrones consisten en estructuras estáticas formadas por tipos y relaciones entre ellos. El profile está enfocado hacia la aplicación de patrones de diseño orientados a objetos, ya que los patrones son instanciados (desplegados) como diagramas de clases/objetos. Como se ha comentado anteriormente, esto da lugar a problemas para representar otros tipos de diagramas útiles durante una etapa de modelado conceptual. Además, no establecen ningún tipo de restricción sobre la instanciación del modelo (p. ej. la cantidad de instancias posibles para cada uno de sus elementos) y cómo se

[9] BFOP es un conjunto de patrones de objetos que se organizan en una estructura jerárquica multicapa. Esto facilita la herencia y mantiene la consistencia con el modelo de objetos

vería afectado el despliegue. El profile se centra más bien en cómo construir nuevos patrones de diseño a partir de otros más simples, mediante operaciones de herencia y composición de patrones.

Dong y Yang [2003] definen un profile para la identificación de patrones de diseño dentro de los modelos, asociando a los elementos de modelado el nombre del patrón e instancia a la que pertenecen, así como el rol que desempeñan dentro de éste. Sin embargo, no dicen nada acerca de cómo representar la vista interna de éstos. Además sólo tienen en cuenta las clases, atributos y operaciones como los únicos elementos de un patrón.

Sanada y Adams [2002] proponen un profile para representar patrones de diseño y frameworks. Sin embargo, éstos se centran únicamente en patrones de diagramas de clases y la extensión incorpora sólo aquellos elementos encaminados a facilitar la comprensión de los frameworks.

Por último, Debnath et al. [2006] muestra una forma de definir patrones de diseño mediante profiles, sugiriendo la definición de un profile para cada patrón. Según los autores, como los profiles extienden el vocabulario de UML y los patrones definen un vocabulario común para los diseñadores, es posible usar los profiles para definir un vocabulario de patrones en UML. Desde nuestro punto de vista, creemos que esta perspectiva no mejora otras que ya hemos comentado. Tampoco justifica ni vemos práctica la utilización de este mecanismo, cuyo propósito, en principio, es la adaptación de UML a diferentes dominios o plataformas. Aunque los autores presentan como una ventaja el hecho de que no sea necesario definir una notación especial, más que un beneficio, en la práctica esto puede suponer una desventaja, ya que la definición de cada profile requiere de la definición de un estereotipo por cada elemento del patrón y la especificación OCL de sus restricciones asociadas, lo que podría complicar aún más el entendimiento, frente a la utilización de una notación visual específica.

4. PMP (Pattern Modelling Profile): Un Perfil para el Modelado de Patrones

4.1. Introducción

UML dispone de un conjunto de conceptos y notaciones que permiten modelar la mayoría de los escenarios que nos encontramos durante el desarrollo de software. Sin embargo, muchas veces necesitamos modelar situaciones que van más allá del estándar. Por ejemplo, como hemos podido comprobar anteriormente (v. sec. 2.2), UML presenta ciertas dificultades para modelar convenientemente los patrones de software.

Para salvar esta circunstancia existen fundamentalmente dos vías. La primera consiste en modificar directamente el metamodelo[10] de UML. En este caso, habría que tener en cuenta que estaríamos modificando la definición de un estándar, lo que no es muy recomendable. La segunda forma, la cual debería ser contemplada antes que cualquier otra opción, sería la utilización de los mecanismos de extensión del propio lenguaje.

4.2. Mecanismos de extensión de UML

Estos mecanismos permiten extender el lenguaje de forma controlada. Mediante ellos es posible añadir nuevos tipos de elementos de modelado y asociar información diversa con los elementos ya existentes. Los *estereotipos*, las *etiquetas* y las *restricciones* son los mecanismos utilizados por UML para este fin.

Un *estereotipo* es un metaelemento definido por el usuario que mantiene una estructura similar a la de un metaelemento predefinido (su clase base) en UML. Dicho de otro modo, los estereotipos permiten la definición virtual de subclases de metaclases del metamodelo, con metaatributos y semántica adicional, permitiendo la extensión del vocabulario de modelado como si fueran bloques de construcción primitivos.

[10] La inclusión explícita de elementos en el metamodelo de UML es posible si la herramienta de modelado soporta OMG Meta Object Facility (MOF).

Su declaración aparece en la capa de "modelo", conocida como M1, el tercer nivel en la jerarquía de cuatro capas[11] de la arquitectura de metamodelado de UML, aunque conceptualmente pertenece a la capa inmediatamente superior, la capa de "metamodelo" o M2, ya que permite definir o extender el lenguaje para especificar los modelos (v. Fig. 4.4).

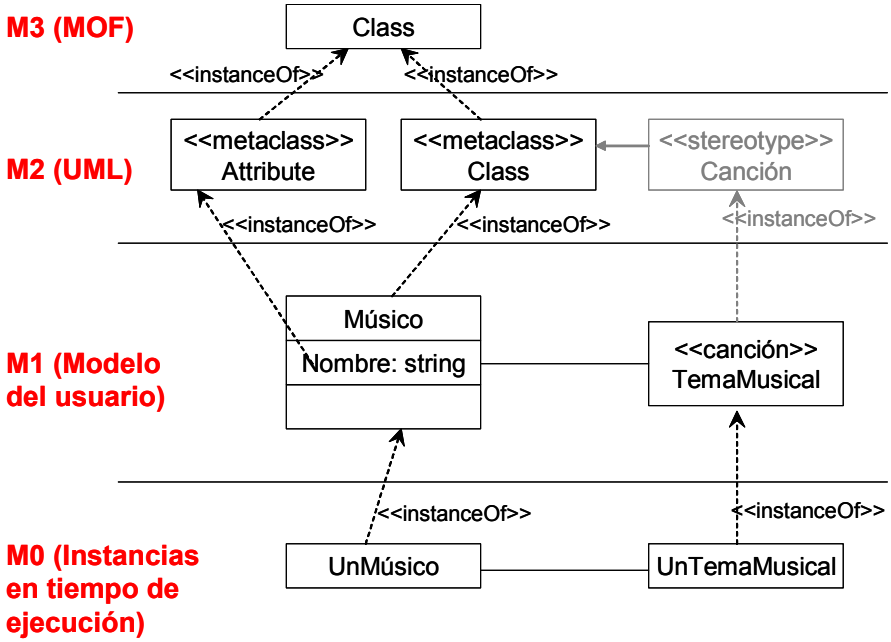


Fig. 4.4: Estereotipos y jerarquía de metamodelado a cuatro capas en UML

Un estereotipo se denota por un nombre entre comillas tipográficas y se sitúa junto al nombre del elemento que está estereotipando. Opcionalmente se puede proporcionar un icono que permite distinguir visualmente las abstracciones. Este icono puede aparecer a la derecha del nombre o ser usado como la representación gráfica del elemento estereotipado (v. Fig. 4.5).

[11] De arriba hacia abajo en la jerarquía: capa de meta-metamodelo (M3), capa de metamodelo (M2), capa de modelo (M1) y capa de objetos de usuario (M0). Un ejemplo de meta-metamodelo sería MOF (Meta Object Facility), mientras que UML sería una instancia de este meta-metamodelo, es decir, cada elemento de UML es una instancia de un elemento en MOF.

Fig. 4.5: Varias notaciones para un elemento con el estereotipo `<<canción>>`

Además de añadir nuevos elementos, es posible asociar nuevas propiedades a los elementos mediante la definición de *etiquetas* a las que se asignan valores, denominados valores etiquetados. Se especifican mediante una cadena de caracteres entre llaves que incluye el nombre de la etiqueta, un separador (el símbolo =) y el valor etiquetado asignado (v. Fig. 4.6). También es posible especificar solamente el valor de la etiqueta cuando no existe ambigüedad.

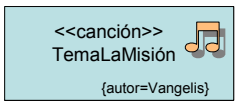


Fig. 4.6: Etiqueta `autor` asociada al elemento `TemaLaMisión` estereotipado como `<<canción>>`

Por último, las *restricciones* permiten refinar la semántica de los elementos o modificar las reglas existentes de UML. Éstas pueden ser expresadas de una manera informal, mediante lenguaje natural, o formalmente, haciendo uso del lenguaje OCL (Object Constraints Language) [OMG, 2006], integrado en UML. La ventaja de utilizar una notación formal reside en que una herramienta podría interpretar sus expresiones y obligar su cumplimiento durante el modelado. Las restricciones se representan como una cadena de caracteres entre llaves junto al elemento asociado (v. Fig. 4.7).

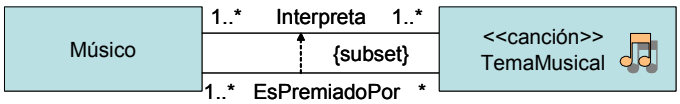


Fig. 4.7: Restricción `{subset}` indicando que las obras musicales por las cuales es premiado un músico son un subconjunto del total de obras que interpreta

Tanto los valores etiquetados como las restricciones pueden aplicarse a elementos ya existentes o a elementos estereotipados.

4.3. Definición de PMP

Un *perfil* o *profile* es un conjunto de estereotipos, restricciones y/o etiquetas que adaptan UML con objeto de personalizar el modelado para distintos dominios, plataformas o métodos.

En nuestro caso, vamos a definir un profile denominado *PMP* (*P*attern *M*odelling *P*rofile o *P*erfil para *M*odelado de *P*atrones) que incluye las extensiones a UML necesarias para permitir el modelado de patrones de software consistentes en modelos o diagramas de cualquier tipo reutilizables durante las etapas de análisis o diseño de software.

4.3.1. Objetivos

Una de las cualidades más interesantes que poseen los patrones es la de facilitar el modelado de software.

Como ya sabemos, entre otras posibles ventajas, la aplicación de patrones durante el modelado de software facilita la creación de modelos mediante la reutilización de otros (patrones) que actúan como guías o bloques de construcción. Reutilizamos la experiencia de otros modeladores para modelar situaciones similares. Además, la visualización de los patrones empleados facilita la descripción de los modelos mejorando su comprensión, comunicación y mantenimiento.

Para la representación[12] interna y externa de un patrón es preciso dotar al lenguaje de modelado de una sintaxis (notación) y semántica específica. Por un lado, el modelo que proporciona el patrón (vista interna) debe ser lo suficientemente genérico y flexible como para que pueda ser instanciado en cualquier situación que cumpla las condiciones de aplicación del patrón. Por otro lado, es preciso identificar en el modelo generado los

[12] Aunque la descripción completa de un patrón incluye muchos otros aspectos (contexto de aplicación, relaciones con otros patrones, ejemplo de aplicación, etc.), los cuales son recogidos en estructuras tabulares conforme a un formato de descripción uniforme (v. cap. III, sec. 2.4), entendemos que son su vista externa e interna los aspectos que requieren de una notación específica.

patrones que han sido aplicados y cómo éstos se relacionan con los elementos de modelado existentes (vista externa).

Tradicionalmente, el modelado de patrones se ha caracterizado por la representación de estructuras formadas por clases/objetos reutilizables durante el diseño. Sin embargo, cuando modelamos un sistema, además de objetos y clases, usamos otros tipos de elementos (estados, actividades, paquetes, eventos, actores, roles, etc.) que nos permiten reflejar muchos otros aspectos relevantes (flujos de trabajo, estructura social de los usuarios, dinámica de la organización, protocolos sociales, cooperación entre actores, etc.) los cuales son recogidos principalmente durante la fase de análisis.

Así pues, partiendo de una perspectiva más general, definimos un patrón como un modelo (formado por elementos de cualquier tipo) genérico y flexible que representa una familia de modelos (las instancias del patrón). La idea es usar este modelo como guía para la creación y/o descripción de otros modelos semejantes, mediante la correspondencia de los elementos del patrón con los elementos que forman sus instancias.

Para conseguir el grado de generalización y flexibilidad necesario, consideramos un patrón como un modelo parametrizado o plantilla en la que sus elementos pueden llevar asociadas restricciones de multiplicidad, optatividad, agrupamiento, incertidumbre, etc., de los elementos que representan. De esta forma, dado un patrón, los modelos que pueden ser generados o descritos a partir de éste van a poder adoptar configuraciones muy diversas.

Decimos que un modelo es una *instancia u ocurrencia de un patrón* cuando cumple las restricciones que el patrón impone en cuanto al tipo, relaciones y número de elementos que deben aparecer.

Nuestra visión va más allá de la idea que mantiene UML acerca de las plantillas, las cuales consisten en modelos rígidos parametrizados que dan lugar a copias de éstos por sustitución de sus parámetros. Nuestra intención es incluir en éstas un cierto grado de flexibilidad que permita generar o describir las posibles variantes (instancias) de un mismo patrón.

4.3.2. Estereotipos

En la Tabla 4.1 se definen los estereotipos necesarios para el modelado de patrones de software.

Las columnas de la tabla se interpretan como sigue:

- *Estereotipo*: Nombre del estereotipo definido.
- *Clase base*: Elemento del metamodelo de UML que sirve como base para el estereotipo.
- *Etiquetas*: Una lista con todas las etiquetas de los valores que pueden asociarse al estereotipo. Si no es aplicable se indica "N/A".
- *Restricciones*: Condiciones que los elementos estereotipados deben cumplir para que el modelo esté bien formado. Cada restricción tiene una referencia asociada que es interpretada en la Tabla 4.2.
- *Descripción*: Semántica asociada al estereotipo.

Tabla 4.1: Definición de estereotipos

Estereotipo	Clase Base	Etiquetas	Restr.	Descripción
<<Pattern>>	Package	Clasifi-cation	(1)	Un patrón (pattern) es un paquete parametrizado que incluye un modelo genérico reutilizable. La definición de dicho modelo va a permitir que pueda ser usado como una plantilla extensible y adaptable para la construcción y/o descripción de modelos concretos semejantes (instancias del patrón).
<<PatternElement>>	Element	N/A	N/A	Cada uno de los elementos de modelado que forman parte del modelo de un patrón. Son abstracciones que representan a los distintos elementos de modelado que componen una instancia concreta de éste.
<<PatternNamedElement>>	NamedElement	N/A	N/A	Es un <<PatternElement>> que posee un identificador.
<<UncertainElement>>	Element	N/A	N/A	Permite representar aquellos elementos indeterminados del

Estereotipo	Clase Base	Etiquetas	Restr.	Descripción
				contexto, fronterizos al patrón, que necesariamente tendrán que conectarse con sus instancias para que éstas estén bien formadas.
<<PatternBind>>	Dependency	N/A	(2) (3) (4)	Indica que el elemento de la cola de la dependencia (cliente) está siendo representado, dentro de un patrón, por el <<PatternNamedElement>> al que apunta (proveedor). Si no está visible el símbolo del <<PatternNamedElement>> la dependencia puede apuntar directamente al paquete que simboliza el patrón.
<<MultiplicityBind>>	Constraint	N/A	N/A	Delimita la cantidad de elementos de una instancia que pueden relacionarse mediante una dependencia de tipo <<PatternBind>> con el <<PatternNamedElement>> afectado por la restricción.
<<UsedPattern>>	Dependency	N/A	N/A	Es una relación entre dos patrones en la que el patrón de la cola de la dependencia (cliente) se compone del patrón de la cabeza de la dependencia (proveedor).
<<PatternCollection>>	Package	N/A	N/A	Agrupación de patrones que forman un catálogo o colección aplicable en un dominio específico.
<<PatternView>>	Model	N/A	N/A	Vista que incluye las relaciones que existen entre los patrones utilizados durante el modelado del sistema.

Tabla 4.2: Tabla de restricciones

Restricción	Descripción
(1)	Debe tener al menos un parámetro.
(2)	El elemento cliente debe ser del mismo tipo que el elemento proveedor, a no ser que el elemento proveedor no esté visible y la dependencia apunte directamente al paquete que representa el patrón.
(3)	El número de clientes debe conformar con la restricción de multiplicidad especificada en

Restricción	Descripción
	el elemento proveedor.
(4)	Dados dos elementos cliente pertenecientes a una misma instancia de un patrón, entre ellos deben aparecer al menos las mismas relaciones que existen entre sus respectivos elementos proveedores.

A continuación, de acuerdo con todos los estereotipos que acabamos de describir, definimos nuestro profile (v. Fig. 4.8) incluyendo todas las extensiones realizadas a las metaclases del metamodelo de UML:

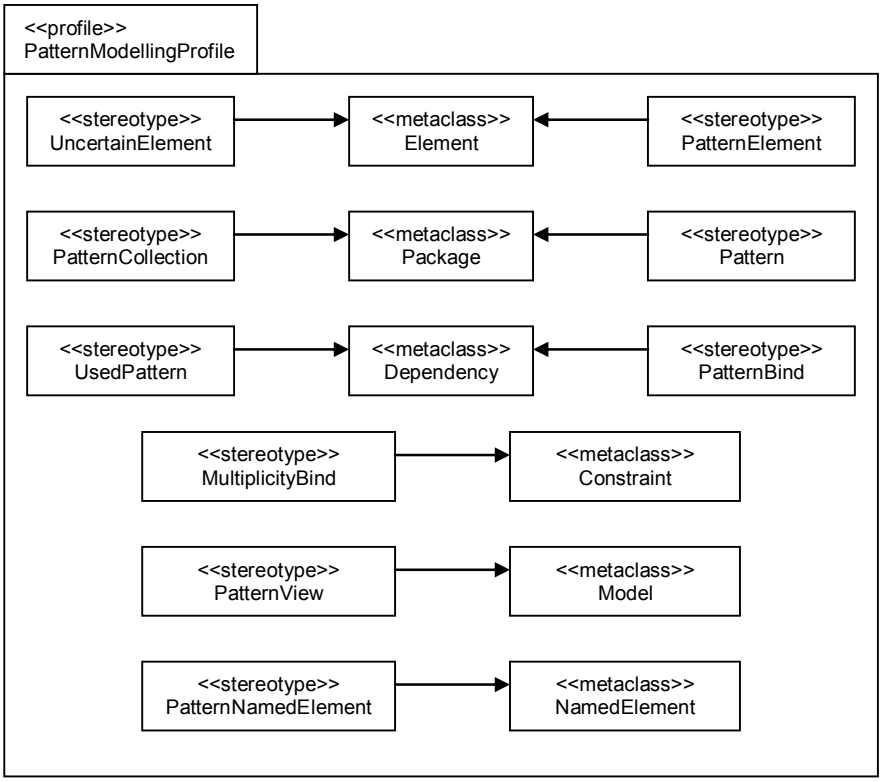


Fig. 4.8: Definición de PMP (`PatternModellingProfile`)

La Figura 4.9 muestra el metamodelo que especifica las relaciones estructurales existentes entre los distintos estereotipos definidos:

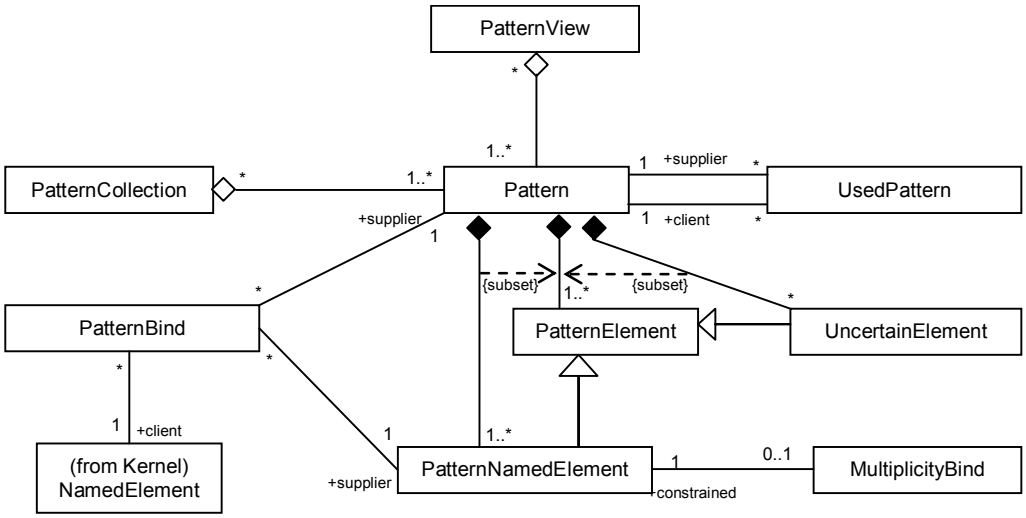


Fig. 4.9: Metamodelo correspondiente a los elementos de PMP

4.3.3. Etiquetas

La siguiente tabla presenta las etiquetas correspondientes a los valores etiquetados que pueden asociarse con algunos de los estereotipos. El significado de cada una de las columnas es el siguiente:

- *Etiqueta*: Nombre de la etiqueta.
- *Estereotipo*: Nombre del estereotipo al que se aplica la etiqueta.
- *Tipo*: El tipo de los valores que pueden asociarse a la etiqueta.
- *Multiplicidad*: Máximo número de valores que puede tener una instancia de etiqueta.
- *Descripción*: Explicación informal de la semántica asociada a la etiqueta.

Tabla 4.3: Definición de etiquetas

Etiqueta	Estereotipo	Tipo	Multiplicidad	Descripción
Type	Pattern	PrimitiveTypes::String	1	Indica el tipo de patrón según alguna catalogación previamente establecida.

4.3.4. Notación para los estereotipos

4.3.4.1. <<Pattern>>

Un patrón lo vamos a representar mediante un paquete parametrizado, estereotipado como `<<Pattern>>` (v. Fig. 4.10), que incluye un modelo genérico reutilizable que se define en base a un conjunto de elementos de tipo `<<PatternElement>>`, `<<PatternNamedElement>>`, `<<UncertainElement>>` y restricciones `<<MultiplicityBind>>`.

Los argumentos van ser cadenas de caracteres que sustituyen a los parámetros del patrón (todos de tipo string) para especificar los identificadores de ciertos `<<PatternNamedElement>>` que pueden variar de una instancia a otra del patrón.

Los parámetros se reconocen dentro del diagrama por ir encerrados entre ángulos (véase `<P1>`, `<P2>` y `<P3>` en Fig. 4.10).

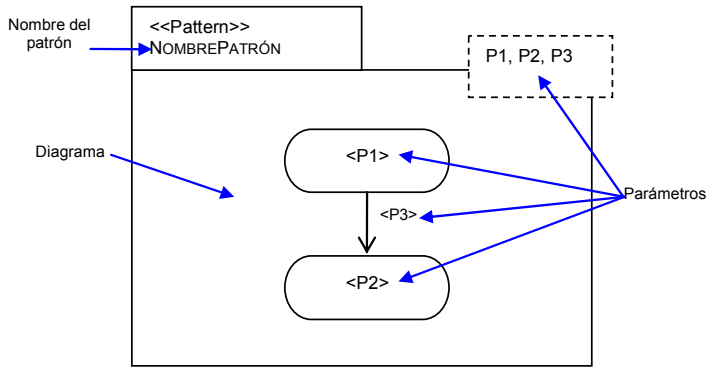


Fig. 4.10: Paquete parametrizado y estereotipado como `<<Pattern>>` que representa el patrón NOMBREPATRÓN

Los parámetros formales suelen aparecer dentro de un rectángulo punteado sobre la esquina superior derecha del paquete. La lista de parámetros nunca debería estar vacía. Éstos deben aparecer separados por comas. Su sintaxis usando notación BNF es la siguiente:

<ListaParámetros> ::= <NombreParámetro> [(',' <NombreParámetro>)]*

Donde *<NombreParámetro>* es el identificador del parámetro dentro del ámbito del patrón. Es de tipo string.

Para mantener la compatibilidad con UML, los patrones de diseño orientado a objetos (diagramas de clases/objetos que interaccionan) podrán también ser modelados como colaboraciones parametrizadas[13] (v. Fig. 4.3).

4.3.4.2. <<PatternElement>> y <<PatternNamedElement>>

Estos elementos no poseen un símbolo especial que los diferencie de aquellos que están representando. No va a ser necesario estereotiparlos ya que los vamos a reconocer por su inclusión en un paquete estereotipado como `<<Pattern>>`. La adición de un estereotipo únicamente contribuiría a complicar la legibilidad del diagrama.

Véase, por ejemplo, cómo los patrones de la Figura 4.11, donde el PATRÓN-A representa un diagrama de actividad y el PATRÓN-B un diagrama de clases, son modelados en la Figura 4.12 sin necesidad de incorporar estos estereotipos, lo que redunda en una mejora de la claridad.

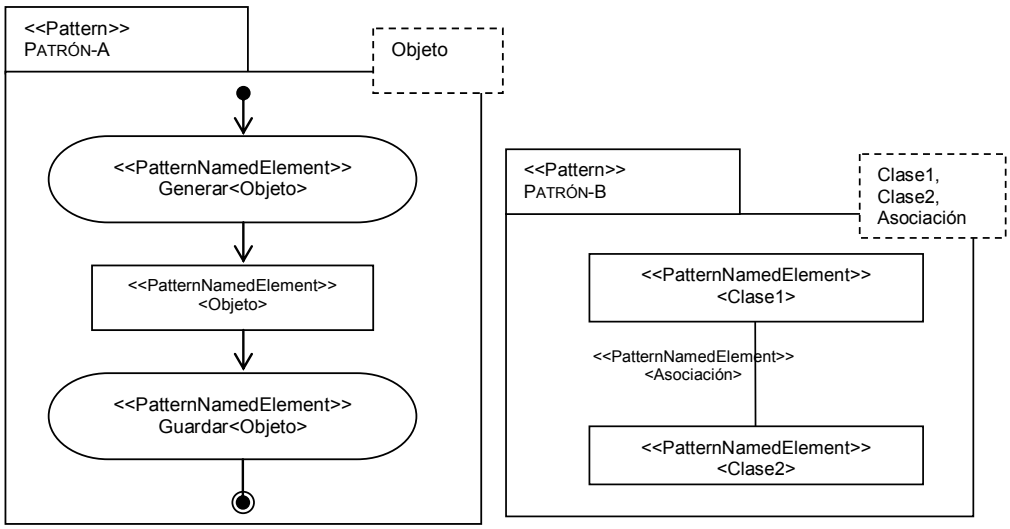


Fig. 4.11: PATRÓN-A y PATRÓN-B incluyendo sus elementos estereotipados

[13] En este mismo capítulo (v. sec. 2.2) hemos analizado los problemas que supone la utilización de colaboraciones parametrizadas para el modelado de patrones de software.

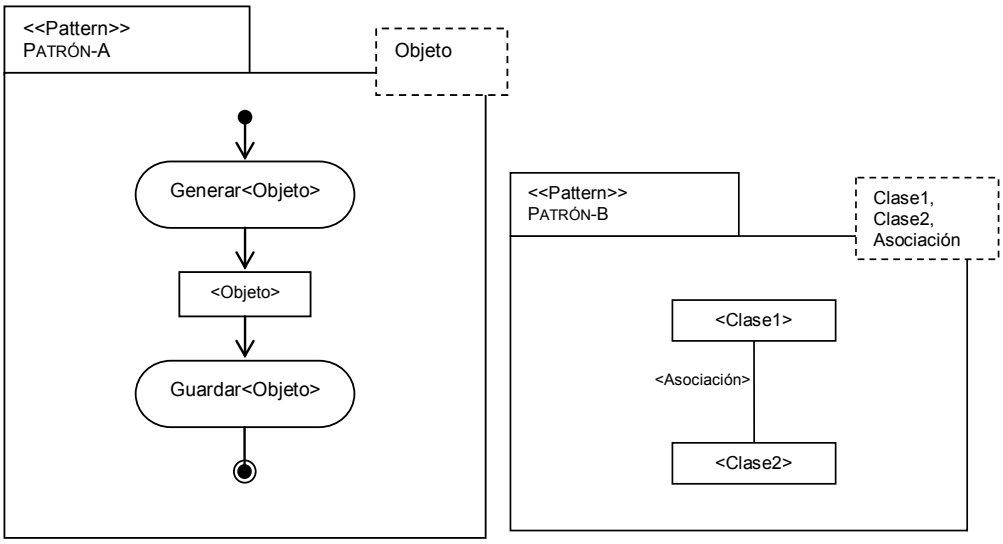


Fig. 4.12: PATRÓN-A y PATRÓN-B obviando los estereotipos de sus elementos para aumentar la legibilidad de los diagramas

Estos elementos pueden incluir parámetros, de forma que cuando el patrón recibe los argumentos necesarios para generar una instancia concreta, éstos son sustituidos por sus correspondientes valores. Los parámetros pueden existir de forma aislada (p. ej., `<Clase1>` en el PATRÓN-B) o formando parte de otra cadena de caracteres (p. ej., `Generar<Objeto>` en el PATRÓN-A, donde si el parámetro `Objeto` recibe como argumento el valor `"Mensaje"` el nombre de la actividad en la instancia sería `GenerarMensaje`).

4.3.4.3. <<UncertainElement>>

En ocasiones es necesario simbolizar aquellos elementos indeterminados del contexto, fronterizos al patrón, que necesariamente tendrán que conectarse con sus instancias para que éstas estén bien formadas. Actúan como puntos de enganche obligatorios de la instancia con elementos de modelado de su entorno. Esta parte desconocida[14] del modelo se representa mediante un hexágono punteado con un símbolo de interrogación en su interior.

[14] Autores como Cañete et al. [2004] ya pusieron de manifiesto la necesidad de introducir y controlar la incertidumbre en los modelos.

Por ejemplo, en la Fig. 4.13, el estado `A` es un estado alcanzable (cuando se activa la transición T) pero desconocemos, o no nos interesa modelar, a través de qué otros estados se puede llegar a éste.

La introducción de la incertidumbre facilita el modelado de patrones más generales, reduce la complejidad de los mismos y permite conectar éstos con los elementos de modelado que forman parte de su entorno.

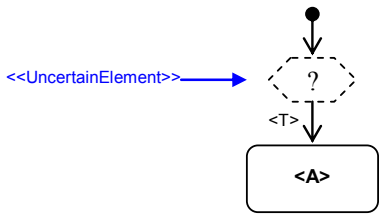


Fig. 4.13: Notación utilizada para representar el elemento `<<UncertainElement>>`

4.3.4.4. <<MultiplicityBind>>

Se denota como una expresión de multiplicidad encerrada entre llaves, es decir:

{*multiplicidad*}
donde *multiplicidad* es del tipo `UML::Datatypes::Multiplicity`.

Este tipo de restricciones se muestran junto a los `<<PatternNamedElement>>` afectados siguiendo las recomendaciones de UML. Sin embargo, cuando estas restricciones afectan a grupos de elementos en bloque, las indicaciones de UML no son suficientes. En este caso, encerramos los elementos afectados dentro de un rectángulo (o varios rectángulos si hay varios grupos dispersos) y conectamos éste (o éstos) a una elipse que incluye la restricción, todo dibujado con trazos discontinuos (v. Fig. 4.15).

Cuando esta restricción da lugar a una ligadura múltiple[15] en la que cada uno de los argumentos ligados de la instancia se debe identificar de

[15] Varios elementos de una misma instancia son ligados a un único `<<PatternNamedElement>>`

manera diferente, el identificador del parámetro debe llevar al final una variable entre corchetes. Esto va a permitir distinguir cada parámetro en la ligadura por medio de un índice numérico consecutivo entre corchetes (comenzando a partir de 1). Si el identificador del parámetro no está indexado y forma parte de una ligadura múltiple se entiende que todos sus argumentos van a poseer el mismo identificador.

En el siguiente ejemplo (v. Fig. 4.14), la multiplicidad de ligadura denota la existencia de un grupo de cero o más parámetros `Intermedio[i]` cuyos identificadores se diferenciarán por un índice (`Intermedio[1]`, `Intermedio[2]`, ...) en la especificación de su ligadura. Así, cada uno de estos parámetros podrá recibir un argumento diferente.

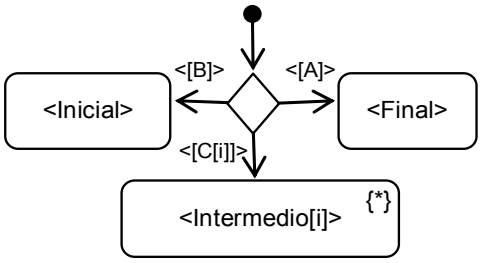


Fig. 4.14: Ligadura múltiple `{*}` asociada al parámetro múltiple `<Intermedio[i]>`

A continuación, la Figura 4.15 indica la existencia de un conjunto de ramas paralelas (al menos dos, véase la restricción de multiplicidad de ligadura `{2..*}`) que contienen actividades (`ActividadParalela[i]`) las cuales terminan con el envío de una misma señal (`FinActividad`).

Los nombres de las actividades que actúan como argumentos podrán tener nombres diferentes, identificándose los parámetros como `ActividadParalela[1]`, `ActividadParalela[2]`, etc. En el caso de la señal, todas compartirán el mismo identificador (varias ocurrencias de la misma señal), con lo que no hay que indexar el identificador del parámetro.

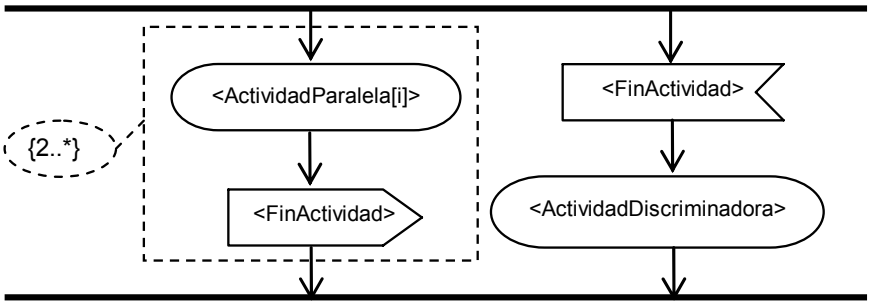


Fig. 4.15: Multiplicidad de ligadura para un bloque de elementos

La ligadura de un elemento puede ser opcional. En la Figura 4.16, el elemento `B` está restringido por una multiplicidad de ligadura `{0..1}` para indicar tal circunstancia. En este caso, el parámetro `<B>` sería opcional.

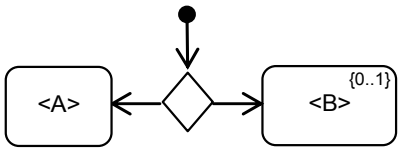


Fig. 4.16: Multiplicidad de ligadura para indicar la opcionalidad de un elemento

4.3.4.5. Relaciones

4.3.4.5.1. <<PatternBind>> y la ligadura de patrones

Las instancias de un patrón las construimos mediante la sustitución de sus parámetros por valores concretos dentro de un modelo. A este proceso lo denominamos *ligadura* (en inglés *binding*).

Decimos que una *ligadura* es *completa* cuando todos los parámetros incluidos en los `<<PatternNamedElement>>` obligatorios[16] reciben algún valor y se respetan las restricciones de multiplicidad asociadas.

[16] Aquellos que no están afectados por alguna restricción de multiplicidad de ligadura <<MultiplicityBind>> que especifique cero o más elementos.

Si un <<PatternNamedElement>> tiene asociada una restricción de multiplicidad indicando opcionalidad[17] y contiene algún parámetro no ligado, esto es, en su ligadura no se especifica ningún valor para ese parámetro, la instancia no incluirá elemento alguno relacionado con dicho <<PatternNamedElement>>.

Una relación de dependencia con el estereotipo <<PatternBind>> nos permite representar con qué elementos de sus instancias se relaciona cada uno de los <<PatternNamedElement>> de un patrón. De esta forma, podemos indicar que el elemento de la cola de la dependencia (cliente) es representado dentro del patrón por el <<PatternNamedElement>> al que apunta dicha dependencia (proveedor), o lo que es lo mismo, revelamos el papel que juega el elemento cliente dentro del patrón.

Lógicamente, como los parámetros del patrón forman parte de los <<PatternNamedElement>>, esta relación nos permite conocer cuáles son los argumentos que participan en la ligadura.

Por ejemplo, la Figura 4.17 muestra, por medio de relaciones <<PatternBind>>, qué elementos del modelo forman parte de una instancia concreta del patrón JOINT VENTURE[18] y con qué <<PatternNamedElement>> se corresponden. De esta manera es posible conocer el valor que cada parámetro ha asumido en dicha instancia, en concreto:

```
Socio -> "GrupoSubproyecto"
Administración -> "AdministraciónProyecto"
RepresentanteSocio -> "InvestigadorPrincipalSubproyecto"
Secretario -> "Secretario"
Coordinador -> "CoordinadorSubproyectos"
Director -> "DirectorProyecto"
```

[17] Cero o más elementos.

[18] Este patrón forma parte de nuestra propuesta inicial para un catálogo de patrones aplicables durante el modelado conceptual de sistemas cooperativos. Para una descripción detallada de todos estos patrones el lector puede consultar el Apéndice B de esta tesis.

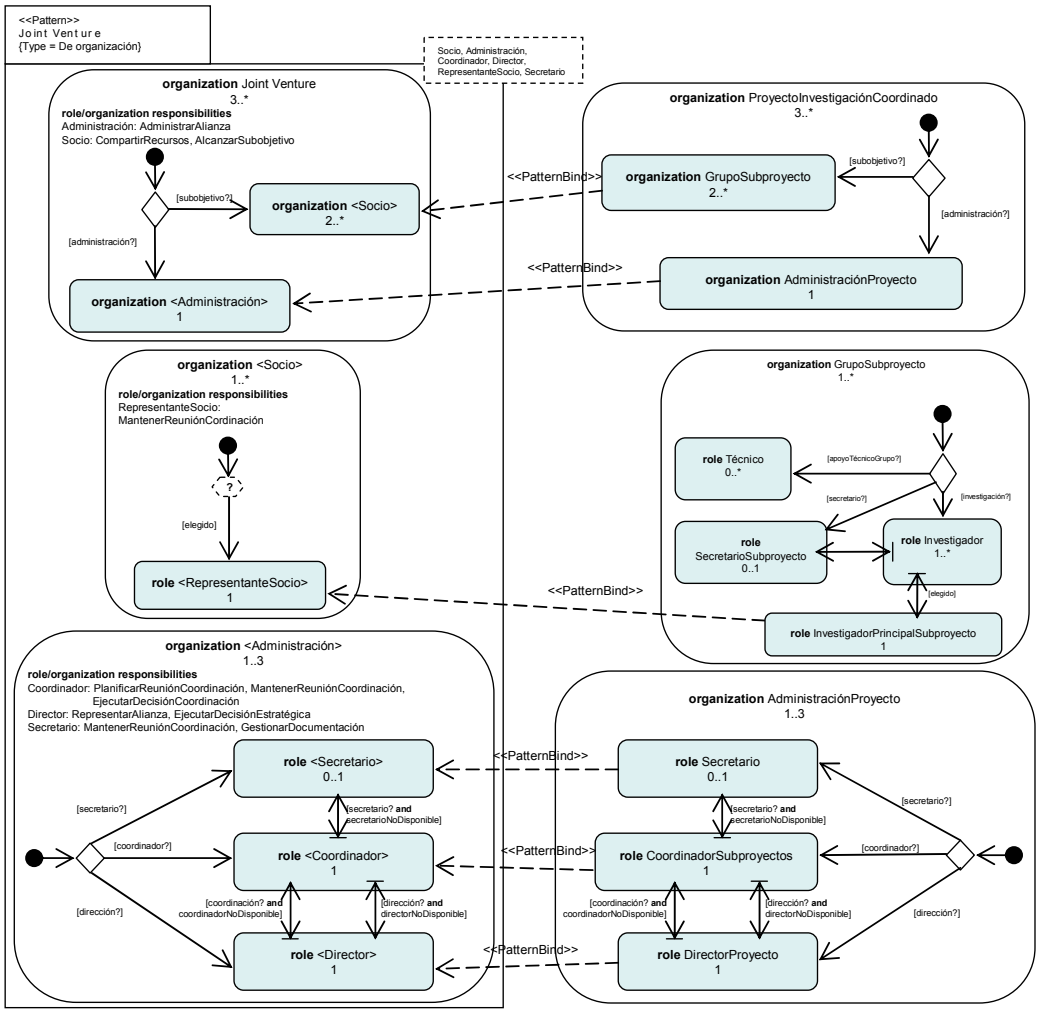


Fig. 4.17: Relación <<PatternBind>> entre los elementos contenidos en el patrón JOINT VENTURE y sus análogos dentro de un modelo concreto

A menudo el diagrama del patrón no está visible, quedando sus <<PatternNamedElement>> ocultos. En este caso, cada uno de los elementos que forman parte de la instancia del patrón puede apuntar directamente al paquete que simboliza el patrón mediante dependencias <<PatternBind>>, teniendo en cuenta que es necesario especificar de alguna forma cómo se ha llevado a cabo la ligadura.

A fin de no complicar gráficamente los diagramas y obtener una mayor flexibilidad, claridad y precisión, abogamos por la incorporación de notas que especifican la ligadura mediante la inclusión de expresiones de acuerdo con el siguiente formato en notación BNF:

'pattern ' <NombrePatrón> 'binding {' <ExpresiónBinding> '} ' [('OR 'pattern ' <NombrePatrón> 'binding {' <ExpresiónBinding> '} ')]*

donde:

<ExpresiónBinding> ::= (<ExpresiónLigaduraParámetro> | <ExpresiónLógica>) [(',' <ExpresiónLigaduraParámetro> | <ExpresiónLógica>)]*

<ExpresiónLigaduraParámetro> ::= <NombreParámetro> '->' (<valorArgumento> | '$')

*<ExpresiónLógica> ::= <ExpresiónAnd> (' OR ' <ExpresiónAnd>)**

<ExpresiónAnd> ::= '(' <ExpresiónLigaduraParámetro> [(' AND ' <ExpresiónLigaduraParámetro>)] ')'*

Un parámetro que recibe el valor '$' significa que su argumento coincide con el identificador del parámetro.

Las expresiones lógicas permiten especificar lo que hemos denominado la *ligadura dinámica* de patrones. Éstas serán tratadas convenientemente al final de esta sección.

La Figura 4.18 muestra la aplicación[19] del patrón JOINT VENTURE, cuyo modelo permanece oculto en el paquete que lo representa, mediante la conexión de dependencias `<<PatternBind>>` desde cada uno de los elementos de su instancia al símbolo del patrón.

Como podemos apreciar, mediante una etiqueta ligada al patrón indicamos la especificación de ligadura que da lugar a la instancia representada. En dicha etiqueta expresamos que `Socio`, `Administración`, `Coordinador`, `Director` y `RepresentanteSocio` se ligan con `"GrupoSubproyecto"`, `"AdministraciónProyecto"`, `"Coordinador Subproyectos"`, `"DirectorProyecto"` e `"InvestigadorPrincipal Subproyecto"` respectivamente, mientras que el parámetro `Secretario`

[19] Una explicación detallada acerca de lo que estamos representando en este modelo puede encontrarse en el Capítulo VI

recibe el valor '$', o lo que es lo mismo, el identificador del propio parámetro: la cadena "`Secretario`".

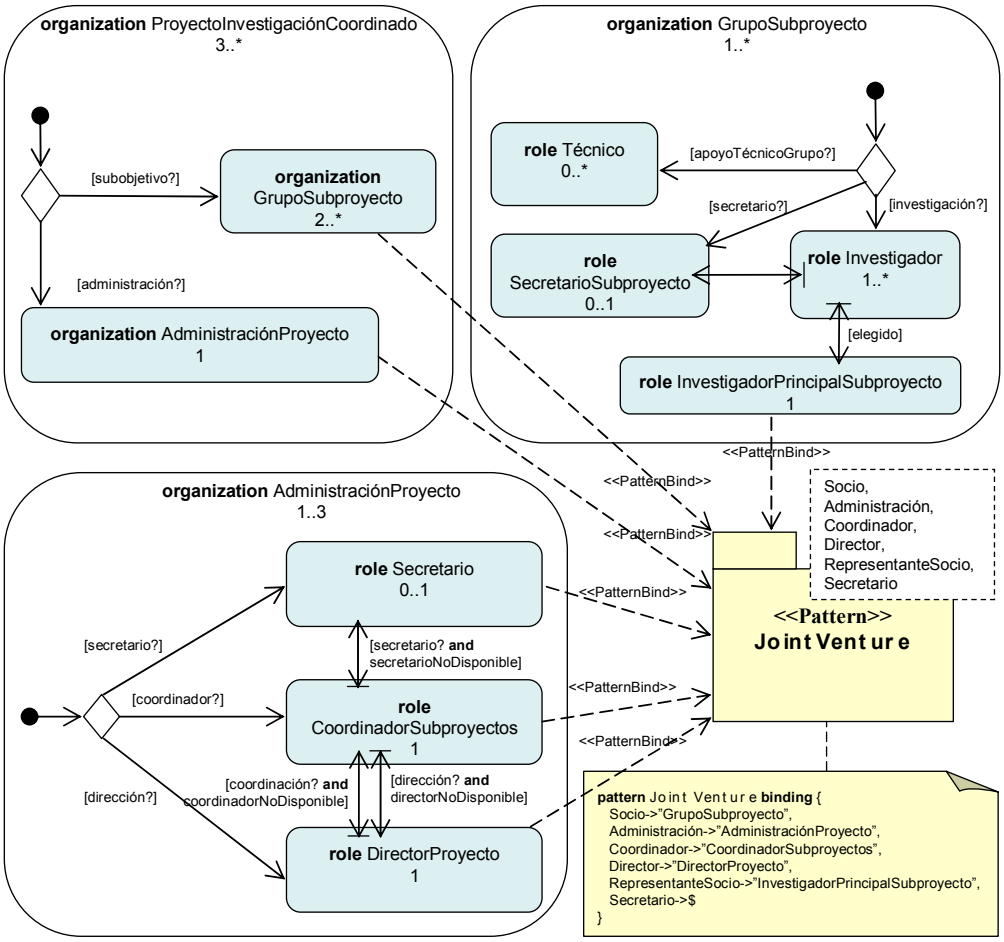


Fig. 4.18: Patrón JOINT VENTURE oculto en el paquete y su aplicación de acuerdo con la especificación de ligadura incluida en una nota asociada al paquete

También es posible mostrar este tipo de etiquetas conectándolas directamente con todos y cada uno de los elementos que forman la instancia o encerrando la instancia dentro de un rectángulo punteado y conectando la etiqueta a dicho rectángulo.

Por ejemplo, la Figura 4.19 muestra una instancia del patrón PETICIÓN-RESPUESTA MÚLTIPLE[20] (Fig. 4.20) y la especificación de la ligadura utilizando el primer procedimiento.

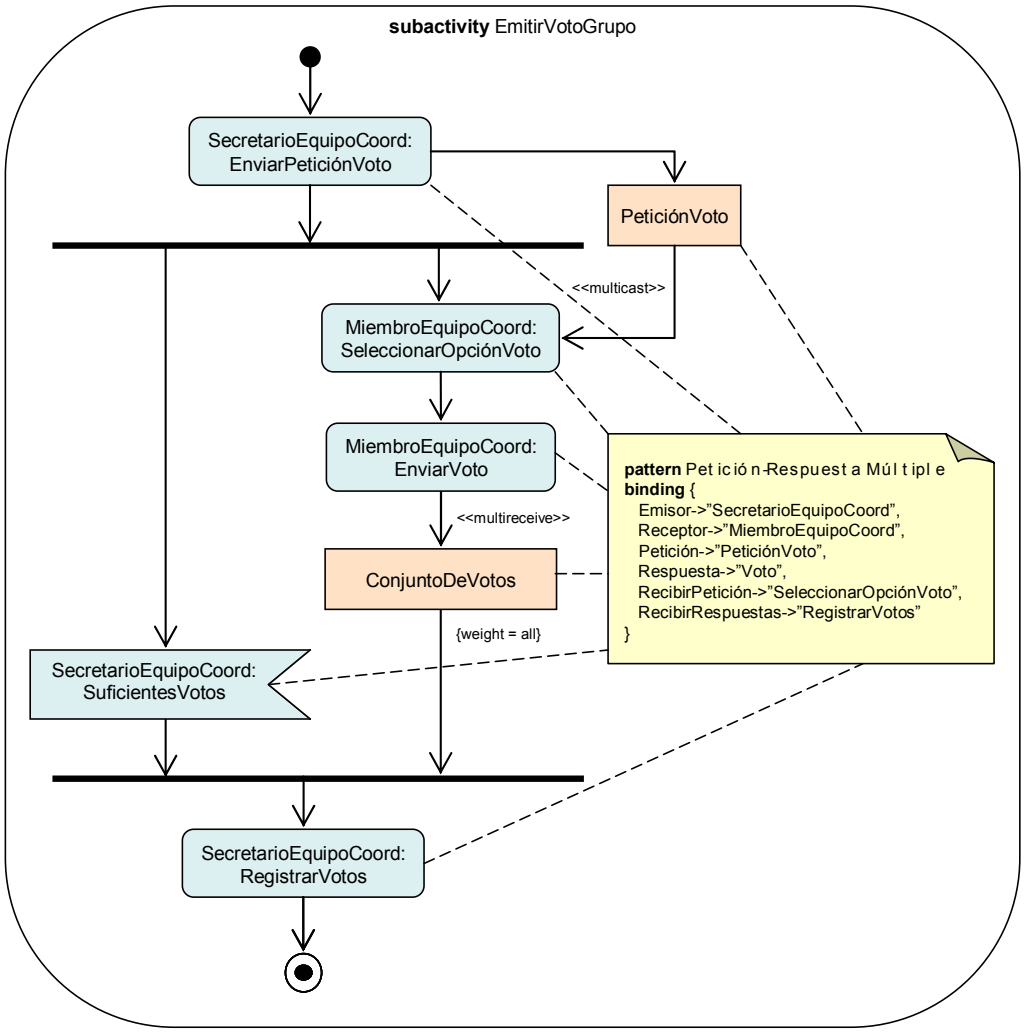


Fig. 4.19: Visualización de la ligadura del patrón PETICIÓN-RESPUESTA MÚLTIPLE para el modelado de la subactividad `EmitirVotoGrupo` conectando una expresión de `binding` directamente con los elementos que componen la instancia

[20] Para una descripción detallada de este patrón puede consultarse el Apéndice B de esta tesis

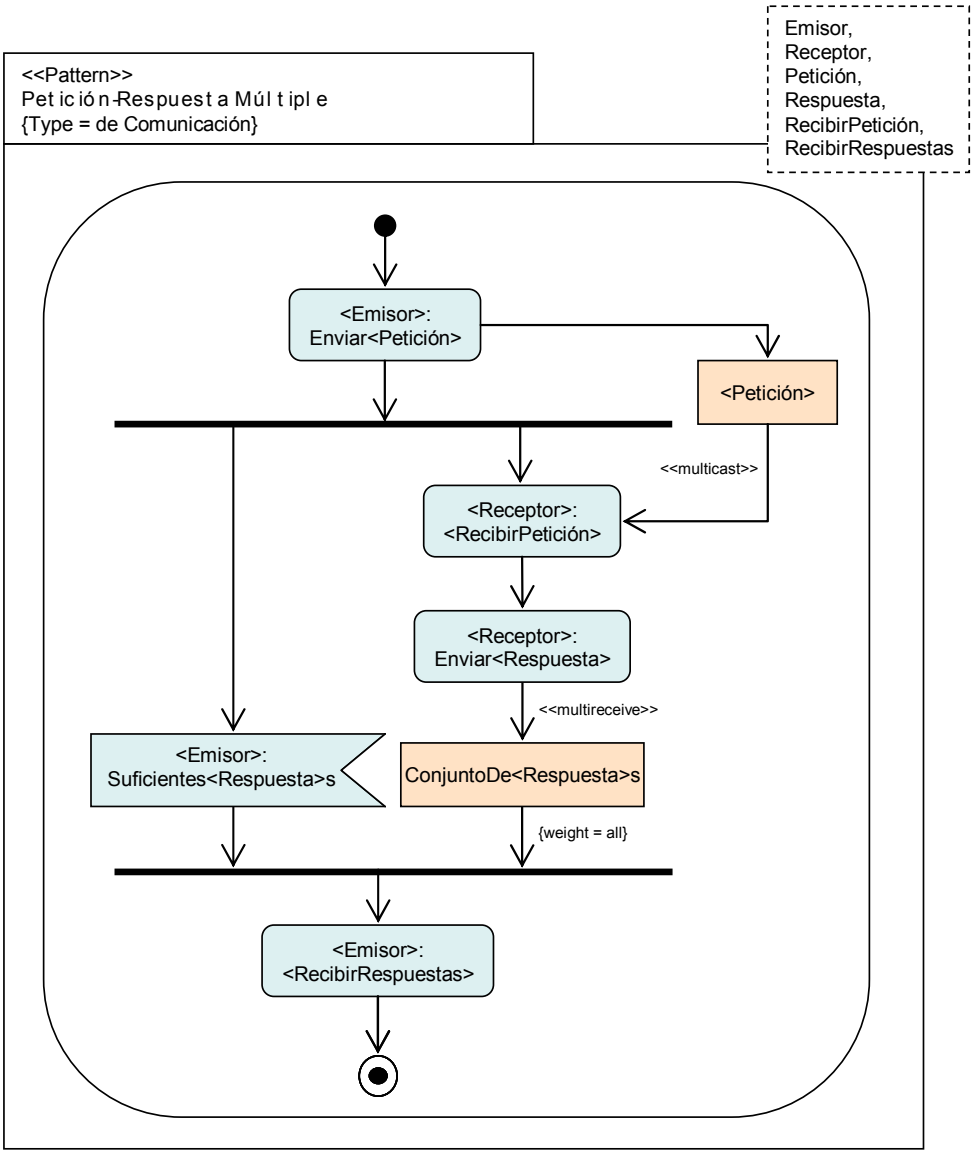


Fig. 4.20: Definición del patrón PETICIÓN-RESPUESTA MÚLTIPLE

La Figura 4.22 muestra una instancia del patrón PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO[21], cuyo modelo presentamos en la Figura 4.21, y la especificación de una posible ligadura utilizando el segundo procedimiento.

[21] Para una descripción detallada de este patrón puede consultarse el Apéndice B de esta tesis

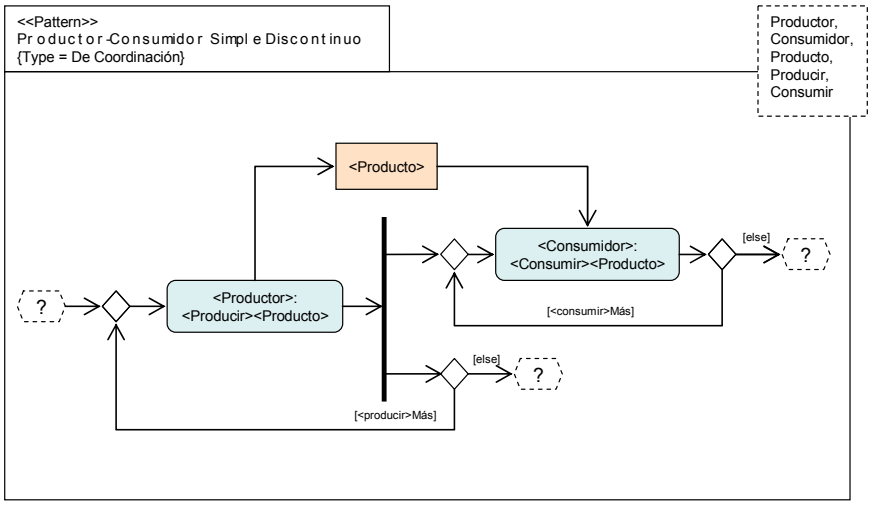


Fig. 4.21: Modelo correspondiente al patrón PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO

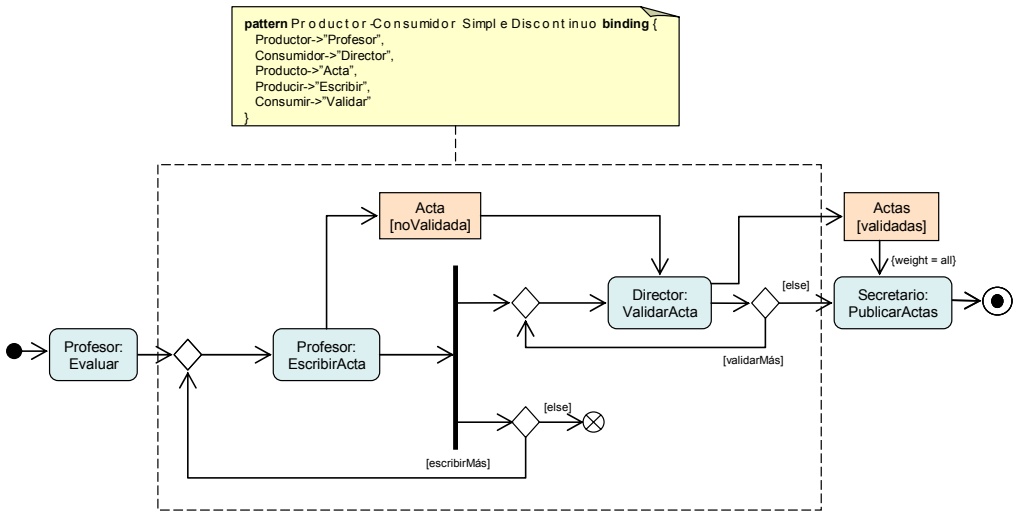


Fig. 4.22: Visualización de una ligadura del patrón PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO conectando una expresión de `binding` al rectángulo punteado que encierra la instancia

Como hemos podido comprobar anteriormente en la Figura 4.19, la subactividad `EmitirVotoGrupo` se define completamente a partir del patrón PETICIÓN-RESPUESTA MÚLTIPLE. Por ello, para aumentar la claridad del diagrama, también podemos conectar dicha etiqueta directamente con el símbolo de la subactividad (Fig. 4.23).

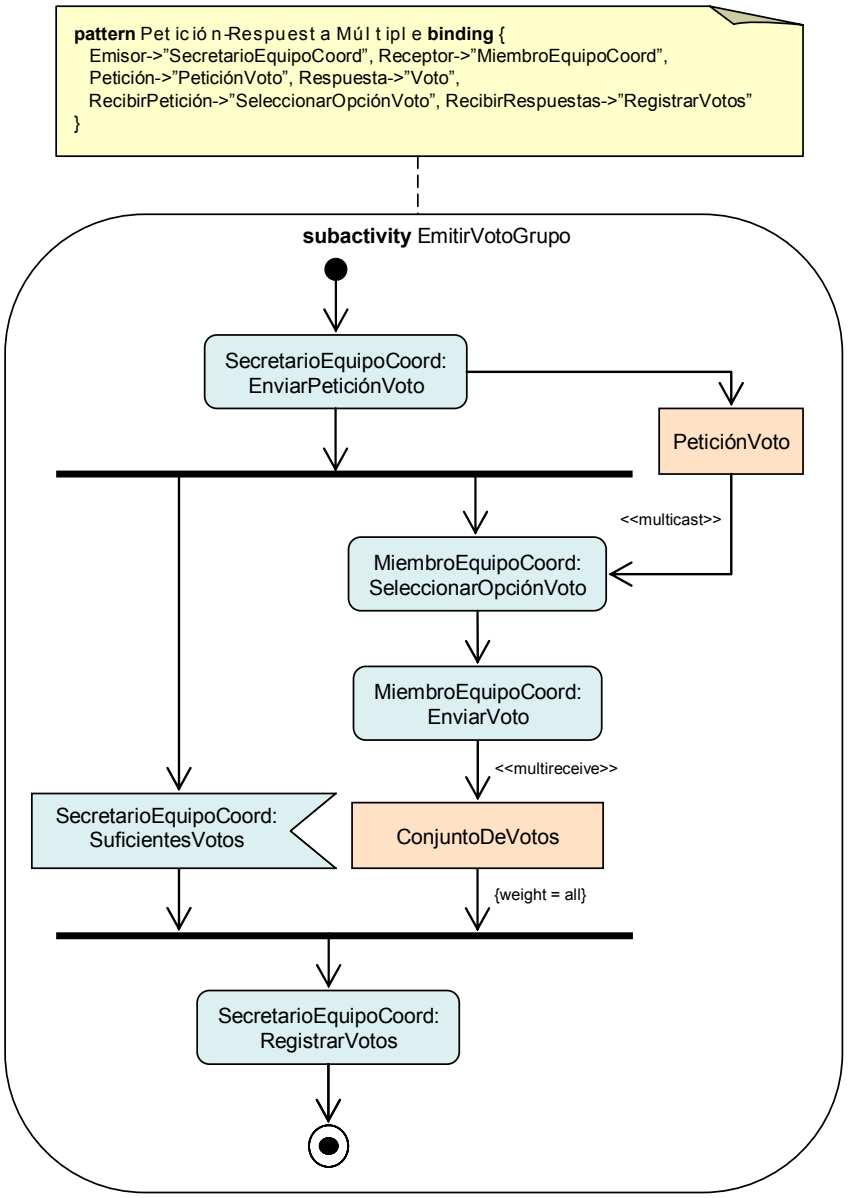


Fig. 4.23: Aplicación del patrón PETICIÓN-RESPUESTA MÚLTIPLE para el modelado de la subactividad `EmitirVotoGrupo` conectando una expresión de `binding` directamente con la subactividad

Hasta ahora, una ligadura siempre la hemos especificado visualizando el modelo de la instancia del patrón. Sin embargo, a menudo es muy útil dejar anunciada dicha ligadura por medio de etiquetas, sin que su instancia llegue a ser mostrada. Esto es posible hacerlo cuando el modelo resultante se define totalmente a partir del patrón, esto es, no incluye elementos extra que no están definidos por el patrón, pudiéndose conocer el modelo implícito sin necesidad de desplegarlo. Esto supone una ventaja muy interesante, ya que

permite simplificar los modelos, facilitando así su reutilización, comprensión, comunicación y mantenimiento.

Por ejemplo, la Figura 4.24 muestra parte de un modelo en el que la acción `EmitirVotoGrupo` queda completamente definida simplemente conectando la especificación de una ligadura concreta del patrón PETICIÓN-RESPUESTA MÚLTIPLE. En consecuencia, no es necesario desplegar dicha actividad para conocer su detalle y saber lo que sucede cuando ésta se ejecuta.

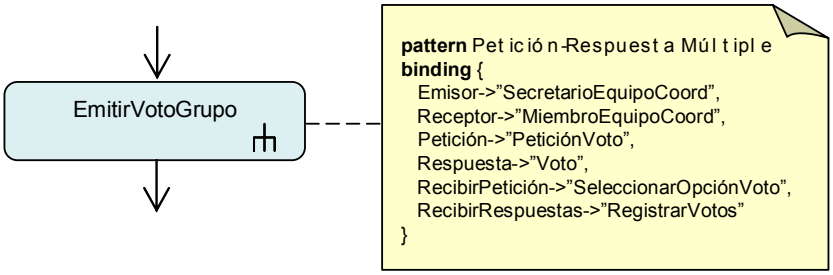


Fig. 4.24: Definición de la subactividad `EmitirVotoGrupo` a partir de la especificación de una ligadura del patrón PETICIÓN-RESPUESTA MÚLTIPLE

UML dispone del estereotipo estándar `<<bind>>`, el cual puede ser aplicado sobre una relación de realización para poder especificar, con las limitaciones que veremos a continuación, un tipo de ligadura parecida.

Una realización se define como una dependencia semántica entre dos conjuntos de elementos de modelado en la que uno establece una especificación (el proveedor) y el otro su implementación (el cliente), no entendiendo el término "implementación" en el sentido estricto de la palabra, sino más bien como una forma más refinada o elaborada del modelo proveedor.

El establecimiento de una relación `<<Bind>>` entre un paquete y una plantilla UML es equivalente a la copia exacta del contenido de esa plantilla en el paquete, reemplazando los parámetros de la plantilla con los argumentos especificados en la relación de ligadura (deben aparecer todos).

Por ejemplo, en la Figura 4.25 hemos definido el patrón *Composite* [Gamma et al., 1995] como una plantilla UML ligada al paquete `GestiónDibujo`, lo que sería equivalente al diagrama que aparece en la

Figura 4.26. La plantilla asociada al patrón crearía una copia idéntica en el paquete `GestiónDibujo`, sustituyendo sus parámetros por los elementos que aparecen en la ligadura, no admitiendo otras configuraciones posibles para este mismo patrón como, por ejemplo, la existencia de varios tipos de clases `Hoja` incluyendo múltiples operaciones, o la posibilidad de que cada objeto de la clase `Componente` tenga conocimiento de quién es su padre. Para cada una de estas configuraciones habría que crear una plantilla diferente.

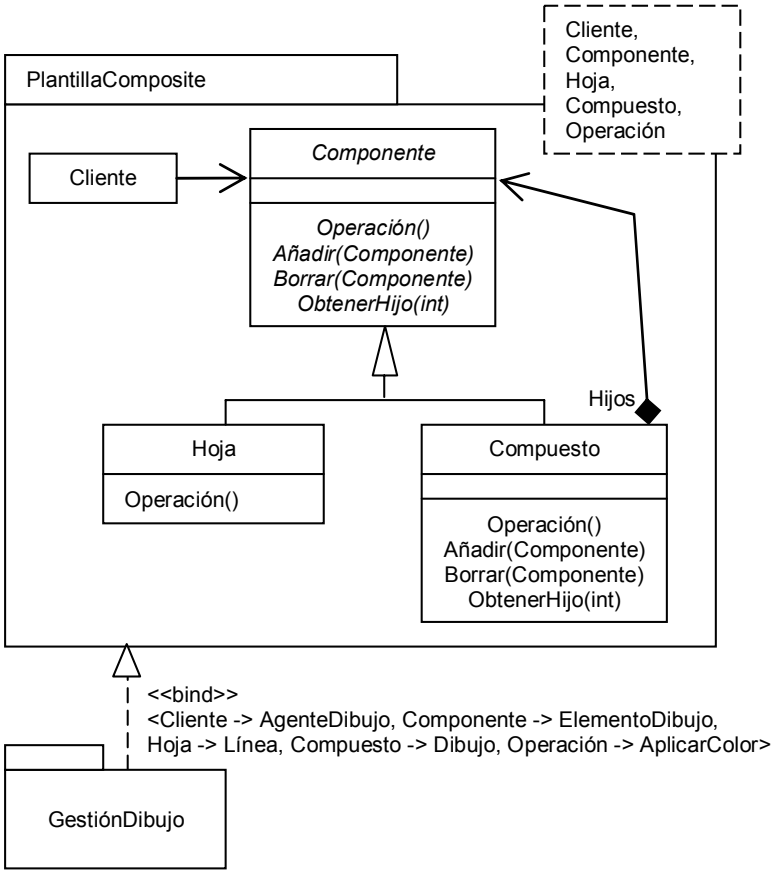


Fig. 4.25: Plantilla UML asociada al patrón de diseño `Composite` y una posible ligadura `<<bind>>` con el paquete `GestiónDibujo`

A diferencia con el estándar, el profile PMP (Pattern Modelling Profile) que presentamos en este capítulo permite la definición de plantillas que no tienen porqué dar lugar a copias exactas del contenido del patrón, aunque podría darse el caso, sino un modelo, a menudo incompleto, que puede ser diferente en función de los argumentos transferidos, pero siempre respetando

las restricciones que el patrón impone (invariante del patrón[22]) a los elementos que participan en sus instancias.

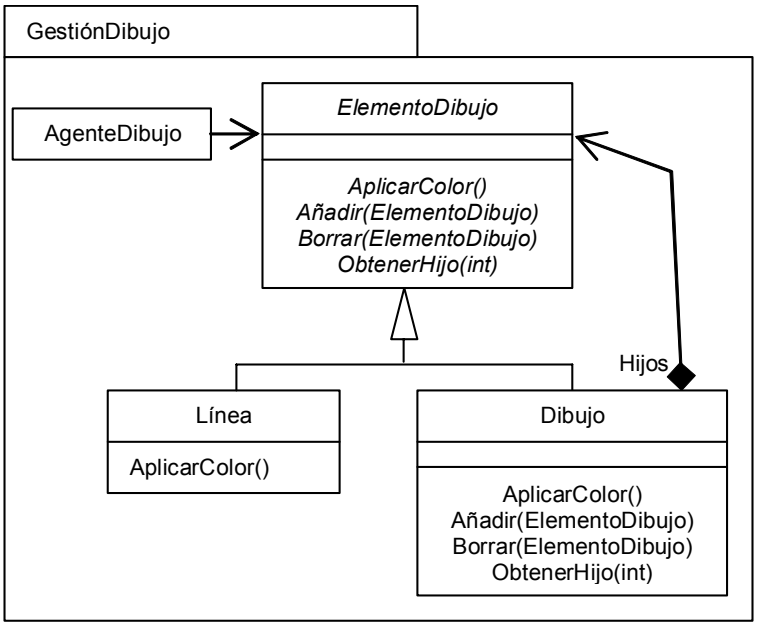


Fig. 4.26: Paquete `GestiónDibujo` que resulta de una ligadura `<<bind>>` con el patrón de diseño `Composite`

Aunque en esta tesis aportamos muchos otros aspectos que hacen que el tratamiento de los patrones sea más completo, preciso y eficiente que en el estándar UML, pensamos que la simple utilización de nuestro profile para la definición de estas plantillas dotaría al estándar de una mayor versatilidad para el modelado y aplicación de patrones. Por ejemplo, de acuerdo con nuestro profile, en la Figura 4.27 presentamos una redefinición de la plantilla que aparece en la Figura 4.25, así como una posible ligadura con el paquete `GestiónDibujo`.

[22] El invariante del patrón es: (1) Cada elemento en la instancia debe ser del mismo tipo que su semejante en el patrón, es decir, el <<PatternNamedElement>> que lo representa, (2) el número de elementos de la instancia relacionados con cada semejante en el patrón debe conformar con la restricción de multiplicidad de ligadura, si la hay, asociada a éste último, y (3) dados dos elementos de la instancia, entre ellos deben aparecer relaciones similares a las que existen entre sus elementos semejantes en el patrón

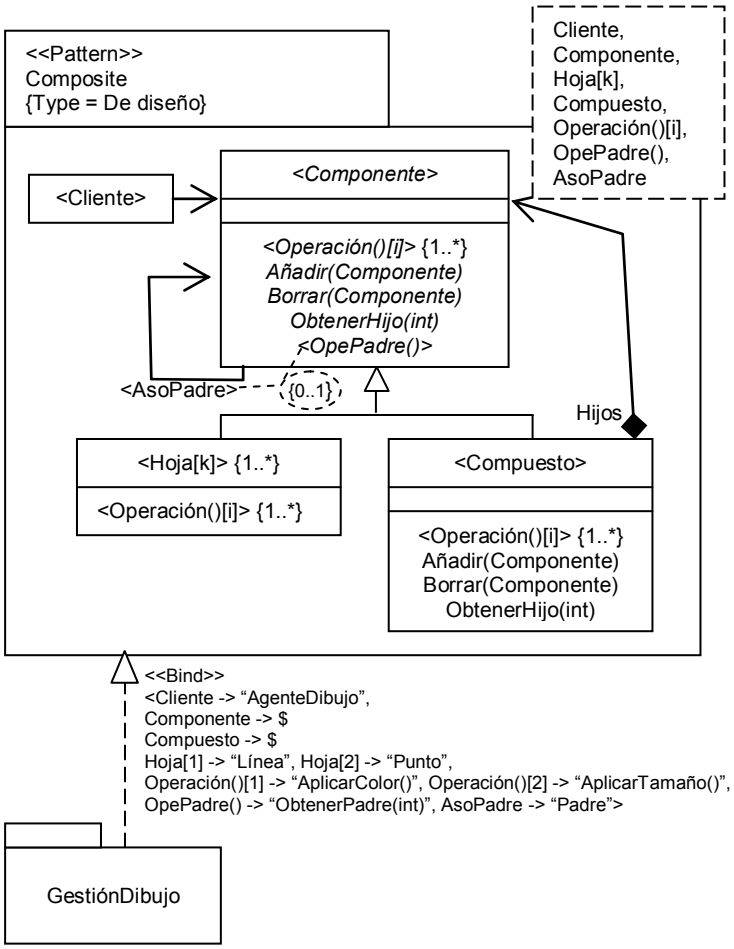


Fig. 4.27: Plantilla PMP correspondiente al patrón `Composite` y una posible ligadura `<<Bind>>` con el paquete `GestiónDibujo`

Podemos observar la existencia de restricciones de multiplicidad de ligadura (`<<MultiplicityBind>>`) que afectan a ciertos elementos/parámetros del patrón, lo que va a permitir la realización de ligaduras múltiples. En concreto, haciendo una interpretación relajada de la relación `<<bind>>`, y en línea con nuestro profile, podríamos realizar una ligadura múltiple de la clase `hoja` (multiplicidad `{1..*}`). En este caso, el parámetro indexado (`hoja[k]`) permite el enlace de distintos argumentos (`hoja[1] -> "Línea"` y `hoja[2] -> "Punto"`). Asimismo, también es posible ligar diferentes operaciones (`Operación[1] -> "AplicarColor"` y `Operación[2] -> "AplicarTamaño"`) que son comunes a las clases `Componente`, `Hoja` y `Compuesto`. Además, existen dos parámetros (`OpePadre()` y `AsoPadre`) que son optativos (véase la multiplicidad `{0..1}` asociada simultáneamente a ambos elementos), los cuales reciben sendos

argumentos ("ObtenerPadre(int)" y "Padre" respectivamente) para esta instancia concreta. Al ser opcionales, estos parámetros podrían simplemente no haber aparecido en el <<Bind>>, con lo que dichos elementos no participarían y el resultado sería diferente.

Podemos ver que hemos ligado el valor $ a los parámetros Componente y Compuesto, asumiendo de esta forma que los nombres de estos elementos en la instancia coinciden con los identificadores de los parámetros en el patrón.

El resultado de esta ligadura sería, por tanto, equivalente al diagrama que aparece en la figura 4.28 mediante la sustitución y copia del contenido del patrón en el paquete GestiónDibujo pero, en este caso, respetando las restricciones de multiplicidad/opcionalidad de algunos de sus parámetros. Una ligadura diferente podría haber dado lugar a una instancia con una configuración estructural muy distinta a la que aquí presentamos.

Aparte de los obstáculos que UML presenta para la definición de plantillas flexibles, el otro gran inconveniente que encontramos está en la utilización de <<Bind>> para la instanciación de estas plantillas. A diferencia de lo que ocurre con la relación <<PatternBind>>, la cual permite ligar individualmente cada uno de los elementos de una instancia con su elemento representante en el patrón, la relación <<Bind>> no permite realizar esto, lo que dificulta identificar los elementos que forman las instancias del patrón entre todos los elementos del modelo.

Según la especificación más reciente del estándar [OMG, 2007b], una dependencia de tipo <<Bind>> sólo puede establecerse entre un templateableElement y una plantilla. Sin embargo, durante el modelado conceptual de sistemas cooperativos a menudo necesitamos ligar una plantilla (patrón) a elementos que no son templateableElement, como por ejemplo los estados compuestos o las actividades, para definirlos sin tener que desplegar el modelo que referencian o contienen (v. Fig. 4.24).

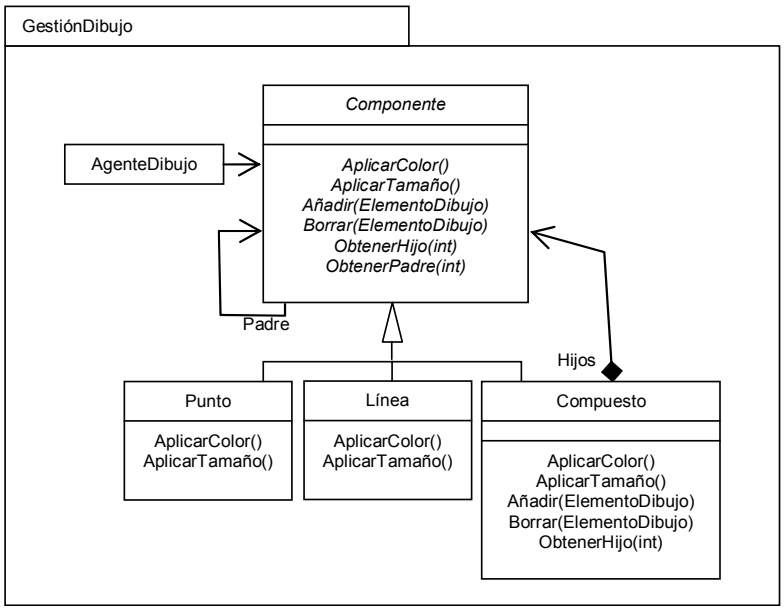


Fig. 4.28: Paquete `GestiónDibujo` según la ligadura establecida con el patrón de diseño `Composite` definido mediante el profile PMP

En ocasiones, el modelo que proporciona un patrón puede informar acerca del patrón o patrones que pueden usarse para definir alguno de sus elementos, sin mostrar ninguna ligadura en concreto. Esta característica ayuda al diseñador en la selección de aquellos patrones que podrían aplicarse posteriormente. Para la realización de este tipo de especificación usamos notas que incluyen expresiones con el siguiente formato en notación BNF:

'pattern ' <NombrePatrón> [('OR 'pattern ' <NombrePatrón>)]*

El modelo que aparece en la Figura 4.29, correspondiente a la definición del patrón REUNIÓN[23], presenta esta característica. En éste se muestran dos casos distintos de actividades que pueden ser construidas a partir de patrones. En un caso, hay una nota indicando que la actividad `VotarPunto` puede ser construida a partir de una ligadura concreta del patrón VOTACIÓN[24]. En el otro caso, la actividad `DebatirPunto` está conectada a una nota que incluye la especificación de dos patrones (DEBATE MODERADO y

[23] Véase Apéndice B de esta tesis para una descripción completa

[24] Para una descripción detallada de este patrón consúltese el Apéndice B

DEBATE NO MODERADO)[25] unidos por el operador lógico `OR`. Esta expresión permite indicar que la definición de la actividad `DebatirPunto` puede ser realizada ligando en tiempo de diseño cualquiera de estos patrones. Sin embargo, ¿tendría sentido que en el tiempo de diseño especifiquemos las dos ligaduras posibles, una por cada uno de los patrones, para que en tiempo de ejecución se realice la que más interese en cada momento?. En esta tesis afirmamos que no sólo es posible, sino que, además, es necesario poder especificarlo en determinadas ocasiones. Esto es lo que denominamos *ligadura dinámica* de patrones.

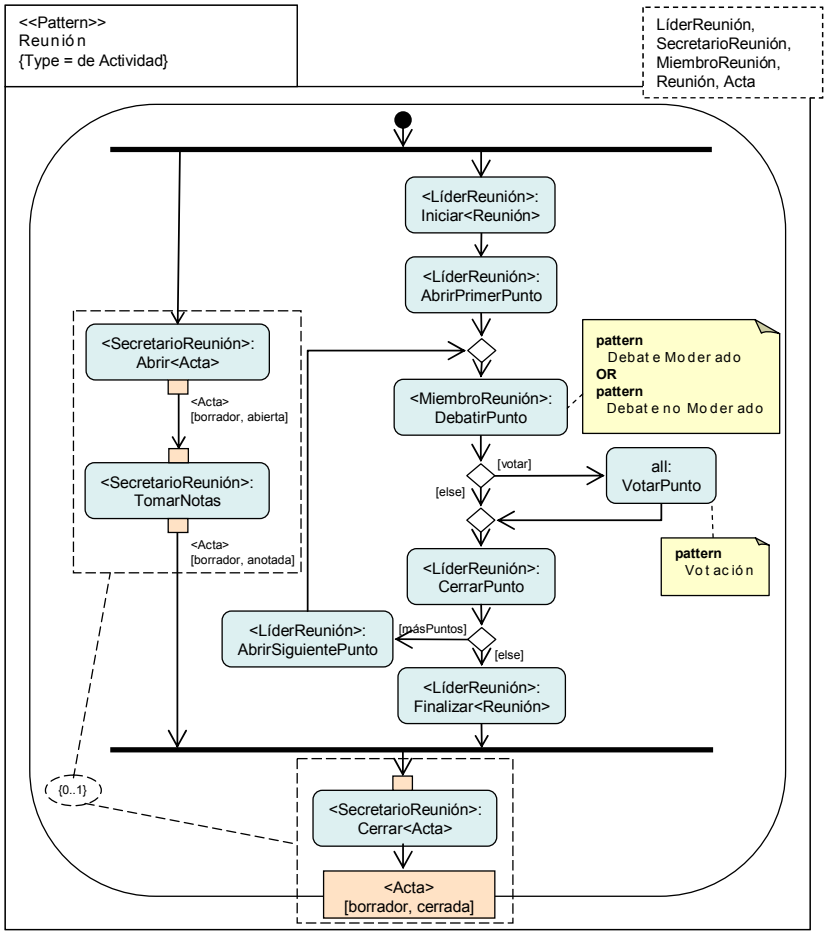


Fig. 4.29: Definición del patrón REUNIÓN incluyendo notas informativas sobre los posibles patrones a ligar para la definición de algunas de sus actividades

[25] Para una descripción detallada de estos patrones consúltese el Apéndice B

Una *ligadura dinámica* es aquella que se produce en tiempo de ejecución, lo que puede dar lugar a ligaduras diferentes en distintos momentos de la ejecución de las actividades. Las ligaduras pueden variar de un momento a otro porque cambia el patrón ligado o porque se realizan ligaduras diferentes de un mismo patrón.

Una de las características esenciales del trabajo cooperativo es su naturaleza dinámica. El grupo puede cambiar espontáneamente su estrategia de trabajo para abordar de manera diferente los problemas que tratan. El concepto de ligadura dinámica nos permite expresar esta característica cuando modelamos sistemas dinámicos como son los sistemas cooperativos.

Para especificar una ligadura dinámica nos basamos en el mismo formato que hemos utilizado desde un principio para la formulación de las expresiones de binding. Sin embargo, ahora intervienen las expresiones lógicas que, incluidas en el formato de descripción, hemos dejado de lado hasta este momento adrede.

La Figura 4.30 muestra un ejemplo donde la ligadura dinámica varía el patrón ligado. En este caso, una instancia concreta del patrón REUNIÓN define la actividad `DebatirPunto` mediante la ligadura dinámica de los patrones DEBATE MODERADO y DEBATE NO MODERADO. El grupo de trabajo puede cambiar en cualquier momento el protocolo de comunicación utilizado para el debate, ya sea dentro de un mismo ciclo (debate de un punto concreto) o entre ciclos distintos (debate de puntos diferentes).

La Figura 4.31 muestra otro ejemplo de ligadura dinámica, pero en este caso expresando diferentes ligaduras para un mismo patrón. En particular, presentamos la ligadura del patrón PETICIÓN-RESPUESTA SIMPLE[26] (Figura 4.32) para la definición de la actividad `ResolverDuda`. Como podemos observar, en la actividad intervienen tanto el rol de `Profesor` como el de `Alumno`, sin embargo, un mismo actor puede intervenir en unos momentos como `Emisor` (haciendo preguntas) y en otros como `Receptor` (respondiendo a preguntas). Esto se especifica mediante una expresión lógica `(Receptor ->`

[26] Una descripción detallada de este patrón puede consultarse en el Apéndice B de esta tesis

```
”Profesor” AND Emisor -> ”Alumno”) OR (Receptor -> ”Alumno” AND
Emisor -> ”Profesor”).
```

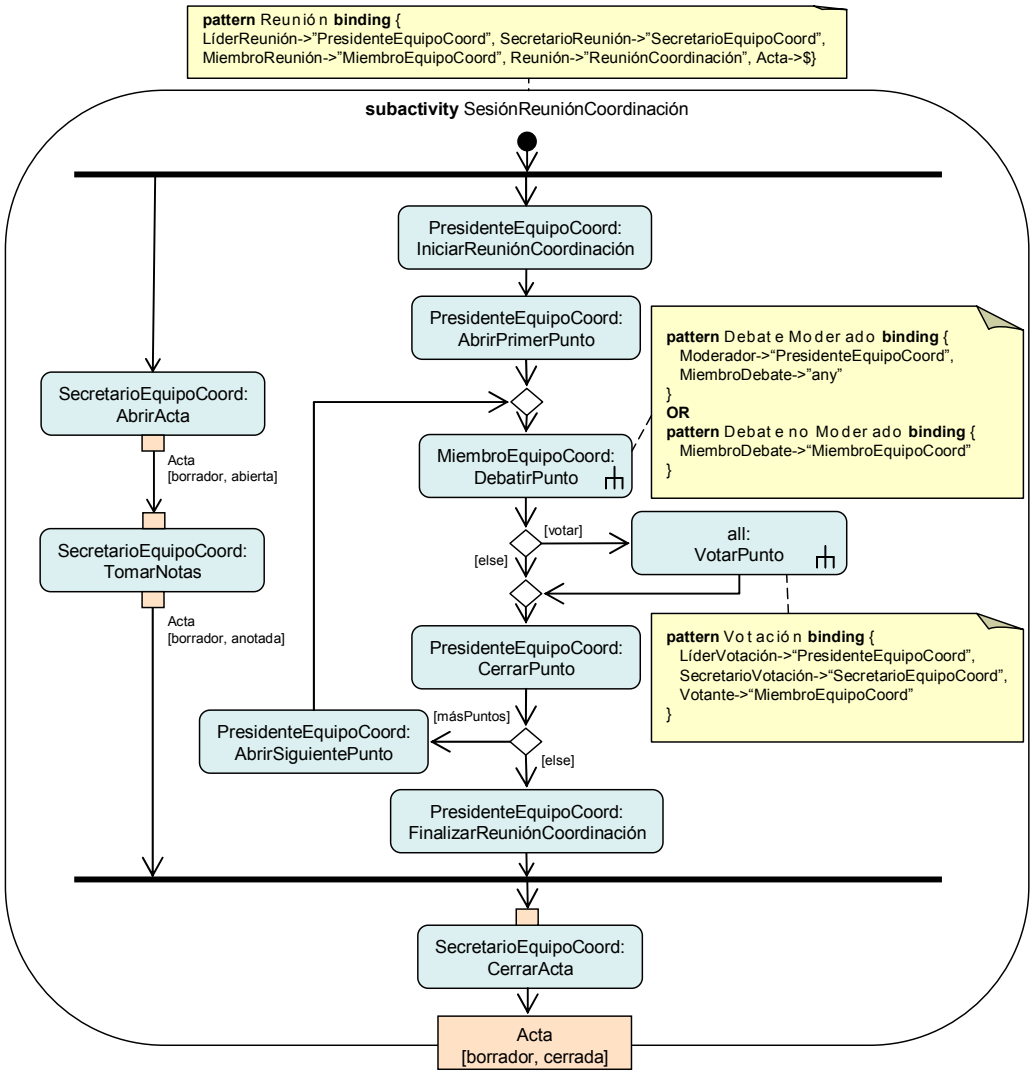


Fig. 4.30: Instancia del patrón REUNIÓN incluyendo una ligadura dinámica para la actividad `DebatirPunto`

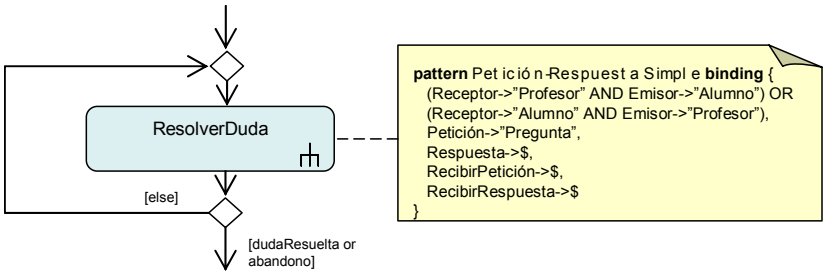


Fig. 4.31: Ligadura dinámica del Patrón PETICIÓN-RESPUESTA SIMPLE

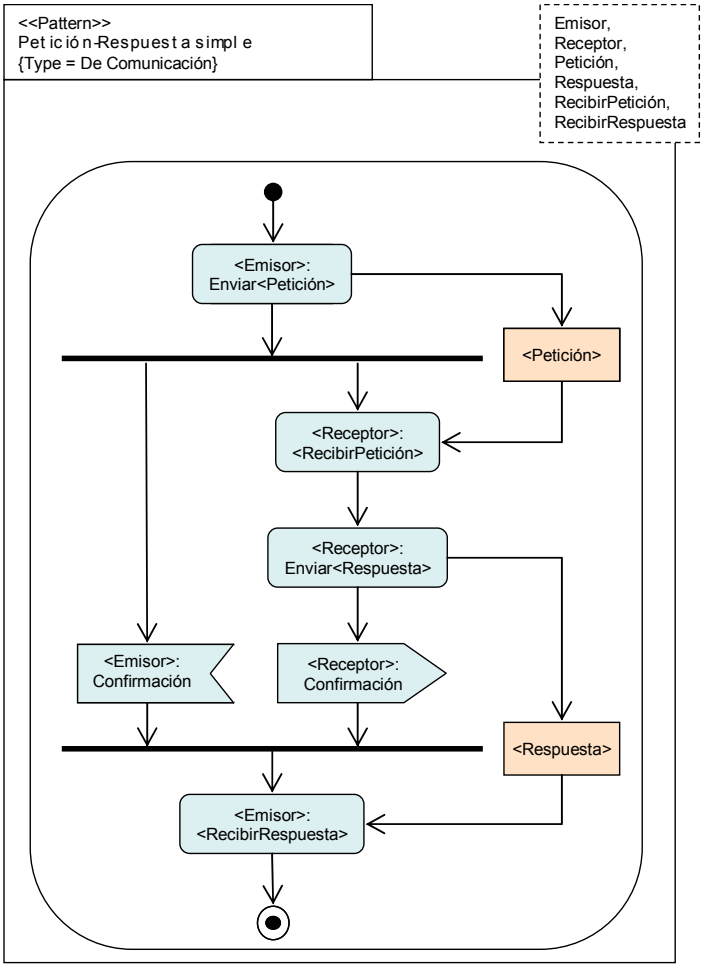


Fig. 4.32: Patrón PETICIÓN-RESPUESTA SIMPLE

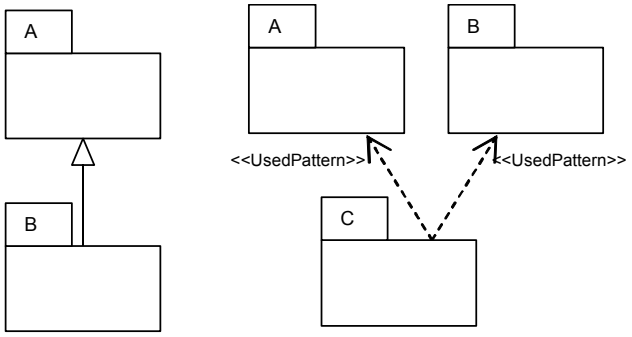
4.3.4.5.2. <<UsedPattern>>

Una de las relaciones que pueden establecerse entre patrones es la dependencia de uso o `<<UsedPattern>>`.

Una *dependencia de uso* indica que un patrón se modela en base a la utilización de otro u otros patrones. Esta relación permite representar la composición de patrones (v. Fig. 4.33).

4.3.4.5.3. Generalización

El otro tipo de relación que se puede dar entre patrones es la *generalización/especialización o herencia*. Ésta permite reflejar la relación existente entre dos patrones en el que uno es especialización de otro, es decir, extiende al otro (v. Fig. 4.33).



El patrón B especializa el patrón A

El patrón C usa los patrones A y B para su composición

Fig. 4.33: Relación de herencia y dependencia de uso

4.3.4.6. <<PatternCollection>>

Un paquete que contiene un conjunto de patrones útiles dentro de un dominio, y sus posibles relaciones, forma una colección de patrones. En la Figura 4.34, `PatronesDeOrganización` es una colección de patrones.

Según UML, también es posible mostrar el contenido del paquete usando una estructura de árbol (v. Fig. 4.35).

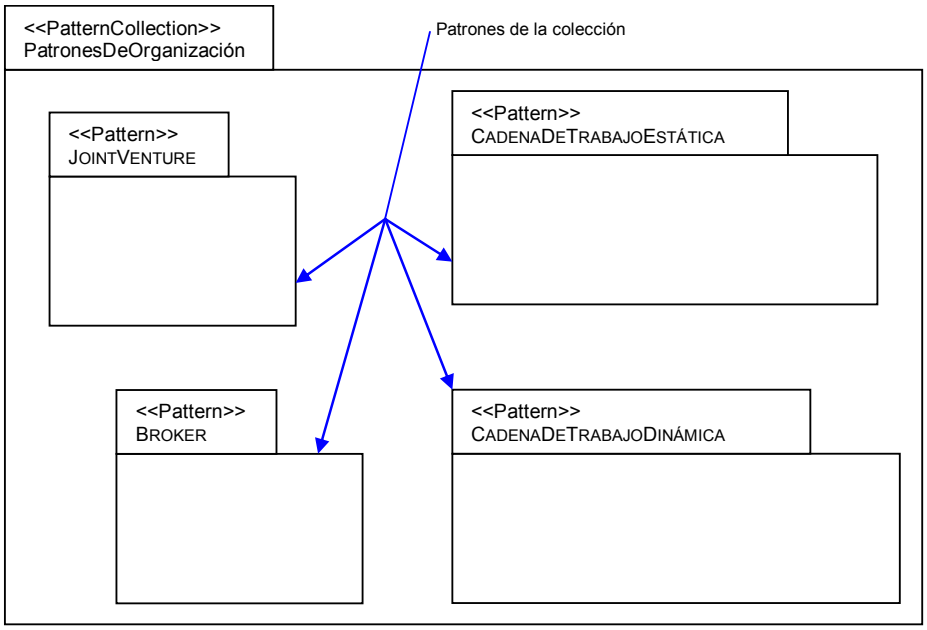


Fig. 4.34: Notación utilizada para modelar una colección de patrones

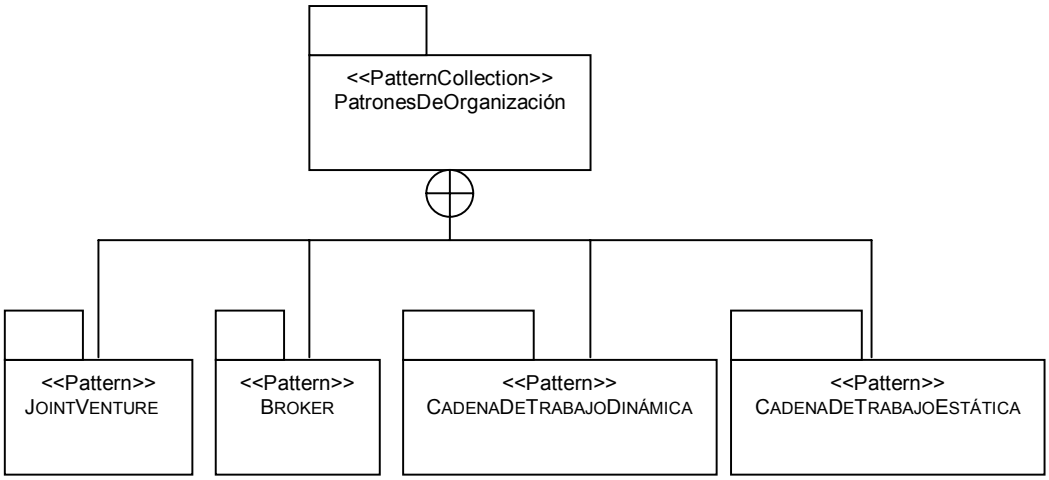


Fig. 4.35: Otra notación utilizada para modelar una colección de patrones

4.3.4.7. <<PatternView>>

La vista del sistema que contiene los patrones utilizados y sus relaciones se modela a través de un paquete con el estereotipo `<<PatternView>>`. Una aplicación muy interesante de este mecanismo consiste en agrupar los patrones en diferentes vistas, dependiendo de la fase de desarrollo en la que son aplicados, y relacionar los patrones entre las

capas vecinas mediante dependencias por refinamiento. De esta manera podemos hacer ingeniería inversa (subimos de nivel) o directa (bajamos de nivel), siguiendo la traza que conecta los patrones a través de los distintos niveles de modelado dentro del proceso de desarrollo.

Conforme a la notación UML, si no hay visible ningún diagrama utilizaremos un triángulo en la esquina superior derecha del rectángulo grande, en caso contrario, lo pondremos en la pestaña, a la derecha del nombre.

Utilizaremos el estereotipo estándar `<<trace>>` para especificar las conexiones entre patrones que representan el mismo problema en diferentes modelos (v. Fig. 4.36).

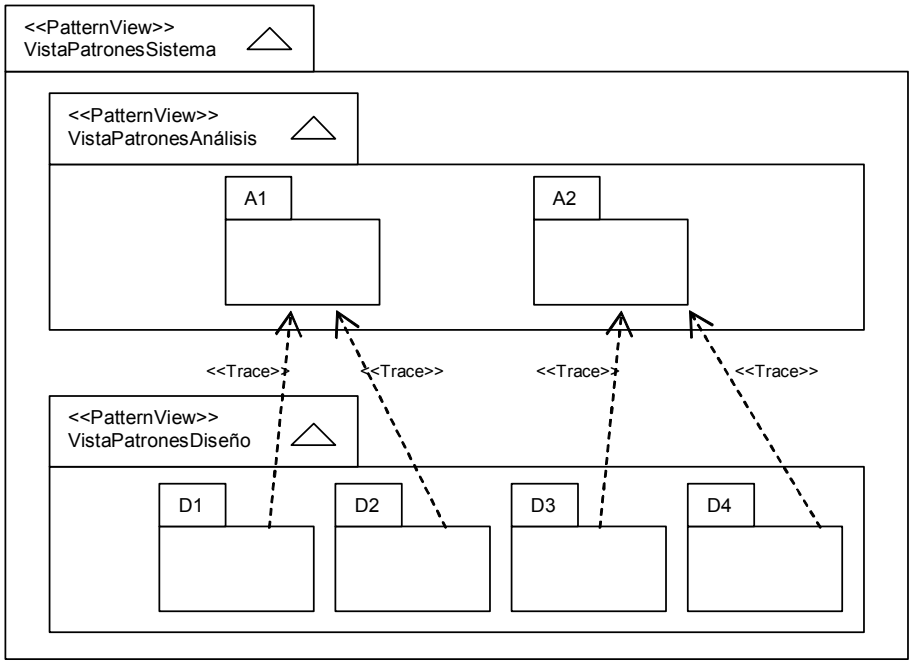


Fig. 4.36: Distribución de los patrones entre las diferentes vistas del sistema y sus relaciones de refinamiento

5. Conclusiones del capítulo

A menudo los patrones son utilizados para facilitar el modelado de software. En este caso, los patrones suelen proveer modelos que dan solución a determinados problemas de diseño o que describen situaciones comunes dentro de un dominio específico. Idealmente, estos modelos deberían poder

ser usados a modo de guías o bloques de construcción reutilizables durante el modelado. Además de agilizar la creación de los modelos, la identificación de los patrones en éstos mejoraría su comprensión, comunicación y mantenimiento, así como la documentación en general. Sin embargo, con demasiada frecuencia, la forma de expresar dichos modelos limita su capacidad de reutilización.

El modelo que define un patrón (vista interna) debería ser lo suficientemente genérico y flexible como para que pueda ser convenientemente reutilizado e instanciado. Una vez aplicado el patrón, su instancia debería reconocerse fácilmente dentro de los modelos generados (vista externa), identificando los elementos que la componen y el papel que juega cada uno de éstos dentro del patrón.

La búsqueda de una notación adecuada que facilite dicha representación ha sido un tema candente a lo largo de la historia de los patrones de software. Como no podía ser de otra forma, este tema también ha sido un tópico de especial interés a lo largo de las diferentes versiones del lenguaje estándar de modelado UML.

En este capítulo hemos revisado y analizado el tratamiento que sobre los patrones se ha venido realizando en UML, concluyendo que éste presenta los siguientes inconvenientes:

- Se centra en un solo tipo de patrones, en concreto, los patrones de diseño orientados a objetos. No se tiene en cuenta la existencia de otros tipos de patrones, por ejemplo, los utilizados en fases previas al diseño (patrones conceptuales o de análisis) cuyos modelos no tienen porqué estar formados meramente por estructuras de clases u objetos que interaccionan.

- La representación de patrones se basa exclusivamente en la utilización de colaboraciones parametrizadas, lo que limita enormemente los tipos de modelos que se pueden instanciar.

- Dicha representación es demasiado rígida, ya que está basada en el caso concreto, no permitiendo la utilización de parámetros opcionales o variables en número para posibilitar la ligadura de diferentes tipos de instancias cuyas estructuras son compatibles con el invariante patrón.

- Lo único que se puede parametrizar son los roles, no pudiendo usar otros elementos (atributos, métodos, estados, acciones, condiciones, etc.) como parámetros. Además, los roles dependen de clasificadores o asociaciones base predefinidos, lo que carece de sentido si tenemos en cuenta que es el patrón el que nos debería ayudar a definir estos elementos y no al revés.

Alternativamente muchos autores han propuesto notaciones para el modelado de patrones. Después de recorrer los principales trabajos existentes relacionados con este asunto, consideramos que cada una de éstas propuestas adolece, en mayor o menor medida, de alguno de los siguientes inconvenientes:

- Emplea una notación generalmente compleja por su formalidad.
- Basada en algún lenguaje de modelado que es poco conocido.
- Limitada al modelado de patrones de diseño orientado a objetos.
- Centrada exclusivamente en la vista interna o externa del patrón.
- Modifica directamente el metamodelo de UML.

Para solventar estas dificultades, en este capítulo hemos especificado *PMP (Pattern Modelling Profile)*, un perfil UML para el modelado de patrones de software en general. A partir de un reducido número de elementos, esta extensión nos permite representar de manera simple, intuitiva y fácil de comprender, tanto la vista interna como externa de un patrón. Los patrones son definidos como diagramas genéricos, flexibles y totalmente parametrizables que pueden reutilizarse durante la construcción de los modelos que se crean en las distintas fases de desarrollo, incluyendo, por supuesto, los modelos conceptuales que construimos durante el estudio de los sistemas cooperativos en el marco de la metodología AMENITIES. De este modo, los patrones son vistos como plantillas flexibles que representan familias de modelos semejantes (instancias), los cuales pueden usarse como guía para la creación y/o descripción de modelos, o partes de éstos, por medio de la correspondencia (ligadura) de los elementos del patrón con los elementos que forman sus instancias. De acuerdo con este perfil, un modelo podrá ser instancia de un patrón siempre y cuando cumpla con las

restricciones que la propia definición del patrón impone en cuanto al tipo, relaciones y número de elementos.

A través de diversos ejemplos hemos ilustrado la capacidad expresiva que tiene este perfil para el modelado y aplicación de patrones en general, y particularmente, de aquellos patrones que son útiles durante el modelado conceptual de sistemas cooperativos con la metodología AMENITIES. En el siguiente capítulo introducimos la creación de un catálogo de patrones aplicables en este ámbito específico, el cual saca partido de este perfil para modelar los patrones que lo componen y aplicarlos durante la construcción del Modelo Cooperativo de un sistema según esta metodología.

Capítulo V

Construcción del Modelo Cooperativo de AMENITIES en base a Patrones

Contenido

<<No basta con adquirir sabiduría, es preciso además saber usarla>>

—Cicerón (106 AC - 43 AC)

Escritor, orador y político romano

1. Introducción

Para modelar un sistema cooperativo hay que tener en cuenta un amplio conjunto de conceptos interrelacionados. El establecimiento de un marco conceptual que refleje y relacione los distintos conceptos que aparecen en este dominio es fundamental. En concreto, los modelos que usamos están basados en los marcos conceptuales de trabajo propuestos en Garrido [2003] y Gutiérrez et al. [2006b][1], los cuales, a su vez, integran elementos

[1] Este marco conceptual forma parte de los trabajos que hemos realizado dentro del proyecto ADACO (`http://adaco.dynalias.org/`). En este proyecto se desean proporcionar soluciones a los problemas que surgen en el desarrollo de sistemas colaborativos basados en Web.

procedentes de la Teoría de la Actividad [Nardi, 1995] y de ontologías de Modelos de Tareas [Van Welie et al., 1998] entre otros.

Sin embargo, aunque consideramos que estos marcos conceptuales son un punto de partida ineludible, los patrones conceptuales son también una herramienta importante durante el modelado. Éstos representan modelos reutilizables que facilitan la construcción del modelo conceptual de un sistema, que en el caso de AMENITIES, como ya sabemos, es conocido como el *Modelo Cooperativo* del sistema (v. Cap. II).

Los patrones conceptuales modelan abstracciones comunes en el dominio del problema, enriqueciendo el vocabulario y extendiendo nuestro marco conceptual de trabajo. Estos patrones guían la percepción que se tiene del mundo, actuando como "*convenciones de pensamiento*" [Hay, 1996] que ayudan a comprender y describir el dominio. Cuando modelamos pensamos en términos de patrones.

Como vimos en el Capítulo III, el modelado conceptual de un sistema se puede beneficiar enormemente con la aplicación de patrones. Sin embargo, para ello es primordial:

- Utilizar un formato de descripción uniforme que facilite el estudio, identificación, comparación y uso de los distintos patrones dentro de un dominio concreto.

- Disponer de una colección de patrones aplicables en dicho dominio.

- Crear un catálogo que organice y relacione los patrones de la colección, agilizando las búsquedas dentro de éste.

- Determinar un método que, teniendo en cuenta los tres elementos anteriores, marque las pautas para la selección y aplicación efectiva de los patrones.

En este capítulo introducimos todos estos elementos con el propósito de facilitar la aplicación efectiva de patrones durante el modelado conceptual de sistemas cooperativos en el marco de la metodología AMENTIES.

Más tarde ponemos de relieve los principales requisitos que debería satisfacer una herramienta para el modelado basado en patrones de acuerdo con nuestra propuesta.

A continuación, con ánimo de allanar el camino que va desde el dominio del problema al dominio de la solución, hacemos una primera aproximación a las posibles conexiones que pueden establecerse entre los patrones conceptuales que hemos propuesto y algunos patrones de diseño que aparecen en la literatura.

Terminamos el capítulo emitiendo nuestras conclusiones.

2. Hacia un catálogo de patrones para el modelado conceptual de sistemas cooperativos con AMENITIES

2.1. Una plantilla para la descripción uniforme de patrones

Aunque el objetivo final de los patrones que vamos a describir es presentar un modelo que de solución a un problema concreto de modelado, la descripción de éstos no puede basarse únicamente en dicho modelo. Necesitamos completar dicha descripción con información que nos ayude a entenderlos, decidir sobre su uso, compararlos y aplicarlos.

Casi todos los autores utilizan formatos de descripción basados en el llamado "formato canónico o Alejandrino" (v. Cap. III, sec. 2.4). No obstante, la mayoría optamos por una versión simplificada y adaptada al dominio de aplicación específico.

Así, pues, describimos los patrones mediante una plantilla uniforme que estructura su descripción en una serie de secciones que facilitan su estudio y aplicación. En la Tabla 5.1 explicamos las secciones concretas que forman parte de esta plantilla.

Tabla 5.1: Formato para la descripción uniforme de los patrones del catálogo

Sección	Descripción
Nombre/Alias	Debe ser significativo y reflejar la esencia del patrón en pocas palabras. Forma parte del vocabulario que facilita la comunicación de las abstracciones. Si es conocido también por otro nombre deberemos especificar su alias.

Clasificación	Vista a la que pertenece, fase de especificación y tipo de patrón según la clasificación establecida en el catálogo.
Intención	¿Cuál es el escenario que pretendemos modelar?.
Contexto	¿En qué situaciones se puede aplicar?, ¿cómo reconocer dichos escenarios?
Solución	Modelo genérico que provee el patrón, definido en base al perfil *PMP* (*Pattern Modelling Profile*), y que facilita la generación de la instancia que modela el escenario concreto.
Explicación	Descripción de la solución que se propone.
Ejemplo	Aplicación del patrón a un caso concreto.
Patrones relacionados	Otros patrones que forman parte del mismo catálogo y con los cuales se relaciona.

La utilización de un lenguaje sencillo, preciso y homogéneo, contribuye también al aumento de la claridad y comprensión de las descripciones. Para favorecer esta circunstancia, disponemos de un glosario de términos (Apéndice C) que define, entre otros, los conceptos que forman parte de nuestro marco de trabajo. Asimismo, la notación que empleamos para definir los modelos puede ser consultada en el Capítulo IV (Perfil para el modelado de patrones), Apéndice A (Notación COMO-UML) y Apéndice D (Diagramas de actividad con UML 2). También se han adoptado una serie de convenciones tipográficas que facilitan la lectura (v. Cap. I).

De acuerdo con esta plantilla, en el Apéndice B describimos algunos de los patrones que componen nuestro catálogo.

2.2. Estructura del catálogo

2.2.1. Tipos de patrones

Como vimos en el Capítulo III (v. sec. 2.3), existen muchas formas posibles de clasificar los patrones. Los criterios que principalmente se usan son el nivel de abstracción, el dominio de aplicación y el tipo de problema tratado.

El objetivo primario de los patrones que componen este catálogo es facilitar el modelado conceptual de un sistema cooperativo con AMENITIES [Isla et al., 2006a, 2007], o lo que es lo mismo, agilizar la construcción de su Modelo Cooperativo (v. Cap. II, sec. 3.4), núcleo central de la metodología. Por ello, vamos a denominar a los patrones que forman parte de nuestro catálogo *patrones cooperativos.*

El Modelo Cooperativo es un modelo conceptual formado por un conjunto de modelos interrelacionados (de comportamiento y de tareas) que permiten describir, usando un alto nivel de abstracción, la organización y manera de proceder de los grupos de trabajo de un sistema, independientemente de su implementación. Por consiguiente, los patrones que van a formar parte de este catálogo son de alto nivel y su dominio de aplicación es el modelado conceptual de sistemas cooperativos con la metodología AMENITIES.

En este sentido, podemos definir un *patrón cooperativo* como aquel que especifica un modelo conceptual reutilizable durante la construcción del Modelo Cooperativo de un sistema con la metodología AMENITIES.

El Modelo Cooperativo se describe a partir de cuatro vistas complementarias (*organizacional, cognitiva, de interacción* y *de información*), las cuales modelan diferentes perspectivas del sistema, a la par que ordenan el proceso de construcción. Asimismo, dentro de cada una de estas vistas se abordan diferentes tipos de problemas.

De cara a facilitar la selección durante el proceso de construcción del Modelo Cooperativo, pensamos que una buena forma de organizar los patrones del catálogo consiste en clasificarlos según la vista en la que participan, fase de especificación del Modelo Cooperativo en la que se suelen aplicar y aspecto concreto que abordan, donde este último define concretamente el tipo del patrón. La Tabla 5.2 muestra los nombres de los patrones del catálogo organizados en base a estos criterios.

Como hemos mencionado en la introducción a este capítulo, no todos los patrones incluidos en este catálogo los hemos especificado con ayuda de la plantilla de descripción definida en la sección anterior. Los patrones así especificados los identificamos mediante un subíndice al final de su nombre, el cual señala la sección precisa del Apéndice B donde aparece su

descripción. Por supuesto, todos los patrones que aplicamos en el siguiente capítulo, dedicado a dos casos de estudio concretos, están convenientemente detallados en dicho apéndice.

Tabla 5.2: Clasificación de los patrones del catálogo

Vista	Fase de especificación	Tipo de patrón	Ejemplos
Organizacional	Organización	De organización	* JOINT VENTURE(2.1.1) * CADENA DE TRABAJO(2.1.2) * BROKER * ESTRUCTURA EN 5 * PIRÁMIDE * MANDO-SUBMANDO
	Tareas	De equipo	* CIRCULO DE CALIDAD(2.2.1) * EQUIPO DE DIRECCIÓN
Cognitiva	Roles	De rol	* COORDINADOR(2.3.1) * SECRETARIO
	Tareas	De actividad	* PROCESO DE REUNIÓN(2.4.1) * REUNIÓN(2.4.2) * VOTACIÓN(2.4.3) * NEGOCIACIÓN NO MODERADA(2.4.4) * CONVOCATORIA DE REUNIÓN * LLAMAMIENTO PARA PROPUESTAS * TORMENTA DE IDEAS * CREAR PLAN DE TRABAJO * ENRUTAR FORMULARIO
	Tareas, Roles	De coordinación	* PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO(2.5.1) * PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO(2.5.2) * SALVAVIDAS(2.5.3) * PRODUCTOR-CONSUMIDOR MÚLTIPLE DISCONTINUO * PRODUCTOR-CONSUMIDOR MÚLTIPLE CONTINUO * TURNO DE PALABRA

Interacción	Tareas, Protocolos de Interacción[2]	De comunicación	* DEBATE MODERADO(2.6.1) * DEBATE NO MODERADO(2.6.2) * PETICIÓN-RESPUESTA SIMPLE(2.6.3) * PETICIÓN-RESPUESTA MÚLTIPLE(2.6.4) * EXPOSICIÓN(2.6.5) * MENSAJES ENCOLADOS * PUBLICACIÓN-SUSCRIPCIÓN
Información	Modelo Conceptual de Datos	De estructura	* ACTA DE REUNIÓN(2.7.1) * CALENDARIO DE EVENTOS * PLAN DE TRABAJO
	Tareas, Roles	De acceso	* AUTORIZADO(2.8.1) * PRIMERO EN LLEGAR-PRIMERO EN SERVIRSE * ORDEN DE PRIORIDAD * TURNO DE ACCESO

2.2.1.1. Vista organizacional

Esta vista incluye la especificación de dos visiones distintas de los grupos de trabajo.

1) Especificación desde el punto de vista de la *organización* formal, permanente y preestablecida de los grupos, mediante la identificación de los roles que intervienen, así como las restricciones y capacidades que afectan a la evolución del comportamiento desde el punto de vista de los cambios de rol de sus miembros (*Patrones de organización*).

2) Especificación desde la perspectiva de los *equipos de trabajo* asociados a tareas cooperativas, o actividades dentro de éstas, donde sus miembros se deciden dinámicamente a partir de responsabilidades concretas. Esta especificación es más relajada (*Patrones de equipo*).

[2] Como podrá comprobarse más adelante, actualmente esta fase se encuentra integrada con la de especificación de tareas.

2.2.1.1.1. Patrones de organización

Diferentes estudios realizados sobre organizaciones [Mintzberg, 1992; Morabito et al., 1999] han propuesto estructuras organizativas que, por su eficacia, a menudo rigen el contexto organizativo de un sistema. Este es el caso de estilos organizativos como los de ESTRUCTURA EN 5, JOINT VENTURE, PIRÁMIDE, ESTRUCTURA VERTICAL, etc., los cuales dotan a toda la organización de una estructura que facilita la distribución de sus integrantes (unidades organizativas o individuos) para conseguir objetivos globales. También existen otras estructuras sociales, a priori de grano más fino que las anteriores, que suceden a menudo en el seno de las propias organizaciones. En la mayoría de los casos, éstas describen cómo se organizan un conjunto de actores que persiguen un objetivo particular. De este tipo son las estructuras sociales de BROKER, MONITOR, EMBAJADA, MEDIADOR, etc., utilizadas, por ejemplo, para el diseño de arquitecturas de sistemas basados en agentes [Kolp et al., 2002, 2003; Fuxman, 2001].

Es importante señalar que muchas de estas estructuras organizativas se dan en diferentes contextos, tanto a nivel de macro-organizaciones (p. ej. empresas, alianzas empresariales, etc.) como a nivel de micro-organizaciones (p. ej. grupos de trabajo en un aula, departamentos, etc.). Precisamente, esta capacidad de reutilización hace que dichas estructuras organizativas sean tan interesantes durante el modelado de la vista organizacional de un sistema.

Podemos definir un *patrón de organización* como aquel que especifica una estructura organizativa común.

En el marco de esta tesis, hemos publicado numerosos trabajos [Gutiérrez et al., 2003; Isla et al., 2004a, 2004b, 2005b, 2006b, 2006c] relacionados con la descripción, modelado y aplicación de patrones de organización bajo la metodología AMENITIES, facilitando así el modelado de la vista organizacional de los sistemas. La capacidad expresiva de la notación que proponemos permite modelar tanto la estructura de la organización (y suborganizaciones) como su comportamiento, en base a los roles que pueden asumir sus miembros en cada instante, dependiendo de sus propias capacidades o las leyes que regulan la organización.

Los patrones de organización también especifican las responsabilidades de cada rol u organización en el contexto del patrón. Esto lo hacemos a través de expresiones con el siguiente formato:

'role/organization responsibilities ' ((<NombreRol> | <NombreOrganización>) ':' <responsabilidad> [(',' <responsabilidad>)] ';')**

Esta información va a ser la base para descubrir las tareas concretas a asignar a cada rol durante la fase posterior de especificación de roles. Cuando una responsabilidad sea compartida por varios roles implicará que la tarea diseñada para alcanzarlo tendrá que ser especificada como cooperativa y definida en cada uno de los roles participantes.

En la Tabla 5.3 describimos la intención de los distintos patrones de organización que, a modo de ejemplo, exponemos en nuestro catálogo. Para una descripción completa de los patrones JOINT VENTURE(2.1.1) y CADENA DE TRABAJO(2.1.2) puede consultarse el Apéndice B.

Tabla 5.3: Intención de cada patrón de organización del catálogo

Nombre	Intención
JOINT VENTURE(2.1.1)	Modelar una estructura organizativa en la que un grupo de socios, cada uno especializado en la realización de una tarea concreta, unen sus capacidades y recursos para alcanzar objetivos más ambiciosos. Así obtienen una serie de ventajas colectivamente (inversión parcial, costes de mantenimiento más bajos, mayores beneficios, recursos compartidos, etc.).
CADENA DE TRABAJO(2.1.2)	Modelar una estructura organizativa en la que varios actores colaboran para alcanzar una meta común en varias fases, normalmente la realización de un producto o servicio (p. ej. la cadena de venta o de montaje de un artículo), cada una con un objetivo concreto. Los actores se distribuyen entre las distintas etapas que componen la cadena. Para lograr el objetivo local de una etapa, antes se ha tenido que alcanzar el objetivo local de la etapa anterior, si ésta existe, y así sucesivamente.
BROKER	Modelar una estructura organizativa en la que un actor (Broker) intermedia entre un actor que provee unos determinados servicios (Proveedor) y otro que los consume (Consumidor). El broker dispondrá de acceso directo a los servicios que ofrece el proveedor y se encargará de satisfacer las demandas de los

	consumidores.
ESTRUCTURA EN 5	Modelar una estructura organizativa compuesta de cinco sub-organizaciones [Mintzberg, 1992]: Núcleo Operacional (encargada de las tareas directamente relacionadas con la producción), Punta Estratégica (responsable de la toma decisiones ejecutivas y de la definición de la estrategia global), Línea Media (facultada para supervisar y coordinar las actividades del núcleo operacional), Tecno-estructura (comprometida con que el trabajo de los demás sea más efectivo) y Soporte (encargada de proporcionar servicios especializados en cualquier nivel de la jerarquía).
PIRÁMIDE	Modelar una estructura organizativa en la que los actores que están en los niveles más bajos son supervisados directamente por aquellos que forman parte del nivel inmediatamente superior y así sucesivamente. Los jefes o supervisores que forman los niveles intermedios hacen que se cumplan las decisiones que se toman en el nivel más alto de la jerarquía. Ellos pueden coordinar y tomar decisiones propias en su ámbito local, siempre con el objetivo de que se cumplan las decisiones que proceden de niveles superiores.
MANDO-SUBMANDO	Modelar una estructura organizativa en la que existe un actor que tiene cierta autoridad o poder de mando en la organización (Mando, Director, Jefe, etc.) y otro que sirve inmediatamente a las órdenes del mando o le sustituye en sus funciones (Submando, Subdirector, Subjefe, etc.)

2.2.1.1.2. Patrones de equipo

Podemos definir un *patrón de equipo* como aquel que especifica las responsabilidades concretas (roles) que típicamente asumen un conjunto de actores (miembros del equipo) que colaboran para llevar a cabo actividades específicas.

Según su finalidad podemos hablar de equipos de producción (generan un producto o prestan un servicio concreto), de solución de problemas (realizan tareas de resolución de problemas o dan soporte a los mismos), de resolución de conflictos (afrontan situaciones de conflicto entre diferentes partes de la organización o de ésta con el exterior), etc.

Los patrones de equipo facilitan la representación de equipos durante la etapa de especificación de tareas con AMENITIES. La especificación de un

equipo de trabajo consiste, básicamente, en una redefinición de los roles que desempeñan los actores "oficialmente" dentro de la organización origen, redistribuyendo sus miembros en función de los requisitos concretos de trabajo.

En la Tabla 5.4 relatamos la intención de los patrones de equipo CÍRCULO DE CALIDAD(2.2.1) y EQUIPO DE DIRECCIÓN del catálogo. El primero de ellos, un tipo de equipo muy conocido en el mundo de la organización del trabajo, lo describimos detalladamente en el Apéndice B.

Tabla 5.4: Intención de cada patrón de equipo del catálogo

Nombre	Intención
CIRCULO DE CALIDAD(2.2.1)	Especificar un equipo cuyos miembros colaboran para analizar los problemas propios de su actividad, elaborar soluciones y presentar dichas mejoras normalmente a la dirección. Abarcan áreas tales como la mejora de los procesos de producción, mejora de la seguridad, etc. Suele haber un miembro que actúa como coordinador, encargado de coordinar y dirigir las reuniones.
EQUIPO DE DIRECCIÓN	Especificar un equipo en el que sus miembros colaboran para dirigir un proyecto colectivo. Suele haber un líder, que tiene la máxima autoridad dentro del proyecto, y uno o varios miembros responsables de aspectos específicos dentro de éste. Normalmente, uno de estos miembros sustituye al líder cuando éste no se encuentra disponible.

2.2.1.2. Vista cognitiva

Esta vista representa el conocimiento que cada miembro del grupo tiene o adquiere en relación con las actividades que desarrolla dentro del contexto organizacional.

Incluye la especificación de los roles y la especificación de las tareas. La especificación de roles conecta la especificación de la organización con las tareas individuales/colaborativas a realizar por sus integrantes. Para cada tarea se pueden identificar las tareas por las cuales puede ser interrumpida, si es o no cooperativa, el número de actores que intervienen en caso de ser cooperativa, los eventos que la habilitan y las capacidades/leyes necesarias

para poder desempeñarla. La especificación de tareas describe las tareas y las actividades que las componen por medio de diagramas de actividad.

2.2.1.2.1. Patrones de rol

Definimos un *patrón de rol* como aquel que especifica las responsabilidades que son comunes a un determinado tipo de rol.

En la Tabla 5.5 exponemos la intención de los distintos patrones de rol que hemos añadido al catálogo. Dentro del Apéndice B aparece descrito en detalle el patrón COORDINADOR(2.3.1).

Tabla 5.5: Intención de cada patrón de rol del catálogo

Nombre	Intención
COORDINADOR(2.3.1)	Especificar el diagrama de rol que especifica las tareas comunes de un rol de Coordinador. Entre las tareas típicas de este tipo de rol se encuentran la coordinación de un grupo de trabajo, la realización de reuniones con el grupo, el reparto de tareas entre sus miembros y el seguimiento de las distintas tareas.
SECRETARIO	Especificar el diagrama de rol que especifica las tareas comunes de un rol de Secretario. Entre las tareas características de este tipo de rol están la convocatoria de reuniones por orden de un líder (coordinador, director, presidente, etc.), el registro de sucesos/decisiones en un acta y la gestión administrativa.

2.2.1.2.2. Patrones de actividad

Podemos definir un *patrón de actividad* como aquel que especifica una actividad que se realiza típicamente en situaciones de trabajo cooperativo.

Representan el flujo de control y de objetos que sucede durante la realización de la actividad, así como los roles que participan en cada una de

las acciones que comprende. Su modelado se basa en los diagramas de actividad de UML 2 (v. Apéndice D), cuya semántica está inspirada en las redes de Petri[3], lo que facilita la especificación de procesos de negocio distribuidos, como son los flujos de trabajo cooperativo.

En la Tabla 5.6 describimos la intención de los distintos patrones de actividad que aparecen en el catálogo. Para una descripción completa de los patrones PROCESO DE REUNIÓN(2.4.1), REUNIÓN(2.4.2), VOTACIÓN(2.4.3) y NEGOCIACIÓN NO MODERADA(2.4.4) véase el Apéndice B.

Tabla 5.6: Intención de cada patrón de actividad del catálogo

Nombre	**Intención**
PROCESO DE REUNIÓN(2.4.1)	Modelar el flujo general de trabajo que conlleva el proceso de realización de una reunión. Desde su planificación por parte de un líder, hasta el envío de las actas, si las hay, una vez realizada la sesión de reunión.
REUNIÓN(2.4.2)	Modelar el flujo general de trabajo que comprende una sesión de reunión, en la que existe un líder que la conduce y una serie de miembros que debaten y, llegado el caso, votan los distintos puntos del orden del día. Los aspectos más relevantes de la sesión pueden ser registrados en un acta por alguien que desempeña el papel de secretario.
VOTACIÓN(2.4.3)	Modelar el flujo general de trabajo que comporta una sesión de votación, desde que alguien, actuando como líder o secretario, informa de las alternativas posibles de voto, hasta que éste informa del resultado obtenido en la votación.
NEGOCIACIÓN NO MODERADA(2.4.4)	Modelar el flujo general de trabajo que define una negociación no moderada, es decir, no existe un moderador responsable de coordinar las intervenciones. Se modela desde que los negociadores plantean sus necesidades, hasta que entre éstos se acuerda una propuesta o se cierra la negociación sin éxito.

[3] Si desea obtener información detallada sobre este formalismo puede consultar http://www.informatik.uni-hamburg.de/TGI/PetriNets/

CONVOCATORIA DE REUNIÓN	Modelar el flujo general de trabajo necesario para realizar la convocatoria de una reunión.
LLAMAMIENTO PARA PROPUESTAS	Modelar el flujo general de trabajo que implica un llamamiento para propuestas sobre algún producto, servicio o necesidad de información, desde que un iniciador emite dicho llamamiento para un grupo de participantes, hasta que el iniciador recibe y selecciona las propuestas que se ajustan a sus necesidades.
TORMENTA DE IDEAS	Modelar el flujo general de trabajo asociado con la dinámica de grupo conocida como tormenta de ideas o brainstorming, empleada para generar abundantes ideas sobre un determinado asunto.
CREAR PLAN DE TRABAJO	Modelar el flujo general de trabajo que conlleva la creación de un plan de trabajo. Desde el establecimiento de las tareas a realizar, sus interdependencias y su duración, hasta la asignación de los distintos recursos para cada tarea (personas, materiales, espacios, etc.).
ENRUTAR FORMULARIO	Modelar el flujo general de trabajo que describe las acciones que definen la ruta que debe seguir un determinado formulario, las tareas a realizar en cada instante, los actores autorizados y los tiempos a emplear en cada una de éstas.

2.2.1.2.3. Patrones de coordinación

La *coordinación* permite que cada unidad o parte de un todo sepa cómo y cuándo actuar para conseguir un objetivo mayor. Según Malone y Crowston [1990], la coordinación es "*la gestión de dependencias entre actividades*".

La utilización de diagramas de actividad facilita la representación de la coordinación entre las distintas acciones que componen una actividad y, por consiguiente, la coordinación entre los actores que las llevan a cabo, estableciendo cómo y cuándo interviene cada uno de ellos. Del mismo modo, los diagramas de rol permiten reflejar ciertas formas de coordinación mediante la especificación de eventos que activan las tareas en momentos determinados. Incluso los diagramas de organización permiten reflejar ciertos tipos de coordinación relacionados con posibles cambios de rol cuando suceden determinados sucesos. La coordinación también es necesaria cuando compartimos recursos, para indicar cuándo y cómo podemos acceder a ellos.

Podemos definir un *patrón de coordinación* como aquel que especifica un mecanismo común de coordinación entre actividades o actores.

El caso particular de los mecanismos de coordinación necesarios para acceder a la información o, en general, a los recursos compartidos, lo hemos incluido en los patrones que hemos denominado *de acceso*, los cuales forman parte de la vista de información.

Existen muchos trabajos que tratan el tema de la coordinación[4], sin embargo hay uno especialmente interesante para nuestro propósito. Van Der Aalst et al. [2003] identifica 21 patrones que representan construcciones de flujo de control típicamente usadas durante el modelado de flujos de trabajo. Éstas se clasifican en: *patrones de control básico* (definen aspectos fundamentales de control de procesos), *patrones de bifurcación avanzada y sincronización*, *patrones estructurales* (estructuración de procesos), *patrones de instancias múltiples* (describen situaciones donde puede haber más de una instancia de actividad activa al mismo tiempo dentro de la misma instancia de proceso), *patrones basados en el estado* (caracterizan escenarios de un proceso donde la siguiente ejecución está determinada por el estado de la instancia del proceso) y *patrones de cancelación* (maneras de terminar un proceso o sus componentes cuando se cumplen ciertas circunstancias).

Estos patrones aportan mecanismos típicos de coordinación de actividades que pueden ser incorporados perfectamente a nuestro catálogo. Es más, hay autores como Russell et al. [2006] y White [2004] que ya han modelado muchos de estos patrones por medio de diagramas de actividad de UML 2.0, lo nos facilita aún más esta labor. Simplemente podríamos generalizarlos, describirlos con la plantilla, definirlos con ayuda de nuestro profile y ligarlos a los modelos que generamos. Sin embargo, esto nos puede acarrear un problema de sobrecarga de información en los modelos. Muchos de estos patrones son tan básicos que no merece la pena identificarlos en los modelos. Algunos, incluso, están recogidos como estructuras elementales de la notación. No obstante, aunque no se liguen explícitamente, el conocimiento

[4] El lector puede encontrar multitud de referencias consultando el *Estudio Interdisciplinar de la Coordinación* de Malone y Crowston [1994].

de estos patrones facilita enormemente la construcción de los diagramas de actividad.

En la Tabla 5.7 indicamos la intención de varios patrones de coordinación del catálogo que no están incluidos en el trabajo de Van Der Aalst et al. [2003]. Estos patrones los usamos como mecanismos de coordinación de alto nivel, ya que gestionan las dependencias entre actividades/acciones en el dominio del problema. Para una descripción completa de los patrones PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO(2.5.1), PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO(2.5.2) y SALVAVIDAS(2.5.3) consúltese el Apéndice B.

Tabla 5.7: Intención de cada patrón de coordinación del catálogo

Nombre	Intención
PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO(2.5.1)	Modelar una estructura de coordinación en la que existe un actor (productor) que genera un producto y un actor (consumidor) que lo consume. La acción de producir o consumir debe terminar antes de volver a producir o consumir, respectivamente, otro producto.
PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO(2.5.2)	Modelar una estructura de coordinación en la que existe un actor (productor) que genera un producto y un actor (consumidor) que lo consume. La producción y la consumición es un flujo continuo. No es necesario que termine la producción de un producto para comenzar la producción del siguiente, al igual que no es necesario que finalice la consumición de un producto para empezar a consumir el siguiente.
SALVAVIDAS(2.5.3)	Modelar una estructura de coordinación en la que si surge un problema mientras un actor está realizando una determinada acción (acción protegida), existe otro actor específicamente encargado de darle solución, pudiendo continuar el primero con la acción que estaba realizando.
PRODUCTOR-CONSUMIDOR MÚLTIPLE DISCONTINUO	Modelar una estructura de coordinación en la que existen varios actores (productores) que generan productos y varios actores (consumidores) que los consumen. Los actores no comienzan la producción o consumición de un nuevo producto antes de terminar de producir o consumir, respectivamente, el anterior producto.
PRODUCTOR-CONSUMIDOR MÚLTIPLE CONTINUO	Modelar una estructura de coordinación en la que existen varios actores (productores) que generan productos y varios actores (consumidores) que los consumen. La producción y la consumición es un flujo

	continuo. No es necesario que termine la producción de un producto para comenzar la producción del siguiente, al igual que no es necesario que finalice la consumición de un producto para empezar a consumir el siguiente.
TURNO DE PALABRA	Modelar una estructura de coordinación donde la acción de un actor puede comenzar sólo cuando un actor moderador le concede su turno de palabra.

2.2.1.3. Vista de interacción

Esta vista contempla la especificación de la interacción entre participantes. AMENITIES ha venido utilizando los llamados protocolos de interacción para la realización de esta especificación. Según Garrido et al. [2002], éstos son conjuntos de reglas que especifican los pasos generales a seguir para realizar la actividad en base a protocolos sociales, los cuales implican sincronización y comunicación de cualquier clase de información (gestos, mensajes, documentos, etc.). Ejemplos de éstos pueden ser petición-respuesta, mensajes encolados, etc.

Uno de los motivos que inspiraron la aplicación de patrones en dicha metodología fue precisamente el concepto de protocolo de interacción. Asociar un protocolo a una actividad servía para especificar que ésta se llevaba a cabo siguiendo determinadas pautas de comunicación y coordinación entre sus participantes (protocolo social) con lo que, conociendo dichas pautas de comportamiento previamente, no era necesario desplegar dicha actividad para saber lo que ocurría dentro de ella. Sin embargo, la descripción de estos protocolos se hace en lenguaje natural, por lo que no está exenta de imprecisiones.

La noción de patrón va más allá de la idea de protocolo de interacción, tal y como fue concebida en su momento por AMENITIES. Los patrones, aparte de proporcionar modelos genéricos y precisos que pueden ser reutilizados durante el modelado de cualquier aspecto de un sistema (incluyendo, por supuesto, la interacción entre los participantes), añaden conocimiento útil para facilitar su aplicación.

En consecuencia, dentro de un marco de trabajo basado en patrones, creemos que ya no es necesario utilizar los protocolos de interacción en AMENITIES. Es más, en nuestro catálogo disponemos de patrones que

modelan exactamente lo que los protocolos de interacción expresaban narrativamente. Un ejemplo de este tipo de patrones son los patrones de comunicación.

2.2.1.3.1. Patrones de comunicación

Podemos definir un *patrón de comunicación* como aquel que especifica una forma de interacción típica entre participantes para el intercambio de información.

Tal y como ocurría con los protocolos de interacción, la ligadura de este tipo de patrones suele ir acompañada de etiquetas que indican los requisitos de comunicación de la actividad (v. `communication requirements` en el Apéndice A), siempre que éstos no estén previamente definidos en una actividad de nivel superior.

En la Tabla 5.8 aparece la intención de cada uno de los patrones de comunicación que hemos reunido en el catálogo. Los patrones DEBATE MODERADO(2.6.1), DEBATE NO MODERADO(2.6.2), PETICIÓN-RESPUESTA SIMPLE(2.6.3), PETICIÓN-RESPUESTA MÚLTIPLE(2.6.4) y EXPOSICIÓN(2.6.5) los describimos detalladamente en el Apéndice B.

Tabla 5.8: Intención de cada patrón de comunicación del catálogo

Nombre	Intención
DEBATE MODERADO(2.6.1)	Modelar una conversación grupal moderada. Los actores solicitan la palabra a un moderador, quien registra el turno y decide el momento en el que debe intervenir cada cual.
DEBATE NO MODERADO(2.6.2)	Modelar una conversación libre entre varios participantes.
PETICIÓN-RESPUESTA SIMPLE(2.6.3)	Modelar una forma de comunicación sincrónica donde un participante emite una petición para, a continuación, recibir una respuesta por parte del receptor, la cual es esperada por el emisor antes de continuar.
PETICIÓN-RESPUESTA MÚLTIPLE(2.6.4)	Modelar una forma de comunicación sincrónica donde un participante emite una petición para, a continuación, recibir respuesta por parte de un grupo de receptores. El emisor espera la llegada de suficientes

	respuestas antes de continuar.
EXPOSICIÓN(2.6.5)	Modelar una forma de comunicación en la que un actor (ponente) expone un tema a un grupo de participantes y resuelve las dudas que pudieran plantearle.
MENSAJES ENCOLADOS	Modelar una forma de comunicación asincrónica en la que un participante emite una petición y no espera la respuesta del receptor o receptores para poder continuar. Es un tipo de patrón petición-respuesta asíncrono.
PUBLICACIÓN-SUSCRIPCIÓN	Modelar una forma de comunicación asincrónica en la que los emisores (publicadores) no envían directamente sus mensajes a receptores específicos (suscriptores). Los mensajes publicados son divididos en clases. Los suscriptores expresan su interés en una o más clases y, sin tener conocimiento de los publicadores, sólo reciben los mensajes en los que están interesados.

2.2.1.4. Vista de información

Esta vista recoge la información que es compartida o comunicada dentro del escenario de trabajo. De acuerdo con el nivel de abstracción de AMENITIES, la información puede aparecer implícita en actividades/acciones o, si se cree conveniente, de manera explícita como nodos de objeto dentro de los diagramas de actividades. Claramente, la vista de información conecta, por tanto, con la vista cognitiva y de interacción.

Además de representar la información desde el punto de vista del flujo de datos u objetos que existe entre actividades/acciones, también es posible especificar e incluir en esta vista su estructura conceptual por medio de diagramas de clases de UML.

En el catálogo distinguimos dos tipos de patrones que se encuadran dentro de la vista de información: los patrones de estructura y los patrones de acceso.

Existen, no obstante, otros trabajos que pueden complementar y extender los patrones y los tipos que aquí presentamos. Por ejemplo, Russell et al. [2005] propone una ampliación a la labor desarrollada por Van Der Aalst et al. [2003] con el objetivo de capturar las diferentes maneras en las que los datos son representados y utilizados en los modelos de workflow. En este trabajo se clasifican los patrones en: *patrones de visibilidad de los datos*

(ámbito en el que los datos son visibles y pueden ser utilizados), *patrones de interacción de datos* (maneras en las que los datos fluyen según el tipo de los componentes de un proceso), *patrones de transferencia de datos* (distintos modos de transferencia a través de las interfaces de los elementos de un proceso) y *patrones de enrutamiento basado en datos* (precondiciones, postcondiciones, disparo de tareas en base a eventos, etc.).

2.2.1.4.1. Patrones de estructura

Vamos a definir un *patrón de estructura* como aquel que especifica la estructura conceptual de un objeto de información típico en un sistema cooperativo.

Desde un punto de vista general, autores como Fowler [1997], Hay [1996], Coad et al. [1995] y Nicola [2001] nos proveen de un amplio surtido de patrones útiles durante el modelado conceptual de datos.

En la Tabla 5.9 mostramos la intención de los diferentes patrones de estructura que pertenecen al catálogo. En el Apéndice B describimos detalladamente el patrón ACTA DE REUNIÓN(2.7.1).

Tabla 5.9: Intención de cada patrón de estructura del catálogo

Nombre	Intención
ACTA DE REUNIÓN(2.7.1)	Modelar la estructura conceptual correspondiente al acta de una reunión.
CALENDARIO DE EVENTOS	Modelar la estructura conceptual correspondiente a un calendario de eventos.
PLAN DE TRABAJO	Modelar la estructura conceptual correspondiente a un plan de trabajo.

2.2.1.4.2. Patrones de acceso

En situaciones de trabajo cooperativo los miembros del grupo compiten continuamente por el acceso a determinados recursos compartidos. Estos recursos pueden ser fisicos (p. ej. un ordenador) o lógicos (p. ej. un documento de texto almacenado en el ordenador). Por ello, a menudo, es

necesario establecer los mecanismos de coordinación adecuados que permitan determinar cuándo y cómo un actor puede acceder a ellos.

La autorización y control de acceso a los recursos compartidos es una importante dimensión a tener en cuenta desde las primeras etapas de desarrollo de los sistemas cooperativos. La seguridad dependerá en gran medida de estos mecanismos. Basándonos en un modelo de organización que considera los aspectos necesarios para representar las políticas de control de acceso y autorización, en Gutiérrez et al. [2006a, 2007] y en Sánchez [2006] proponemos una arquitectura orientada a servicios (SOA) para la implementación de un servicio basado en roles encargado de la gestión del control de acceso en sistemas colaborativos empresariales. Este servicio usa modelos del sistema construidos en base a patrones para controlar el acceso a los recursos y actividades por parte de los usuarios y otros subsistemas. Una de las principales características de este servicio es su capacidad para adaptarse a los cambios. Éstos pueden ser modificaciones en los modelos (cambios evolutivos) que varían la funcionalidad del sistema en tiempo real (p. ej., podemos cambiar la política de asignación de permisos para un cierto recurso añadiendo un nuevo rol dentro de la organización y distribuyendo las funciones entre el nuevo rol y los preexistentes) o cambios en la estructura organizacional (cambios adaptativos) que están predeterminados en el modelo (p. ej., cuando el director de una sucursal bancaria se encuentra ausente, éste puede delegar en el subdirector la autorización para la concesión de un préstamo).

Definimos un *patrón de acceso* como aquel que especifica un mecanismo común de coordinación para el acceso a la información u otros tipos de recursos compartidos.

La Tabla 5.10 resume la intención de los patrones de acceso que hemos propuesto para el catálogo. El patrón AUTORIZADO(2.8.1) puede consultarse en el Apéndice B.

Tabla 5.10: Intención de cada patrón de acceso del catálogo

Nombre	**Intención**
AUTORIZADO(2.8.1)	Modelar un mecanismo de coordinación en el que un actor o, lo que es lo mismo, la acción que realiza, podrá acceder a un determinado recurso compartido si

	dispone de los privilegios necesarios para ello y el recurso está disponible.
PRIMERO EN LLEGAR-PRIMERO EN SERVIRSE	Modelar un mecanismo de coordinación en el que un actor o, lo que es lo mismo, la acción que realiza, puede acceder inmediatamente a un determinado recurso compartido sólo si éste se encuentra libre.
ORDEN DE PRIORIDAD	Modelar un mecanismo de coordinación en el que un actor o, lo que es lo mismo, la acción que realiza, puede acceder a un determinado recurso compartido si la prioridad que tiene asignada se lo permite y el recurso está disponible.
TURNO DE ACCESO	Modelar un mecanismo de coordinación en el que un actor o, lo que es lo mismo, la acción que realiza, puede acceder a un determinado recurso compartido sólo cuando alguien o algo le asigna el turno de acceso.

2.2.2. Relaciones entre los patrones

Existen muchas formas posibles de relacionar los patrones. Entre ellas, las que más nos interesan son aquellas que nos facilitan la elección y aplicación del patrón más adecuado en cada momento.

Por ejemplo, la propia estructura del catálogo nos permite formar distintas familias de patrones, las cuales pueden ser de gran ayuda para satisfacer nuestro propósito. Así por ejemplo, podemos hablar de patrones relacionados por su aplicabilidad en el modelado de un cierto aspecto del sistema (organización, equipos, tareas de rol, actividades, etc.), por su participación en una determinada fase de construcción o por su relación con una determinada vista.

Complementariamente, también es posible constituir otros tipos de relaciones que conducen también a nuestro objetivo, por ejemplo: patrones que pueden aplicarse antes o después de uno dado (relación de proximidad), patrones que son usados dentro de otros (relación de uso), patrones que son similares (relación de similitud), etc.

A continuación, en la Figura 5.1 relacionamos los patrones del catálogo según diferentes criterios.

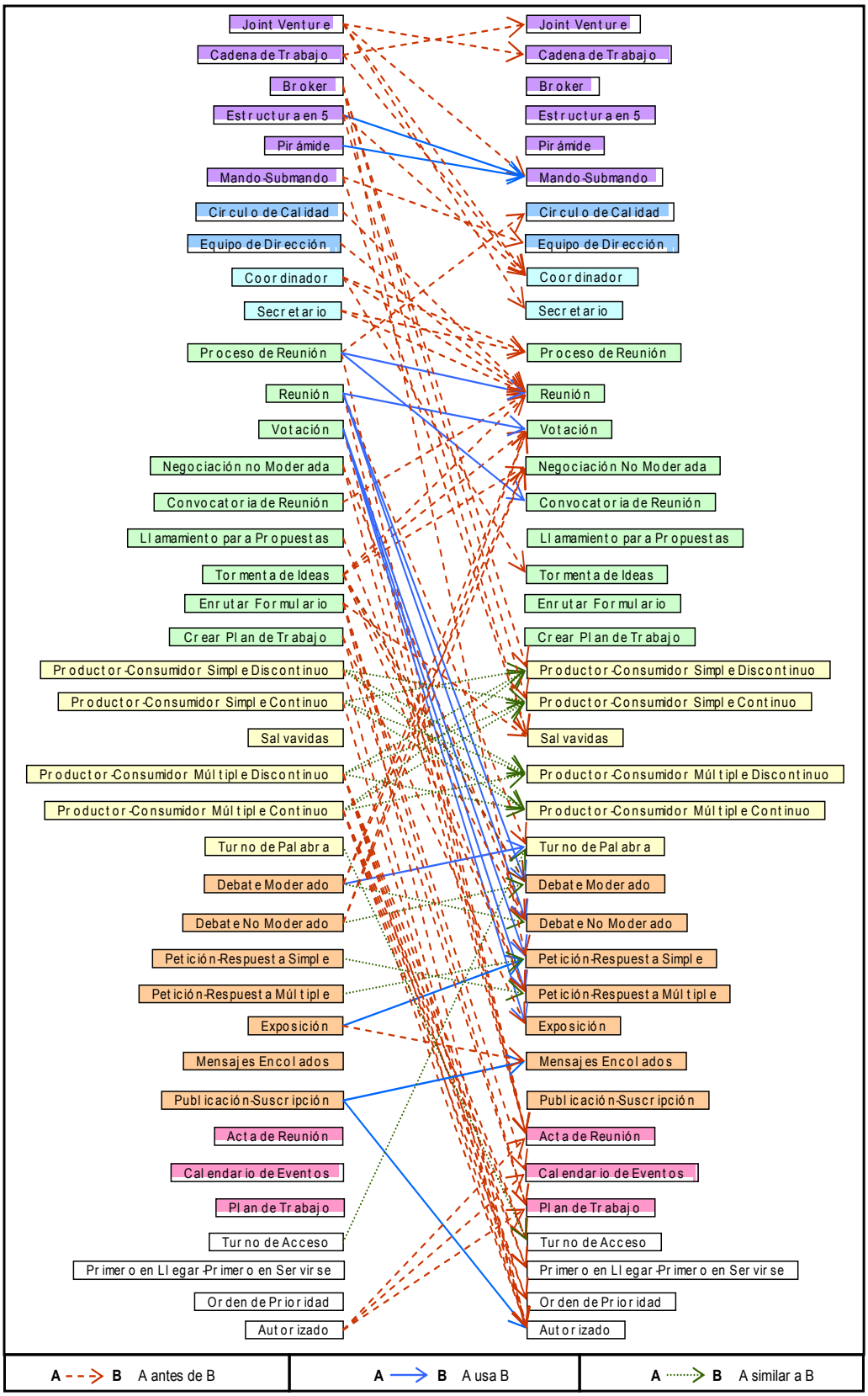


Fig. 5.1: Relaciones entre los patrones del catálogo

Los patrones son interconectados atendiendo a los siguientes tipos de relaciones:

- (A ——> B, A usa B), donde el patrón A referencia al patrón B en su definición.

- (A ----> B, A antes de B), donde el patrón A, el cual no usa B, se suele aplicar antes que el patrón B en determinadas ocasiones. Hay una relación de proximidad y no de uso. Los patrones pueden formar parte de la especificación de vistas diferentes.

- (A ·····> B, A similar a B), donde el patrón A resuelve un problema semejante al de B, pero sus contextos de aplicación son diferentes.

Estas relaciones están construidas en base a nuestra experiencia. No obstante, estamos seguros de que el tiempo va a jugar un papel muy importante en la extensión y mejora de esta red, sobre todo conforme se vayan introduciendo nuevos patrones en el catálogo.

Para no complicar más el diagrama hemos preferido no etiquetar las relaciones. Es posible conocer más detalles sobre éstas consultando directamente la descripción completa de cada patrón en el Apéndice B y leyendo la sección dedicada a “patrones relacionados”.

En el siguiente punto explicamos cómo sacar provecho de este catálogo para guiar la construcción del Modelo Cooperativo de un sistema en base a patrones.

3. Una propuesta metodológica para la construcción del Modelo Cooperativo de AMENITIES en base a patrones

De poco o nada sirve disponer de un catálogo de patrones si no sabemos cómo usarlo. Nuestra pretensión, como hemos dicho reiteradamente, es aplicar los patrones durante la construcción del Modelo Cooperativo de un sistema en el marco de la metodología AMENITIES.

Como vimos en la sección 3.4 del Capítulo II, el método de construcción que sugiere AMENITIES se compone de las siguientes fases:

1) Especificación de la organización.

2) Especificación de los roles.

3) Especificación de las tareas.

4) Especificación de los protocolos de interacción.

Tal y como apuntamos en la sección 2.2.1.3 de este mismo capítulo, bajo un enfoque basado en patrones no tiene demasiado sentido hablar de protocolos de interacción. Pensamos, por tanto, que esta cuarta etapa se puede obviar, quedando la especificación de estos protocolos integrada dentro de la etapa anterior (especificación de tareas) como ligaduras de patrones de comunicación con ciertas actividades.

Por otro lado, creemos necesario añadir una nueva fase, la cual sería entonces la cuarta y que podríamos denominar "*especificación del modelo conceptual de datos"*, para representar a nivel conceptual la estructura estática de la información manejada dentro del escenario. Mientras que otras consideran los datos como su principal foco de atención, la metodología AMENITIES se centra esencialmente en la especificación de la estructura y comportamiento de grupos dentro de escenarios de trabajo cooperativo, desde el punto de vista de su organización y de las tareas que realizan. De esta manera, la información (documentos, mensajes, eventos, recursos, etc.) que se comparte/comunica en el escenario, unas veces la modelamos como flujo de información entre actividades/acciones (nodos de objeto en los diagramas de actividad) y otras, simplemente, no la modelamos, asumiendo que ésta queda implícita en las actividades/acciones. Sin embargo, a menudo necesitamos especificar el modelo conceptual de datos asociado con dicha información, el cual nos ayuda a mejorar el conocimiento que tenemos sobre el dominio del problema.

A lo largo de las diferentes fases hemos de tomar multitud de decisiones que afectan a qué y cómo modelar. Justamente, los patrones nos asisten en la toma de esta clase de decisiones, ya que guían la percepción que tenemos del dominio y nos ayudan a identificar, comprender y especificar los escenarios que encontramos comúnmente.

No obstante, si no los empleamos con sentido común, la aplicación de patrones no está exenta de riesgos. Los patrones no son la panacea. Nunca debemos forzar la aplicación de un patrón si no se ajusta realmente a nuestro problema, ya que lo único que podemos conseguir es modelar un problema diferente. Es muy peligroso usarlos ciegamente sin cuestionarnos escrupulosamente su utilidad para el problema que estamos tratando. No debemos olvidar el acertado principio de modelado enunciado por Fowler:

"Los patrones son un punto de partida, no un destino."

—Fowler, 1997

El método que proponemos para la integración de patrones durante la construcción del Modelo Cooperativo de un sistema con AMENITIES consta básicamente de dos pasos:

1) Selección del patrón más adecuado para tratar el problema que tenemos entre manos.

2) Aplicación del patrón anteriormente seleccionado en el escenario concreto que pretendemos modelar.

Una vez aplicado un patrón, no deberíamos modificar la instancia, salvo que dichos cambios no afecten a las restricciones que éste impone, es decir, a su invariante (v. sec. 3.2). Si la modificación afecta a su invariante, deberíamos eliminar cualquier referencia a la ligadura utilizada para su construcción.

Según la experiencia que hayamos adquirido con el catálogo, será más o menos fácil saber si existe un patrón adecuado para cada uno de los problemas que encontramos. Por tanto, antes de hacer uso de este método por primera vez, recomendamos encarecidamente haber leído con detenimiento la descripción de los patrones que componen dicho catálogo. Además, es totalmente imprescindible conocer el perfil UML propuesto para el modelado de patrones de software (v. Cap. IV, sec. 4).

En los dos apartados siguientes explicamos las pautas generales que cada uno de estos pasos comprenden. Después, a modo de síntesis, en la Figura 5.2 mostramos el proceso general de selección y aplicación modelado mediante diagramas de actividad. No obstante, pretendemos que este método

sea flexible y que, en función de su experiencia, cada diseñador lo vaya adaptando a sus propias necesidades.

3.1. Selección

Ésta se basa en la aplicación de tres filtros consecutivos, ordenados desde el grano más grueso al más fino.

El primer filtro aprovecha la clasificación de patrones presentada en la Tabla 5.2, donde los patrones son organizados según la vista en la que participan, la fase de especificación del Modelo Cooperativo y el tipo de problema que manejan. También tiene en cuenta las relaciones de uso y las de proximidad (Fig. 5.1) que facilitan la búsqueda a partir de un patrón dado. La aplicación de este filtro da lugar a un conjunto de patrones potencialmente útiles.

El segundo filtro se basa en el examen del *nombre* y la *intención* de los patrones extraídos anteriormente. Para ello, tiene en cuenta las tablas de intención (Tabla 5.3 hasta Tabla 5.10) y las relaciones de similitud existentes en la Figura 5.1. Esto reduce aún más nuestro campo de búsqueda.

El tercer filtro se fundamenta en el estudio minucioso de las secciones *contexto*, *solución*, *explicación* y *ejemplo* correspondientes a las plantillas que describen detalladamente los patrones obtenidos con el segundo filtro. Éstas pueden ser localizadas en el apartado del Apéndice B que se referencia al final del nombre de cada patrón.

Si llegados a este punto no hemos encontrado aún ningún patrón que se ajuste a nuestras necesidades, no debemos considerar el tiempo perdido. Habremos aprendido algo más sobre los patrones y sobre nuestro problema, con lo que será más fácil para nosotros encontrar una solución adecuada.

Ahora bien, si después de haber estudiado el patrón en profundidad pensamos que tenemos delante el patrón idóneo debemos comenzar con la fase de aplicación.

3.1.1. Primer filtro

- Dependiendo de la fase de especificación en la que nos hallemos dentro del método de construcción del Modelo Cooperativo, podemos centrarnos en aquellos tipos de patrones que están relacionados con ésta (v. Tabla 5.2). En concreto:
  - Especificación de la organización: patrones de organización.
  - Especificación de roles: patrones de rol, patrones de coordinación y patrones de acceso.
  - Especificación de tareas: patrones de actividad, patrones de equipo, patrones de coordinación, patrones de comunicación y patrones de acceso.
  - Especificación de protocolos de interacción[5]: patrones de comunicación.
  - Especificación del modelo conceptual de datos: patrones de estructura.
- Dependiendo de la vista en la que estemos trabajando, podemos centrar nuestra búsqueda en los siguientes tipos de patrones (v. Tabla 5.2):
  - Vista organizacional: patrones de organización y patrones de equipo.
  - Vista cognitiva: patrones de rol, patrones de actividad y patrones de coordinación.
  - Vista de interacción: patrones de comunicación.

[5] Actualmente esta fase se encuentra integrada con la de especificación de tareas. Los protocolos de interacción, tal y como fueron empleados inicialmente por AMENITIES, han sido remplazados por la idea de patrones de comunicación o de actividad capaces de capturar protocolos sociales concretos y ligarlos con determinadas actividades.

- Vista de información: patrones de estructura y patrones de acceso.

- Dependiendo del problema concreto que estemos abordando, podemos ir directamente a los patrones cuyo tipo está relacionado con esta clase de problemas según la Tabla 5.2.

- Si ya hemos aplicado algún patrón y seguimos modelando el mismo escenario de trabajo, podemos navegar por la red de relaciones que mostramos en la Figura 5.1 para conocer cuáles son los patrones que pueden aplicarse después de éste, independientemente de la vista en la que participen. También puede ser conveniente ver cuáles son los patrones usados por éste.

  Podemos conocer los eventuales patrones que pueden ser aplicados después de uno dado, buscando su nombre en la primera columna y, con ayuda de la relación (A ----> B, A antes de B), viendo las conexiones que existen con los nombres de la segunda columna.

  En cuanto a los patrones que son usados por otro, éstos los podemos extraer viendo las relaciones de uso (A ——> B, A usa B) u observando directamente el modelo que define el patrón.

3.1.2. Segundo filtro

Para cada uno de los patrones obtenidos con la aplicación del primer filtro hacer lo siguiente:

- Leer su nombre. Puede que su identificador sea lo suficientemente significativo como para apostar por él.

- Repasar su intención. Aunque su descripción detallada con la plantilla ofrece también esta información, con objeto de facilitar este proceso, hemos dispuesto un conjunto de tablas (v. desde Tabla 5.3 hasta Tabla 5.10), cada una especializada en un tipo concreto, que muestra la intención de cada patrón. Hemos de elegir aquellos patrones cuya intención coincida con el escenario que pretendemos especificar o que sea similar. Para ello, podemos recorrer la red (Fig. 5.1) siguiendo las relaciones de similitud existentes (A ·····> B, A similar a B). Por ejemplo, partiendo del patrón PRODUCTOR-CONSUMIDOR SIMPLE

DISCONTINUO(2.4.1) podemos ver cómo los patrones similares son el resto de patrones de la clase productor-consumidor.

3.1.3. Tercer filtro

Accedemos a la sección del Apéndice B donde se encuentra cada uno de los patrones obtenidos con la aplicación del segundo filtro (v. referencia que aparece al final del nombre de cada patrón) y estudiamos detenidamente las siguientes secciones:

- Contexto. Debemos verificar si reconocemos estas circunstancias en el escenario que pretendemos modelar.

- Solución, explicación y ejemplo. Comprender la solución que propone el patrón y comprobar cómo es posible aplicarlo a un caso concreto, puede ser definitivo para saber si por fin hemos dado con el patrón que necesitamos.

3.2. Aplicación

Una vez que hemos encontrado el patrón que buscamos, ahora es el momento de aplicarlo.

Para entender las instrucciones que vamos a proporcionar a continuación, es necesario que el lector tenga un buen conocimiento del perfil *PMP (Pattern Modelling Profile)* propuesto en la sección 4 del Capítulo IV.

Las etiquetas definidas que incluyen las expresiones de ligadura correspondientes a las instancias del patrón deben corresponderse con *ligaduras completas*, es decir, todos los parámetros incluidos en los `<<PatternNamedElement>>` obligatorios[6] tienen que recibir algún valor y se

[6] Aquellos que no están afectados por alguna restricción de multiplicidad <<MultiplicityBind>> que especifique cero o más elementos.

deben respetar las restricciones de multiplicidad especificadas (v. Cap. IV, sec. 4.3.4.5.1).

La aplicación de un patrón va a depender de si el modelo correspondiente a su instancia necesita ser desplegado o no:

- Si no necesitamos desplegar su instancia, por ejemplo, debido a que aparece un símbolo (actividad, paquete, estado compuesto, etc.) que la va a contener o referenciar, basta con conectar a este símbolo una etiqueta que incluya la expresión de ligadura que define la instancia deseada. Esto es posible hacerlo sólo cuando la instancia se puede especificar totalmente a partir del patrón, es decir, no incluye elementos extra al patrón ni contiene elementos (por ejemplo subactividades) que necesitemos desplegar (v. por ejemplo Fig. 4.24 en Cap. IV).

- Si necesitamos desplegar la instancia, hacemos lo siguiente:

  - Definimos una etiqueta con la expresión de ligadura que define la instancia que queremos crear.

  - Desplegamos un modelo análogo (la instancia) al que provee el patrón[7] y que cumple con las restricciones que éste impone, o lo que es lo mismo, satisface su invariante.

    Para ello hacemos lo siguiente:

    - Sustituimos los parámetros por los valores que hemos utilizado en la expresión de ligadura. Estos valores se corresponderán

[7] Esto se puede hacer también creando y conectando cada elemento de la instancia con el `<<PatternNamedElement>>` del patrón que lo representa por medio de dependencias `<<PatternBind>>`. Recordemos que este tipo de relación exige: (1) El elemento cliente debe ser del mismo tipo que el elemento proveedor. (2) El número de clientes debe conformar con la restricción de multiplicidad de ligadura especificada en el elemento proveedor. (3) Dados dos elementos cliente pertenecientes a la misma instancia de un patrón, entre ellos deben aparecer al menos las mismas relaciones que existen entre sus respectivos elementos proveedores.

con entidades pertenecientes al escenario de trabajo concreto que estamos modelando. Para ello, crearemos tantos elementos como necesitemos, teniendo en cuenta que tenemos que respetar las condiciones que impone el invariante del patrón (v. más abajo).

- Si un `<<PatternNamedElement>>` tiene asociada una restricción de multiplicidad indicando opcionalidad (0 ó más elementos) y contiene algún parámetro no ligado, esto es, en su expresión de ligadura no se especifica ningún valor para ese parámetro, la instancia no debe incluir elemento alguno relacionado con dicho `<<PatternNamedElement>>`.

- Los elementos `<<UncertainElement>>` tienen que ser sustituidos por elementos de modelado pertenecientes al contexto del patrón. Funcionan como puntos de enganche con los elementos fronterizos.

- En el modelo generado no deben aparecer elementos propios de la notación usada para la definición de patrones.

El modelo construido debe cumplir con las restricciones que impone el patrón (su invariante), a saber:

- Cada elemento en la instancia debe ser del mismo tipo que su semejante en el patrón, esto es, el `<<PatternNamedElement>>` con el que se corresponde.

- El número de elementos de la instancia relacionados con cada semejante en el patrón debe conformar con la restricción de multiplicidad de ligadura (`<<MultiplicityBind>>`), si la hay, asociada con éste último. En su ausencia, por defecto, la cantidad de elementos a relacionar es uno.

- Dados dos elementos de la instancia, entre ellos deben aparecer relaciones similares a aquellas que existen entre sus elementos semejantes en el patrón.

– Conectamos la etiqueta que contiene la expresión de ligadura:

  - Con todos y cada uno de los `<<NamedElement>>` que forman la instancia (v. por ejemplo Fig. 4.19 en Cap. IV), o bien

  - Encerrando la instancia dentro de un rectángulo punteado y conectando la etiqueta a dicho rectángulo (v. por ejemplo Fig. 4.22 en Cap. IV), o bien

  - Directamente con el elemento que contiene o referencia a la instancia (p. ej. el símbolo de la actividad que contiene o referencia el diagrama de actividades que ha generado el patrón), si no incluye otros elementos que no son definidos por el patrón (v. por ejemplo Fig. 4.23 en Cap. IV).

- Tanto si necesitamos desplegar la/s instancia/s, como si no, aplicaríamos el mismo método en el caso de una *ligadura dinámica* (v. Cap. IV, sec. 4.3.4.5.1). Este tipo de ligaduras las utilizamos, bien porque puede variar de un momento a otro el patrón ligado (p. ej., el grupo puede cambiar dinámicamente su estrategia de trabajo), o bien porque se pueden realizar ligaduras diferentes con un mismo patrón (p. ej., los actores pueden cambiar dinámicamente sus roles manteniendo la misma estrategia de trabajo). La única diferencia está en que, de precisarlo, tendríamos que desplegar una instancia por cada una de las posibles ligaduras que se pueden definir a partir de la ligadura dinámica.

A continuación, por medio de varios diagramas de actividad en UML 2 (Figs. 5.2, 5.3, 5.4 y 5.5) representamos y resumimos todo este proceso (selección y aplicación).

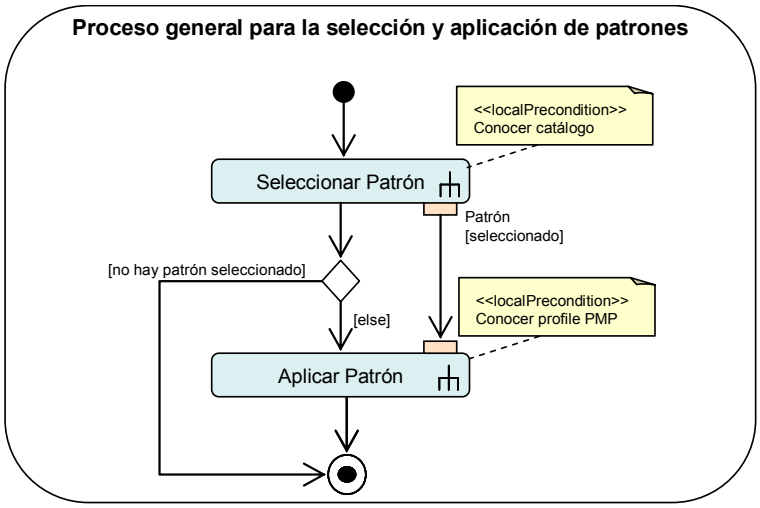


Fig. 5.2: Proceso general para la selección y aplicación de patrones con AMENITIES

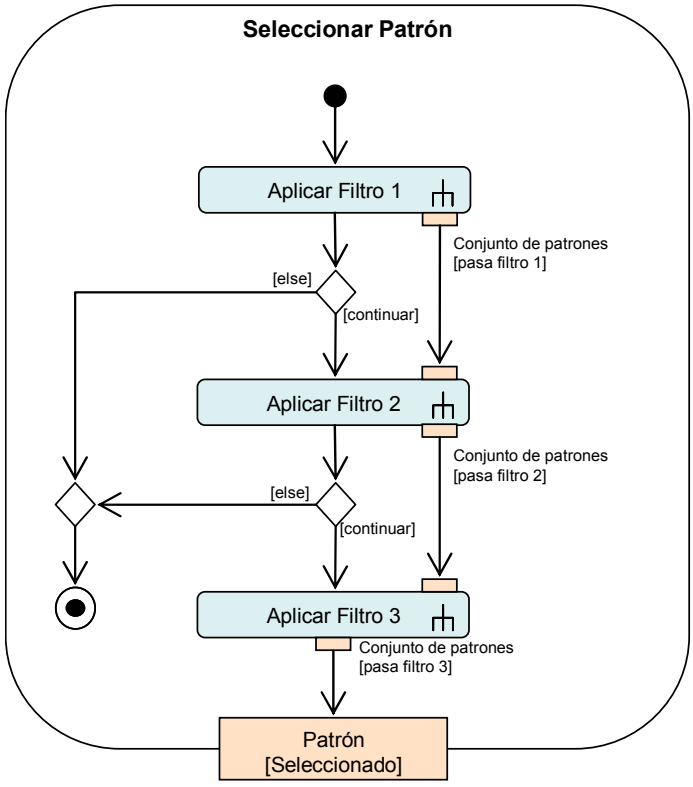


Fig. 5.3: Actividad `Seleccionar Patrón`

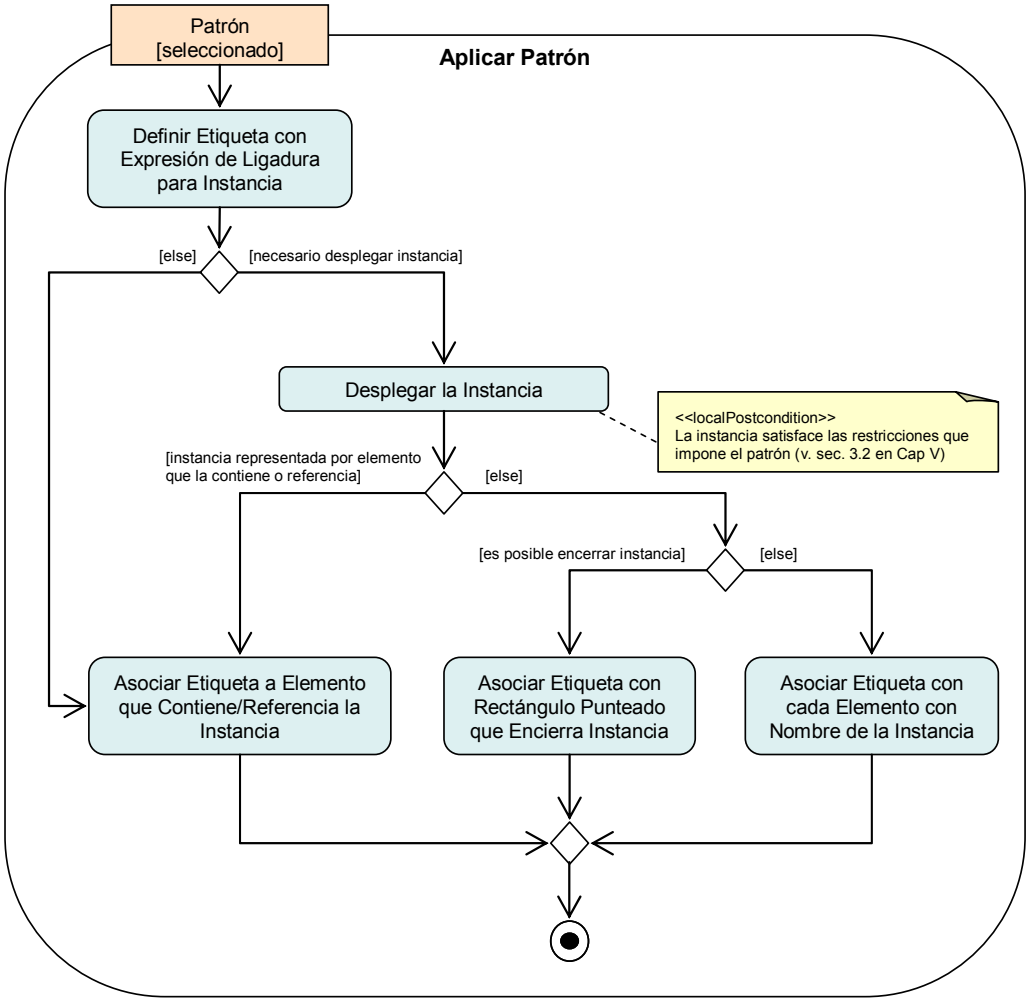


Fig. 5.4: Actividad `Aplicar Patrón`

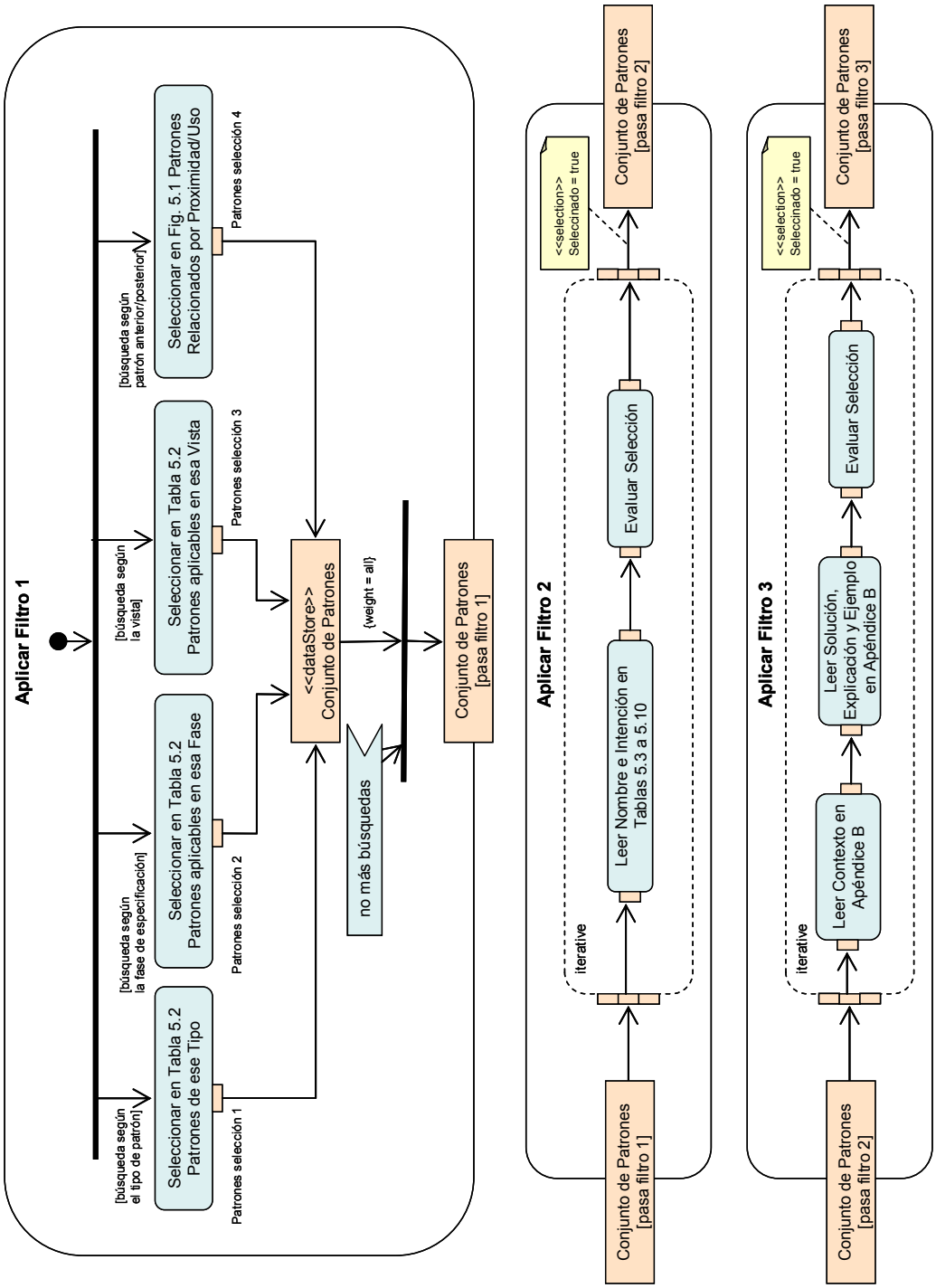


Fig. 5.5: Actividades `Aplicar Filtro 1`, `Aplicar Filtro 2` y `Aplicar Filtro 3`

Entre nuestros objetivos más próximos está la implementación de una herramienta de modelado que facilite la selección y aplicación de patrones durante el proceso de construcción del Modelo Cooperativo de un sistema. Actualmente disponemos de una herramienta que permite construir dicho modelo, pero no cuenta con capacidad para el tratamiento de patrones. No obstante, aunque no es nuestro propósito presentar aquí, con el rigor que exige la Ingeniería de Requisitos, un documento de especificación de requisitos detallado y exhaustivo, creemos oportuno referenciar algunos de los requisitos más significativos, relacionados con el tratamiento de patrones, que debería satisfacer una herramienta de este tipo.

Aparte de garantizar la satisfacción de una serie de propiedades generales (requisitos no funcionales) como son la robustez, la portabilidad, la usabilidad, el rendimiento, la mantenibilidad, etc., es necesario que la herramienta cubra una serie de requisitos funcionales. Éstos últimos los vamos a dividir entre aquellos que tienen por objeto facilitar la selección de patrones y aquellos orientados a su aplicación.

- Requisitos relacionados con la selección de patrones:

  - Creación y mantenimiento de un catálogo de patrones.

    Por un lado, la herramienta debería facilitar la construcción y mantenimiento de una estructura arborescente capaz de albergar a los diferentes tipos de patrones que pudieran surgir. Esta estructura la utilizaríamos para organizar los patrones de acuerdo con la clasificación realizada en la Tabla 5.2. Debería permitir mover, copiar y editar sus elementos. Por otro lado, debería ser capaz de realizar las altas, bajas y modificaciones de las descripciones correspondientes a cada uno de los patrones, en base a la plantilla propuesta en la Tabla 5.1. Esta herramienta debería estar facultada, por tanto, para el modelado de patrones de acuerdo con la notación establecida en PMP. En este sentido, sería muy ventajoso que la herramienta permitiese la generalización y modelado de patrones con PMP a partir de la selección de modelos preexistentes.

Sería también interesante que prestara apoyo para la edición compartida del catálogo y sus patrones. La realización de reuniones electrónicas para discutir y refrendar las solicitudes de creación o cambio, la habilitación de algún tipo de servicio (sincrónico o asíncrono) para el intercambio de opiniones con otras personas del equipo que ya tengan experiencia con algún patrón concreto, etc., serían medidas tendentes a la mejora de la calidad de esta base de conocimientos.

- Consulta y selección de patrones. Debería posibilitar el empleo de diferentes criterios de búsqueda. En concreto, debería facilitar las búsquedas relacionadas con cada uno de los filtros propuestos en el método. Asimismo, en cada una de estas fases de filtrado, deberíamos poder seleccionar/deseleccionar los patrones que deseamos que pasen a la siguiente fase. Otro tipo de búsqueda que consideramos realmente interesante consiste en poder realizar la localización de patrones por proyectos concretos en los que fueron aplicados.

- Requisitos relacionados con la aplicación de patrones:

  - Definición de ligaduras. Este proceso debería estar guiado por la propia herramienta. En función del patrón, debería presentarnos sus parámetros, indicándonos la multiplicidad de ligadura de cada uno de ellos, y facilitarnos la introducción de valores respetando esta restricción. También debería dar soporte a la definición de ligaduras dinámicas.

  - Generación de la instancia. Si necesitamos desplegar la instancia, la herramienta podría generar automáticamente, a partir de la definición de la ligadura, el modelo que cumple con las restricciones que impone el patrón (v. sec. 3.2). En caso de existir elementos `<<UncertainElement>>` debería solicitarnos los elementos del contexto que van a actuar como puntos de enganche.

  - Asociación de etiquetas de ligadura. Debería poder ofrecer la mejor alternativa para conectar la etiqueta, aunque para ello tenga que reubicar los elementos del modelo. La localización e identificación

de las instancias a través de sus etiquetas, facilita la comprensión, documentación, modificación y comunicación de los modelos.

- Vista de patrones y documentación. Sería interesante que la herramienta permitiera observar una panorámica general de los patrones que han sido aplicados y sus relaciones. Además sería muy útil que la herramienta pudiera generar automáticamente información relacionada con los patrones utilizados de cara a la documentación del proyecto.

- Chequeo del invariante de los patrones. En todo momento, se deberían verificar las restricciones que cada patrón impone sobre sus instancias. Cualquier modificación posterior puede provocar que su invariante no llegue a cumplirse. Sería interesante que la herramienta dispusiera de un "reparador" que, una vez detectada la violación de alguna restricción y previa autorización del usuario, fuera capaz de corregir el problema automáticamente.

- Propagación de modificaciones. Cualquier tipo de cambio realizado a los patrones debería propagarse automáticamente hacia sus instancias.

- Ayuda al diseño. A partir de los patrones conceptuales aplicados, la herramienta debería proveer información acerca de cuáles son los patrones de diseño que podrían participar posteriormente durante una fase de diseño inicial. En la siguiente sección introducimos este tema.

4. Un puente hacia el diseño

Como vimos en el Capítulo III, hay autores [Schümmer et al., 1999; Guerrero y Fuller, 1999; Schümmer, 2004a, 2004b, 2004c; Lukosch y Schümmer, 2004; etc.] que proveen diversos patrones aplicables durante las primeras fases de diseño de un sistema cooperativo. Estos patrones abordan aspectos relacionados con el diseño de la interacción, la comunicación, la colaboración, la interfaz del usuario, la coordinación, la compartición de datos, etc. Claramente, estos tipos de problemas guardan relación con los aspectos que tratan los patrones conceptuales que aportamos en el catálogo.

La diferencia fundamental reside en el nivel de abstracción con el que se tratan estos problemas.

El paso del análisis al diseño no es directo y está lleno de dificultades. Es un proceso complejo en el que hay que tomar multitud de decisiones relacionadas con la forma en que queremos llevar a cabo la implementación del sistema. Dado un problema, suelen existir numerosas alternativas de diseño, entre las cuales tendremos que elegir aquellas que nos permitan satisfacer los requisitos del sistema de manera eficaz.

Si los patrones conceptuales se aplican en el dominio del problema y los patrones de diseño en el dominio de la solución ¿podríamos tender un puente que nos facilite el paso de un dominio a otro a través de las posibles conexiones existentes entre ambos tipos de patrones?

Nosotros vislumbramos que esto es factible y de hecho anunciamos algunas de estas posibles conexiones en la Tabla 5.11. Es necesario señalar que existen ciertos patrones que tratan problemas semejantes, aunque con puntos de vista diferentes. Incluso hay un patrón con el mismo nombre (v. SESSION). La razón es que dichos patrones proceden de autores distintos.

Esto es sólo una aproximación inicial. Profundizar en este campo es para nosotros de un gran interés, ya que nuestro propósito es extender el uso de patrones a todas las fases del proceso de desarrollo de un sistema cooperativo. Con esto en mente, nuestros primeros pasos los hemos dado en González et al. [2007]. En este trabajo analizamos las bondades de los patrones de organización y de interacción para el diseño de interfaces de usuario colaborativas. Por un lado, los patrones de organización están relacionados con la identificación de roles y de actividades ligadas a la dinámica del grupo dentro de la organización. Por otro lado, los patrones de interacción documentan facilidades que la interfaz de usuario debería ofrecer para el desarrollo de las tareas que comúnmente realiza un usuario cuando utiliza un producto software.

Tabla 5.11: Algunas conexiones entre patrones conceptuales y de diseño

Patrón de Diseño/Alias	Propósito y publicación donde aparece	Patrón conceptual relacionado
SESSION/ WORKGROUP	Administrar y coordinar los miembros de los grupos de	* REUNIÓN(2.4.2) * VOTACIÓN(2.4.3)

	trabajo. Mantener una lista de éstos y de los que están actualmente conectados. Solicitar el nombre y la clave a cada usuario que quiere unirse al trabajo y validarlo [Guerrero y Fuller, 1999].	* NEGOCIACIÓN NO MODERADA(2.4.4) * TORMENTA DE IDEAS * CREAR PLAN DE TRABAJO * DEBATE NO MODERADO(2.6.2) * DEBATE MODERADO(2.6.1)
REPOSITORY	Almacenar la información compartida y mantener su consistencia [Guerrero y Fuller, 1999].	* TORMENTA DE IDEAS * ACTA DE REUNIÓN(2.7.1) * CALENDARIO DE EVENTOS * PLAN DE TRABAJO
OBJETO/ META-OBJECT/ INFORMATION OBJECT	Almacenar la información sobre cada uno de los objetos generados (propietario, fecha y hora de creación, tipo de objeto, palabras clave, versión, etc.) [Guerrero y Fuller, 1999].	* ACTA DE REUNIÓN(2.7.1) * CALENDARIO DE EVENTOS * PLAN DE TRABAJO
VIEW	Mostrar sólo una parte del almacén de datos dependiendo del rol del usuario. Seleccionar distintos formatos de presentación [Guerrero y Fuller, 1999].	* COORDINADOR(2.3.1) * SECRETARIO * AUTORIZADO(2.8.1)
FLOOR CONTROL	Disponer de una política de asignación que forme parte del propio objeto y que decida a qué usuario concede el control del objeto (exclusión mutua) [Guerrero y Fuller, 1999].	* TURNO DE ACCESO * PRIMERO EN LLEGAR-PRIMERO EN SERVIRSE * ORDEN DE PRIORIDAD * AUTORIZADO(2.8.1)
BROADCAST	Enviar durante una sesión de trabajo un objeto a los restantes miembros del grupo [Guerrero y Fuller, 1999].	* PETICIÓN-RESPUESTA MÚLTIPLE(2.6.4) * LLAMAMIENTO PARA PROPUESTAS * EXPOSICIÓN(2.6.5)
ROLE	Gestionar los derechos de acceso a la información y las operaciones [Guerrero y Fuller, 1999].	* COORDINADOR(2.3.1) * SECRETARIO
COMMUNICATION CHANNEL	Añadir un conjunto de herramientas para el intercambio de información (diferentes formatos) entre usuarios del entorno de trabajo. Puede actuar automáticamente cuando el sistema entra en un estado concreto (por ejemplo	* DEBATE MODERADO(2.6.1) * DEBATE NO MODERADO(2.6.2) * PETICIÓN-RESPUESTA SIMPLE(2.6.3) * PETICIÓN-RESPUESTA MÚLTIPLE(2.6.4)

	más de un usuario modificando un elemento de información) o bajo demanda. Una buena solución sería integrar herramientas de comunicación multimedia con las aplicaciones a desarrollar [Schümmer et al., 1999].	* EXPOSICIÓN(2.6.5) * MENSAJES ENCOLADOS * PUBLICACIÓN-SUSCRIPCIÓN
SESSION	Proveer un entorno común para toda la gente que trabaja junta. Cada usuario es consciente de lo que está haciendo el resto del grupo, pudiendo ver y manipular los mismos documentos. Los usuarios podrán crear nuevas sesiones, dividirlas, unirlas, eliminarlas, entrar, salir, etc. También se puede establecer el tipo de colaboración (desde relajada a rigurosa) entre los participantes [Schümmer et al., 1999].	* REUNIÓN(2.4.2) * VOTACIÓN(2.4.3) * NEGOCIACIÓN NO MODERADA(2.4.4) * TORMENTA DE IDEAS * CREAR PLAN DE TRABAJO * DEBATE NO MODERADO(2.6.2) * DEBATE MODERADO(2.6.1)
USER ROLE	Para representar los diferentes comportamientos que un usuario puede mostrar dependiendo del contexto de la colaboración. La solución consiste en construir una jerarquía de roles para cada usuario del sistema. El rol puede cambiar dinámicamente dependiendo de la tarea que esté realizando. [Schümmer et al., 1999].	* JOINT VENTURE(2.1.1) * CADENA DE TRABAJO(2.1.2) * BROKER * ESTRUCTURA EN 5 * PIRÁMIDE * MANDO-SUBMANDO
GROUP LOCATION AWARENESS	Proveer una referencia permanente de dónde están los otros usuarios. Se podría solucionar con un objeto navegacional que actúe como un índice dinámico para las localizaciones de los otros usuarios (por ejemplo una lista con los otros usuarios o un mapa de la estructura hipermedia) [Schümmer et al., 1999].	* SALVAVIDAS(2.5.3) * JOINT VENTURE(2.1.1) * CADENA DE TRABAJO(2.1.2) * BROKER * ESTRUCTURA EN 5 * PIRÁMIDE * MANDO-SUBMANDO
AVATAR	Representar el usuario en el sistema. Puede ser un dibujo, un nickname, etc. [Schümmer et al., 1999].	Aplicable en prácticamente cualquier situación
VIRTUAL ROOM/	Estructurar y representar la	* REUNIÓN(2.4.2)

ROOM METAPHOR	colaboración entre usuarios y grupos de una manera natural e intuitiva. A diferencia de una sesión, una habitación virtual puede permanecer incluso habiendo terminado una sesión. Además una habitación podría contener material para una sesión sin estar activa ésta. Es necesario acompañar éstas de funciones como crear, borrar, nombrar, listar, etc., habitaciones. Es el lugar donde tiene lugar una sesión [Schümmer et al., 1999].	* VOTACIÓN(2.4.3) * NEGOCIACIÓN NO MODERADA(2.4.4) * TORMENTA DE IDEAS * CREAR PLAN DE TRABAJO * DEBATE NO MODERADO(2.6.2) * DEBATE MODERADO(2.6.1) * ACTA DE REUNIÓN(2.7.1) * EQUIPO DE DIRECCIÓN * CÍRCULO DE CALIDAD(2.2.1)

5. Conclusiones del capítulo

Una vez presentada la notación que vamos a usar para la representación de nuestros patrones (v. Cap. IV), en este capítulo hemos introducido el resto de elementos que vamos a necesitar para agilizar el modelado conceptual de un sistema cooperativo con la metodología AMENITIES, o lo que es lo mismo, la construcción de su Modelo Cooperativo.

En la primera parte de este capítulo sentamos las bases e iniciamos la creación de un catálogo de patrones conceptuales aplicables durante la construcción de dicho Modelo Cooperativo (*Patrones Cooperativos*). Para ello, definimos una plantilla que permite su descripción uniforme, caracterizamos el tipo de patrones que van a formar parte de nuestro catálogo y establecemos los criterios que nos van a ayudar a clasificarlos e interrelacionarlos. Fruto de nuestra experiencia, hemos recopilado un conjunto inicial de patrones. Estos patrones los hemos clasificado según el catálogo y algunos de ellos los hemos especificado sistemáticamente (v. Apéndice B) con ayuda de la plantilla de descripción citada anteriormente.

Con ánimo de agilizar su búsqueda y aplicación, los criterios que hemos utilizado para clasificar los patrones del catálogo son la vista en la que participan dentro de AMENITIES (organizacional, cognitiva, interacción o información), fase de construcción del Modelo Cooperativo donde se suelen aplicar (especificación de la organización, de los roles, de las tareas, de los protocolos de interacción o del modelo conceptual de datos) y aspecto que abordan dentro del escenario cooperativo. Éste último identifica el tipo

concreto de patrón cooperativo (patrón de organización, de equipo, de rol, de actividad, de coordinación, de comunicación, de estructura o de acceso). A menudo, encasillar un patrón es ciertamente difícil, debido a que diferentes aspectos relacionados con la cooperación suelen entremezclarse dentro de un mismo patrón. Por ejemplo, un patrón de comunicación normalmente incluye alguna forma de coordinación y viceversa, un patrón de actividad puede incluir aspectos de coordinación, comunicación, información, etc. Sin embargo, el objetivo principal con el que utilizamos un determinado patrón es casi siempre único, éste es el que usamos para catalogar a un patrón dentro de un determinado tipo u otro.

La estructura del catálogo nos permite formar familias de patrones relacionados que facilitan la selección y aplicación del patrón más adecuado en cada momento. No obstante, de manera complementaria, hemos constituido una red de conexiones entre patrones en base a otros criterios que nos pueden ayudar también en nuestro empeño: patrones que pueden aplicarse antes o después de uno dado (relación de proximidad), patrones que son usados dentro de otros (relación de uso) y patrones que son similares (relación de similitud).

El catálogo que proponemos está totalmente abierto. Nuestro propósito en esta tesis no ha sido lograr un catálogo exhaustivo y completo[8], sino sentar las bases para que esto sea posible a medida que aumente nuestra experiencia y la de otras personas. Es de esperar, por tanto, que este catálogo evolucione con el tiempo, por medio de la introducción de nuevos patrones y tipos, así como la depuración de los ya existentes. Con todo, clasificamos y relacionamos los patrones que tenemos disponibles por el momento.

En la segunda parte de este capítulo, y sobre la base de los elementos anteriores, hemos explicado el método de construcción propuesto. Éste consta básicamente de dos pasos: 1) seleccionar el patrón más adecuado para

[8] Aunque podríamos haber añadido muchos otros patrones, a fin de no complicar excesivamente el catálogo, especialmente las relaciones entre patrones, hemos creído conveniente exponer tan sólo algunos ejemplos dentro de cada tipo.

el problema que estamos tratando, y 2) aplicar éste en el escenario concreto que queremos modelar. Aunque mostramos las pautas generales que cada uno de estos pasos comprende, pretendemos una aplicación flexible del método, de manera que, dependiendo de la experiencia de cada cual, éste sea usado de manera más o menos estricta. Es conveniente dejar claro que nunca debemos forzar la aplicación de un patrón si no se ajusta exactamente a nuestras expectativas, ya que lo único que podemos conseguir es modelar un escenario diferente. De cualquier forma, el conocimiento que nos aporta un patrón puede servirnos como punto de partida para encontrar una solución que encaje realmente con los requisitos de nuestro problema.

En la segunda parte esbozamos, además, los principales requisitos que debería satisfacer una herramienta para el modelado basado en patrones según nuestra propuesta.

Al final de este capítulo, como una primera aproximación, vislumbramos algunas posibles conexiones entre los patrones conceptuales que hemos propuesto y determinados patrones de diseño que aparecen en la literatura. Creemos que es posible tender un puente que facilite el paso del dominio del problema al dominio de la solución. Profundizar en este asunto es para nosotros de gran interés ya que, además de allanar el dificultoso camino de la transición de un dominio a otro, nuestro propósito es extender la aplicación de patrones hacia la etapa de diseño de sistemas cooperativos con la metodología AMENITIES.

Esta metodología nos ha proporcionado el sustrato ideal para poder desarrollar y sacar provecho de este catálogo. Los patrones comparten el mismo marco conceptual de trabajo que utiliza la metodología. Estos patrones modelan abstracciones comunes en el dominio del problema, con lo que enriquecen, a su vez, el propio marco conceptual. Cuando analizamos el dominio no pensamos sólo en base a los conceptos y relaciones que forman parte de este marco, sino también en términos de patrones. Además, la estructura del catálogo tiene en cuenta las vistas y fases utilizadas en la construcción del Modelo Cooperativo, facilitando la aplicación sistemática de los patrones a lo largo del proceso. No obstante, los escenarios que describen los patrones pertenecen al dominio del problema, el cual es compartido por cualquier metodología interesada en el desarrollo de los sistemas cooperativos. En consecuencia, entendemos que este conocimiento proporcionado por los patrones es de interés universal y que puede ser

reutilizado por cualquier metodología que comparta el mismo marco conceptual de trabajo.

-Esta página se encuentra deliberadamente en blanco-

Capítulo VI Casos de Estudio

Contenido

<<El fin de la ciencia especulativa es la verdad, y el fin de la ciencia práctica es la acción>>

—Aristóteles (384 AC - 322 AC)

Filósofo griego

1. Introducción

Este capítulo se divide en dos partes bien diferenciadas. En primer lugar abordamos la especificación del Modelo Cooperativo correspondiente a un Sistema para la Gestión de la Cooperación dentro de un Proyecto de Investigación Coordinado. En segundo lugar realizamos esta misma especificación en el caso de un Sistema para el Aprendizaje Cooperativo usando una estrategia tipo Jigsaw.

Utilizamos un esquema de descripción común para ambos casos de estudio. Primero introducimos el sistema a abordar y después especificamos los distintos modelos que componen el denominado Modelo Cooperativo del sistema.

No es nuestra pretensión hacer un estudio completo y exhaustivo de estos sistemas, ni explicar detalladamente la aplicación de la metodología AMENITIES. De hecho, hemos simplificado bastante el dominio del problema con objeto de que nuestra especificación sea lo más simple posible. Nuestro interés se centra, más bien, en mostrar cómo nuestra propuesta metodológica basada en patrones facilita el proceso de especificación del Modelo Cooperativo y mejora los modelos resultantes desde el punto de vista de su comprensión, comunicación y mantenimiento.

En función de la experiencia que tengamos con el catálogo de patrones, será posible saber de forma más o menos inmediata si existe algún patrón que encaje con nuestras necesidades. Lógicamente, el autor de esta tesis juega con cierta ventaja ya que, sin necesidad de seguir paso a paso el método propuesto en el capítulo anterior, conoce si existe algún patrón adecuado para cada uno de los problemas de modelado que encontramos. Como ya comentamos en su momento, dicho método tiene vocación de ser flexible, de manera que su aplicación puede ser más o menos rígida según la

experiencia de cada cual. No obstante, aunque no explicamos en este capítulo cada uno de los pasos del método que un modelador neófito en el uso del catálogo debería seguir para seleccionar el patrón más adecuado en cada momento, en ambos casos de estudio, antes de aplicar cada patrón indicamos las circunstancias que nos llevan a su elección.

Terminamos el capítulo presentando las conclusiones que hemos obtenido con esta experiencia.

2. Sistema para la Gestión de la Cooperación dentro de un Proyecto de Investigación Coordinado

2.1. Introducción

El primer caso de estudio se corresponde con el modelado de los aspectos cooperativos de un Sistema para la Gestión de la Cooperación dentro de un Proyecto de Investigación Coordinado. El objetivo es presentar el Modelo Cooperativo de AMENITIES que lo describe y destacar algunos de los patrones que participan en su especificación.

Varios grupos de investigación, interesados en parcelas de conocimiento complementarias y relacionadas con el tratamiento de un mismo problema, pueden unir sus esfuerzos, capacidades y recursos para participar en un proyecto de investigación conjunto que aborde dicho problema de manera integral. Para alcanzar este objetivo, existe la posibilidad de participar en un proyecto coordinado dividido en varios subproyectos, donde la responsabilidad de cada uno de éstos recae en grupos distintos de investigación.

La realización de un conjunto de actividades permite cubrir los diferentes subobjetivos en los que se suele desglosar el objetivo común. Unas veces estas actividades son asignadas a subproyectos determinados y otras son compartidas por varios subproyectos, colaborando los integrantes de diferentes grupos de investigación para su consecución.

Para el correcto desarrollo del proyecto, además de un cronograma que facilita la coordinación de las distintas actividades a realizar y, por consiguiente, de los grupos/subproyectos participantes, es fundamental la

realización de reuniones de coordinación. Estas reuniones favorecen el trabajo en equipo y la conciencia de grupo, a la vez que posibilitan la realización de acciones conjuntas sobre tópicos comunes.

2.2. Construcción del Modelo Cooperativo de AMENITIES

2.2.1. Especificación de la organización

Como señalamos en el Capítulo II, la especificación de la organización consiste, por un lado, en la identificación de los roles que intervienen y sus relaciones, y por otro, en la especificación de las condiciones (leyes de la organización o capacidades exigidas a los actores) que restringen la evolución del comportamiento en base a los posibles cambios de rol.

Si analizamos la organización de un proyecto coordinado, entre otras cosas, podemos extraer que:

- Hay un proyecto común que se compone de varios subproyectos.
- Para cada subproyecto existe un grupo de investigación responsable, el cual está especializado, o tiene un interés especial, en algún aspecto concreto del problema.
- Cada grupo de investigación puede tener su propia organización, pero siempre debe haber alguien encargado de representar al grupo/subproyecto dentro del proyecto coordinado (investigador principal del subproyecto).
- En el proyecto suele haber alguien que desempeña el papel de coordinador durante las reuniones de coordinación de los diferentes grupos/subproyectos (coordinador de subproyectos).
- En ocasiones alguien puede realizar labores de secretario y efectuar tareas como, por ejemplo, la redacción de las actas correspondientes a las sesiones de reunión.
- Existe una persona encargada de dirigir y representar el proyecto coordinado (director del proyecto).

Desde nuestra experiencia, fácilmente podemos comprobar que esta organización puede ser modelada a partir del patrón JOINT VENTURE (2.1.1). (Fig. 6.1). Examinando los patrones de organización del catálogo, podemos ver que la intención y el contexto de aplicación de este patrón encajan perfectamente con los requisitos del escenario que queremos modelar.

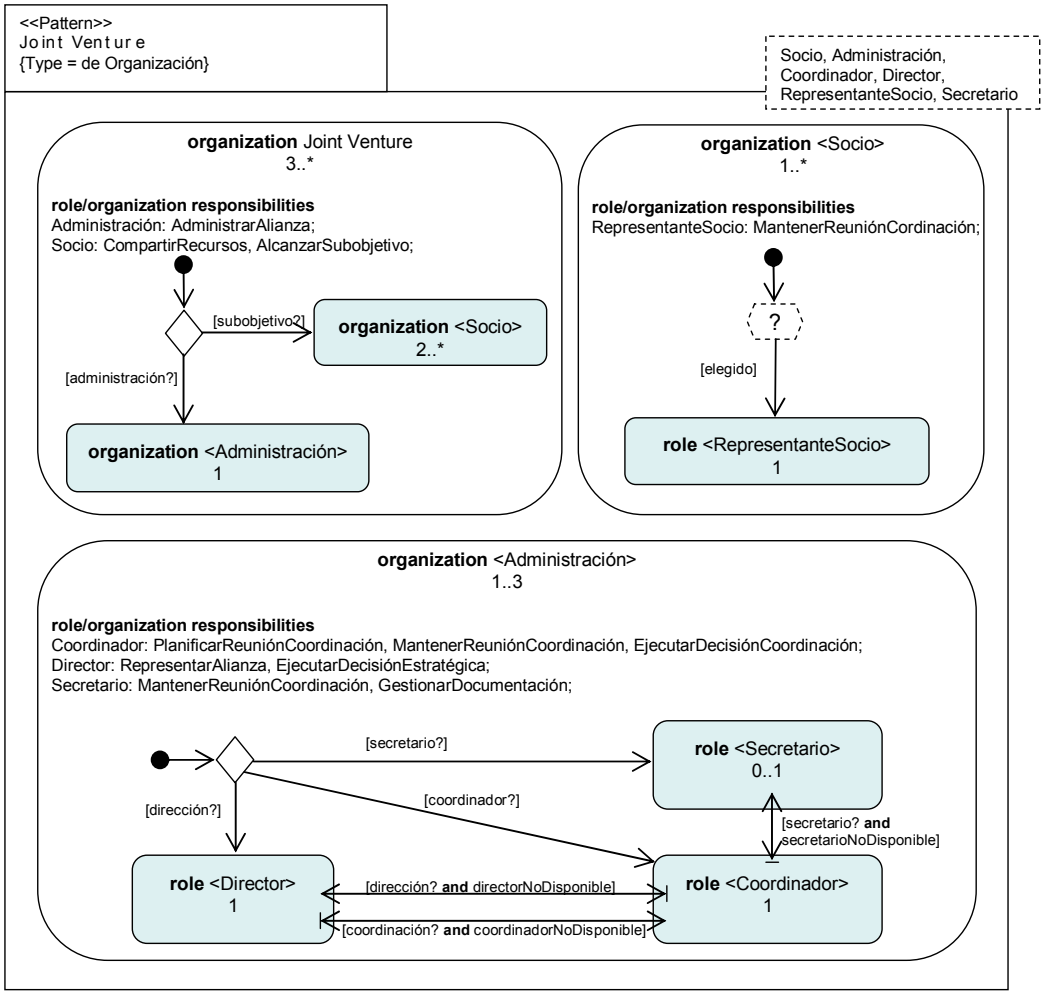


Fig. 6.1: Modelo correspondiente al patrón JOINT VENTURE(2.1.1)

La Figura 6.2 muestra el modelo correspondiente a la organización del proyecto de investigación coordinado construido en base a una ligadura de este patrón. El diagrama de la Figura 6.3 es equivalente al expuesto en la Figura 6.2. La diferencia estriba en que el primero muestra explícitamente las relaciones `<<PatternBind>>`, las cuales enlazan cada uno de los elementos que forman la instancia con el paquete que simboliza el patrón (el modelo que lo define permanece oculto), y el segundo no muestra estas relaciones ni el paquete. En el primer caso, la etiqueta de ligadura se conecta con el paquete

que representa el patrón y en el segundo ésta se conecta directamente con cada uno de los elementos que componen la instancia. Tanto en un caso como en otro la etiqueta especifica que `Socio`, `Administración`, `Coordinador`, `Director` y `RepresentanteSocio` se ligan respectivamente con "`GrupoSubproyecto`", "`AdministraciónProyecto`", "`Coordinador Subproyectos`", "`DirectorProyecto`" e "`InvestigadorPrincipal Subproyecto`", mientras que el parámetro `Secretario` recibe el valor '$', o lo que es lo mismo, recibe la cadena "`Secretario`" (coincide con el propio identificador del parámetro).

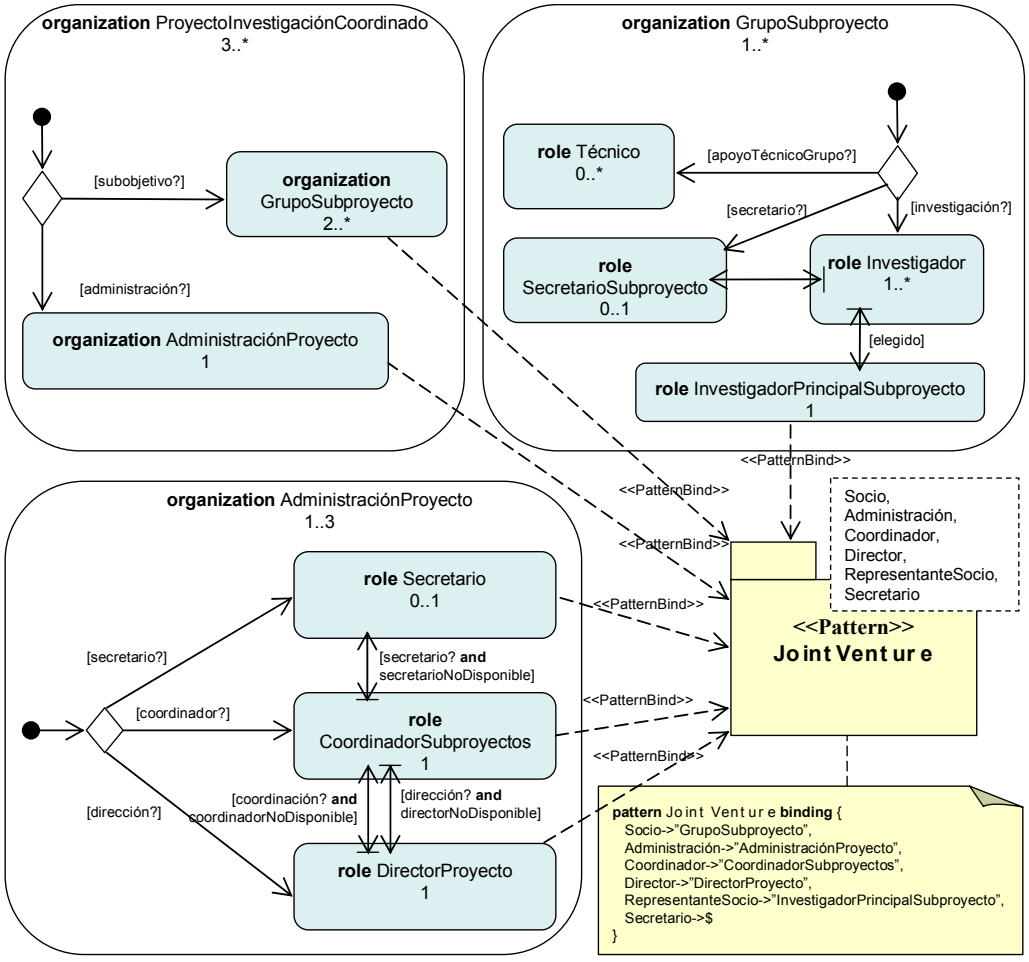


Fig. 6.2: Diagrama de organización correspondiente a un Proyecto de Investigación Coordinado

Una de las ventajas de la aplicación de patrones es que los modelos ligados pueden ser explicados a partir de la propia descripción del patrón, la cual puede consultarse en la sección del Apéndice B, coincidente con el índice que aparece al final del nombre del patrón (en este caso 2.1.1). Por tanto, no

sería necesario añadir en este momento más explicaciones que aquellas que no se contemplan en la descripción del patrón. No obstante, aunque sólo sea por esta vez y a modo de demostración, vamos a interpretar el modelo reproduciendo la explicación del patrón y adaptándola a la instancia concreta que acabamos de modelar[1].

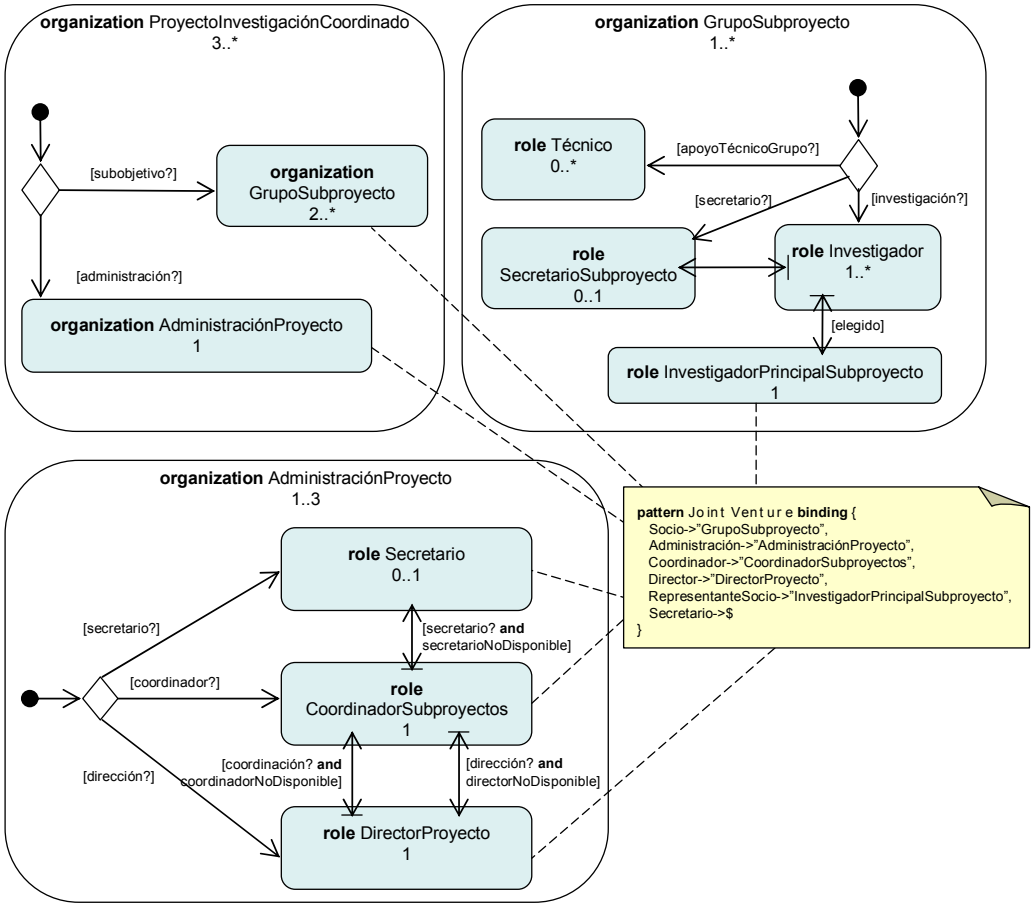


Fig. 6.3: Diagrama de organización equivalente correspondiente a un Proyecto de Investigación Coordinado

Una organización de tipo `ProyectoInvestigaciónCoordinado` es una alianza que se compone básicamente de dos clases de suborganizaciones: `GrupoSubproyecto` y `AdministraciónProyecto`.

[1] El lector puede comparar la explicación que realizamos aquí con la que aparece en la sección "explicación" perteneciente a la plantilla de descripción de este patrón

Los objetivos o responsabilidades de cada una de estas suborganizaciones[2], las cuales se pueden obtener del patrón (`v. role/organization responsibilities`), son las siguientes:

- Organización `GrupoSubproyecto` (dos o más):
  - Alcanzar el subobjetivo global que tiene asignado y en el que está especializada (objetivo `AlcanzarSubobjetivo`).
  - Compartir recursos con el resto de grupos (objetivo `CompartirRecursos`).
- Organización `AdministraciónProyecto` (sólo una)
  - Administrar el proyecto coordinado (objetivo `AdministrarProyecto`).

A su vez, en cada una de estas suborganizaciones existe un conjunto de roles[3] relevantes para la organización del proyecto y con objetivos[4] concretos (`v. role/organization responsibilities`):

- Rol `GrupoSubproyecto::InvestigadorPrincipalSubproyecto` (sólo un actor)
  - Mantener reuniones de coordinación (objetivo compartido `MantenerReuniónCoordinación`).
- Rol `AdministraciónProyecto::DirectorProyecto` (sólo un actor)
  - Representar al proyecto en los contactos con el exterior (objetivo `RepresentarProyecto`).

[2] Junto al nombre de cada una aparece la cantidad de organizaciones que participan (v. expresión de multiplicidad en los estados del diagrama).

[3] Junto al nombre de cada uno de estos roles (obsérvese que para identificarlos se utiliza el formato `organización::rol`) aparece la cantidad de actores que pueden desempeñarlo (v. expresión de multiplicidad en los estados del diagrama)

[4] Por supuesto, estos actores pueden formar también parte de otras organizaciones y tener otros objetivos.

  - Tomar las decisiones estratégicas para el proyecto (objetivo `EjecutarDecisiónEstratégica`).

- Rol `AdministraciónProyecto::CoordinadorSubproyectos` (sólo un actor)

  - Planificar las reuniones de coordinación con los grupos del proyecto (objetivo `PlanificarReuniónCoordinación`).

  - Mantener reuniones de coordinación con los grupos (objetivo compartido `MantenerReuniónCoordinación`).

  - Tomar decisiones para la coordinación de los distintos grupos (objetivo `EjecutarDecisiónCoordinación`).

- Rol `AdministraciónProyecto::Secretario` (uno o ninguno)

  - Asistir a las reuniones de coordinación (objetivo compartido `MantenerReuniónCoordinación`).

  - Gestionar la documentación relacionada con el proyecto (tarea `GestionarDocumentación`).

Aparte de las suborganizaciones, roles y objetivos, el patrón facilita la especificación de las condiciones (capacidades o leyes) que deben cumplir las suborganizaciones para formar parte de la organización, o las que deben cumplir los actores para poder jugar los distintos roles especificados.

Por un lado, las suborganizaciones de tipo `GrupoSubproyecto` tienen que tener capacidad para llevar a cabo alguno de los grandes subobjetivos en los que se divide el objetivo común (v. capacidad `[subobjetivo?]`). Por otro lado, la suborganización `AdministraciónProyecto` debe ser capaz de administrar el proyecto (v. capacidad `[administración?]`).

En cuanto a los roles, para que un actor pueda desempeñar el rol de `DirectorProyecto` debe tener capacidad de dirección (v. capacidad `[dirección?]`), para ser `CoordinadorSubproyectos` capacidad de coordinador (v. capacidad `[coordinador?]`), para actuar como `Secretario` capacidad de secretario (v. capacidad `[secretario?]`) y para ser

`InvestigadorPrincipalSubproyecto` debe haber sido elegido como tal en su propia organización (v. ley `[elegido]`).

Además de especificar la estructura de roles y el comportamiento individual de cada uno de éstos, el patrón facilita la descripción del comportamiento de la organización desde el punto de vista de los posibles cambios de rol que pueden experimentar sus actores a lo largo del tiempo, siempre de acuerdo con las leyes que impone la propia organización y la ocurrencia de determinadas condiciones. Así, hay una transición aditiva que representa bajo qué ley un actor que está desempeñando el rol de `CoordinadorSubproyectos`, sin abandonar éste, puede asumir además las labores correspondientes al rol de `DirectorProyecto`. Como se puede comprobar, esto puede suceder cuando el `CoordinadorSubproyectos` tiene capacidad de dirección y el `DirectorProyecto` no está disponible (v. condición `[dirección? and directorNoDisponible]`). En el momento en que alguna de estas condiciones deja de cumplirse, el actor abandona el rol de `DirectorProyecto` y vuelve a desempeñar exclusivamente su rol inicial como `CoordinadorSubproyectos`. De la misma forma, el `DirectorProyecto` puede desempeñar adicionalmente el rol de `CoordinadorSubproyectos` cuando tiene capacidad de coordinación y éste último no está disponible (v. `[coordinación? and coordinadorNoDisponible]`).

El modelo que define el patrón también representa la incertidumbre que existe dentro del diagrama correspondiente a la suborganización `Socio`, en relación con la posible presencia de otros roles encargados de realizar tareas propias de la suborganización y que pueden no tener nada que ver con el proyecto coordinado. Estos roles no se describen a partir del patrón, con lo que su descripción no está recogida en ningún sitio, al contrario de lo que ha pasado hasta ahora. Aquí, por tanto, es conveniente dar alguna explicación.

Los roles ajenos al patrón que forman parte de la suborganización `GrupoSubproyecto` (suborganización `Socio` en el patrón) son `Técnico`, `SecretarioSubproyecto` e `Investigador`. Cero o más actores pueden desempeñar el papel de `Técnico` y para ello deben ser capaces de dar soporte técnico al grupo (v. capacidad `[apoyoTécnicoGrupo?]`). El rol de `SecretarioSubproyecto` puede ser ejecutado a lo sumo por un actor y debe tener capacidad de secretario (v. capacidad `[secretario?]`). Al menos un actor con capacidad de investigación (v. capacidad `[investigación?]`)

deberá participar como `Investigador` dentro de `GrupoSubproyecto`. Además, el rol de `SecretarioSubproyecto` y de `InvestigadorPrincipalSubproyecto` debe ser jugado por alguien que es también `Investigador`.

2.2.2. Especificación de los roles

La definición de roles facilita la conexión entre la especificación de la organización y las actividades individuales/colaborativas a realizar por sus miembros. Los diagramas COMO-UML usados para realizar esta especificación se denominan *diagramas de rol*.

Como vimos en el capítulo anterior, los tipos de patrones que se pueden aplicar durante esta fase son: patrones de rol, patrones de coordinación y patrones de acceso.

Dentro de los patrones de rol disponemos en el catálogo del patrón COORDINADOR(2.3.1) y del patrón SECRETARIO. De momento, es el primero el único que tenemos completamente descrito en el Apéndice B. Por tanto, aunque sabemos que el segundo es potencialmente aplicable, la pregunta que debemos hacernos en este momento es: ¿puede ser útil el patrón COORDINADOR(2.3.1) (Fig. 6.4) para la especificación de alguno de los roles de esta organización?.

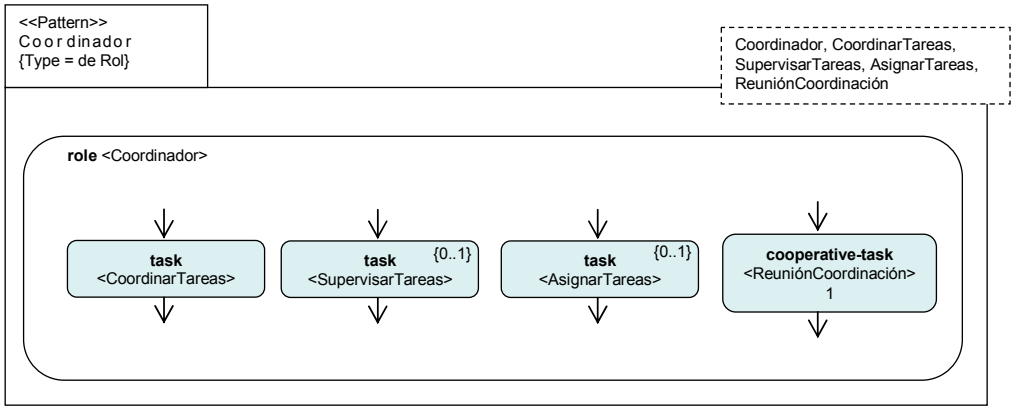


Fig. 6.4: Modelo correspondiente al Patrón COORDINADOR(2.3.1)

Entre los roles de la organización se encuentra el `CoordinadorSubproyectos`, cuyas tareas principales son planificar y participar en las reuniones de coordinación (v. tarea

`ReunióCoordinaciónSubproyectos`) así como ejecutar las decisiones de coordinación (v. tarea `EjecutarDecisiónCoordinación`).

Si echamos un vistazo a la descripción del patrón COORDINADOR(2.3.1) que aparece en el Apéndice B, podemos observar que entre las tareas típicas de este rol están precisamente las anteriormente mencionadas. En cierta medida esto es lógico, ya que el `CoordinadorSubproyectos` es un caso concreto de coordinador. Así pues, como vemos en la Figura 6.5, es posible construir y describir el diagrama de este rol mediante la ligadura mostrada. En este caso podemos observar cómo `<SupervisarTareas>` y `<AsignarTareas>`, los cuales son parámetros optativos del patrón (v. multiplicidad de ligadura `{0..1}`), no han sido utilizados para la generación de la instancia.

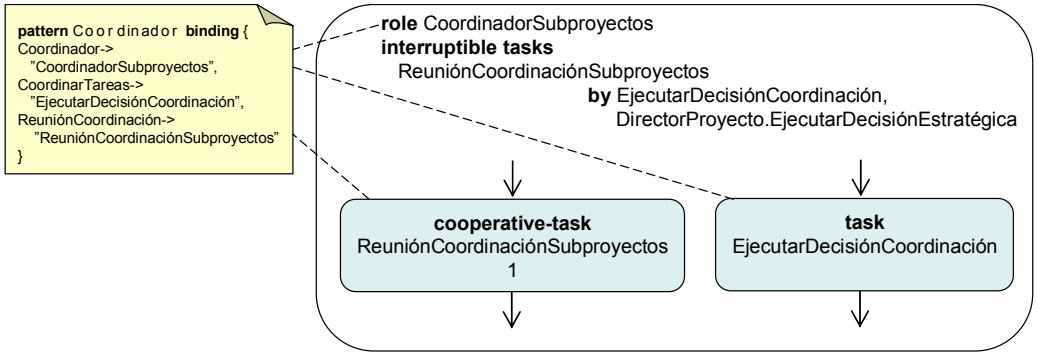


Fig. 6.5: Diagrama correspondiente al rol `CoordinadorSubproyectos` y su construcción/descripción en base a la ligadura del patrón COORDINADOR(2.3.1)

Tanto la tarea cooperativa `ReunióCoordinaciónSubproyectos`, en la que participa un sólo actor con este rol (v. multiplicidad `1`), como la tarea individual `EjecutarDecisiónCoordinación`, no están sincronizadas con ningún evento, ni condicionada su ejecución por alguna restricción, con lo que pueden ser iniciadas en cualquier momento por el actor.

Además de estas tareas, el diagrama añade otra información que el patrón no provee y que está relacionada con la interrupción de unas tareas por otras. En concreto, el diagrama muestra cómo la tarea `ReunióCoordinaciónSubproyectos` puede ser interrumpida cuando el actor emprende la tarea `EjecutarDecisiónCoordinación` o la tarea `EjecutarDecisiónEstratégica`, ésta última si el actor está desempeñando

también el rol de `DirectorProyecto` (v. condición necesaria en el diagrama de organización para que esto ocurra).

El siguiente diagrama de rol es el correspondiente al `InvestigadorPrincipalSubproyecto` (Fig. 6.6), al cual le corresponde participar en las reuniones de coordinación de subproyectos (v. tarea cooperativa `Reunión CoordinaciónSubproyectos` a la que deben concurrir al menos dos actores con este rol) cuando llegue el día y la hora de la reunión (v. evento `DíaHoraReuniónCS`), realizar la solicitud del subproyecto (v. tarea individual `SolicitarSubproyecto`) dentro del plazo de solicitud (v. condición `[PlazoSolicitudSubproyecto]`) y participar en las reuniones de investigación locales al subproyecto (v. tarea cooperativa `Reunión InvestigaciónSubproyecto` en la que interviene un solo actor con este rol).

La única tarea interrumpible es `SolicitarSubproyecto`, la cual puede ser interrumpida por cualquier otra tarea que pueda iniciar el actor, independientemente de su rol.

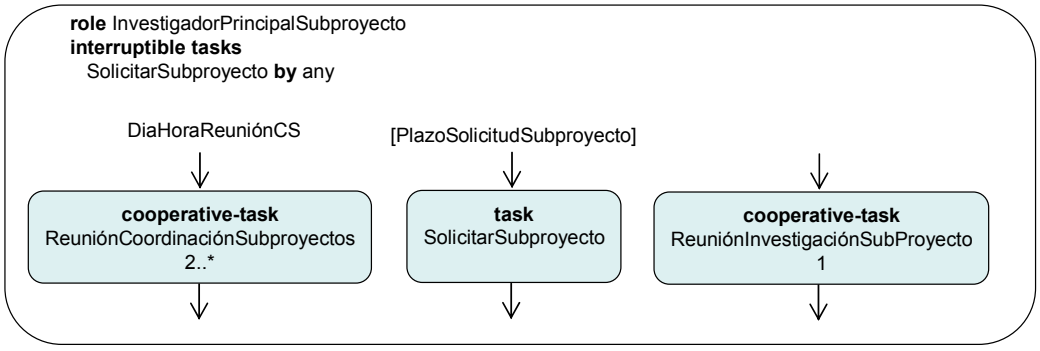


Fig. 6.6: Diagrama correspondiente al rol `InvestigadorPrincipalSubproyecto`

El diagrama asociado al rol de `Secretario` aparece en la Figura 6.7. Este rol se encarga de gestionar la documentación asociada al proyecto (v. tarea individual `GestionarDocumentaciónProyecto`), intervenir en las reuniones de coordinación de subproyectos (v. tarea cooperativa `ReuniónCoordinaciónSubproyectos` en la que participa un solo actor con este rol) desde el momento en que el `CoordinadorSubproyectos` solicita a éste que convoque a los distintos miembros (v. evento `CoordinadorPideConvocarReuniónCS`) y, por último, colaborar en las reuniones de investigación (p. ej. talleres) del proyecto (v. tarea cooperativa

`ReunióInvestigaciónProyecto` en la que participa un único actor con este rol) a partir de que el `DirectorProyecto` le pida a éste que convoque a los miembros (v. evento `CoordinadorPideConvocarReuniónIP`).

La única tarea interrumpible es `GestionarDocumentaciónProyecto`, la cual puede ser interrumpida por cualquier otra tarea que pueda iniciar el actor, independientemente de su rol.

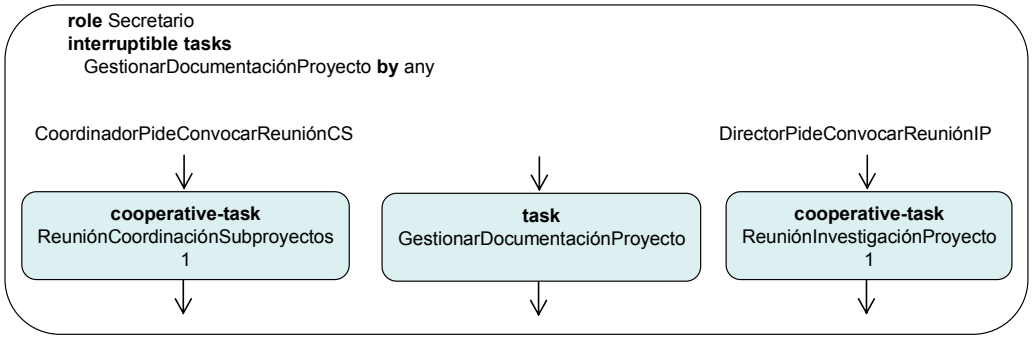


Fig. 6.7: Diagrama correspondiente al rol `Secretario`

El diagrama correspondiente al rol `DirectorProyecto` aparece en la Figura 6.8. Este rol es responsable de efectuar la solicitud del proyecto (v. tarea individual `SolicitarProyecto`) dentro del plazo preceptivo (v. condición `[PlazoSolicitudProyecto]`), ejecutar decisiones estratégicas relacionadas con el proyecto conjunto (v. tarea individual `EjecutarDecisiónEstratégica`) y participar en las reuniones de investigación (p. ej. talleres) del proyecto (v. tarea cooperativa `ReuniónInvestigaciónProyecto` en la que hay un único actor implicado con este rol).

La única tarea interrumpible es `SolicitarProyecto`, la cual puede ser interrumpida por cualquier otra tarea que pueda iniciar el actor, independientemente de su rol.

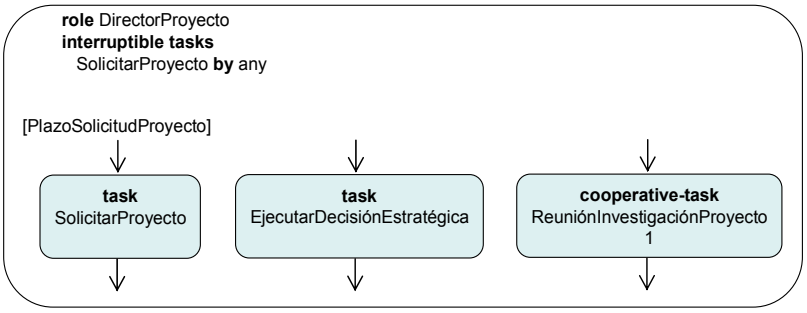


Fig. 6.8: Diagrama correspondiente al rol `DirectorProyecto`

La Figura 6.9 presenta el diagrama correspondiente al rol `Investigador`. Entre sus cometidos está participar en la elección de la persona (investigador principal) que coordina el subproyecto (v. tarea cooperativa `ElegirCoordinadorSubproyecto` en la que intervienen uno o más actores con este rol) cuando llega el momento de la elección (v. evento `DíaHoraElección`), realizar labores de investigación relacionadas con el proyecto (v. tarea individual `Investigar`), mantener reuniones de investigación en el marco del subproyecto (v. tarea cooperativa `ReunIónInvestigaciónSubproyecto` en la que participan dos o más actores con este rol) y reuniones de investigación a nivel de proyecto (v. tarea cooperativa `ReunIónInvestigaciónProyecto`) cuando llega el momento (v. evento `DiaHoraReunIónIP`).

La única tarea interrumpible es `Investigar`, la cual puede ser interrumpida por cualquier otra tarea que pueda iniciar el actor, independientemente de su rol.

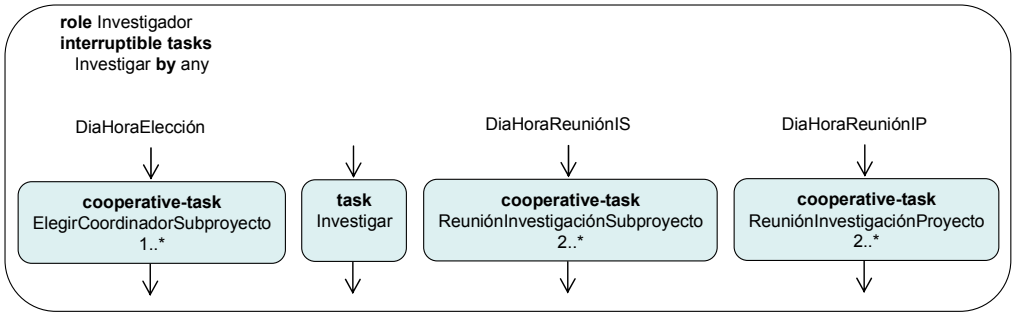


Fig. 6.9: Diagrama correspondiente al rol `Investigador`

El diagrama asociado al rol `SecretarioSubproyecto` aparece en la Figura 6.10. Este rol se dedica a gestionar la documentación relacionada con el subproyecto (v. tarea individual `GestionarDocumentaciónSubproyecto`) e intervenir en las reuniones de investigación del subproyecto (v. tarea cooperativa `ReunIónInvestigaciónSubproyecto` en la que participa sólo un actor con este rol) a partir del instante en que el `InvestigadorPrincipalSubproyecto` solicita a éste que convoque a los miembros (v. evento `InvestigadorPrincipalPideConvocarReunIónIS`).

La única tarea interrumpible es `GestionarDocumentación Subproyecto`, la cual puede ser interrumpida por cualquier otra tarea que pueda iniciar el actor, independientemente de su rol.

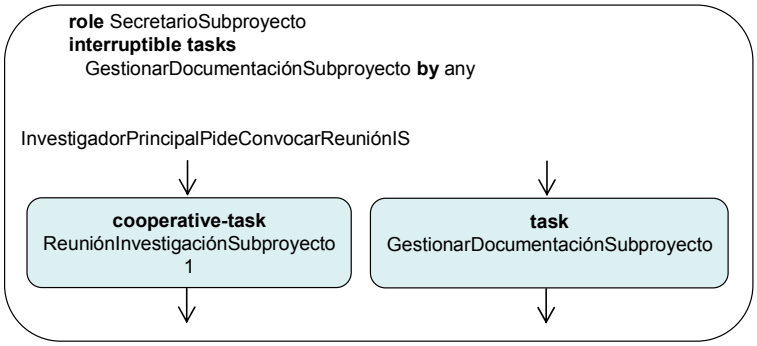


Fig. 6.10: Diagrama correspondiente al rol `SecretarioSubproyecto`

Por último, el diagrama de rol asociado al `Técnico` (Fig. 6.11) consta tan sólo de la tarea individual `PrestarSoporteTécnico`, la cual representa la obligación de prestar apoyo técnico al grupo cuando éste es solicitado (v. evento `PeticiónServicio`). En este caso, la tarea no es interrumpible.

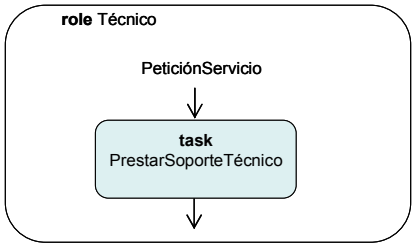


Fig. 6.11: Diagrama correspondiente al rol `Técnico`

2.2.3. Especificación de las tareas

En esta fase se describen las tareas que, junto con la definición de roles realizada en la etapa anterior, constituyen la vista cognitiva del modelo cooperativo. Durante esta fase, y formando parte de la vista de interacción, también se pueden utilizar patrones de comunicación (protocolos de interacción en AMENITIES) que facilitan la especificación de formas comunes de interacción entre actores que participan en actividades de comunicación.

Además, a menudo hemos de especificar equipos de trabajo asociados con actividades cooperativas, los cuales estarían integrados dentro de la vista organizacional.

Para esta especificación usaremos los diagramas de tareas y subactividades de COMO-UML, los cuales están basados en los diagramas de actividad de UML[5].

Nuestra pretensión en esta sección no es especificar todas y cada una de las tareas que aparecen en los diagramas de rol. Con esto no aportaríamos más pruebas a favor de la utilidad de los patrones para la construcción de los modelos, sino una sobrecarga de esta sección. Claramente, nuestro principal interés se centra en la especificación de las tareas cooperativas. Por lo demás, como hemos visto, prácticamente todas las tareas cooperativas hacen referencia a algún tipo de reunión, con lo que sus especificaciones van a resultar similares. Por consiguiente, para nuestro propósito, creemos que basta tan sólo con enfocar nuestra atención en alguna de las tareas cooperativas que aparecen, por ejemplo en la tarea `ReuniónCoordinaciónSubproyectos`.

Como acabamos de ver, esta tarea es común a los siguientes roles: `CoordinadorSubproyectos`, `Secretario` e `InvestigadorPrincipal Subproyecto`. El `CoordinadorSubproyectos` planifica y participa en las sesiones de reunión para la coordinación de subproyectos. El `Secretario` convoca dichas reuniones, interviene en las sesiones de reunión, prepara el acta correspondiente (si procede) y envía ésta a los miembros convocados. Los actores con el rol de `InvestigadorPrincipalSubproyecto` participan también en dichas sesiones de reunión cuando son convocados.

[5] Como se advirtió en el capítulo II, desde su concepción inicial, los diagramas de tareas y subactividades utilizados por AMENITIES han estado basados en los diagramas de actividad, de acuerdo con la versión oficial de UML correspondiente a aquella época (versión 1.5). En dicha versión, éstos eran considerados como un caso especial de diagrama de estados donde sus estados representaban acciones o actividades. Sin embargo, en la versión oficial actual del estándar [OMG, 2007a, 2007b] los diagramas de actividad poseen una nueva y rica sintaxis, así como una semántica inspirada en el formalismo de Redes de Petri. Por esta razón, todos los diagramas de tareas y subactividades los hemos modelado e interpretado de acuerdo con este nuevo tratamiento. En el Apéndice D, se dan a conocer algunas nociones básicas sobre los diagramas de actividad bajo la versión 2 de UML.

Consultando la descripción del patrón PROCESO DE REUNIÓN(2.4.1) podemos comprobar cómo la intención y el contexto de este patrón coinciden con el escenario que pretendemos modelar. Además, si partimos del patrón COORDINADOR(2.3.1) anteriormente aplicado, podemos confirmar que, efectivamente, el patrón PROCESO DE REUNIÓN(2.4.1) se encuentra entre los patrones que pueden aplicarse a continuación de éste.

En la Figura 6.12 mostramos el patrón PROCESO DE REUNIÓN(2.4.1) y, seguidamente, en la Figura 6.13 presentamos el modelo de la tarea cooperativa `ReuniónCoordinaciónSubproyectos` construido a partir de la ligadura siguiente:

```
Reunión->"ReuniónCoordinación"
LíderReunión->"CoordinadorSubproyectos"
SecretarioReunión->"Secretario"
díaHoraReunión->"díaHoraReuniónCS"
Acta->$
```

El patrón PROCESO DE REUNIÓN(2.4.1) incorpora también etiquetas informativas relacionadas con los posibles patrones que pueden aplicarse para el modelado de algunas de las actividades que lo componen. Particularmente, indica que es posible usar el patrón CONVOCATORIA DE REUNIÓN para el modelado de la actividad `ConvocarReuniónCoordinación` y el patrón REUNIÓN(2.4.2) para la actividad `SesiónReuniónCoordinación`.

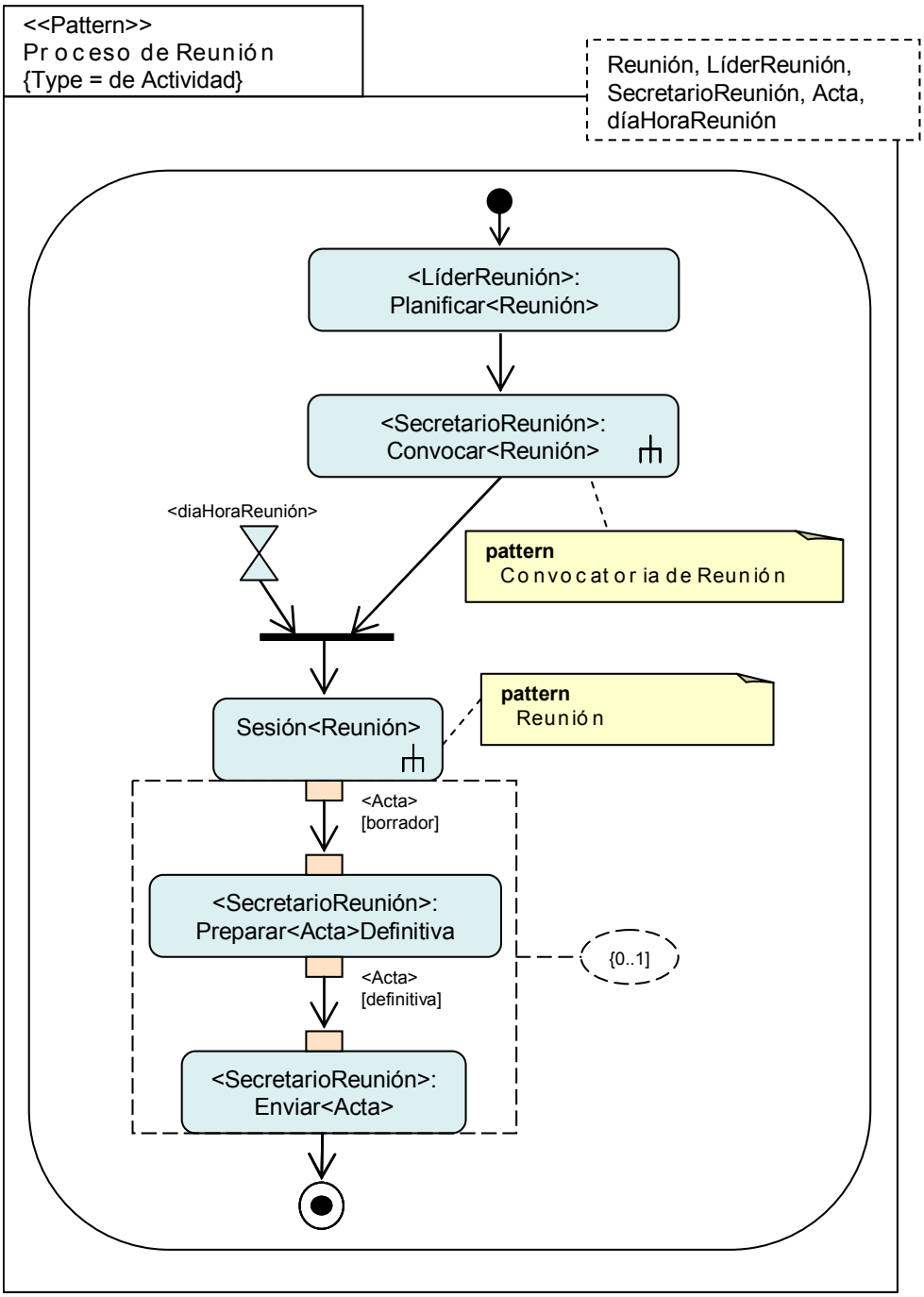


Fig. 6.12: Modelo correspondiente al patrón PROCESO DE REUNIÓN(2.4.1)

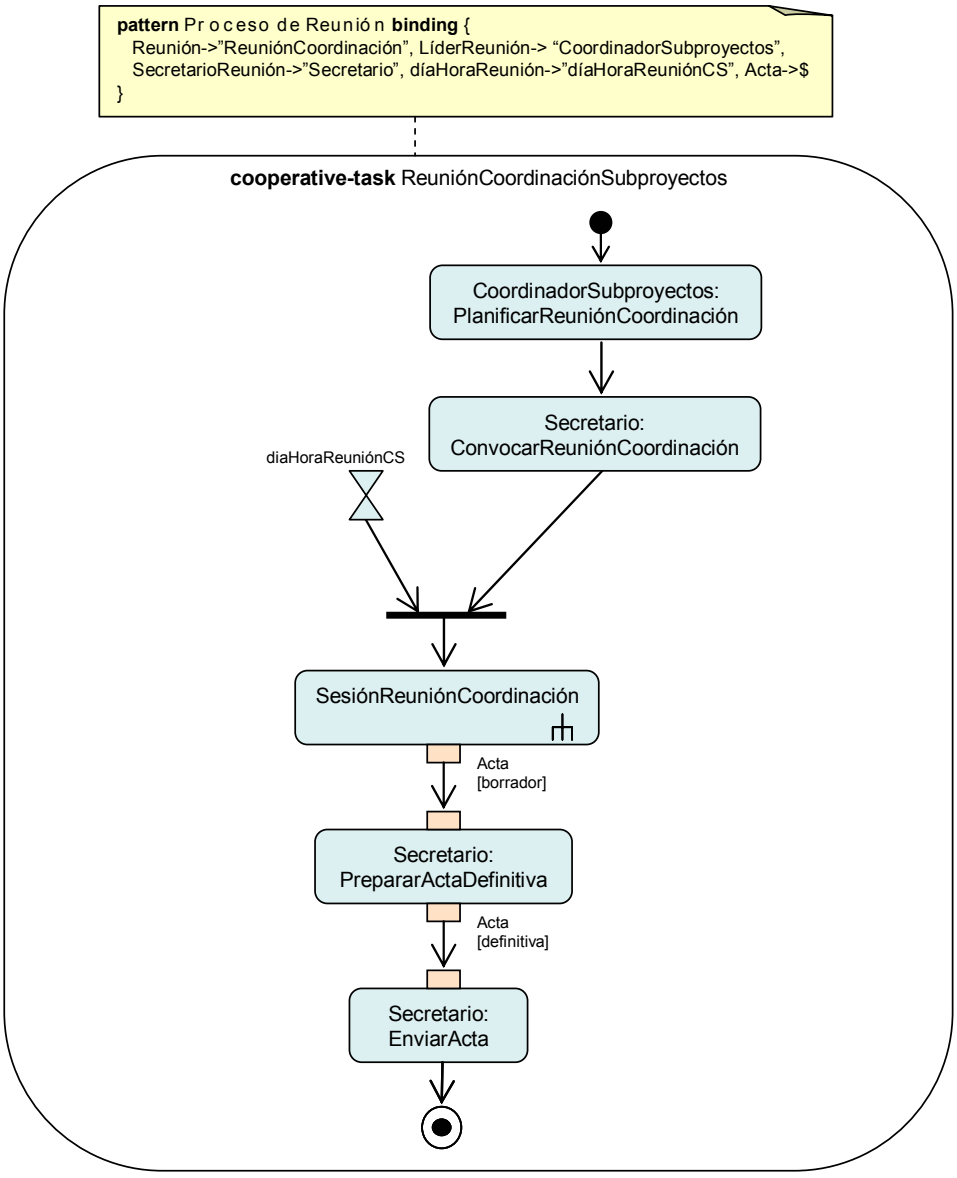


Fig. 6.13: Instancia del patrón PROCESO DE REUNIÓN(2.4.1)

Por su complejidad, y porque es uno de los patrones que tenemos detalladamente descritos en el Apéndice B, estamos especialmente interesados en el despliegue de la actividad `SesiónReuniónCoordinación` con ayuda del patrón REUNIÓN(2.4.2).

Para ello, simplemente, conectamos a la actividad `SesiónReuniónCoordinación` una etiqueta expresando la ligadura necesaria del patrón REUNIÓN(2.4.2), tal y como mostramos en la Figura 6.14.

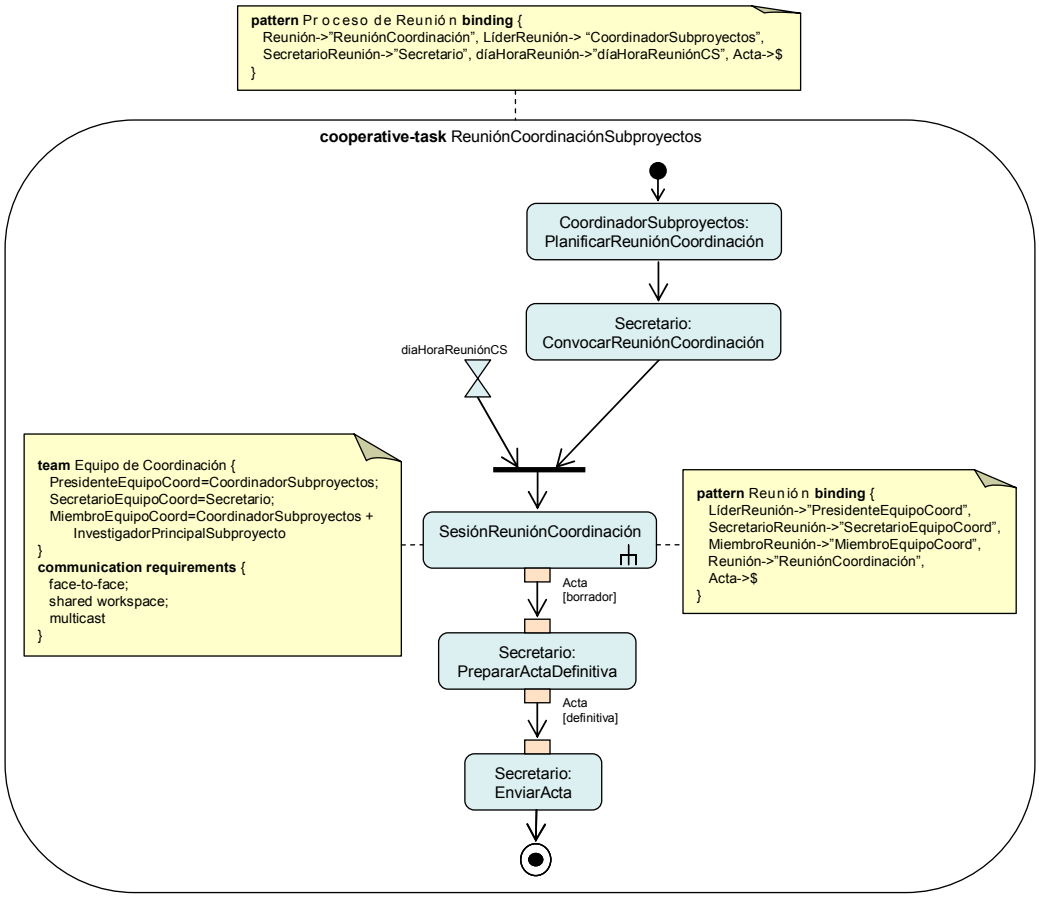


Fig. 6.14: Modelado de la tarea `ReuniónCoordinaciónSubproyectos` de acuerdo con el patrón PROCESO DE REUNIÓN(2.4.1)

Es importante hacer notar que dicha actividad también lleva asociada una etiqueta de equipo, la cual expresa que el grupo de actores encargado de ejecutar esta labor es conocido en la organización como el `Equipo de Coordinación` y que está formado por el `CoordinadorSubproyectos` de la organización, quien juega dentro del equipo el papel de presidente (v. `PresidenteEquipoCoord`), el `Secretario` de la organización, el cual hace también de secretario dentro del equipo (v. `SecretarioEquipoCoord`), y los actores que juegan el rol de `InvestigadorPrincipalSubproyecto` dentro de la organización, que junto al `CoordinadorSubproyectos`, poseen el rol de miembro del equipo (v. `MiembroEquipoCoord`). Además, dentro de esta misma etiqueta, se expresan una serie de requisitos de comunicación para el desarrollo de esta actividad. Éstos revelan que los participantes se comunican cara a cara (v. `face-to-face`), que hacen uso de un espacio de trabajo compartido (v. `shared workspace`) y que los mensajes transmitidos por cada participante llegan a todos los demás (v. `multicast`). Estos requisitos de

comunicación serán heredados por todas las subactividades de la actividad `SesiónReunión Coordinación`.

Gracias al patrón REUNIÓN(2.4.2), cuyo modelo mostramos en la Figura 6.15, y a la ligadura expresada en la Figura 6.14, podemos desplegar fácilmente la actividad `SesiónReunión Coordinación` (Fig. 6.16).

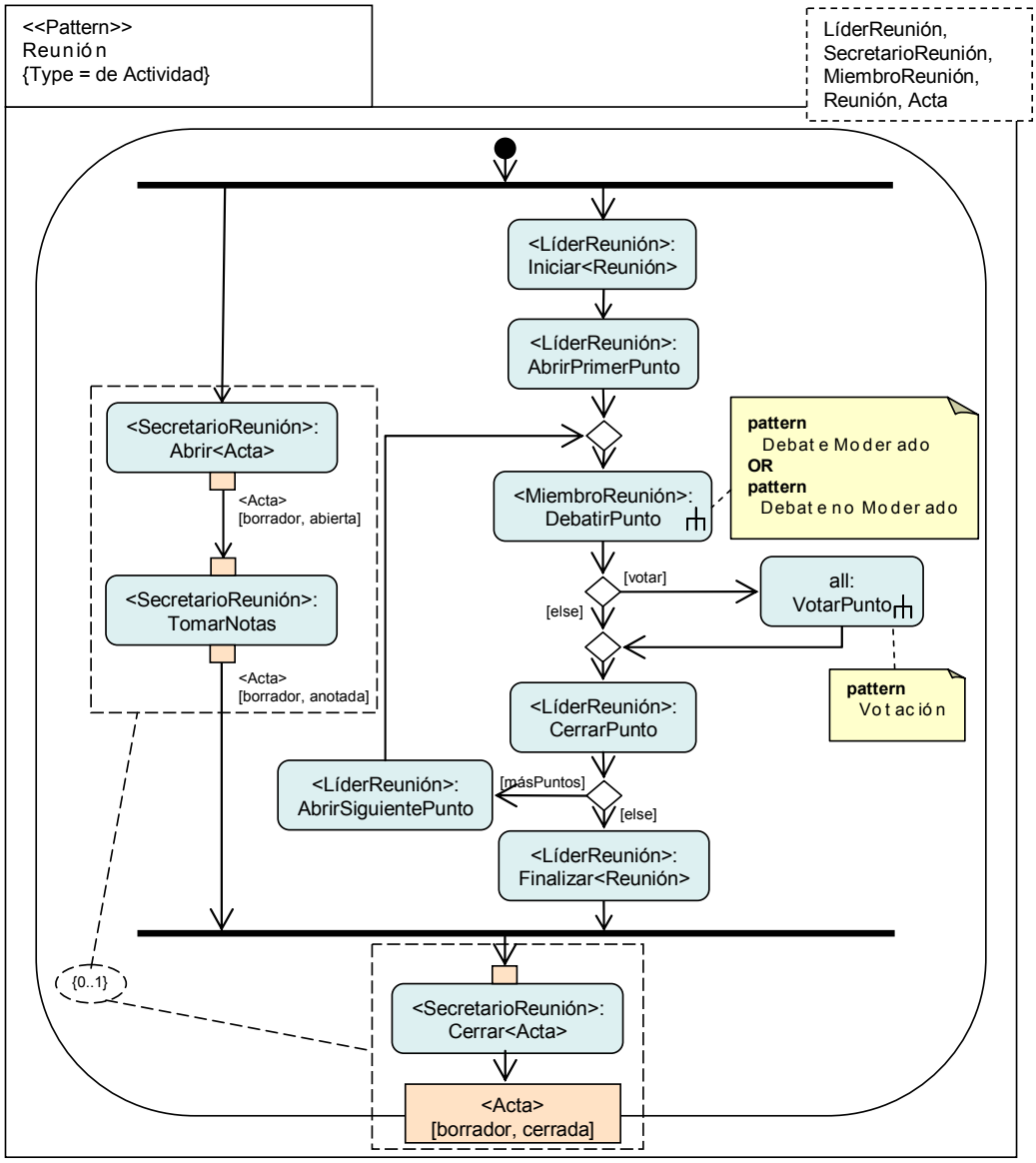


Fig. 6.15: Modelo correspondiente al patrón REUNIÓN(2.4.2)

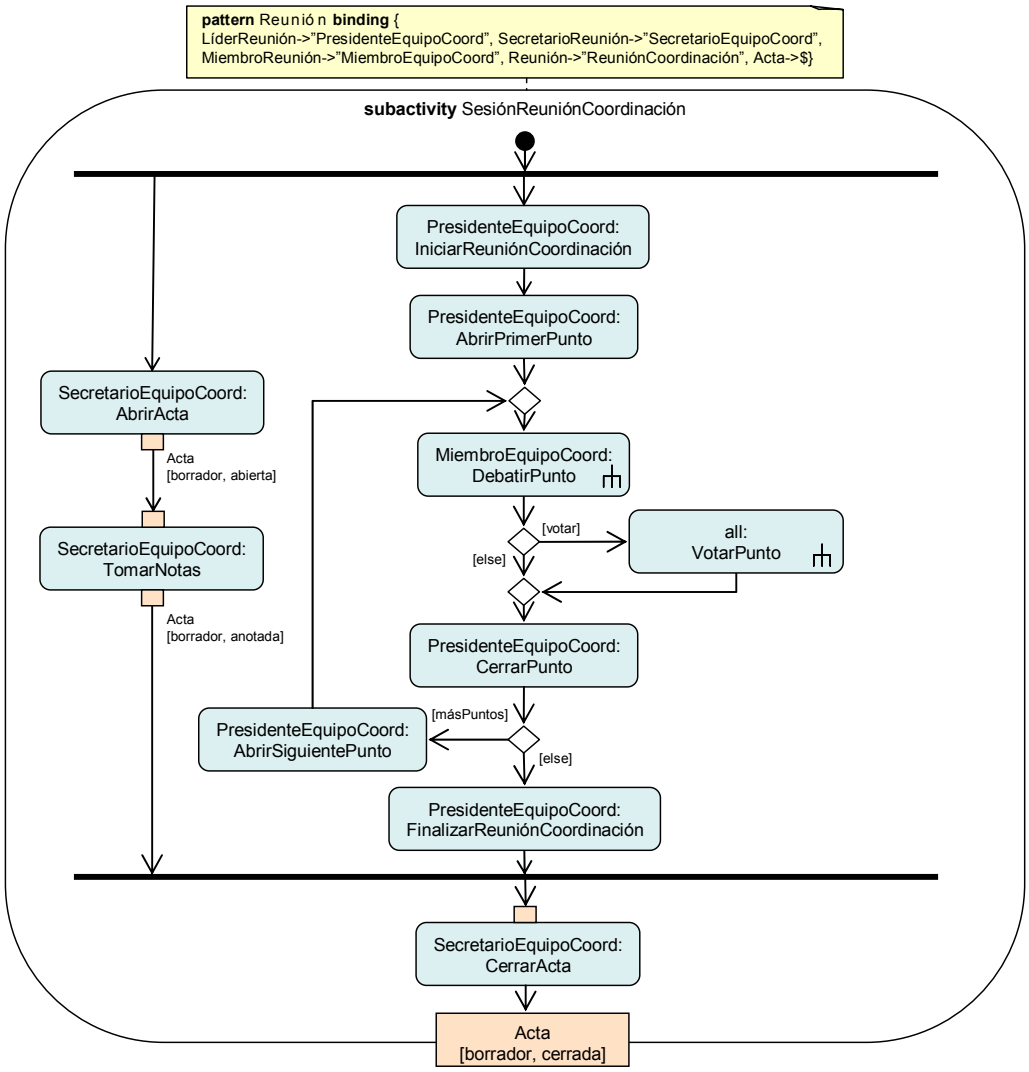


Fig. 6.16: Instancia del patrón REUNIÓN(2.4.2)

No obstante, como podemos apreciar en el modelo correspondiente al patrón REUNIÓN(2.4.2) (Fig. 6.15), éste incluye ciertas etiquetas que informan sobre otros patrones que es posible aplicar para el modelado de algunas de las actividades que lo componen. En concreto, sugiere que podemos emplear el patrón DEBATE MODERADO(2.6.1) o el patrón DEBATE NO MODERADO(2.6.2) para la definición de la actividad `DebatirPunto`. De la misma forma, propone el patrón VOTACIÓN(2.4.3) para la actividad `VotarPunto`.

Así pues, con ayuda de estos patrones vamos a modelar cada una de estas actividades. En la Figura 6.17 aparecen las ligaduras necesarias para ello. Concretamente, `DebatirPunto` la describimos mediante la ligadura dinámica de los patrones de comunicación DEBATE MODERADO(2.6.1) (Fig. 6.18)

y DEBATE NO MODERADO(2.6.2) (Fig. 6.19). Como vimos en el Capítulo IV, una ligadura dinámica es aquella que se produce en tiempo de ejecución, lo que puede dar lugar a ligaduras diferentes en distintos momentos de la ejecución de las actividades. Con esta ligadura pretendemos especificar que el grupo de trabajo puede cambiar en cualquier momento el protocolo de comunicación utilizado para el debate, ya sea dentro de un mismo ciclo (debate de un punto concreto) o entre ciclos distintos (debate de puntos diferentes). De acuerdo con esta ligadura, cuando el debate es moderado quien actúa como `Moderador` es el actor que posee el rol de `PresidenteEquipoCoord` dentro del `Equipo de Coordinación`.

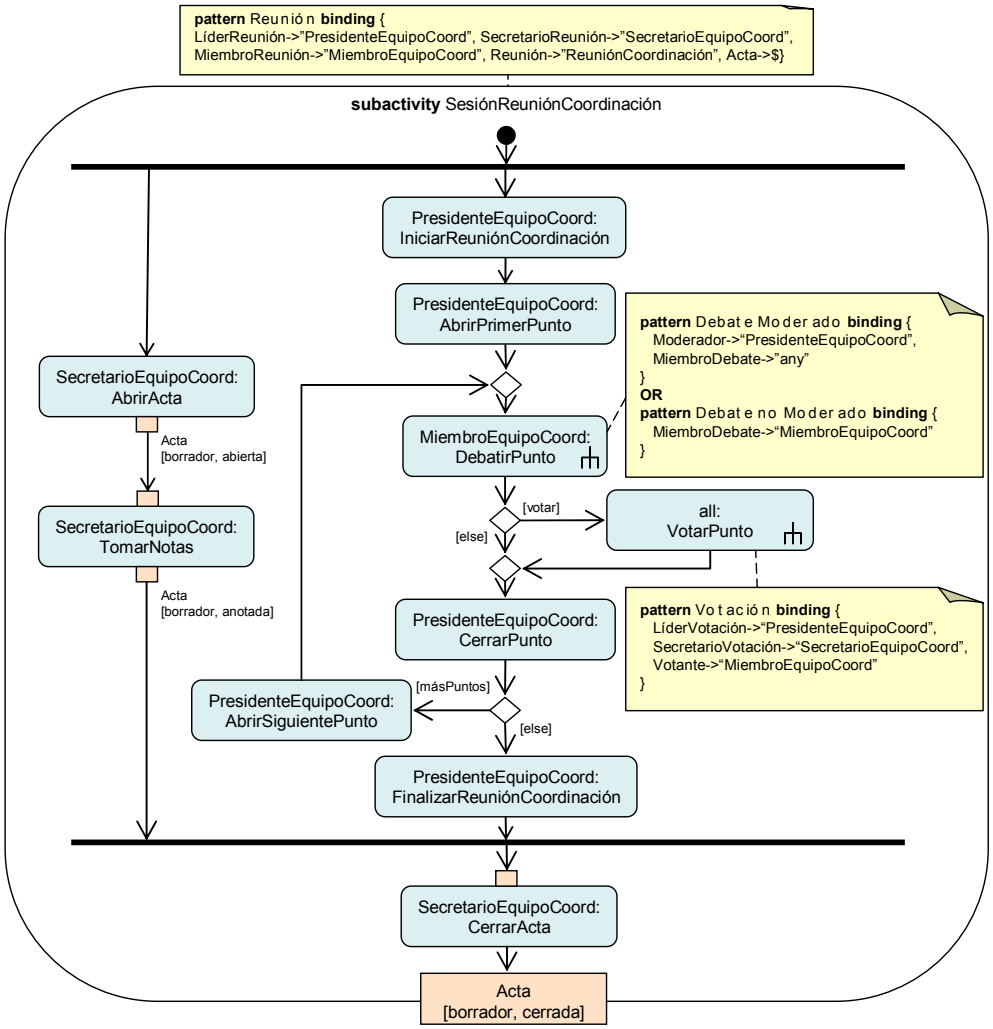


Fig. 6.17: Modelado de la subactividad `SesiónReuniónCoordinación` de acuerdo con el patrón REUNIÓN(2.4.2)

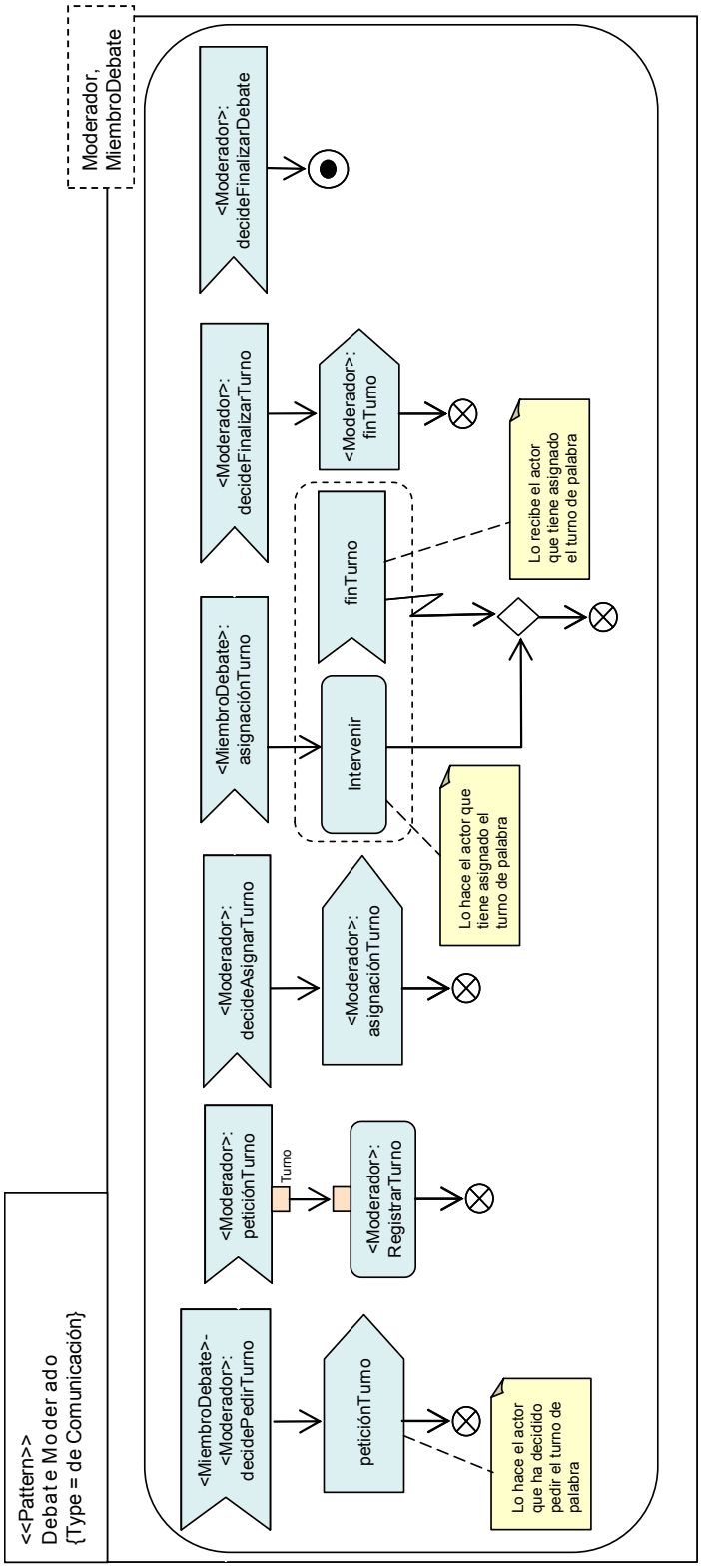


Fig. 6.18: Modelo correspondiente al patrón DEBATE MODERADO(2.6.1)

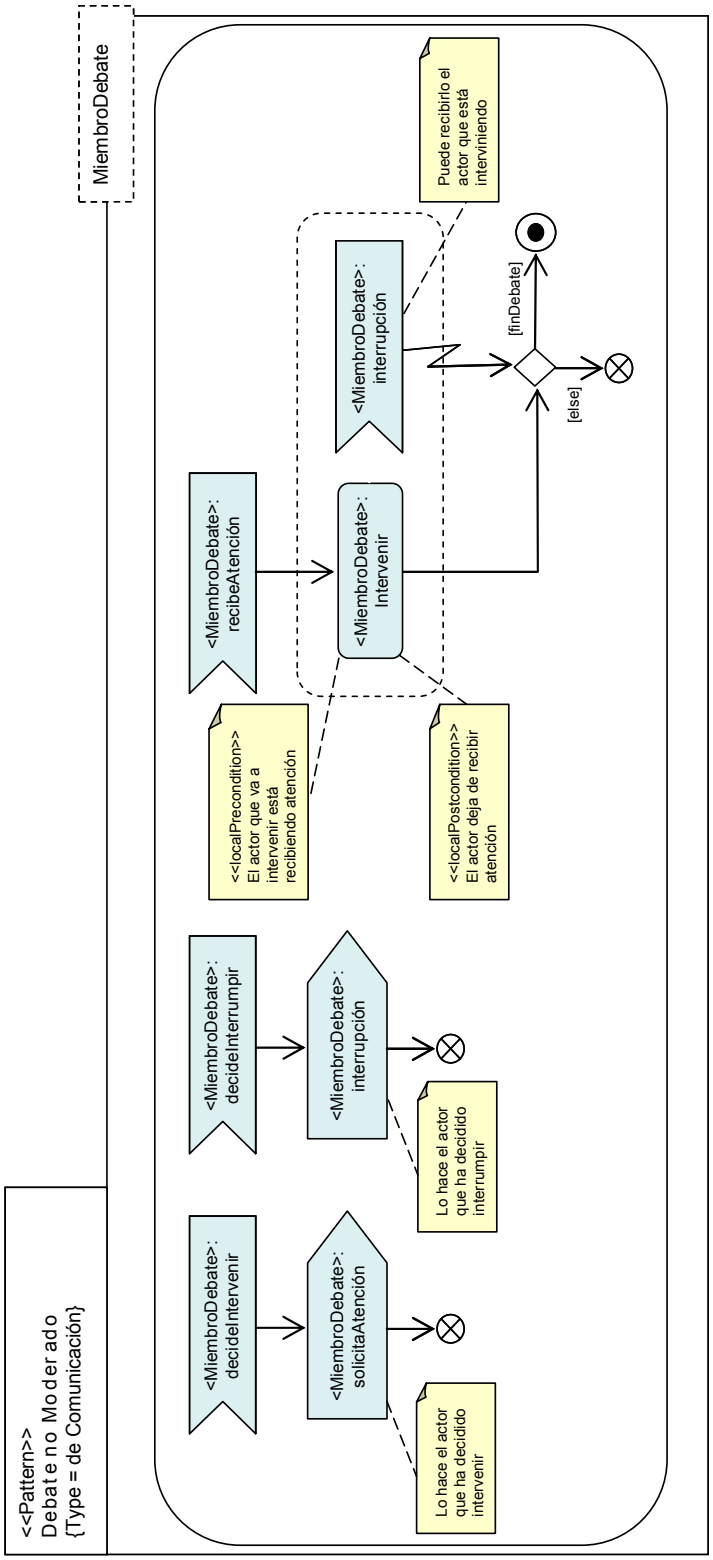


Fig. 6.19: Modelo correspondiente al patrón DEBATE NO MODERADO(2.6.2)

En cuanto a la actividad `VotarPunto`, ésta la definimos a partir del patrón VOTACIÓN(2.4.3) (Fig. 6.20) mediante la siguiente ligadura:

```
LíderVotación->"PresidenteEquipoCoord"

SecretarioVotación->"SecretarioEquipoCoord"

Votante->"MiembroEquipoCoord"
```

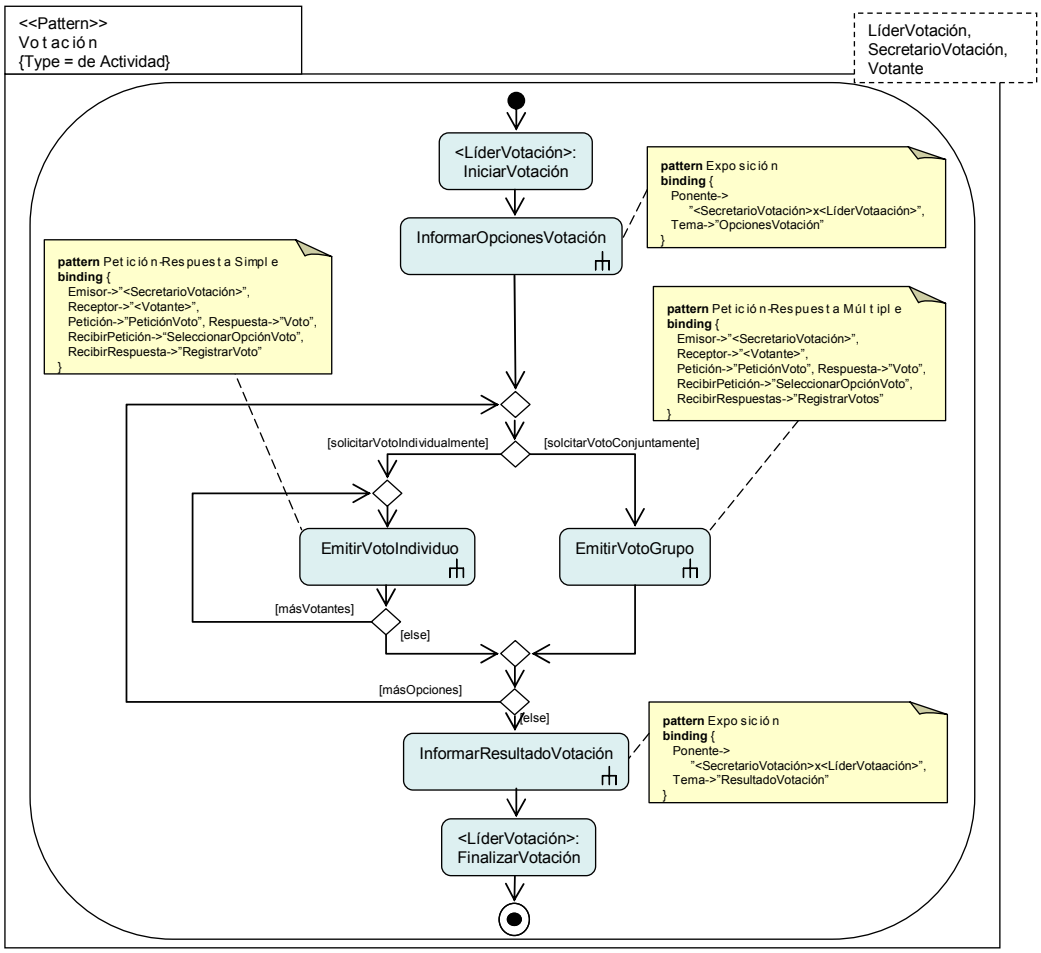


Fig. 6.20: Modelo correspondiente al patrón VOTACIÓN(2.4.3)

Como disponemos en el catálogo de la descripción de los patrones VOTACIÓN(2.4.3), DEBATE MODERADO(2.6.1) y DEBATE NO MODERADO(2.6.2), la especificación de estas etiquetas de ligadura sería, en este caso, suficiente para que el modelo correspondiente a la subactividad `SesiónReuniónCoordinación` (Fig. 6.17) quede completamente definido.

Esto es así debido a que el modelado de estas actividades no incluye ningún elemento adicional, más allá de los que ya provee el patrón. No obstante, presentamos en la Figura 6.21 el despliegue de la subactividad `VotarPunto` en base al patrón VOTACIÓN(2.4.3), en la Figura 6.22 el modelado de la subactividad `DebatirPunto` según el patrón DEBATE MODERADO(2.6.1) y, por último, en la Figura 6.23 el correspondiente a esta misma subactividad de acuerdo con el patrón DEBATE NO MODERADO(2.6.2).

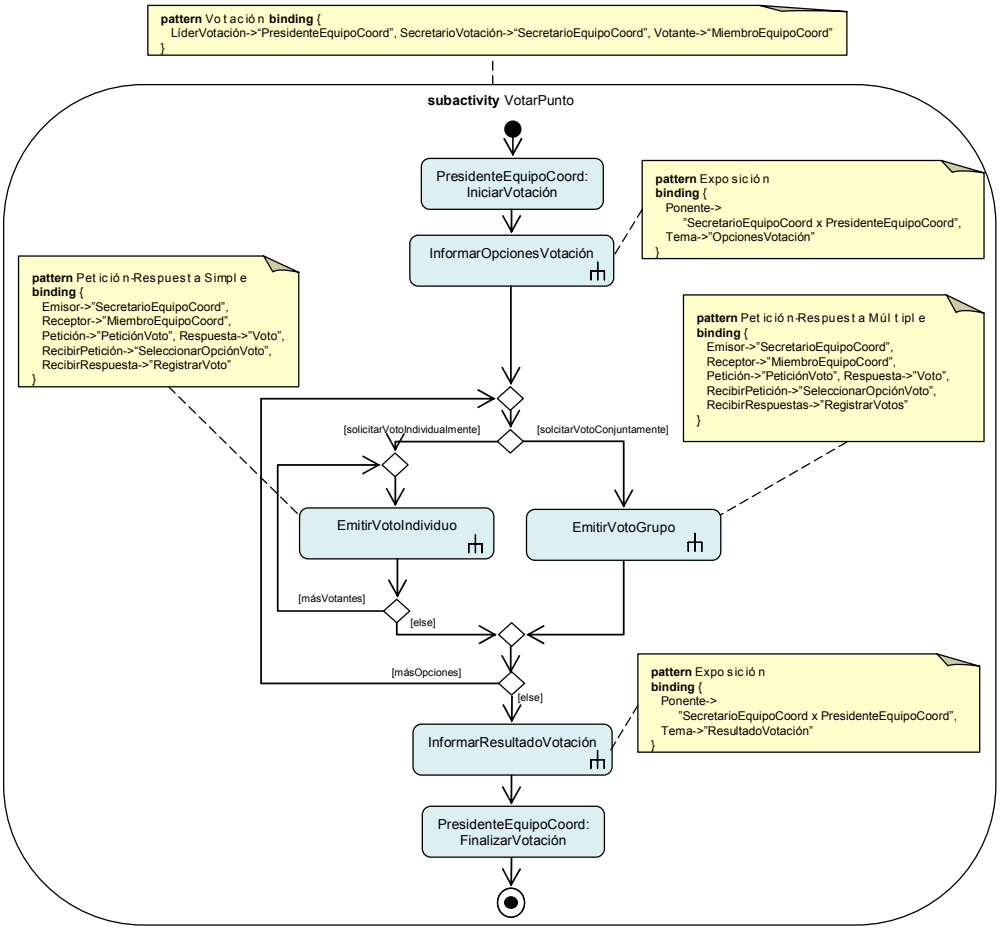


Fig. 6.21: Modelado de la subactividad `VotarPunto` de acuerdo con el patrón VOTACIÓN(2.4.3)

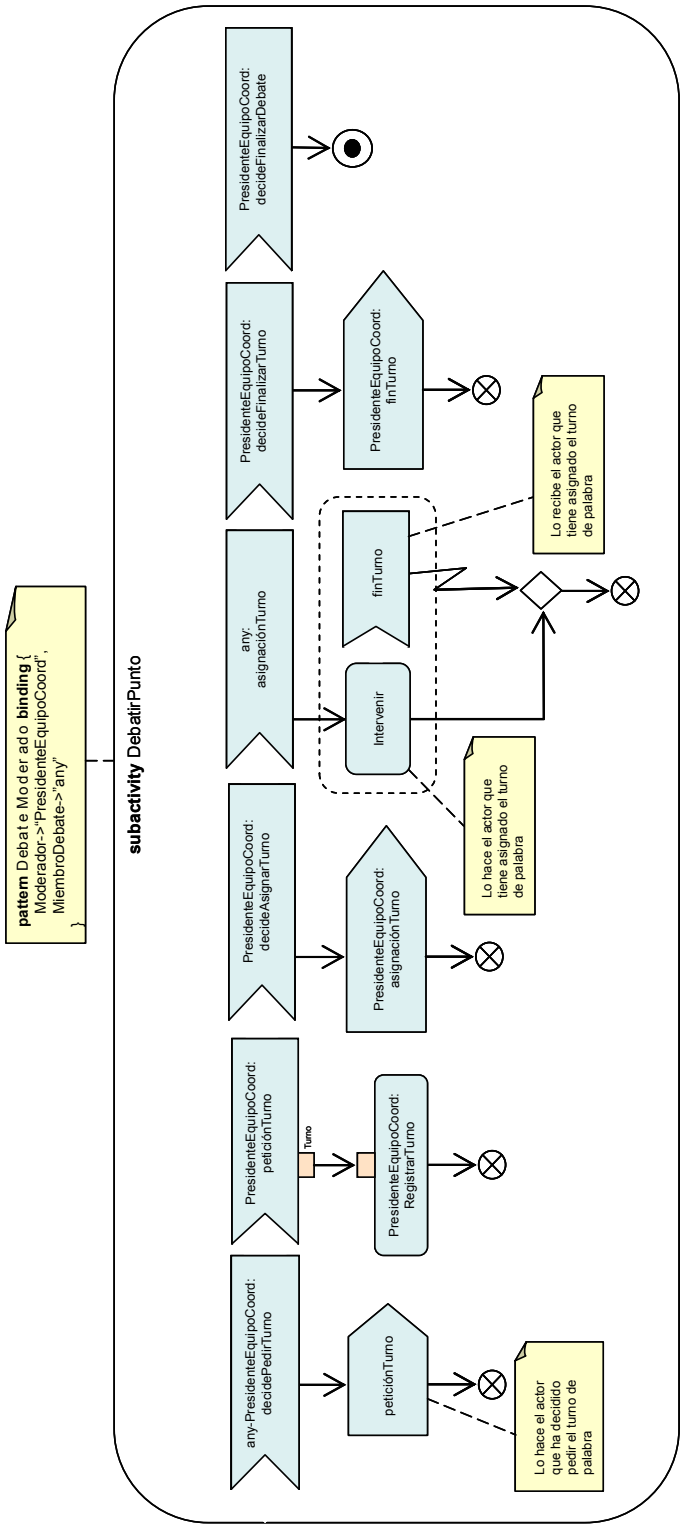


Fig. 6.22: Modelado de la subactividad `DebatirPunto` de acuerdo con el patrón DEBATE MODERADO(2.6.1)

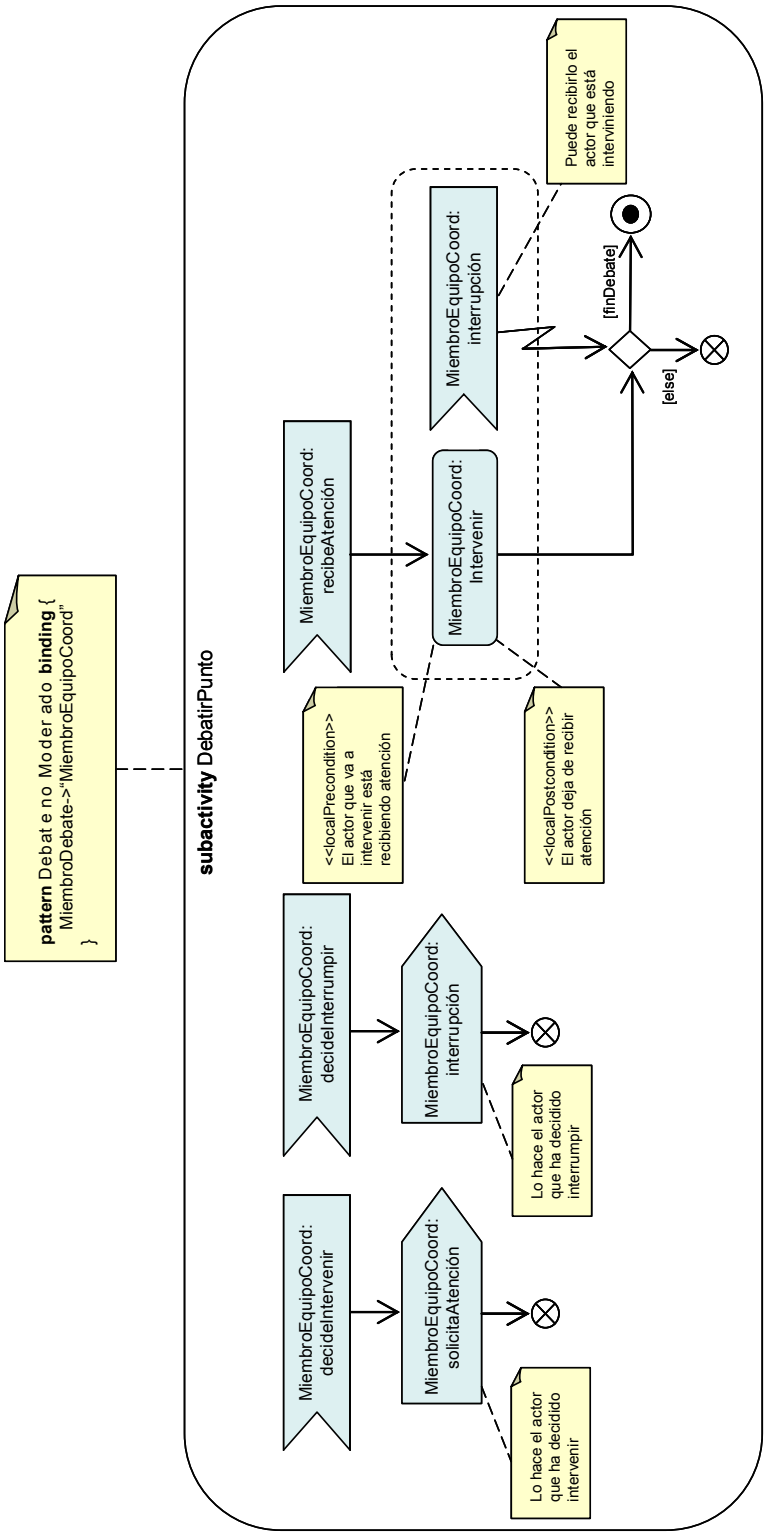


Fig. 6.23: Modelado de la subactividad `DebatirPunto` de acuerdo con el patrón DEBATE NO MODERADO(2.6.2)

2.2.4. Especificación del modelo conceptual de datos

En AMENITIES la información que se comparte/comunica dentro de los escenarios de trabajo es representada, una veces, en forma de flujo de información entre actividades/acciones (nodos de objeto en los diagramas de actividad), y otras, se asume implícita a éstas. Sin embargo, el conocimiento que tenemos sobre el dominio del problema mejora bastante si detallamos, a nivel conceptual, la estructura de la información manejada.

En nuestro ejemplo, hay un objeto protagonista en varias de las actividades que hemos modelado, el cual es utilizado por el actor que desempeña la labor de `Secretario` para registrar los acontecimientos relevantes que suceden durante las sesiones de reunión de coordinación. Sin duda nos estamos refiriendo al objeto `Acta`. Teniendo en cuenta el conjunto de patrones de estructura que actualmente existen en el catálogo, parece obvio que el candidato idóneo para este caso es el patrón ACTA DE REUNIÓN(2.7.1) (Fig. 6.24). Así, por ejemplo, la ligadura de este patrón a un paquete nos permite expresar que su contenido es el modelo conceptual de datos correspondiente al objeto `Acta`, generado a partir del patrón (Fig. 6.25).

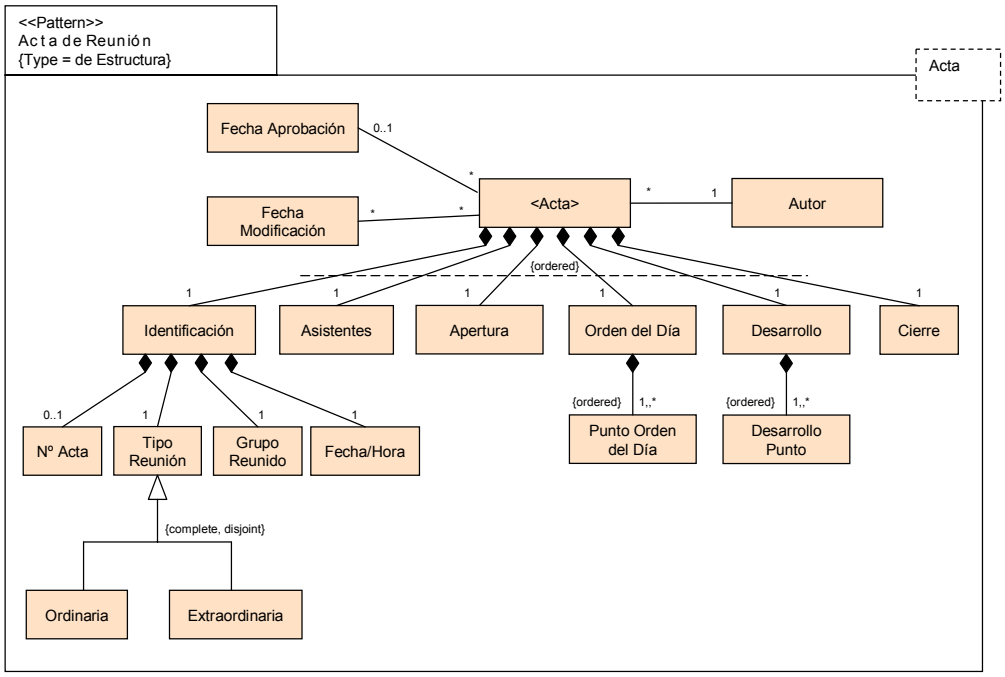


Fig. 6.24: Modelo correspondiente al patrón ACTA DE REUNIÓN(2.7.1)

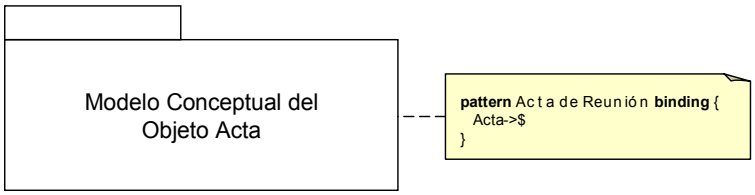


Fig. 6.25: Paquete que incluye el modelo conceptual de datos del objeto `Acta` generado a partir del patrón ACTA DE REUNIÓN(2.7.1)

3. Sistema para el Aprendizaje Cooperativo usando Jigsaw

3.1. Introducción

Con la aparición del aprendizaje asistido mediante TICs[6], también conocido como *e-Learning*, la construcción de herramientas informáticas para el soporte del aprendizaje cooperativo se ha convertido en un tópico de especial interés entre investigadores y profesionales procedentes de diversas disciplinas, entre las que se incluyen: pedagogía, psicología, sociología o informática. Esto ha dado lugar a un nuevo campo de estudio denominado *CSCL* (Computer Supported Collaborative Learning).

El aprendizaje cooperativo hace referencia a métodos instruccionales [Sharan, 1994] donde grupos pequeños de estudiantes, supervisados por un profesor, colaboran para alcanzar metas comunes (la resolución de un ejercicio, la comprensión de un tema, la realización de un proyecto, etc.), teniendo en cuenta que el éxito de los compañeros es tan importante como el éxito propio. El aprendizaje se adquiere a través de la puesta en práctica de métodos de trabajo en grupo. Se caracteriza por la interacción y el aporte de todos en la construcción del conocimiento. Esta forma de aprendizaje fomenta la interdependencia positiva, la responsabilidad individual y colectiva, las habilidades de comunicación e interacción con los demás, la crítica constructiva, etc.

Aunque algunos autores tienden a no diferenciarlos, se suele hacer distinción entre el paradigma de aprendizaje colaborativo y el cooperativo. En

[6] Tecnologías de la Información y la Comunicación

el caso del aprendizaje colaborativo, cada uno de los miembros del grupo interviene en todas y cada una de las partes en las que se divide el proyecto o problema, teniendo que coordinarse con los demás para resolver cada parte. Los alumnos son quienes diseñan la estructura de las interacciones y mantienen el control sobre las diferentes decisiones que repercuten en su aprendizaje. En cambio, en el caso del aprendizaje cooperativo, cada participante tiene asignada una tarea concreta de la cual es responsable, realizando un trabajo más individual dentro del proyecto común. Es el profesor quien diseña y mantiene, casi por completo, el control en la estructura de las interacciones y de los resultados que se han de obtener.

Más allá de las consideraciones pedagógicas, desde un punto de vista tecnológico, las herramientas necesarias para dar soporte a ambos tipos de aprendizaje son, a menudo, las mismas, con lo que raramente se hace distinción entre herramientas para aprendizaje colaborativo y cooperativo. En nuestro caso, hemos optado por usar el término “cooperativo”.

Básicamente, la puesta en práctica del aprendizaje cooperativo consiste en implementar en el aula las siguientes pautas de trabajo:

1) Se distribuyen a los alumnos en grupos pequeños y heterogéneos (distinto sexo, rendimiento académico, raza, grupo social, etc.).

2) Se asignan roles bien definidos a los componentes de cada grupo (sería interesante que estos roles fuesen rotatorios).

3) Se elige la tarea que debe realizar cada uno de los grupos (debería facilitar la colaboración y evitar el individualismo o el desequilibrio de cargas de trabajo entre sus miembros).

4) El profesor supervisa el progreso del grupo en todo momento e interviene sólo cuando es necesario. Desempeña el papel de guía/facilitador del aprendizaje.

5) El resultado del trabajo es compartido con el resto de estudiantes.

6) Los alumnos se autoevalúan y forman parte del proceso de evaluación habitualmente.

El sistema concreto que vamos a modelar tiene por objetivo servir de apoyo al aprendizaje cooperativo mediante la implementación de una estrategia basada en el método conocido como *Jigsaw* (Rompecabezas) [Aronson et al., 1978; Aronson, 2000], muy extendido en el ámbito pedagógico. Aunque se han creado varias versiones del método original [Kagan, 1989; Slavin, 1986], esencialmente comprende las siguientes fases:

1) Se realiza una distribución de los alumnos de la clase en grupos pequeños con la misma cantidad de miembros (idealmente de 5 a 6 personas).

2) Se divide el tema de trabajo (ejercicio, lectura material, etc.) en tantas partes como miembros haya en cada grupo.

3) Cada miembro del grupo elige, o el profesor asigna, una parte diferente del tema común de trabajo.

4) Una vez que cada alumno ha preparado su parte, éstos se reúnen con los compañeros de otros grupos que tienen asignada esa misma parte, creando "grupos de expertos" temporales para discutirla, mejorarla y estudiarla en profundidad. A continuación, se les proporciona tiempo suficiente para que cada cual actualice y prepare su fragmento con el objetivo de presentarlo a sus compañeros de grupo.

5) Cuando vuelve a su grupo de origen, cada estudiante expone a sus compañeros lo que ha aprendido sobre el tema (una de las partes del rompecabezas) y aclara todas aquellas dudas que surjan dentro del grupo.

6) Al final del tiempo asignado los alumnos son examinados de todas las partes que el tema comprende. Se realiza una evaluación individual, o bien un portavoz del grupo presenta a toda la clase el tema en su totalidad, recibiendo todos sus miembros la misma calificación.

3.2. Construcción del Modelo Cooperativo de AMENITIES

3.2.1. Especificación de la organización

Los roles que pueden jugar los actores involucrados en una experiencia de aprendizaje cooperativo de este tipo son:

- `Alumno`: Entre sus objetivos principales están el estudio y la realización de las tareas Jigsaw que el profesor demanda.

- `Profesor`: Planifica las tareas Jigsaw que deben hacer los alumnos e interviene en dichas tareas facilitando su realización y evaluando a los alumnos.

- `Administrador`: Se encarga de gestionar los alumnos y los recursos necesarios para poder llevar a cabo la experiencia.

En la Figura 6.26 exhibimos el diagrama de organización (v. `AulaJigsaw`) que representa a los distintos roles involucrados, así como la dinámica de los actores desde el punto de vista de los posibles cambios de rol que éstos pueden experimentar, dependiendo de sus capacidades o normas de la organización.

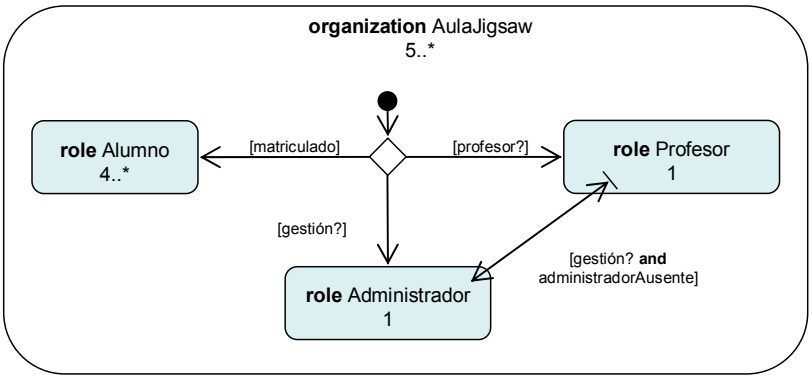


Fig. 6.26: Diagrama de organización asociado al sistema para el aprendizaje cooperativo

El diagrama revela que para que un actor pueda participar en una experiencia de aprendizaje de este tipo como `Alumno` es necesario que esté matriculado en un curso (v. condición `[matriculado]`). Aunque según los especialistas lo ideal es que haya del orden de 5 ó 6 alumnos por grupo,

pensamos que el número mínimo de éstos para poder poner en práctica esta estrategia es de cuatro (v. multiplicidad de rol `4..*`), o lo que es lo mismo, dos grupos de dos alumnos. También nos muestra que para que un actor pueda participar como `Profesor` debe tener capacidad para ello (v. capacidad `[profesor?]`). El número de profesores necesario en este caso es de uno (v. multiplicidad de rol `1`). Respecto al rol de `Administrador` el modelo refleja que para que un actor pueda desempeñarlo debe tener capacidad para gestionar los alumnos y los recursos necesarios (v. capacidad `[gestión?]`).

En cuanto a los posibles cambios de rol que pueden producirse en la organización, podemos ver que cuando el `Administrador` no se encuentra disponible y el `Profesor` cuenta con capacidad suficiente como para realizar las labores de gestión necesarias (v. condición `[gestión? and administradorAusente]`), éste último puede sustituir al primero, sin abandonar su cargo como `Profesor`, mientras se siga dando la condición que ha provocado este cambio[7].

Para la construcción de este sencillo diagrama de organización no hemos necesitado ningún patrón del catálogo.

3.2.2. Especificación de los roles

Los diagramas de rol que especifican las tareas (objetivos generales) asociadas a cada uno de los roles que acabamos de mencionar, así como las condiciones que dan lugar a su activación se muestran en la Figura 6.27.

Las tareas que aparecen se corresponden ostensiblemente con los objetivos indicados para cada uno de los roles en la sección anterior.

Entre todas las tareas la única que es cooperativa es `RealizarJigsaw`, en la que participa un actor con el rol de `Profesor` y al menos cuatro con el rol de `Alumno`.

También podemos observar que la única tarea cuya ejecución está sujeta a la aparición de un determinado evento es la de `RealizarJigsaw` por

[7] Téngase en cuenta que el cambio de rol se ha representado como una transición aditiva

parte del rol `Alumno`. Ésta tarea no es emprendida hasta que el `Profesor` lo señala (v. evento `PeticiónProfesor`).

Respecto a la posibilidad de interrupción, la única tarea que no puede ser interrumpida bajo ninguno de los roles que cooperan en su realización es `RealizarJigsaw`. El resto de tareas pueden ser interrumpidas por cualquier otra.

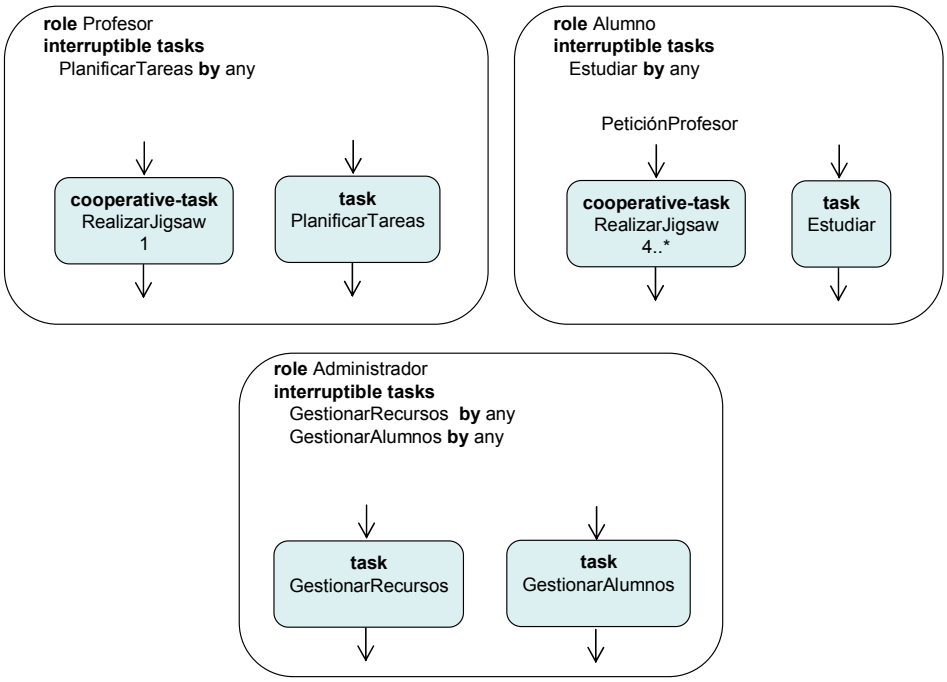


Fig. 6.27: Diagramas de rol asociados al sistema para el aprendizaje cooperativo

3.2.3. Especificación de las tareas

Por simplicidad, en esta fase atendemos sólo la especificación de la única tarea cooperativa (la tarea `RealizarJigsaw`) común a los roles `Profesor` y `Alumno`. Teniendo en cuenta las fases[8] que comprende la puesta en práctica de una sesión de Jigsaw, modelamos en la Figura 6.28 esta tarea.

[8] Véase la sección 3.1 de este mismo capítulo

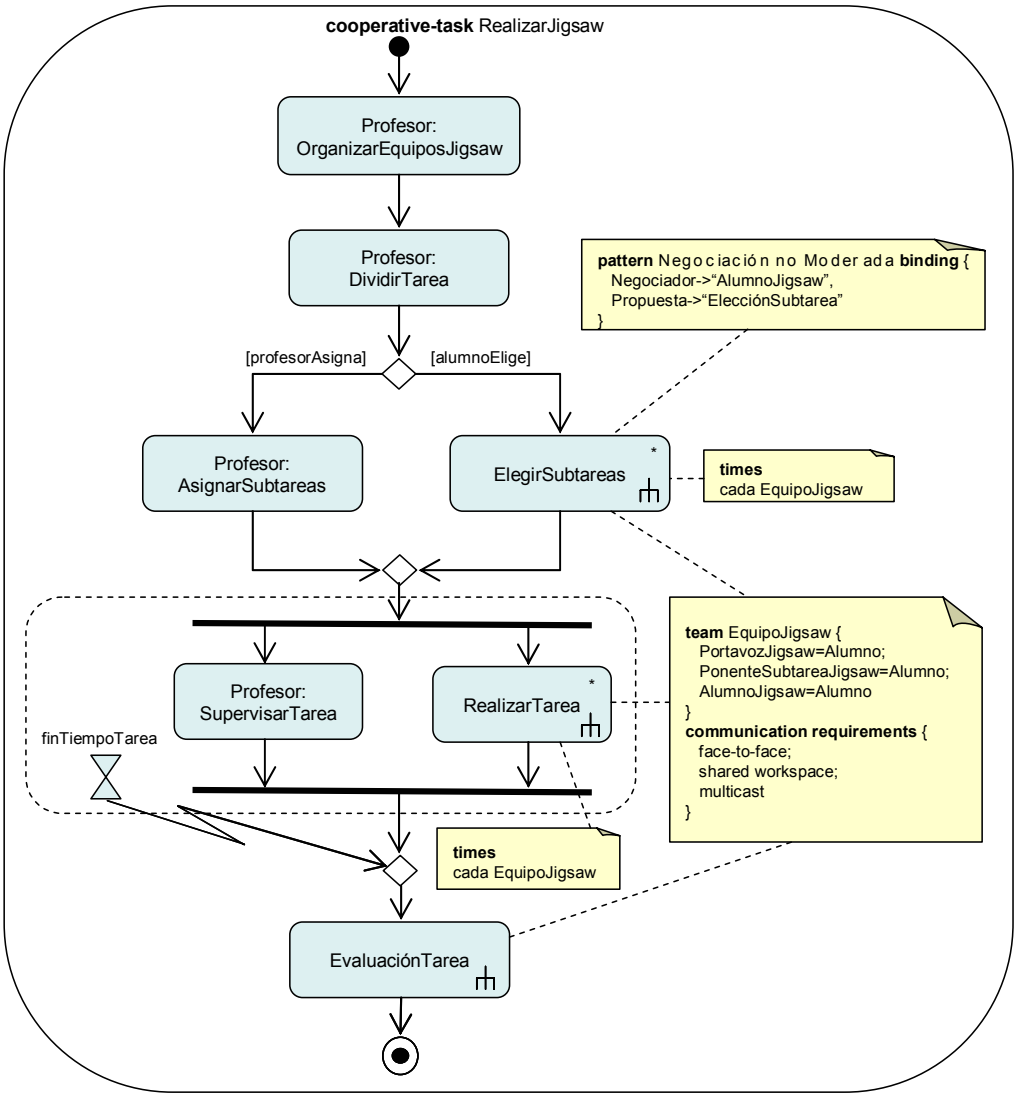


Fig. 6.28: Modelado de la tarea cooperativa `RealizarJigsaw`

El `Profesor` comienza esta tarea distribuyendo a los alumnos de la clase en grupos pequeños con la misma cantidad de miembros (v. acción `OrganizarEquiposJigsaw`). Seguidamente, el `Profesor` divide la tarea a realizar en tantas subtareas como miembros haya en cada grupo (v. acción `DividirTarea`). A continuación, el `Profesor` reparte las tareas (v. condición `[profesorAsigna]`) entre los miembros de cada equipo (v. acción `AsignarSubtareas`) o son los propios alumnos (v. condición `[alumnoElige]`) de cada equipo (v. cláusula `times cada EquipoJigsaw`) quienes se reparten las distintas subtareas (v. actividad paralela `ElegirSubtareas`). Para ello, los miembros de cada equipo tendrán que ponerse de acuerdo y negociar su reparto. Como destacamos en el diagrama, para la especificación de esta

tarea hemos seleccionado el patrón NEGOCIACIÓN NO MODERADA(2.4.4) (Fig. 6.29), cuya descripción se ajusta a las necesidades de modelado de dicha tarea. Para ello, expresamos una ligadura donde el parámetro `Negociador` asume el valor "`AlumnoJigsaw`" y `Propuesta` recibe "`ElecciónSubtarea`". En la Figura 6.30 desplegamos dicha subactividad en base a este patrón.

En el diagrama se expresa además que la realización de la actividad `ElegirSubtareas` es llevada a cabo de forma paralela por distintos equipos (v. etiqueta `team EquipoJigsaw`), formados por alumnos organizados en base a tres roles: `PortavozJigsaw` (actuando como representante del equipo), `PonenteSubtarea` (responsable de exponer una determinada subtarea al resto del equipo) y `AlumnoJigsaw` (rol que posee todo miembro del equipo). Los requisitos de comunicación (v. `communication requirements`) necesarios para realizar esta actividad se expresan dentro de esta misma etiqueta. Éstos revelan que los participantes se comunican cara a cara (v. `face-to-face`), que hacen uso de un espacio de trabajo compartido (v. `shared workspace`) y que los mensajes transmitidos por cada participante llegan a todos los demás (v. `multicast`).

Una vez que las subtareas están asignadas entre los miembros de cada equipo, comienza la realización de la tarea en sí (v. actividad paralela `RealizarTarea`) por parte de cada uno de estos equipos (v. cláusula `times cada EquipoJigsaw`). Obsérvese que la etiqueta de equipo también se conecta con la actividad `RealizarTarea`, indicando que ésta es realizada por un equipo de la clase `EquipoJigsaw`. Mientras que cada equipo ejecuta la actividad `RealizarTarea`, el `Profesor` supervisa en todo momento su desarrollo (v. acción `SupervisarTarea`). En cuanto finaliza el tiempo previsto para la realización de dicha tarea (v. acción de aceptación del evento de tiempo `finTiempoTarea`) se interrumpen las actividades (v. región de interrupción de actividad) y el flujo de control se pasa a la siguiente actividad, durante la cual los alumnos de cada equipo son evaluados (v. actividad `EvaluaciónTarea`).

Tal y como hemos apuntado, el modelo correspondiente a la subactividad `ElegirSubtareas` (Fig. 6.30) se construye en base al patrón

NEGOCIACIÓN NO MODERADA(2.4.4) (Fig. 6.29). Conociendo el patrón[9] y la ligadura que hemos realizado, en teoría no es necesario añadir ningún otro comentario para poder comprender rápidamente este modelo. Como fijamos en el Capítulo III, entre las ventajas de la aplicación de patrones está la reutilización del conocimiento y el establecimiento de un vocabulario común que favorece su comunicación.

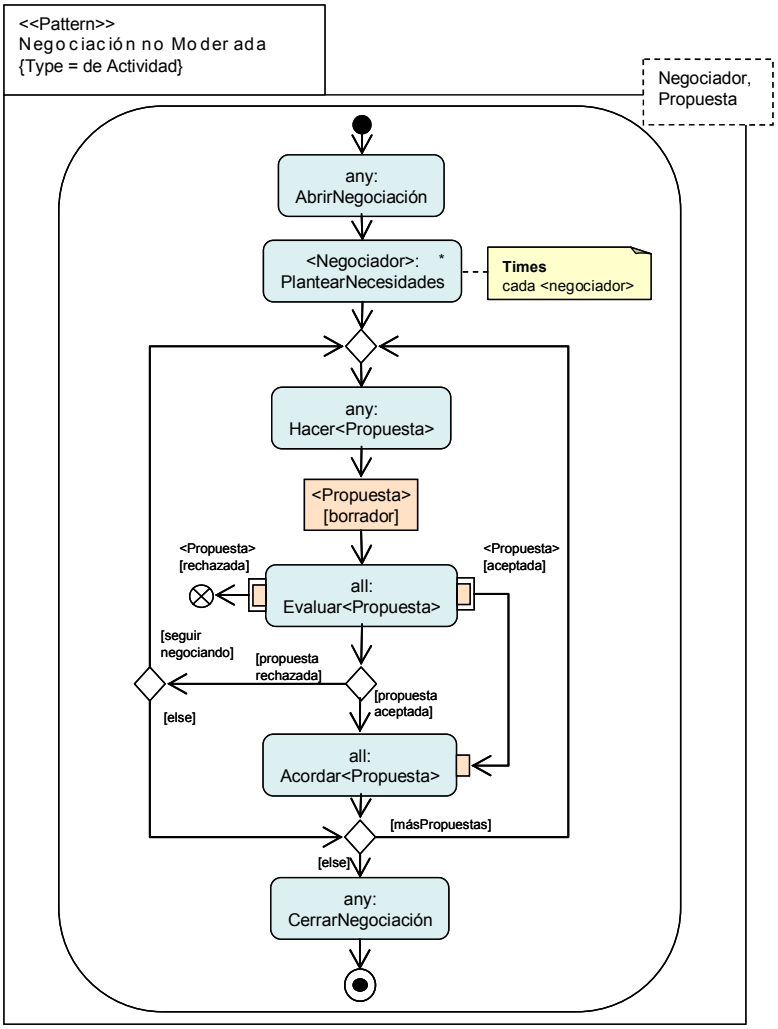


Fig. 6.29: Modelo correspondiente al patrón NEGOCIACIÓN NO MODERADA(2.4.4)

[9] En caso de no conocer este patrón y con objeto de obtener información detallada acerca de éste, recomendamos encarecidamente al lector que consulte la sección 2.4.4 del Apéndice B.

La siguiente actividad que vamos a modelar es `RealizarTarea` (Fig. 6.31). Partimos del hecho de que ésta es realizada por todos y cada uno de los equipos que intervienen en el Jigsaw (v. etiqueta `times cada EquipoJigsaw`) y que los requisitos de comunicación especificados son heredados por todas sus subactividades.

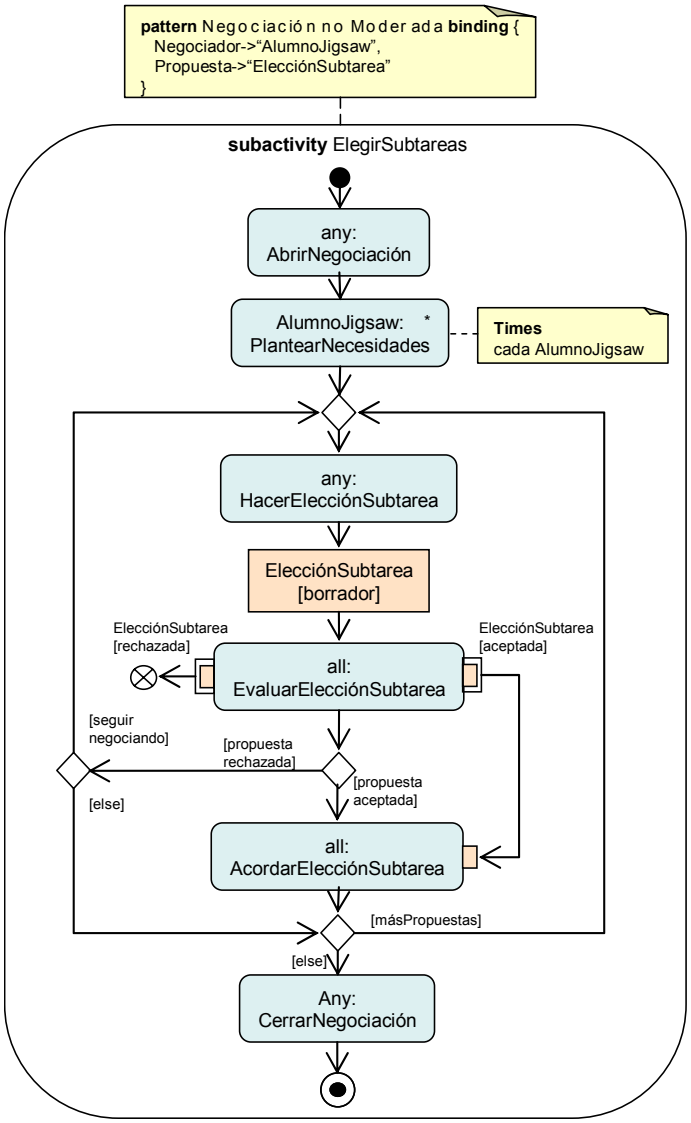


Fig. 6.30: Modelado de la subactividad `ElegirSubtareas` de acuerdo con el patrón NEGOCIACIÓN NO MODERADA(2.4.4)

En primer lugar cada uno de los alumnos que forman parte del equipo (`times cada AlumnoJigsaw`) realiza la subtarea que tiene asignada (v. acción paralela `RealizarSubtarea`). En el momento en que se agota el tiempo previsto para la realización de la subtarea (v. acción de aceptación del evento de tiempo `finTiempoSubtarea`) se interrumpe la actividad (v. región de

interrupción de actividad) y el flujo de control pasa a la siguiente acción (v. acción `OrganizarEquiposExpertos`), en la que el `Profesor` organiza equipos compuestos por miembros de los distintos grupos que tienen asignada esa misma subtarea, creando así "grupos de expertos" temporales para discutirla, estudiarla y, a ser posible, mejorarla. Inmediatamente, cada uno de los equipos de expertos creados (v. `times cada EquipoExpertos`) debaten la subtarea en la que están especializados (v. actividad `DebatirSubtarea`) hasta que el `Profesor` decide finalizar el debate (v. acción de recepción de evento `decideTerminarDebate`) interrumpiendo dicha actividad.

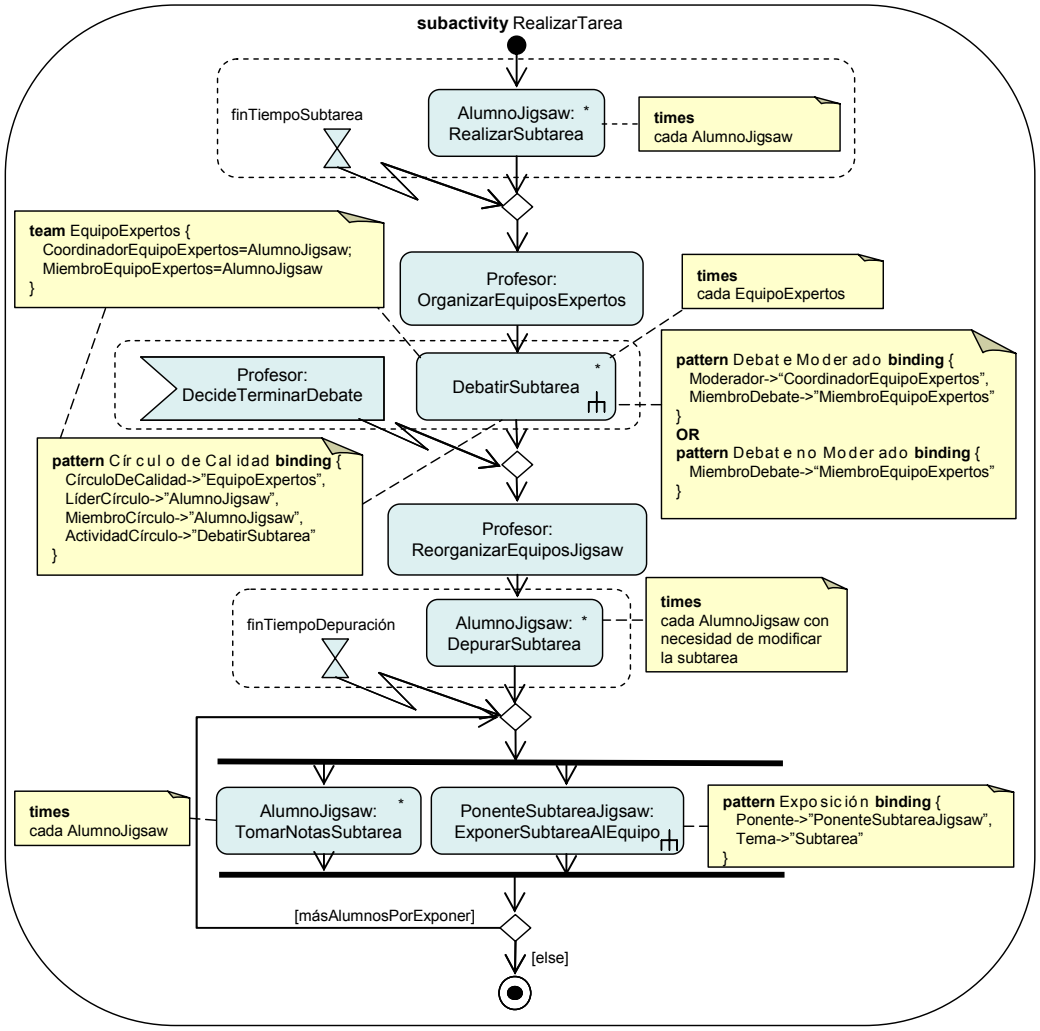


Fig. 6.31: Modelado de la subactividad `RealizarTarea`

Si los equipos de expertos se montan con el objetivo de mejorar, en la medida de los posible, el desarrollo de una determinada subtarea, analizando la intención del patrón de equipo CIRCULO DE CALIDAD(2.2.1) (Fig. 6.32) parece razonable que dicho patrón pueda ser empleado para realizar la especificación de un equipo de expertos.

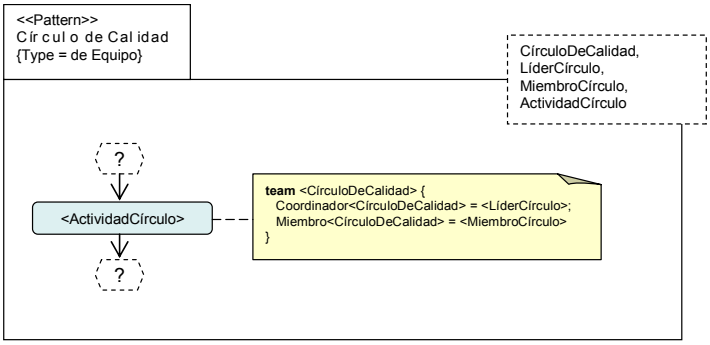


Fig. 6.32: Modelo correspondiente al patrón CÍRCULO DE CALIDAD(2.2.1)

Para ello, conectamos con la actividad `DebatirSubtarea` y con la etiqueta que define el equipo de expertos (v. `team EquipoExpertos`) una etiqueta conteniendo la siguiente expresión de ligadura:

```
CírculoDeCalidad->"EquipoExpertos"

LíderCírculo->"AlumnoJigsaw"

MiembroCírculo->"AlumnoJigsaw"

ActividadCírculo->"DebatirSubtarea"
```

En cuanto a la actividad `DebatirSubtarea`, de manera similar a lo que hicimos en el caso de estudio anterior, dicha actividad se puede describir perfectamente mediante la ligadura dinámica de los patrones de comunicación DEBATE MODERADO(2.6.1) (Fig. 6.18) y DEBATE NO MODERADO(2.6.2) (Fig. 6.19), pudiendo el equipo de expertos cambiar en cualquier momento el protocolo de comunicación utilizado para el debate. Como vemos, en caso de un debate moderado la persona que actuaría como `Moderador` sería quien ocupa el rol de `CoordinadorEquipoExpertos` dentro del equipo. Terminado el debate, el `Profesor` recompone cada uno de los equipos de tipo `EquipoJigsaw` iniciales (v. acción `ReorganizarEquiposJigsaw`), volviendo cada experto a su equipo origen. A continuación, los alumnos revisan y depuran, si es necesario, la subtarea que tienen asignada (v. acción `DepurarSubtarea`). Finalizado el tiempo previsto para la depuración (v.

acción de recepción de evento de tiempo `finTiempoDepuración`) uno de los alumnos acepta el rol de `PonenteSubtareaJigsaw` y expone al resto del equipo la subtarea que tiene asignada una vez depurada (v. actividad `ExponerSubtareaAlEquipo`). Mientras tanto, el resto de miembros del equipo toma nota de la subtarea que el/la compañero/a está exponiendo (v. acción `TomarNotasSubtarea`). Si quedan más alumnos por exponer su subtarea (v. condición `[másAlumnosPorExponer]`) otro/a alumno/a asume el rol de `PonenteSubtareaJigsaw` y el ciclo se repite. Cuando ya no quedan más alumnos por exponer, la subactividad `RealizarTarea` concluye.

Hemos utilizado el patrón de comunicación EXPOSICIÓN(2.6.5) (Fig. 6.33) para especificar la actividad `ExponerSubtareaAlEquipo`.

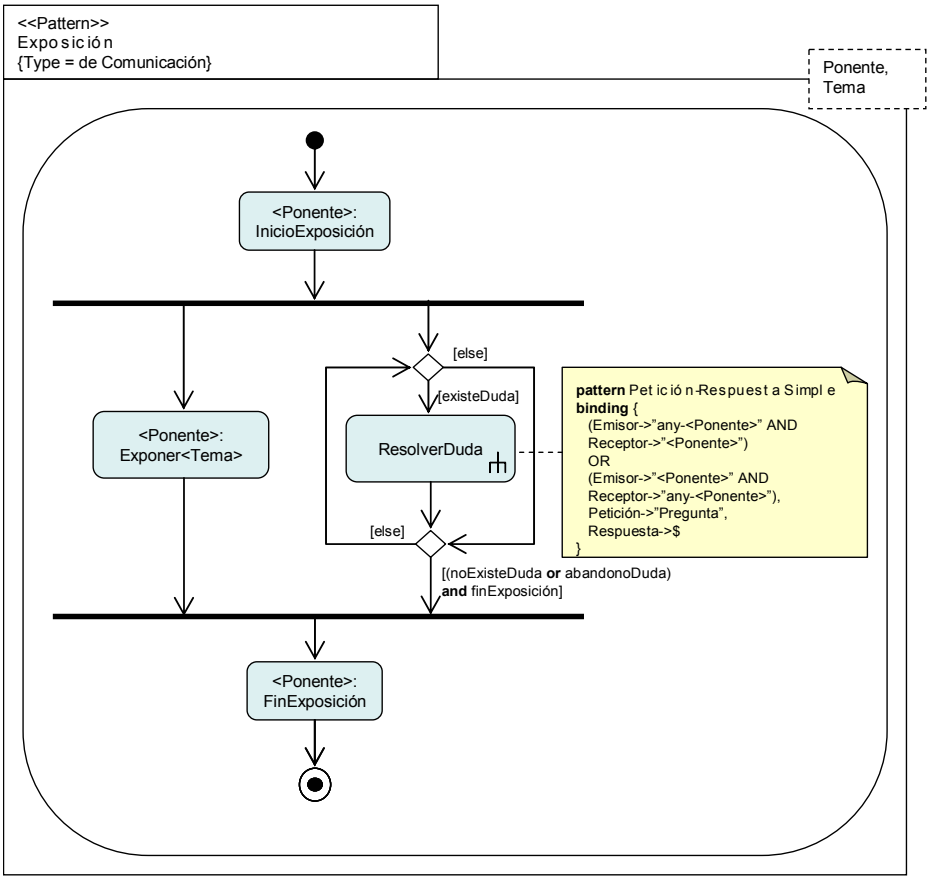


Fig. 6.33: Modelo correspondiente al patrón EXPOSICIÓN(2.6.5)

Para su ligadura necesitamos saber simplemente quién va actuar como `Ponente`, en este caso el miembro del `EquipoJigsaw` que tiene el rol de `PonenteSubtareaJigsaw` en ese momento, y el `Tema` a exponer, o sea, la

`Subtarea`. Aunque con la especificación de la ligadura es suficiente, el despliegue de esta actividad puede contemplarse en la Figura 6.34.

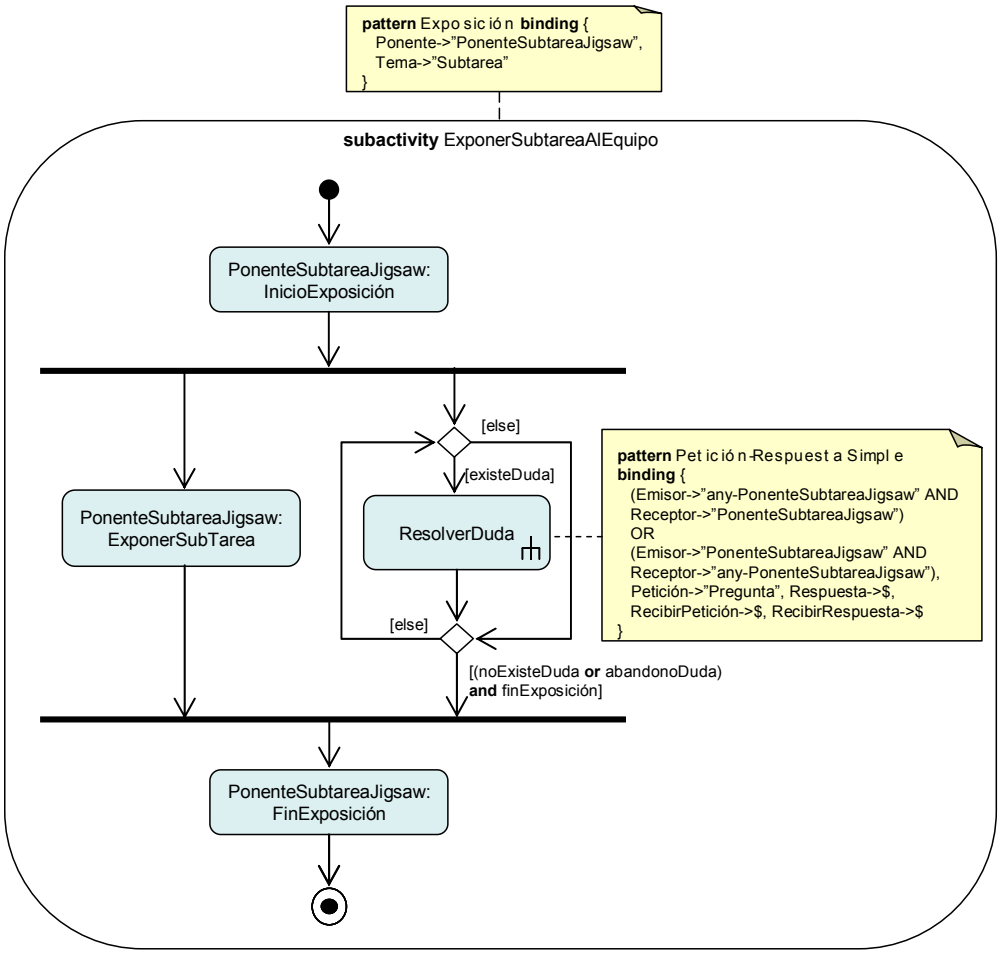


Fig. 6.34: Modelado de la subactividad `ExponerSubtareaAlEquipo` de acuerdo con el patrón EXPOSICIÓN(2.6.5)

A su vez, vemos que el patrón EXPOSICIÓN(2.6.5) hace uso del patrón PETICIÓN-RESPUESTA SIMPLE(2.6.3) (Fig. 6.35) para facilitar la especificación de la tarea `ResolverDuda` por medio de una ligadura dinámica del mismo patrón. Esta ligadura expresa que, dependiendo del ciclo, unas veces es el `PonenteSubtareaJigsaw` quien envía la respuesta, siendo cualquier otro quien emite la pregunta (`(Emisor -> "any-PonenteSubtareaJigsaw" AND Receptor -> "PonenteSubtareaJigsaw")`), y en otras ocasiones es el `PonenteSubtareaJigsaw` quien emite la pregunta, siendo cualquier otro quien la responde (`(Emisor -> "PonenteSubtareaJigsaw" AND Receptor -> "any-PonenteSubtareaJigsaw")`). En ambos casos, el resto de

parámetros recibe los siguientes valores: `Petición->"Pregunta"`, `Respuesta->$`, `RecibirPetición->$` y `RecibirRespuesta->$`.

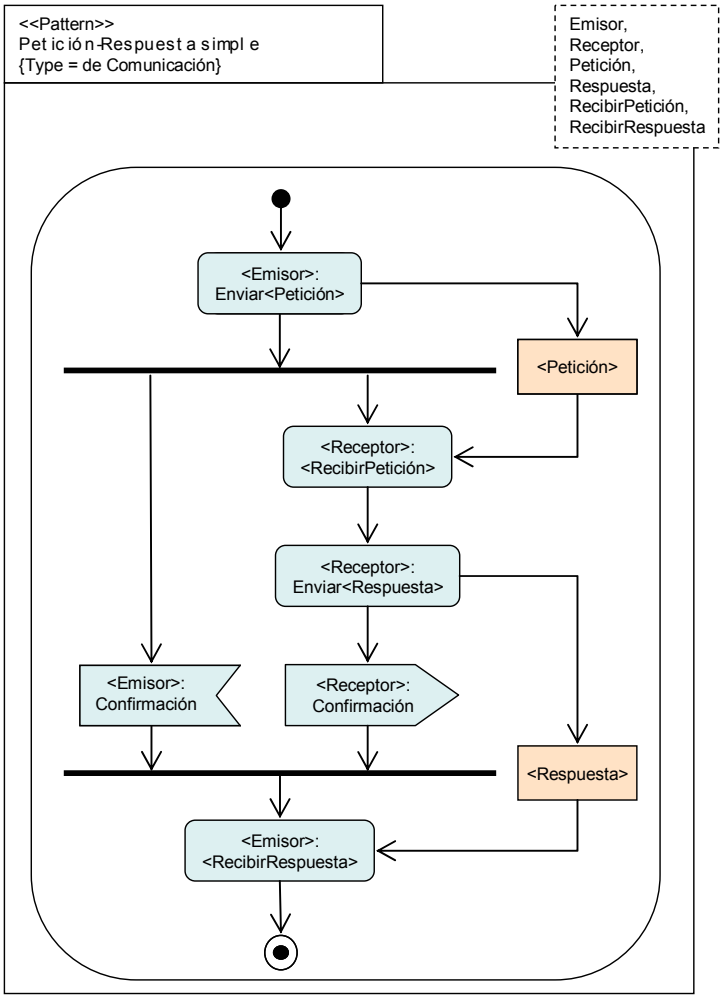


Fig. 6.35: Modelo correspondiente al patrón PETICIÓN-RESPUESTA SIMPLE(2.6.3)

Por último, en la Figura 6.36 modelamos la subactividad `EvaluaciónTarea`, incluida en la tarea cooperativa `RealizarJigsaw`, la cual se activa cuando se agota el tiempo asignado (v. acción de aceptación del evento de tiempo `finTiempoTarea`) para la actividad `RealizarTarea`.

La subactividad arranca dando el `Profesor` comienzo a la evaluación (v. acción `IniciarEvaluación`). Como sabemos, dicha evaluación puede ser individual (v. condición `[evaluaciónIndividual]`) o grupal (v. condición `[evaluaciónGrupal]`). En el primer caso cada alumno (v. `times cada AlumnoJigsaw`) realiza su propio examen de manera paralela al resto de alumnos (v. acción concurrente `HacerExamenIndividual`). En el segundo, de

manera secuencial y mientras haya presentaciones pendientes (v. condición `[másPresentaciones]`), el portavoz (rol `PortavozJigsaw`) de cada uno de los grupos va exponiendo (v. acción `PresentarTareaPúblicamente`) a toda la clase la tarea que ha realizado su equipo. Más tarde, el `Profesor` califica a los alumnos (v. acción `Evaluar`), dando por finalizada la evaluación (v. acción `FinalizarEvaluación`).

Podemos apreciar que, al igual que sucedió con la actividad `ExponerSubtareaAlEquipo`, para la especificación de la actividad `PresentarTareaPúblicamente` hemos recurrido nuevamente al patrón EXPOSICIÓN(2.6.5) (Fig. 6.33). En este caso, la ligadura indica que el `Ponente` es el actor que tiene el rol de `PortavozJigsaw` y el `Tema` a exponer es la `Tarea`.

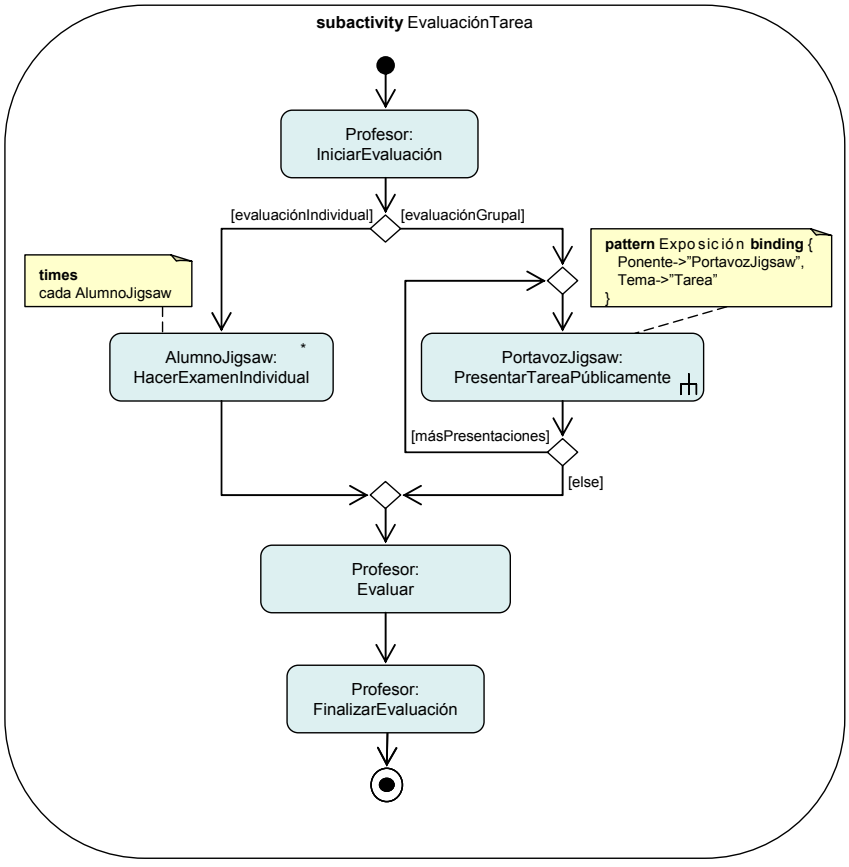


Fig. 6.36: Modelado de la subactividad `EvaluaciónTarea`

3.2.4. Especificación del modelo conceptual de datos

Nuestro catálogo no dispone actualmente de ningún patrón de estructura aplicable a los objetos de información que hemos contemplado en este caso de estudio.

4. Conclusiones del capítulo

En el presente capítulo hemos mostrado un par de casos de estudio en los que, a modo de ejemplo, aplicamos el perfil y el catálogo para la construcción del Modelo Cooperativo de dos sistemas particulares. El primero es un sistema para la gestión de la cooperación en el contexto de un proyecto de investigación coordinado. El segundo consiste en un sistema para el aprendizaje cooperativo a través de la implantación de una estrategia de Jigsaw.

Nuestra pretensión ha sido comprobar cómo nuestra propuesta agiliza el proceso de especificación de ambos Modelos Cooperativos, a la vez que mejora los modelos construidos desde el punto de vista de su comprensión, comunicación y mantenimiento. Para no añadir complejidades innecesarias, hemos creído conveniente no abordar estos sistemas en toda su extensión, simplificando el dominio de aplicación y limitando nuestra especificación a aquellos escenarios cooperativos que consideramos más relevantes.

El resultado ha sido que determinados patrones[10] del catálogo nos han permitido identificar algunos de los escenarios que comúnmente aparecen en el ámbito del trabajo cooperativo. Además, estos patrones nos han facilitado enormemente la labor de modelado de estos escenarios. Unas veces guiando el despliegue y descripción de la instancia correspondiente, y otras, simplemente, a través de la especificación de una nota conteniendo la expresión de ligadura que define la instancia, sin necesidad de desplegarla.

[10] Queremos hacer constar que, aunque a lo largo de este proceso hemos visto la oportunidad de aplicar también otros patrones referenciados en el catálogo, tan sólo hemos aplicado aquellos que tenemos completa y detalladamente descritos en el Apéndice B.

Debido a nuestra experiencia con el catálogo, la selección de estos patrones ha sido prácticamente directa. Ahora bien, si no disponemos de tal experiencia, la aplicación de la secuencia de filtros de selección recomendados en el capítulo anterior (sec. 3.1) puede conducirnos hasta el patrón más adecuado en cada momento. De todas formas, para cualquiera de los patrones elegidos, el lector puede comprobar fácilmente cuál hubiera sido el camino que le habría llevado hasta él. Por ejemplo, antes de aplicar el patrón JOINT VENTURE(2.1.1), primero habríamos seleccionado todos los patrones de organización (filtro 1), después habríamos descubierto que la intención del patrón JOINT VENTURE(2.1.1) encaja con nuestras expectativas (filtro 2) y, por último, tras la revisión de las secciones *contexto*, *solución*, *explicación* y *ejemplo* de su plantilla de descripción (filtro 3) confirmaríamos que dicho patrón es el adecuado para nuestro problema. No obstante, como hemos podido observar en ambos casos de estudio, la aplicación de la mayoría de los patrones se deriva directamente de la aplicación previa de otros que hacen uso de éstos.

Resumidamente, durante el primer caso de estudio hemos observado cómo la aplicación del patrón JOINT VENTURE(2.1.1) nos facilita la especificación de la organización que rige un proyecto de investigación coordinado, ayudando a identificar los distintos roles involucrados, sus responsabilidades y su comportamiento. Seguidamente, durante la fase de especificación de roles, hemos visto que es posible emplear el patrón COORDINADOR(2.3.1) para modelar el rol `CoordinadorSubproyectos`. Más tarde, durante la etapa de especificación de tareas, y centrando nuestro interés en la tarea cooperativa `ReuniónCoordinaciónSubproyectos`, hemos podido constatar que el modelado de dicha tarea se ve favorecido por la aplicación del patrón PROCESO DE REUNIÓN(2.4.1). Este patrón, a su vez, nos informa que es posible utilizar el patrón REUNIÓN(2.4.2) para el modelado de la actividad `SesiónReuniónCoordinación`. Aplicado el patrón, comprobamos nuevamente que, dicho patrón también hace uso de otros patrones para la especificación de dos de sus actividades. En concreto, tal y como nos sugiere el patrón, definimos la actividad de comunicación `DebatirPunto` por medio de la ligadura dinámica de los patrones DEBATE MODERADO(2.6.1) y DEBATE NO MODERADO(2.6.2). Con esta ligadura queremos expresar que el grupo de trabajo puede cambiar en cualquier momento el protocolo de comunicación utilizado para el debate. Del mismo modo, la actividad `VotarPunto` la definimos mediante la ligadura del patrón VOTACIÓN(2.4.3). Es necesario subrayar que la

simple especificación de tales ligaduras es suficiente para que los modelos correspondientes a estas actividades queden completamente definidos, sin necesidad de desplegarlos ni dar más detalles sobre ellos. Esto eleva el nivel de abstracción y simplifica bastante los modelos. Incluso, como hemos podido percibir, el patrón VOTACIÓN(2.4.3) hace uso de otros patrones, los cuales definen algunas de las subactividades que lo componen. La sola ligadura del patrón VOTACIÓN(2.4.3) desencadena automáticamente en la instancia la ligadura de estos patrones para dichas subactividades. Particularmente, la actividad `EmitirVotoIndividuo` se define a partir del patrón PETICIÓN-RESPUESTA SIMPLE(2.6.3), `EmitirVotoGrupo` por medio de PETICIÓN-RESPUESTA MÚLTIPLE(2.6.4) y las actividades `InformarOpcionesVotación` e `InformarResultadoVotación` se describen mediante el patrón EXPOSICIÓN(2.6.5). Este caso de estudio termina con la especificación del modelo conceptual de datos correspondiente al objeto `Acta`, soportado por la aplicación del patrón ACTA DE REUNIÓN(2.7.1).

En cuanto al segundo caso de estudio, por su particularidad, no hemos encontrado en el catálogo patrones útiles para la especificación de la organización o de los roles que intervienen. Es durante la especificación de tareas, y más concretamente durante el modelado de la tarea cooperativa `RealizarJigsaw`, cuando emprendemos la aplicación de patrones. Aquí, el patrón NEGOCIACIÓN NO MODERADA(2.4.4) se ajusta perfectamente a las necesidades de modelado de la actividad `ElegirSubtareas`, quedando ésta totalmente definida a través de la ligadura de este patrón. Luego, durante el modelado de la actividad `RealizarTarea` hemos usado el patrón de equipo CIRCULO DE CALIDAD(2.2.1) para especificar `EquipoExpertos` junto a la actividad en la que participan (`DebatirSubtareas`), la cual hemos definido como una ligadura dinámica de los patrones DEBATE MODERADO(2.6.1) y DEBATE NO MODERADO(2.6.2). Aparte, la actividad `RealizarTarea` incluye también la subactividad `ExponerSubtareaAlEquipo`, que hemos conectado con una ligadura del patrón EXPOSICIÓN(2.6.5). Al mismo tiempo, este patrón hace uso de una ligadura dinámica del patrón PETICIÓN-RESPUESTA SIMPLE(2.6.3), lo que permite expresar que, durante la subactividad `ResolverDuda`, unas veces es el `PonenteSubtareaJigsaw` quien hace las preguntas y otras quien las responde. Por último, la subactividad `PresentarTareaPúblicamente`, incluida en la actividad `EvaluaciónTarea`, la hemos definido también a partir del patrón EXPOSICIÓN(2.6.5).

La aplicación de patrones durante el modelado conceptual de ambos sistemas nos ha reportado importantes beneficios, en particular nos ha facilitado:

- El reconocimiento de abstracciones clave recurrentes en el dominio del problema.

  Los patrones guían la percepción que tenemos del dominio, ayudándonos a encontrar, comprender y describir los escenarios de trabajo cooperativo que aparecen comúnmente.

- El modelado de estas abstracciones en contextos concretos mediante la reutilización del modelo que proporciona el patrón.

  El perfil PMP para modelado de patrones de software que proponemos en esta tesis (Cap. IV) permite la instanciación de dichas abstracciones en contextos específicos. Esta reutilización simplifica la construcción y ayuda a reducir o eliminar los errores de modelado.

- La utilización de un vocabulario común que nos permite comunicar y razonar sobre dichas abstracciones.

  Los patrones proporcionan un lenguaje de comunicación que facilita el entendimiento entre las personas. Así, por ejemplo, una persona conocedora de los patrones aplicados en los casos de estudio expuestos, podría leer los dos párrafos que hemos escrito anteriormente, donde describimos someramente cómo hemos utilizado los patrones, y comprender fácilmente una buena parte de la especificación realizada.

- La mejora de la comprensión, comunicación y mantenimiento de los modelos, así como de la documentación en general.

  El perfil PMP facilita la visualización de los patrones aplicados en la construcción de los modelos y la identificación de sus instancias, lo que permite elevar el nivel de abstracción de éstos e interpretar los modelos en términos de patrones. Además, a menudo, la simple conexión de una nota con el elemento que se pretende especificar, expresando la ligadura que define la instancia concreta del patrón que lo modela, basta para que el modelo quede completamente definido.

Como hemos oportunidad de ver en este mismo capitulo, esto permite la simplificación de los modelos, a la par que enriquece su semántica.

La propia descripción de los patrones usados forma parte de la documentación asociada. Esto permite que durante la generación de la documentación se puedan referenciar, mediante el nombre del patrón, las soluciones empleadas sin tener que entrar en el detalle, lo que simplifica y mejora la comprensión. Durante la redacción de los casos de estudio hemos dedicado muy poco tiempo a comentar los modelos que son generados a partir de los patrones, ya que su explicación sería redundante con la que ya forma parte de la plantilla de descripción del patrón. Es más, dicha explicación puede incluso ser utilizada a modo de patrón lingüístico, con el fin de generar rápidamente la documentación correspondiente a la instancia concreta. Excepcionalmente, y a modo de demostración, hemos explicado la organización del proyecto de investigación coordinado a partir de la explicación contenida en la plantilla del patrón JOINT VENTURE(2.1.1).

- La reducción del tiempo de modelado.

  El conocimiento apropiado del catálogo y del perfil PMP permite alcanzar eficazmente los beneficios citados y, como consecuencia, acelerar considerablemente el tiempo de modelado.

Capítulo VII

Conclusiones y Trabajos Futuros

Contenido

<<La verdadera investigación consiste en buscar a oscuras el interruptor de la luz. Cuando la luz se enciende, todo el mundo lo ve muy claro>>

—Anónimo

1. Principales aportaciones

Como conclusión, complementariamente a las reflexiones vertidas al final de cada uno de los capítulos, destacamos las contribuciones de esta tesis que consideramos más significativas:

- Revisamos los principales trabajos publicados en relación con la aplicación de patrones durante las fases tempranas de modelado de los sistemas software y, en particular, de los sistemas cooperativos, así como una reflexión acerca del estado del arte.

- Analizamos el tratamiento que hace UML de los patrones y exponemos los principales inconvenientes que encontramos para el modelado adecuado de éstos.

- Realizamos un recorrido por las distintas aproximaciones que se han hecho para el modelado de los patrones de software, haciendo especial

hincapié en sus limitaciones en comparación con la propuesta que planteamos.

- Definimos un perfil UML para el modelado de patrones de software (*PMP*, Pattern Modelling Profile) que permite representar tanto la vista interna como externa de un patrón, independientemente de su tipo. A partir de un conjunto mínimo de elementos, los patrones se definen de manera simple, intuitiva y fácil de comprender como plantillas flexibles que representan familias de modelos semejantes (instancias del patrón), las cuales pueden usarse como guía para la creación y/o descripción de modelos, o partes de éstos, por medio de la correspondencia (ligadura) de los elementos del patrón con los elementos que forman sus instancias. Este perfil es usado por la notación COMO-UML, propia de AMENITIES y basada en UML, para poder modelar los patrones conceptuales que usamos durante la construcción del Modelo Cooperativo de un sistema en el marco de dicha metodología, así como de cualquier otro tipo de patrón aplicable en fases posteriores de desarrollo.

- Introducimos el concepto de patrón cooperativo como aquel que especifica un modelo conceptual reutilizable durante la construcción del Modelo Cooperativo de un sistema.

- Diseñamos una plantilla para la descripción uniforme de los patrones cooperativos, la cual facilita su estudio, comparación y aplicación con la metodología AMENITIES.

- Fruto de nuestra experiencia durante el modelado conceptual de sistemas cooperativos, recopilamos una colección de patrones que describimos por medio de la plantilla mencionada anteriormente.

- Constituimos un catálogo inicial que organiza los patrones de la colección, a la par que estructura el dominio del problema, favoreciendo la selección del patrón más adecuado en cada momento. Para ello, de acuerdo con AMENITIES, clasificamos los patrones según la vista a la que pertenecen, fase/s de construcción del Modelo Cooperativo donde se suelen aplicar y aspecto concreto que abordan dentro del dominio del problema, lo que nos permite distinguir, de momento, entre varios tipos de patrones cooperativos (de

organización, de equipo, de rol, de actividad, de coordinación, de comunicación, de estructura y de acceso).

- Proveemos una red de interconexión entre patrones que, aparte de las familias que se forman de manera natural a partir del catálogo, permite relacionar los patrones de la colección en base a otros criterios que nos pueden ayudar en la localización del patrón que estamos buscando, en particular: patrones que pueden aplicarse antes o después de uno dado (relación de proximidad), patrones que son usados por otros (relación de uso) y patrones que son similares (relación de similitud).

- Determinamos un método flexible que, en base a los elementos anteriores, marca las pautas para la selección y aplicación efectiva de los patrones del catálogo.

- Esbozamos los principales requisitos que debería satisfacer una herramienta para el modelado basado en patrones de acuerdo con nuestra propuesta.

- Vislumbramos y establecemos algunas de las posibles conexiones que pueden darse entre los patrones conceptuales que proponemos y determinados patrones de diseño que aparecen en la literatura. Aunque esto es tan sólo una primera aproximación, nuestra intención es tender un puente que facilite el tránsito del dominio del problema al dominio de la solución. Estamos muy interesados en extender la aplicación de patrones hacia la etapa de diseño de sistemas cooperativos con AMENITIES.

- Exponemos un par de casos de estudio, a través de los cuales hemos podido comprobar los beneficios que supone la aplicación de nuestra propuesta para la construcción en base a patrones del Modelo Cooperativo de dos sistemas particulares. En concreto, nos ha facilitado:

  - El reconocimiento, entendimiento y descripción de abstracciones clave recurrentes en el dominio del problema.

  - El modelado de estas abstracciones en contextos concretos mediante la reutilización del modelo que proporciona el patrón, lo

que simplifica y acelera la construcción, a la vez que reduce los errores de modelado.

- El establecimiento de un vocabulario común que nos permite compartir dichas abstracciones y pensar en términos de éstas.

- La mejora de la comprensión, comunicación y mantenimiento de los modelos, así como de la documentación en general.

1.1. Publicaciones científicas y proyectos de investigación relacionados

Esta tesis se asienta en una serie de publicaciones que han permitido dar a conocer y reafirmar nuestra línea de trabajo, así como obtener una retroalimentación de indudable valor durante la participación del doctorando en diferentes encuentros científicos. Dichos trabajos, los cuales han sido convenientemente citados en el texto y referenciados en la bibliografía, son por orden cronológico:

- ISLA, J. L. (2001). *Un Modelo para la Especificación Estructural de Patrones y su Aplicación a UML*, Tesina, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Granada, Director: Dr. José Parets Llorca

- ISLA, J. L. (2002). "A New Approach for Modelling Design Patterns with UML". *The 12th PHDOOS workshop (ECOOP2002)* (Málaga, Spain, 10-14 june 2002). Abstract in Hernández and Moreira (eds.), *ECOOP 2002 Workshops and Posters*, Springer-Verlag, LNCS 2548

- ISLA, J. L.; GUTIÉRREZ, F. L. (2002). "Diseño en Base a Patrones. Aplicación a Sistemas Hipermedia Colaborativos". En *Actas del I Taller en Sistemas Hipermedia Colaborativos y Adaptativos*, VII Jornadas de Ingeniería del Software y Bases de Datos, (El Escorial, Madrid, 18 al 22 de Noviembre de 2002)

- GUTIÉRREZ, F. L.; ISLA, J. L.; GEA, M. (2003). "Modelando Patrones de Organización". En *Actas del II Taller de Sistemas Hipermedia Colaborativos y Adaptativos*, Jornadas de Ingeniería del Software y Bases de Datos (JISBD03), Alicante, España

- ISLA, J. L.; GUTIÉRREZ, F. L. (2003). "Structural Modeling of Design Patterns: REP Diagrams". In Hanmer and Andrade (eds.), *Proceedings of the SugarLoafPLoP 2003* (Porto de Galinhas, Pernambuco, Brazil, August 12-15, 2003), pp. 301-309

- ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M. (2004a). "Patrones de Organización. Integración en un Proceso de Desarrollo Centrado en el Grupo", *Actas del Congreso Internacional Interacción 2004*, Lorés y Navarro (eds.), Lleida, 2004, pp. 172-179

- ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M.; GARRIDO, J. L. (2004b). "Descripción de Patrones de Organización y su Modelado con AMENITIES", *Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'04)*, Dieste y Moreno (eds.), Madrid, pp. 3-14

- ISLA, J. L.; GUTIÉRREZ, F. L.; PADEREWSKI, P. (2005a). "Un Profile para el Modelado de Patrones de Software". En *Actas de las X Jornadas en Ingeniería del Software y Bases de Datos*, Thomson Paraninfo, pp. 265-270

- ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M. (2005b). "Organization Modelling of the Collaborative Process: A Pattern-Based Approach". In *Proceedings of The 9th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2005)*, vol. 2, Coventry, UK, pp. 944-949

- GUTIÉRREZ, F. L.; ISLA, J. L.; PADEREWSKI, P.; SÁNCHEZ, M. (2006a). "Organization Modelling to Support Access Control for Collaborative Systems". In *Proceedings of The 2006 International Conference on Software Engineering Research and Practice*, CSREA Press, Las Vegas, Nevada, USA, pp. 757-763

- GUTIÉRREZ, F. L.; PENICHET, V. M. R.; ISLA, J. L.; MONTERO, F.; LOZANO, M. D.; GALLUD, J. A.; RODRÍGUEZ, M. L. (2006b). "Un Marco Conceptual para el Modelado de Sistemas Colaborativos Empresariales". En *Actas del VII Congreso Internacional de Interacción Persona-Ordenador (Interacción 2006)*, Puertollano (Ciudad Real), España, 13-17 de noviembre de 2006

- ISLA, J. L.; GUTIÉRREZ, F. L.; PADEREWSKI, P. (2006a). "Una Aproximación Basada en Patrones para el Modelado Conceptual de Sistemas Cooperativos". En Riquelme, J. y Botella, P. (eds.), *Actas de las XI Jornadas de Ingeniería del Software y Bases de Datos*, CIMNE, Barcelona, pp. 305-314

- ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M. (2006b). "Supporting Social Organization Modelling in Cooperative Work Using Patterns". In Shen, W. et al. (eds.), *Computer Supported Cooperative Work in Design II*, LNCS 3865, Springer, pp. 112-121

- ISLA, J. L.; GUTIÉRREZ, F. L.; GARRIDO, J. L.; HURTADO, M. V.; HORNOS, M. J. (2006c). "Integration of Organisational Patterns into a Group-Centred Methodology". *HCI Related Papers of Interacción 2004*, Springer-Verlag, Dordrecht, Netherlands, pp. 137-146

- SANCHEZ, M.; JIMÉNEZ, B.; GUTIÉRREZ, F. L.; PADEREWSKI, P.; ISLA, J. L. (2006). "Modelo de Control de Acceso en un Sistema Colaborativo". En *Actas del VII Congreso Internacional de Interacción Persona-Ordenador*, Puertollano (Ciudad Real), pp. 227-237

- GONZÁLEZ, P.; GRANOLLERS, T.; GUTIÉRREZ, F. L.; ISLA, J. L.; MONTERO, F. (2007). "De los Patrones de Organización e Interacción al Diseño de Interfaces de Usuario Colaborativas". En *Actas del 5º Taller en Sistemas Hipermedia Colaborativos y Adaptativos (SHCA 2007)* dentro de las Jornadas de Ingeniería del Software y Bases de Datos (JISBD07), (Zaragoza, Spain, 11 de Septiembre de 2007), pp. 9-16

- GUTIÉRREZ, F. L.; ISLA, J. L.; PADEREWSKI, P.; SÁNCHEZ, M.; JIMÉNEZ, B. (2007). "An Architecture for Access Control Management in Collaborative Enterprise Systems Based on Organization Models". *Science of Computer Programming*, vol. 66, nº 1, 2007, pp. 44–59

- ISLA, J. L.; GUTIÉRREZ, F. L.; PADEREWSKI, P. (2007). "Una Aproximación Basada en Patrones para el Modelado Conceptual de Sistemas Cooperativos". *IEEE Latin America Transactions*, Vol. 5, Nº 4, July 2007, pp. 204-210

Buena parte de las publicaciones que se derivan de esta tesis han sido financiadas en el marco del proyecto de investigación siguiente:

- *Análisis y modelado de sistemas colaborativos-AMENITIES. Proyecto coordinado ADACO*. Investigador responsable: Dr. Fco. Luis Gutiérrez Vela. Entidad que financia: CICYT, fondos FEDER. Entidades participantes: Universidad de Granada junto con la Universidad de Castilla-La Mancha y la Universidad de Lleida. Referencia: TIN2004-08000-C03-02. Fecha Inicial: 2004. Fecha final: 2007.

2. Trabajos futuros

Sin duda, nuestro trabajo no finaliza aquí. Tratándose de ciencia, todo es discutible y mejorable. Además, el conocimiento siempre abre nuevos caminos que es necesario sondear. De momento, hemos dejado varios frentes abiertos, en los cuales ya estamos empezando a trabajar:

- Ampliar y mejorar el catálogo propuesto, tanto desde el punto de vista de su estructura organizativa como de los patrones que contiene, conforme vaya aumentando nuestra experiencia y la de otras personas en el uso del catálogo y conocimiento del dominio del problema.

- Mantener un sitio web colaborativo, por ejemplo una wiki, que permita la discusión, edición y almacenamiento de patrones concretos para su difusión entre la comunidad interesada.

- Implementar una herramienta para el modelado basado en patrones que facilite la selección y aplicación de éstos de acuerdo con nuestra propuesta. Al final de la sección 3.2 del Capítulo V, destacamos algunos de los requisitos que debería poseer dicha herramienta.

- Profundizar en el estudio de las conexiones que pueden establecerse entre patrones conceptuales y patrones aplicables en una etapa de diseño de alto nivel. El objetivo es intentar suavizar la transición del dominio del problema al dominio de la solución.

- En relación con el punto anterior, extender la aplicación de patrones a lo largo de todo el proceso de desarrollo de sistemas cooperativos con la metodología AMENITIES.

- En la actualidad estamos trabajando en una arquitectura basada en servicios web que permita gestionar los aspectos clave de un sistema colaborativo. En esta línea, pensamos que los patrones nos van a permitir definir elementos tan importantes como pueden ser las políticas de control de acceso, de gestión de flujos de trabajo o de compartición de recursos.

- A lo largo de esta tesis hemos buscado y modelado patrones bajo la metodología AMENITIES, esto no quiere decir que las ideas que se proponen queden circunscritas exclusivamente a esta metodología. Al contrario, pensamos que los tipos de patrones utilizados y la forma de modelarlos pueden ser exportados a otras metodologías. Por ejemplo, en estos momentos estamos trabajando en la aplicación de nuestro lenguaje al modelado de patrones de diseño del interfaz de usuario para aplicaciones colaborativas en el marco de la metodología IDEAS [Lozano, 2001] y su soporte con la herramienta IDEALXML [Montero, 2005; Montero et al., 2005a], enfocada hacia la gestión, manipulación y transferencia de experiencia modelada en forma de patrones.

Apéndice A Notación COMO-UML

Contenido

1. Introducción

COMO-UML es la notación utilizada para la construcción del modelo cooperativo de AMENITIES [Garrido, 2003]. Se basa en UML [OMG, 2003], derivando su sintaxis y semántica principalmente de los diagramas de estados y actividades de este lenguaje.

Se consideran válidos aquellos modelos que siguen la sintaxis y semántica de UML. Sin embargo, existen ciertas diferencias entre ambas notaciones, entre las que cabe destacar:

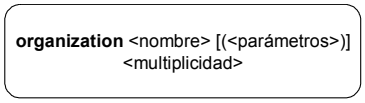
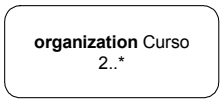
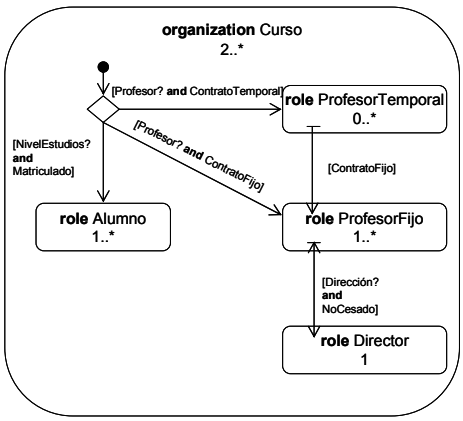
- Se definen dos tipos distintos de transiciones (aditiva y de cambio).
- En lugar de emplear la notación de calles (swimlanes), utilizada por UML para organizar las subactividades o acciones en función de los responsables que las llevan a cabo, es posible especificar los roles responsables dentro del símbolo de cada subactividad o acción, lo que simplifica los diagramas.
- Se incluyen subestados que representan las tareas dentro de los diagramas de rol.

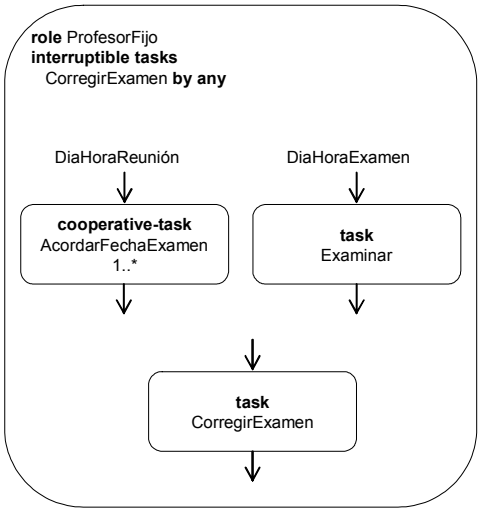
- Se incorpora notación textual para especificar las tareas interrumpibles en los diagramas de rol, multiplicidad en estados (p. ej. cantidad de actores de una organización o un rol), expresiones para asignar roles a subactividades/acciones y equipos de trabajo.

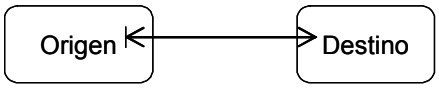
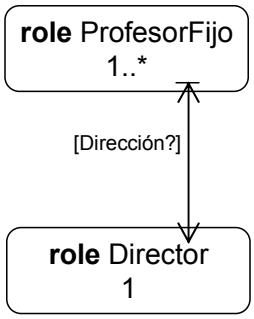
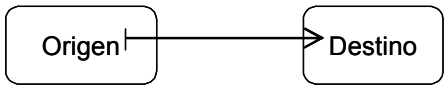
Seguidamente, en la tabla A1.1 repasamos la sintaxis y semántica de aquellos elementos cuya notación o significado difiere de UML.

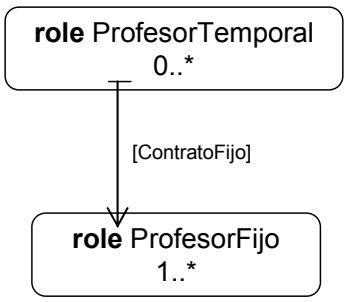
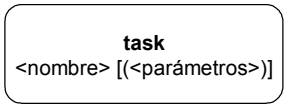
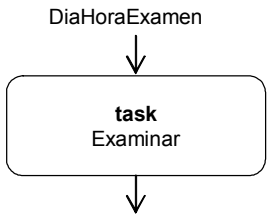
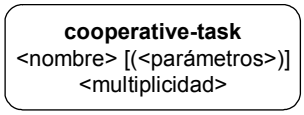
2. Notación

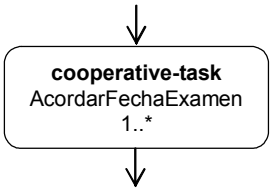
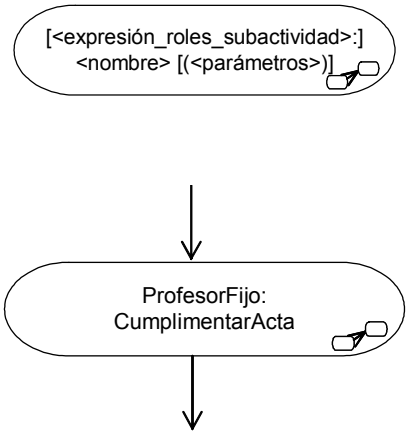
Tabla A.1: Sintaxis y semántica de COMO-UML

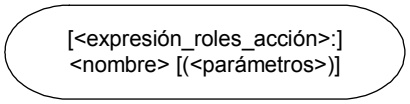
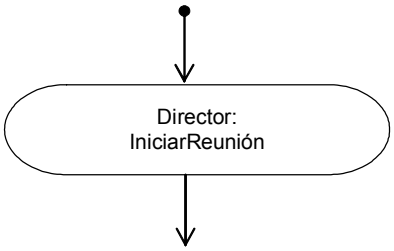
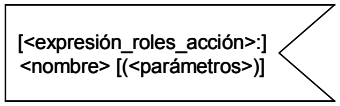
Sintaxis	Semántica
ORGANIZACIÓN  Ej.: Organización `Curso`  Diagrama de organización asociado  	Estado de submáquina que referencia una máquina de estados aparte. Su sintaxis es equivalente a UML, sustituyendo la palabra `include` por `organization`. La submáquina se corresponde con un diagrama de organización. Los subestados de cada diagrama de organización son todos suborganizaciones, o bien, son todos roles. La multiplicidad indica el número de actores que pueden formar parte de la organización (también se expresa la multiplicidad para cada uno de los roles, indicando la cantidad de actores que pueden desempeñarlo). El pseudoestado inicial permite indicar cuál es el primer rol que por defecto juega un actor en la organización, a no ser que exista una transición directa a un rol concreto. Una transición a un estado final refleja que un actor deja de pertenecer a la organización. Una caja de decisión permite elegir entre los distintos roles que un actor puede desempeñar, en virtud del cumplimiento de ciertas condiciones (leyes impuestas por la

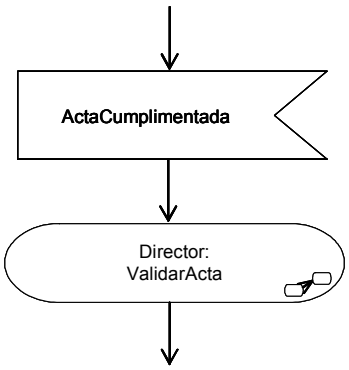
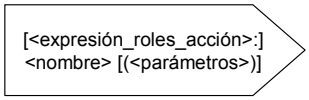
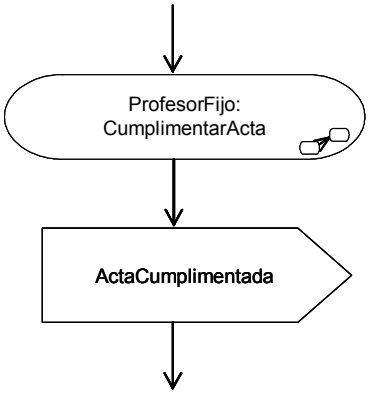
Sintaxis	Semántica
	organización y/o capacidades necesarias del actor) que etiquetan sus transiciones de salida. Si existen varias alternativas evaluadas como verdad, es decisión del propio actor o del sistema elegir una de las alternativas posibles (no determinismo). Los roles están conectados mediante transiciones de cambio y aditivas, las cuales se explican más adelante.
ROL **role** <nombre> [(<parámetros>)] <multiplicidad> Ej.: Diagrama de rol  	Representa un rol (conjunto de tareas) que puede ser jugado por uno o más actores dentro de una organización. Igual que antes, está representado por un estado de submáquina. La máquina de estados asociada describe el diagrama de rol correspondiente. En el ejemplo aparece el diagrama de rol asociado a un `ProfesorFijo` de la organización `Curso`. Para abreviar, aparecen tan sólo tres tareas (`AcordarFechaExamen`, `Examinar` y `CorregirExamen`) representadas mediante estados de submáquina con transiciones. En este caso, las transiciones de entrada a las tareas `AcordarFechaExamen` y `Examinar` están etiquetadas con los eventos que disparan su ejecución. La tarea `CorregirExamen` se realizará en el momento en el que lo decida el propio actor. Puede aparecer una sección `interruptible tasks` para indicar qué tareas pueden ser interrumpidas y por cuáles (de éste u otro rol que pueda desempeñar el mismo actor), de manera que un determinado actor puede estar implicado en varias tareas simultáneamente. En el ejemplo, la tarea `CorregirExamen` puede ser interrumpida por cualquier otra tarea del actor (palabra reservada any). El resto no

Sintaxis	Semántica
	podrán ser interrumpidas (situación por defecto).
TRANSICIÓN ADITIVA  Ej.: Transición aditiva perteneciente al diagrama de organización `Curso`  	Esta notación permite reflejar dentro de los diagramas de organización que un actor, jugando un rol `Origen`, puede desempeñar, además, el rol `Destino`. Si esta transición está etiquetada por alguna condición (ley impuesta por la organización o capacidad necesaria para el actor) ésta deberá cumplirse para que pueda dispararse. Si un actor ha asumido una serie de roles conectados en cadena y deja de cumplirse una de las leyes/capacidades, el actor abandonará el conjunto de roles asumidos a partir de la transición etiquetada por esa ley/capacidad. Al igual que en UML, este tipo de transición también puede tener un evento que la active y una serie de acciones a realizar. En el ejemplo se puede ver cómo un actor, ejerciendo el rol de `ProfesorFijo`, puede desempeñar además el rol de `Director` si tiene capacidad de dirección y no está cesado. Las capacidades se distinguen de las leyes en que la condición de guarda lleva al final el símbolo ? (v. capacidad `Dirección?` y ley `NoCesado`).
TRANSICIÓN DE CAMBIO  Ej.: Transición de cambio perteneciente al diagrama de organización `Curso`	Esta notación permite reflejar dentro de los diagramas de organización que un actor, jugando un rol `Origen`, abandona éste (y todos los que estuviera desempeñando) para pasar a cumplir el rol `Destino`. Si esta transición está etiquetada por alguna condición (ley impuesta por la organización o capacidad necesaria para el actor) ésta deberá cumplirse para que pueda dispararse. Al igual que en UML, esta transición también puede tener un evento que la active y una serie de acciones a realizar.

Sintaxis	Semántica
	En el ejemplo se muestra cómo un actor con el rol de `ProfesorTemporal`, puede abandonar éste y desempeñar exclusivamente el rol de `ProfesorFijo` si tiene un contrato fijo (v. `[ContratoFijo]`).
TAREA Tarea individual  Ej.: Tarea individual `Examinar` perteneciente al diagrama de rol `ProfesorFijo`  Tarea cooperativa  Ej.: Tarea cooperativa AcordarFechaExamen perteneciente al diagrama de rol ProfesorFijo	Estado de submáquina que se corresponde con un diagrama de tarea/subactividad de AMENITIES, el cual es un caso especial de diagrama de actividades de UML. Como se ha visto anteriormente, una tarea en un diagrama de rol puede tener transiciones de entrada y salida (incluyendo eventos, guardas y acciones según la notación UML). Por ejemplo, para que la tarea `Examinar` pueda ejecutarse es necesario que llegue la fecha y hora del examen (v. evento `DiaHoraExamen`). Estas transiciones pueden estar conectadas a otras tareas o no. En caso negativo sería decisión del propio actor su realización, no estando sincronizada su ejecución con ningún otro acontecimiento concreto en el sistema (v. más arriba la tarea `CorregirExamen` correspondiente al diagrama de rol para un profesor fijo). La tarea termina si se han llevado a cabo todas las subactividades/acciones que la definen y existe una transición de salida sin evento, o cuando existe una transición de salida etiquetada con un evento y éste ocurre. En todo caso, si hay alguna condición de guarda se tiene que satisfacer y si existen acciones se deben ejecutar.

Sintaxis	Semántica
DiaHoraReunión  	Cuando la tarea es cooperativa, el número de actores que realiza la tarea viene determinado por la multiplicidad que aparece en cada uno de los roles que intervienen. De acuerdo con el ejemplo, en la tarea cooperativa `AcordarFechaExamen` debe intervenir al menos un profesor fijo.
SUBACTIVIDAD  	Un estado de subactividad invoca un diagrama de subactividades/acciones. La subactividad termina cuando el diagrama anidado llega a un estado final o, menos frecuentemente, ocurre un evento que activa una transición que hace que salga de ese estado de subactividad. Antes del nombre puede haber una expresión de roles para indicar los roles que intervienen en dicha subactividad. Esta expresión es obligatoria cuando la subactividad tiene un protocolo de interacción asociado. Las expresiones se construyen usando los operadores y palabras reservadas que aparecen a continuación entre paréntesis: (`all`) Todos los actores implicados en la tarea participan sin importar su rol. (`-`) Los del rol que aparece a continuación no participan. (`+`) Los del rol que viene a continuación también participan. (`|`) Participa al menos uno de los roles que aparecen a ambos lados del operador. (`x`) Participa sólo uno de los roles que aparecen a ambos lados del operador.
ACCIÓN	En contraposición con las subactividades, una acción es atómica, es decir, no se puede

Sintaxis	Semántica
 	descomponer en otras acciones/subactividades (no tiene diagrama de actividades asociado) y su ejecución es inmediata (no se puede interrumpir). Se utiliza el mismo símbolo que para las subactividades, pero no aparece el símbolo de la esquina inferior derecha representando dos estados conectados. En una acción siempre interviene un único actor. Puede aparecer delante del nombre una expresión de roles de acción de la siguiente manera: - Solamente un rol. - Varios roles con el operador o-exclusivo (x). - Únicamente la palabra reservada `any` (un actor desempeñando un rol cualquiera de los implicados). - La palabra reservada `any` seguido del operador diferencia (−) y uno o más roles.
RECEPCIÓN DE SEÑAL  Ej.: Recepción de señal `ActaCumplimentada` (vista parcial de un diagrama de subactividades/acciones)	Una señal representa un evento asíncrono interno. El estado de recepción de señal acabará cuando se haga efectiva la recepción. Como se aprecia en el ejemplo, el director espera la señal `ActaCumplimentada` para poder realizar la validación del acta.

Sintaxis	Semántica
	
ENVÍO DE SEÑAL  Ej.: Envío de señal `ActaCumplimentada` (vista parcial de un diagrama de subactividades/acciones) 	En este caso, la transición de salida del estado anterior genera el envío de la señal. El remitente de la señal no espera a que el receptor se ocupe de la señal y continúa con su trabajo independientemente. En el ejemplo, una vez que un profesor fijo termina la actividad `CumplimentarActa` se genera la señal `ActaCumplimentada`, pudiendo éste continuar con otras actividades/acciones.
COMENTARIOS	Las notas o comentarios se utilizan en UML para especificar aclaraciones sobre los modelos (requisitos, restricciones, revisiones, explicaciones, observaciones, etc.). Pueden aparecer en cualquiera de los

Sintaxis	Semántica

team JuntaDeCurso {
PresidenteJunta = Director;
RepresentanteProfesorado = Profesor;
RepresentanteAlumnado = Alumno:
SecretarioJunta = Profesor;
};
	diagramas utilizados por AMENITIES.
	
	Se proporcionan varios comentarios estereotipados:
	
	- `Team`. Declara el nombre y los roles de un equipo de trabajo redefiniendo los roles de subactividad.
	
	- `Communication requirements`. Permite especificar diversos requisitos de comunicación en función del tiempo de las interacciones (synchronous, asynchronous, etc.), medios (voice, video, etc.), espacio de trabajo (shared, independent, etc.), tipo de canal de comunicación (link, port, mailbox, multicast, broadcast, etc.) y tipo de interacción (face-to-face, etc.).
	
	- `Protocol`. Por ejemplo request-reply, negotiation, queued-messages, etc.
	
	- `Times`. Para indicar el número de veces que sucede una subactividad/acción. Si aparece * en la subactividad/acción la ejecución es concurrente, en caso contrario sería secuencial.
	
	- `Constraints`. Permite incluir restricciones (leyes o capacidades) asociadas a subactividades/acciones. La restricción estereotipada non-interruptible puede asociarse únicamente con subactividades.

communication requirements {
face-to-face;
shared workspace;
Multicast
}	

protocol negotiation	
times para cada miembro	
constraints non-interruptible	

-Esta página se encuentra deliberadamente en blanco-

Apéndice B

Descripción Completa de Algunos Patrones del Catálogo

Contenido

1. Introducción

En este apéndice hacemos uso de la plantilla de descripción uniforme de patrones (v. Tabla 5.1) que aparece en el Capítulo V para especificar detalladamente algunos de los patrones que componen nuestro catálogo. El objetivo es mejorar la comprensión, selección, comparación y aplicación de los patrones en base a la propuesta metodológica para el modelado conceptual de sistemas cooperativos que hacemos en el Capítulo V.

Describimos todos los patrones que hemos aplicado en el Capítulo VI, el cual está dedicado a dos casos de estudio. En cualquier caso, mostramos al menos un ejemplo de cada tipo de patrón contemplado en el catálogo (v. Tabla 5.2).

Esperamos que este apéndice se vaya extendiendo a medida que se vayan definiendo y poniendo en práctica otros patrones.

2. Descripción de algunos ejemplos de patrones

2.1. Patrones de organización

2.1.1. Patrón JOINT VENTURE

Nombre/alias	JOINT VENTURE / AVENTURA CONJUNTA
Clasificación	▪ Vista: Organizacional. ▪ Fase: Especificación de la Organización. ▪ Tipo: Patrón de Organización.
Intención	Modelar una estructura organizativa en la que un grupo de socios, cada uno especializado en la realización de una tarea concreta, unen sus capacidades y recursos para alcanzar objetivos más ambiciosos. Así obtienen una serie de ventajas colectivamente (inversión parcial, costes de mantenimiento más bajos, mayores beneficios, recursos compartidos, etc.).
Contexto	▪ Hay un objetivo común para la alianza, el cual se descompone en varios subobjetivos. ▪ Para cada subobjetivo existe un socio especializado que es responsable de llevarlo a cabo. ▪ Cada socio puede tener su propia organización, pero debe haber alguien que actúe como representante de ésta dentro de la alianza. ▪ La alianza dispone de un actor que desempeña el papel de coordinador de los distintos socios. ▪ Suele haber alguien que juega el papel de secretario. ▪ Alguien se encarga de dirigir y representar la alianza.
Solución	Véase Figura B.1

Explicación

Una organización de tipo Joint Venture es una alianza que se compone básicamente de dos clases de suborganizaciones: `Socio` y `Administración`.

Los objetivos o responsabilidades de cada una de estas suborganizaciones[1] dentro del Joint Venture son (v. `role/organization responsibilities`):

- Organización `Socio` (dos o más organizaciones)
  - Alcanzar el subobjetivo que tiene asignado y en el que está especializada (objetivo `AlcanzarSubobjetivo`).
  - Compartir sus recursos con el resto de socios (objetivo `CompartirRecursos`).
- Organización `Administración` (sólo una organización)
  - Administrar la alianza (objetivo `AdministrarAlianza`).

A su vez, en cada una de estas suborganizaciones existe un conjunto de roles[2] relevantes para el Joint Venture y con objetivos[3] muy concretos:

- Rol `Socio::RepresentanteSocio` (sólo un actor)
  - Mantener reuniones de coordinación (objetivo compartido `MantenerReunión Coordinación`).
- Rol `Administración::Director` (sólo un actor)
  - Representar a la alianza en los contactos con el exterior (objetivo `RepresentarAlianza`).
  - Tomar las decisiones estratégicas para la alianza (objetivo

[1] Junto al nombre de cada una aparece la cantidad de organizaciones que participan (v. expresión de multiplicidad en los estados del diagrama).

[2] Junto al nombre de cada uno de los roles (obsérvese que para identificarlos se utiliza el formato `organización::rol`) aparece la cantidad de actores que pueden desempeñarlo (v. expresión de multiplicidad en los estados del diagrama)

[3] Por supuesto, estos actores pueden formar también parte de otras organizaciones y tener otros objetivos.

`EjecutarDecisiónEstratégica`).

- Rol `Administración::Coordinador` (sólo un actor)
  - Planificar las reuniones de coordinación con los socios de la alianza (objetivo `PlanificarReuniónCoordinación`).
  - Mantener reuniones de coordinación con los socios (objetivo compartido `MantenerReuniónCoordinación`).
  - Tomar decisiones para la coordinación de los socios (objetivo `EjecutarDecisiónCoordinación`).
- Rol `Administración::Secretario` (opcional)
  - Asistir a las reuniones de coordinación (objetivo compartido `MantenerReuniónCoordinación`).
  - Gestionar la documentación relacionada con la alianza (tarea `GestionarDocumentación`).

Aparte de las suborganizaciones, roles y objetivos, el patrón también representa las condiciones (capacidades o leyes) que deben cumplir las suborganizaciones para formar parte del Joint Venture o las que deben cumplir los actores para poder jugar los distintos roles especificados.

Por un lado, las suborganizaciones de tipo `Socio` tienen que ser capaces de llevar a cabo alguno de los subobjetivos en los que se ha dividido el objetivo común (v. capacidad `[subobjetivo?]`), por ejemplo fabricar alguna de las piezas de un avión cuando el objetivo compartido es la construcción de un avión completo. Por otro lado, la suborganización `Administración` debe tener capacidad para administrar la alianza (v. capacidad `[administración?]`).

En cuanto a los roles, para que un actor pueda desempeñar el rol de `Director` debe tener capacidad de dirección (v. capacidad `[dirección?]`), para ser `Coordinador` capacidad de coordinador (v. capacidad `[coordinador?]`), para actuar como `Secretario` capacidad de secretario (v. capacidad `[secretario?]`) y para ser `RepresentanteSocio` debe haber sido elegido como tal en su propia organización (v. ley

	[elegido]). Además de especificar la estructura de roles y el comportamiento individual de cada uno de éstos, el patrón describe el comportamiento de la organización desde el punto de vista de los posibles cambios de rol que pueden experimentar sus actores a lo largo del tiempo, siempre de acuerdo con las leyes que impone la propia organización y la ocurrencia de determinadas condiciones. Así, hay una transición aditiva que representa bajo qué ley un actor que está desempeñando el rol de Coordinador, sin abandonar éste, puede asumir además las labores correspondientes al rol de Director. Como se puede comprobar, esto puede suceder cuando el Coordinador tenga capacidad de dirección y el Director no esté disponible (v. condición [dirección? and directorNoDisponible]). En el momento en que alguna de estas condiciones deja de cumplirse, el actor abandona el rol de Director y vuelve a desempeñar exclusivamente su rol inicial como Coordinador. De la misma forma, el Director puede desempeñar adicionalmente el rol de Coordinador cuando tiene capacidad de coordinación y éste último no está disponible (v. [coordinación? and coordinadorNoDisponible]). El patrón también representa la incertidumbre que existe dentro del modelo de la suborganización Socio, en relación con la posible presencia de otros roles encargados de realizar tareas propias de la suborganización y que no tienen nada que ver con la alianza. Los parámetros del patrón, expresados entre ángulos dentro del modelo y en la esquina superior derecha del paquete que lo contiene, son los siguientes: <Socio>, <Administración>, <RepresentanteSocio>, <Secretario>, <Director> y <Coordinador>.
Ejemplo	Como este tipo de organización procede del ámbito de las alianzas estratégicas entre empresas, podemos poner como ejemplo la organización que sigue la empresa Airbus, la cual coordina las actividades entre varias empresas que se han asociado para construir y vender aeronaves: Aerospatiale (desarrolla y fabrica

	principalmente la cabina), DASA (el fuselaje), British Aerospace (las alas), CASA (la cola) y, finalmente, el ensamblaje se hace en Aerospatiale. En este caso, todas las operaciones estratégicas, venta, postventa y marketing son realizadas por Airbus Industrie. Otro ejemplo, a menor escala, puede ser el modelado de la estructura organizativa de un proyecto de investigación coordinado que se divide en varios subproyectos asignados a diferentes grupos de investigación (v. Figura B.2). Las labores de coordinación de todos los grupos suelen estar realizadas por alguien que mantiene contactos con representantes de los distintos subproyectos. Además, debe existir una cabeza visible con capacidad para dirigir todo el proyecto. La Figura B.2 muestra cómo es posible construir y describir este modelo a partir del patrón JOINT VENTURE. La etiqueta que define la ligadura indica los parámetros y los argumentos que dan lugar a dicha instancia.
Patrones relacionados	En ocasiones se aplica después: ▪ CADENA DE TRABAJO$_{(2.1.2)}$: Si existe secuencialidad entre los distintos subobjetivos a alcanzar por los socios. ▪ MANDO-SUBMANDO: Si existe una relación de poder entre el `Director` y el `Coordinador`.
Anexo Figuras (v. pág. siguiente)	

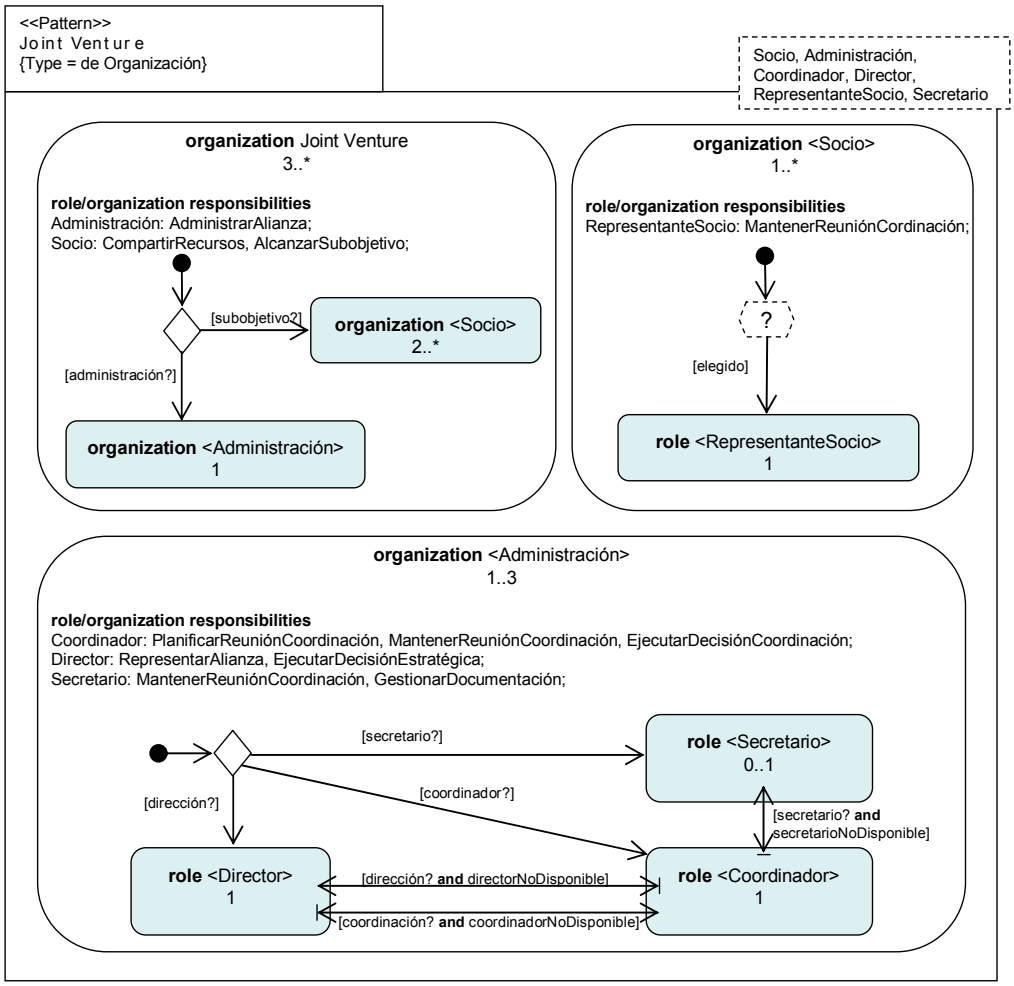


Fig. B.1: Modelado del patrón JOINT VENTURE

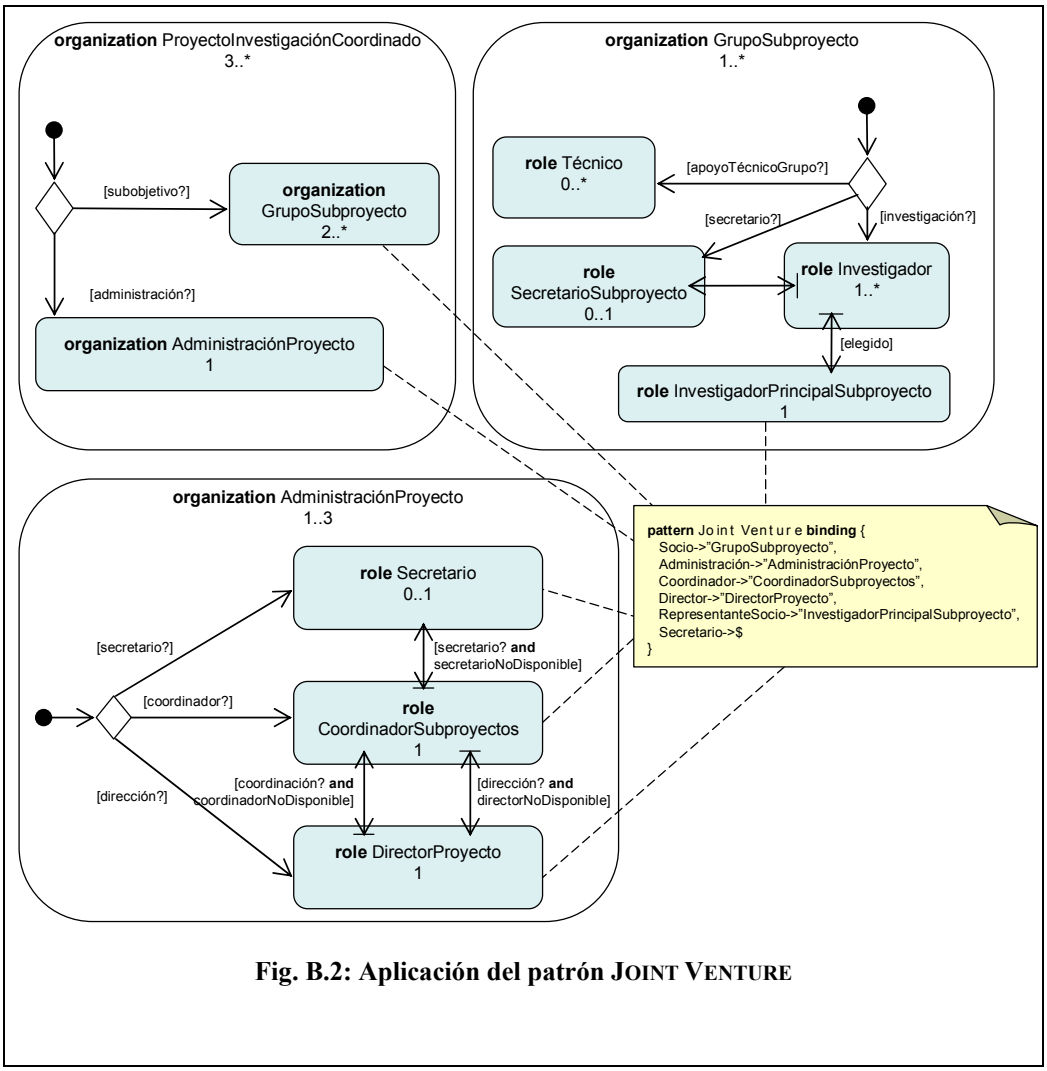


Fig. B.2: Aplicación del patrón JOINT VENTURE

2.1.2. Patrón CADENA DE TRABAJO

Nombre/alias	CADENA DE TRABAJO / PRODUCTION LINE
Clasificación	▪ Vista: Organizacional. ▪ Fase: Especificación de la Organización. ▪ Tipo: Patrón de Organización.
Intención	Modelar una estructura organizativa en la que varios actores colaboran para alcanzar una meta común en varias fases, normalmente la realización de un producto o servicio (p. ej. la cadena de venta o de montaje de un artículo), cada una con un objetivo concreto. Los actores se distribuyen entre las distintas etapas que componen la cadena. Para lograr el objetivo local de una etapa, antes se ha tenido que alcanzar el objetivo local de la etapa anterior, si ésta existe, y así sucesivamente.
Contexto	▪ Hay un objetivo común que se compone de varios subobjetivos. ▪ Cada subobjetivo se alcanza dentro de una determinada fase del trabajo. ▪ Los actores se distribuyen a lo largo de toda la cadena, de manera que para la realización de cada fase debe existir uno o más actores encargados de llevarla a cabo. ▪ Las fases son consecutivas, recibiendo el resultado de la etapa anterior, si ésta existe, para poder alcanzar el subobjetivo que tiene encomendado, y transfiriendo sus resultados a la etapa siguiente, si ésta existe, para que pueda empezar su trabajo. ▪ Hay uno o varios actores encargados de llevar a cabo cada subojetivo. ▪ Puede haber alguien encargado de coordinar a los distintos actores.

Solución	Véase Figura B.3
Explicación	Los actores de esta clase de organización pueden desempeñar básicamente dos tipos de roles: `Eslabón` y `Coordinador`. A su vez, los actores que desempeñan el rol de `Eslabón` pueden jugar alguno de los siguientes subroles: `EslabónInicial`, `EslabónIntermedio` o `EslabónFinal`. Los objetivos o responsabilidades de cada una de estos roles[4] dentro de la cadena de trabajo son (v. `role/organization responsibilities`): ▪ Rol `Eslabón` (dos o más actores) – Realizar la tarea que permite alcanzar el subobjetivo asignado a la etapa (objetivo `EjecutarTareaEtapa`). ▪ Rol `Coordinador` (sólo un actor) – Coordinar los actores de los distintos eslabones (objetivo `CoordinarEslabones`). ▪ Rol `Eslabón::EslabónInicial` (uno o más actores) – Pasar el resultado de su etapa a la etapa siguiente (objetivo `PasarResultadoASiguiente`). ▪ Rol `Eslabón::EslabónIntermedio` (uno o más actores) – Tomar el resultado de la etapa anterior (objetivo `TomarResultadoDeAnterior`). – Pasar el resultado de su etapa a la etapa siguiente (objetivo `PasarResultadoASiguiente`). ▪ Rol `Eslabón::EslabónFinal` (uno o más actores) – Tomar el resultado de la etapa anterior (objetivo

[4] Junto al nombre (obsérvese que para identificarlos se utiliza el formato `organización::rol`) de cada uno de ellos aparece la cantidad de actores que pueden desempeñarlo (v. expresión de multiplicidad en los estados del diagrama)

TomarResultadoDeAnterior).

Aparte de los roles y sus objetivos, el patrón también representa las condiciones (capacidades o leyes) que deben cumplir los actores para poder jugar los distintos roles especificados.

Así, para que un actor pueda desempeñar el rol de Eslabón debe tener capacidad para realizar la/s tarea/s asignada/s a alguna de las etapas de la cadena (v. capacidad [tareaEtapa?]) y para ser Coordinador capacidad para la coordinación (v. capacidad [coordinación?]). En cuanto a los subroles de Eslabón, para actuar como EslabónInicial debe ser capaz de realizar las tareas asociadas a la fase inicial (v. capacidad [tareaInicial?]), para desempeñar un rol de EslabónIntermedio debe ser capaz de realizar las tareas determinadas para alguna fase intermedia (v. capacidad [tareaIntermedia?]) y para acometer el rol de EslabónFinal debe poder ejecutar las tareas propias de la etapa final (v. capacidad [tareaFinal?]).

Además de especificar la estructura de roles y el comportamiento individual de cada uno de éstos, el patrón describe el comportamiento de la organización desde el punto de vista de los posibles cambios de rol que pueden experimentar sus actores a lo largo del tiempo, siempre de acuerdo con las leyes que impone la propia organización y la ocurrencia de determinadas condiciones. Así, hay una transición aditiva que representa bajo qué ley un actor que está desempeñando el rol de Coordinador puede asumir también el rol de Eslabón. Como se puede comprobar, esto puede suceder cuando el Coordinador tenga capacidad para realizar la/s tarea/s asignada/s a alguna de las etapas de la cadena y alguno de los eslabones no esté disponible (v. condición [tareaEtapa? and <Eslabón>NoDisponible]). En el momento en que alguna de estas condiciones deja de cumplirse, el actor abandona el rol de Eslabón y vuelve a desempeñar exclusivamente su rol inicial como Coordinador.

De la misma manera, un actor desempeñando un cierto tipo de eslabón puede acometer, además, el rol correspondiente a otro tipo

	de eslabón, siempre y cuando sea capaz de realizar la tarea correspondiente y haya necesidad (v. etiquetas de las transiciones aditivas). El patrón también representa un par de restricciones de multiplicidad de ligadura para indicar, por una parte, que la aparición del rol `Coordinador` en una instancia es opcional (v. restricción `{0..1}` junto al rol) y, por otra parte, que puede haber cero o más roles de tipo `EslabónIntermedio` en una instancia (v. restricción de multiplicidad `{*}` en dicho rol). Los parámetros del patrón, expresados entre ángulos dentro del modelo y en la esquina superior derecha del paquete que lo contiene, son los siguientes: `<Eslabón>`, `<Coordinador>`, `<EslabónInicial>`, `<EslabónIntermedio[i]>` y `<EslabónFinal>`. Aparece un índice al final de `EslabónIntermedio` para poder identificar cada parámetro de manera diferente en caso de una ligadura múltiple, ya que la restricción de multiplicidad lo permite.
Ejemplo	Podemos usar este patrón para modelar parcialmente la organización correspondiente a un taller de reparación de automóviles. Por lo general, existe un jefe de taller (rol `JefeTaller`) que supervisa todo el proceso. Cuando entra un coche para ser reparado, el cliente es recibido por alguien que desempeña el papel de `Recepcionista`. Una vez que éste ha tomado nota del cliente, del coche y de los problemas que el cliente observa, el coche y la nota pasa a un `Mecánico` (o más de uno) para su revisión y reparación. Por último, el mecánico pasa un parte de reparaciones a la persona encargada de cobrar al cliente (rol `Cobrador`). La Figura B.4 muestra cómo es posible construir y describir este modelo a partir del patrón CADENA DE TRABAJO. La etiqueta que define la ligadura indica los parámetros y los argumentos que dan lugar a dicha instancia. En esta instancia, existe un solo rol (`Mecánico`) que hace de `EslabónIntermedio` en el patrón. Si hubiese más de uno, cada uno de ellos estaría indexado con un valor diferente. En este caso, es necesario señalar que el número

asignado es muy importante, ya que marca el orden de intervención en la cadena de trabajo.

Patrones relacionados

En ocasiones se aplica después:

- COORDINADOR$_{(2.3.1)}$: Para especificar el rol de coordinador de la cadena de trabajo.

Antes se ha podido aplicar:

- JOINT VENTURE$_{(2.1.1)}$: Si existe secuencialidad entre los distintos subobjetivos a alcanzar por los socios.

Anexo Figuras

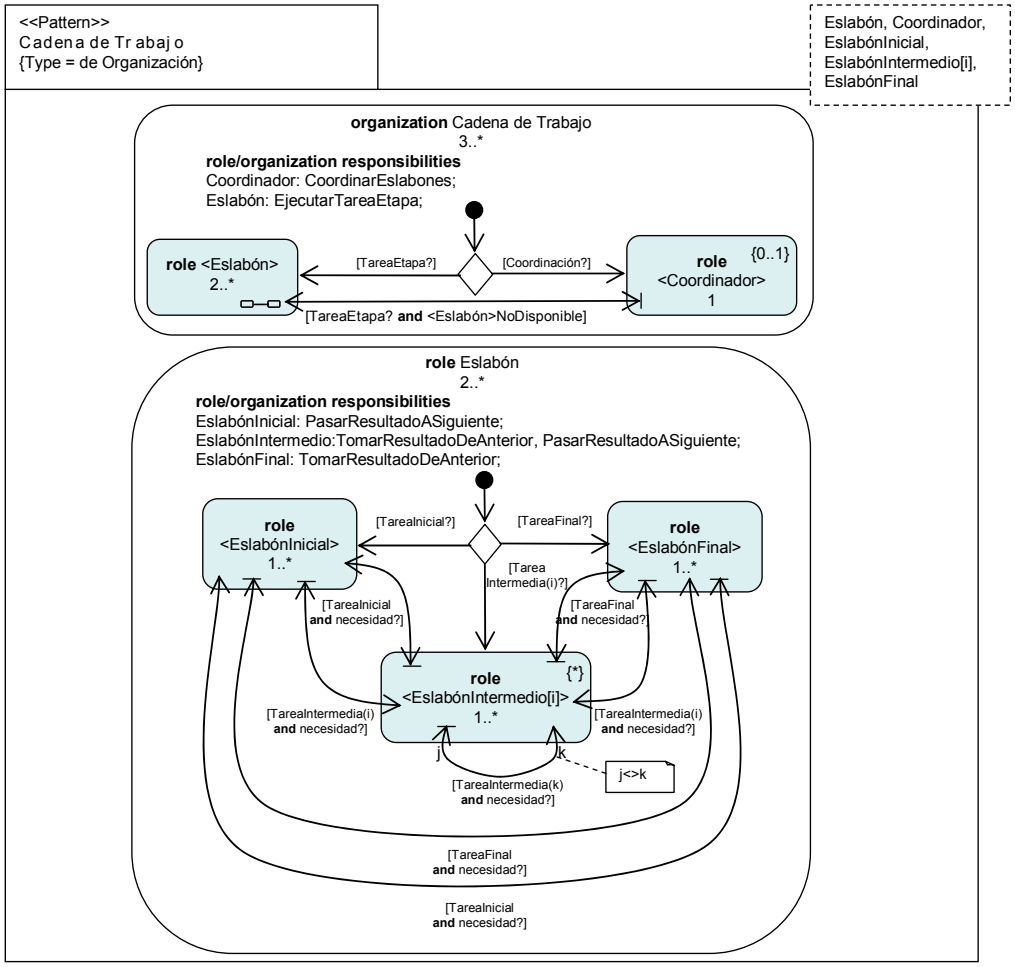


Fig. B.3: Modelado del patrón CADENA DE TRABAJO

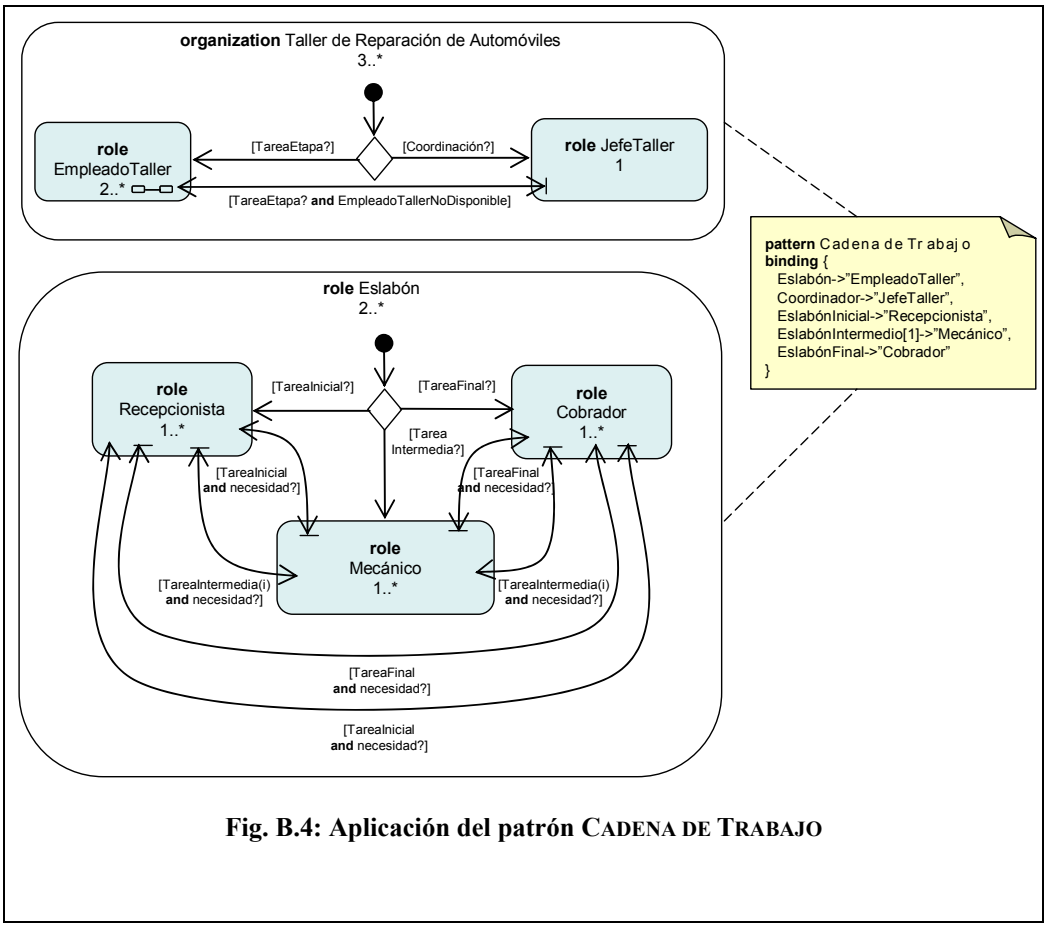


Fig. B.4: Aplicación del patrón CADENA DE TRABAJO

2.2. Patrones de equipo

2.2.1. Patrón CÍRCULO DE CALIDAD

Nombre/alias	CÍRCULO DE CALIDAD / EQUIPO DE MEJORA
Clasificación	▪ Vista: Organizacional. ▪ Fase: Especificación de Tareas. ▪ Tipo: Patrón de Equipo.
Intención	Especificar un equipo cuyos miembros colaboran para analizar los problemas propios de su actividad, elaborar soluciones y presentar dichas mejoras normalmente a la dirección. Abarcan áreas tales como la mejora de los procesos de producción, mejora de la seguridad, etc. Suele haber un miembro que actúa como coordinador, encargado de coordinar y dirigir las reuniones.
Contexto	▪ Hay una actividad cuyo objetivo es la mejora de algún aspecto de la organización. ▪ Para la realización de dicha actividad interviene un equipo propuesto para tal fin. ▪ Existe un miembro que se encarga de liderar las reuniones del equipo.
Solución	Véase Figura B.5
Explicación	Este patrón describe dos elementos básicos: ▪ La actividad a realizar por el equipo (<`ActividadCírculo`>), la cual permite mejorar algún aspecto de la organización. ▪ La especificación del equipo, incluyendo su nombre (<`CírculoDeCalidad`>) y los distintos tipos de roles que lo forman en base a la redefinición de roles “oficiales” en la

	organización, en concreto: – `Coordinador<CírculoDeCalidad>`. Es el rol del líder del equipo y lo puede desempeñar un actor que tenga el rol de `<LíderCírculo>` en la organización. – `Miembro<CírculoDeCalidad>`. Es el rol que ocupa cualquier miembro del equipo y lo puede desempeñar el actor o conjunto de actores que acometan el rol de `<MiembroCírculo>` en la organización. Los parámetros del patrón (expresados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo para especificar un escenario de trabajo concreto, son los siguientes: `<CírculoDeCalidad>`, `<LíderCírculo>`, `<MiembroCírculo>` y `<ActividadCírculo>`.
Ejemplo	Podemos usar este patrón para especificar que, por ejemplo, una Comisión de Ordenación Académica, dependiente de la Junta de Facultad de un determinado Centro, se define y comporta como un equipo de tipo Círculo de Calidad durante la celebración de sus reuniones de ordenación académica (v. actividad `SesiónReuniónOrdenaciónAcadémica`). La Figura B.6 muestra cómo es posible construir y describir este modelo a partir del patrón CÍRCULO DE CALIDAD. La etiqueta que define la ligadura indica los parámetros y los argumentos que dan lugar a dicha instancia.
Patrones relacionados	En ocasiones se aplica después: ▪ REUNIÓN$_{(2.4.2)}$: Si la actividad del equipo se realiza mediante una reunión. Antes se ha podido aplicar: ▪ PROCESO DE REUNIÓN$_{(2.4.1)}$: Si se ha descrito el proceso que conlleva la planificación, celebración y entrega de actas correspondientes a las sesiones de reunión del equipo.

Anexo Figuras

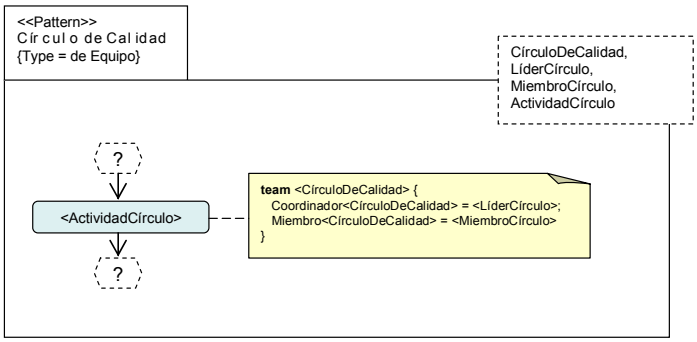


Fig. B.5: Modelado del patrón CÍRCULO DE CALIDAD

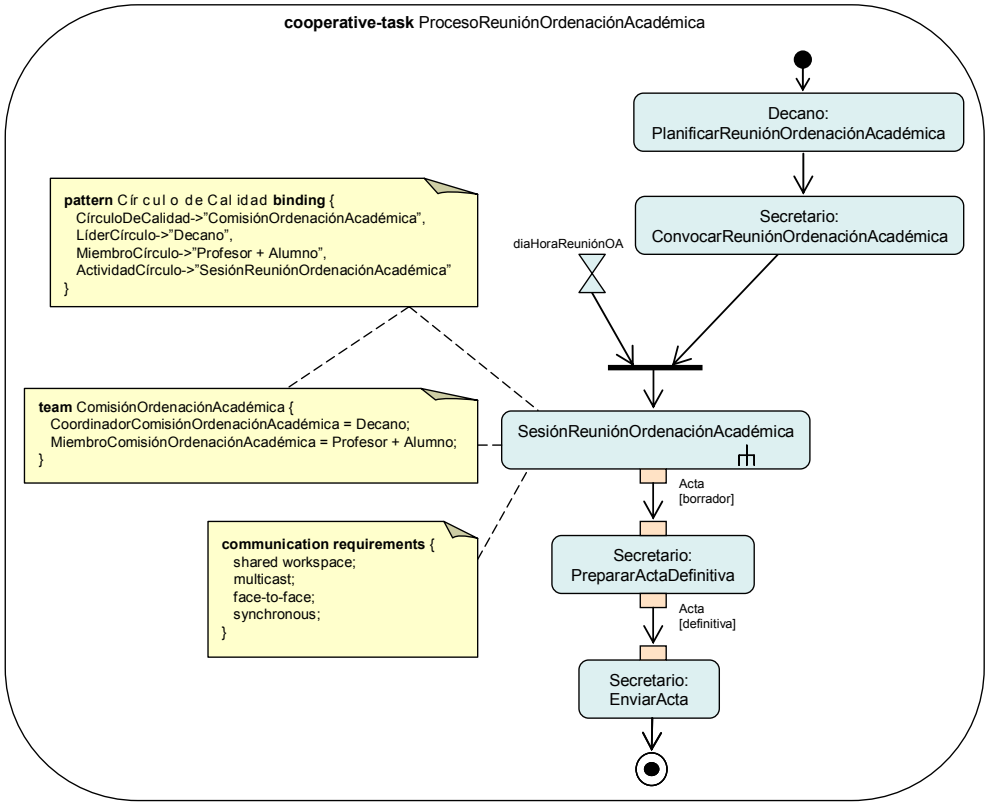


Fig. B.6: Aplicación del patrón CÍRCULO DE CALIDAD

2.3. Patrones de rol

2.3.1. Patrón COORDINADOR

Nombre/alias	COORDINADOR
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Roles. ▪ Tipo: Patrón de Rol.
Intención	Especificar el diagrama de rol que especifica las tareas comunes de un rol de Coordinador. Entre las tareas típicas de este tipo de rol se encuentran la coordinación de un grupo de trabajo, la realización de reuniones con el grupo, el reparto de tareas entre sus miembros y el seguimiento de las distintas tareas.
Contexto	▪ Hay un rol en la organización cuya responsabilidad es la coordinación, asignación y supervisión del trabajo de los distintos miembros de un grupo.
Solución	Véase Figura B.7
Explicación	Este patrón describe las tareas principales que comúnmente realiza el coordinador de un grupo de trabajo. En concreto: ▪ Tarea `<CoordinarTareas>`. Establece el momento en el que comienza y termina la realización de cada una de las tareas que permiten alcanzar el objetivo compartido, así como sus interdependencias. ▪ Tarea `<SupervisarTareas>`. Hace un seguimiento de las tareas para asegurar que éstas se realizan en el tiempo y la forma convenida. ▪ Tarea `<AsignarTareas>`. Reparte las diferentes tareas entre los miembros del grupo de trabajo dependiendo de la

	necesidad, la capacidad, la disponibilidad o el interés de cada uno de ellos. ▪ Tarea `<ReunióonCoordinación>`. Pone en contacto directo a todos los miembros del grupo para debatir y acordar cualquier asunto relacionado la coordinación del trabajo. Los parámetros del patrón (expresados entre ángulos en el diagrama y en la esquina superior derecha del paquete que lo contiene) que permitirán instanciar este modelo para especificar dicho rol dentro de un escenario de trabajo cooperativo concreto, son los siguientes: `<Coordinador>`, `<CoordinarTareas>`, `<SupervisarTareas>`, `<AsignarTareas>` y `<ReunióonCoordinación>`.
Ejemplo	Podemos usar este patrón para facilitar la especificación del rol `CoordinadorSubproyectos`, el cual es una instancia del patrón COORDINADOR en el contexto de la especificación de un sistema para la gestión de un proyecto de investigación coordinado. La Figura B.8 muestra cómo es posible construir y describir este modelo a partir del patrón. La etiqueta que define la ligadura indica los parámetros y los argumentos que dan lugar a dicha instancia. En este caso vemos como `<SupervisarTareas>` y `<AsignarTareas>`, los cuales son parámetros optativos (v. multiplicidad de ligadura `{0..1}`), no han sido utilizados para la generación de la instancia.
Patrones relacionados	Antes se ha podido aplicar: ▪ JOINT VENTURE$_{(2.1.1)}$: El rol de coordinador es uno de los protagonistas dentro de este patrón. ▪ CADENA DE TRABAJO: El rol de coordinador es uno de los protagonistas dentro de este patrón. ▪ ESTRUCTURA EN 5: El rol de coordinador es uno de los protagonistas dentro de este patrón. ▪ PIRÁMIDE: El rol de coordinador es uno de los protagonistas dentro de este patrón.

En ocasiones se aplica después:

- REUNIÓN$_{(2.4.2)}$: Para especificar la reunión de coordinación.
- PROCESO DE REUNIÓN$_{(2.4.1)}$: Para especificar el proceso que conlleva la planificación, celebración y entrega de actas correspondientes a las sesiones de reunión de coordinación.
- TORMENTA DE IDEAS: Este tipo de dinámica suele estar coordinada por un actor que juega el rol de coordinador.

Anexo Figuras (v. pág. siguiente)

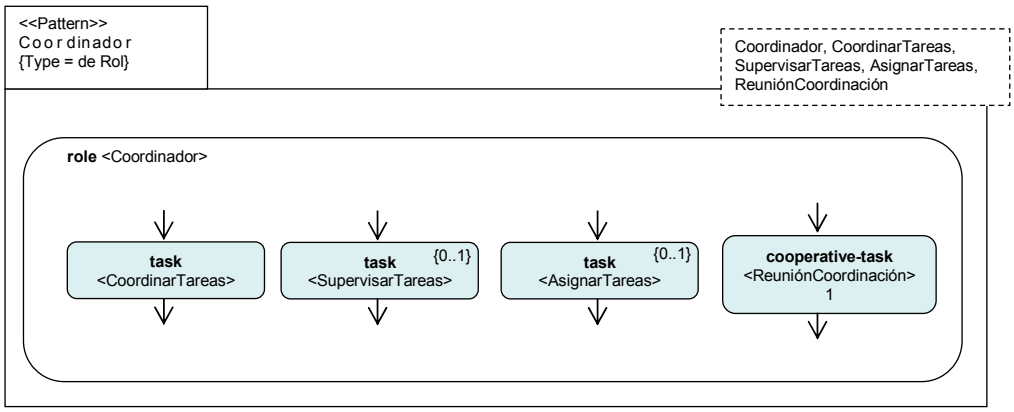


Fig. B.7: Modelado del patrón COORDINADOR

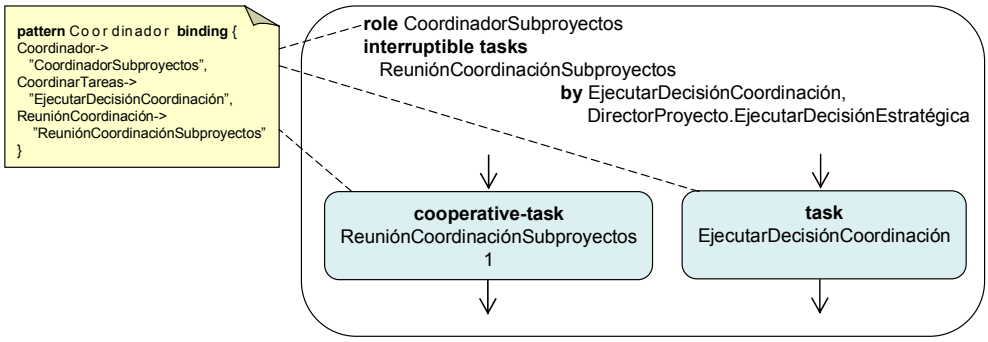


Fig. B.8: Aplicación del patrón COORDINADOR

2.4. Patrones de actividad

2.4.1. Patrón PROCESO DE REUNIÓN

Nombre/alias	PROCESO DE REUNIÓN
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Tareas. ▪ Tipo: Patrón de Actividad.
Intención	Modelar el flujo general de trabajo que conlleva el proceso de realización de una reunión. Desde su planificación por parte de un líder, hasta el envío de las actas, si las hay, una vez realizada la sesión de reunión.
Contexto	▪ Hay un grupo que necesita reunirse formalmente. ▪ Es necesario planificar la celebración de dicha reunión. ▪ Alguien convoca la reunión indicando el lugar, fecha, hora y asuntos a tratar. ▪ Cuando llega el momento se celebra la sesión de reunión. ▪ Una vez celebrada la sesión, si es necesario, la persona que convocó la reunión puede preparar un acta y enviarla a los miembros que fueron convocados.
Solución	Véase Figura B.9
Explicación	Este patrón representa un diagrama de actividades genérico que puede ser instanciado para modelar el proceso asociado con la organización de una determinada reunión. El flujo de trabajo transcurre como sigue:

	Inicialmente el líder de la reunión (rol `<LíderReunión>`) planifica el momento, lugar y orden del día de la reunión (acción `Planificar<Reunión>`). A continuación, alguien que desempeña el papel de secretario (rol `<SecretarioReunión>`) envía la convocatoria a los miembros del grupo que debe reunirse. Cuando llega el día y la hora indicado en la convocatoria (acción de aceptación de evento `<díaHoraReunión>`) se celebra la sesión de reunión (actividad `Sesión<Reunión>`) en la que se tratan los distintos puntos del orden del día. Como se indica en el diagrama, esta actividad puede ser especificada con ayuda del patrón REUNIÓN$_{(2.4.2)}$ (v. etiqueta informativa `Pattern` REUNIÓN). Una vez realizada la reunión, opcionalmente (v. la restricción de multiplicidad de ligadura indicando que el área encerrada dentro del rectángulo punteado es opcional, es decir, su multiplicidad es `{0..1}`), el actor que juega el papel de secretario prepara el acta definitiva (acción `Preparar<Acta>Definitiva`) a partir de las notas que éste tomó durante la celebración de la reunión (objeto `<Acta>` en estado `[borrador]`). Creada el acta definitiva (objeto `<Acta>` en estado `[definitiva]`), ésta se envía a los miembros convocados (acción `Enviar<Acta>`). Los parámetros del patrón (expresados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo para especificar un escenario de trabajo concreto, son los siguientes: `<Reunión>`, `<LíderReunión>`, `<SecretarioReunión>`, `<Acta>` y `<díaHoraReunión>`.
Ejemplo	Podemos usar este patrón para facilitar el modelado del proceso asociado con la organización de una reunión de coordinación de subproyectos, una de las tareas cooperativas típicas que suceden dentro del contexto de trabajo de un proyecto de investigación coordinado (v. actividad `ReuniónCoordinaciónSubproyectos`). La Figura B.10 muestra cómo es posible construir y describir este modelo a partir del patrón PROCESO DE REUNIÓN. La etiqueta que define la ligadura indica los parámetros y los argumentos que dan lugar a dicha instancia. Obsérvese que el parámetro `Acta` recibe el

	valor `$` para indicar que la instancia mantiene el mismo nombre que posee ese elemento dentro del patrón. No se le asigna un nombre específico en la instancia. El diagrama que define la actividad `ReunionCoordinacionSubproyectos` debe estar visible ya que aparecen otros elementos que no son generados a partir del patrón (p. ej. la especificación del `Equipo de Coordinación`, sus requisitos de comunicación y la ligadura del patrón REUNIÓN$_{(2.4.2)}$). En este caso se ha optado por no definir la ligadura del patrón CONVOCATORIA DE REUNIÓN en la especificación de la instancia.
Patrones relacionados	Hace uso o referencia a: ▪ REUNIÓN$_{(2.4.2)}$: Para especificar el flujo de trabajo de las sesiones de reunión. ▪ CONVOCATORIA DE REUNIÓN: Para modelar el flujo de trabajo necesario para realizar la convocatoria de la reunión. En ocasiones se aplica después: ▪ CÍRCULO DE CALIDAD$_{(2.2.1)}$: Si en la sesión de reunión participa un equipo de este tipo. ▪ ACTA DE REUNIÓN: Si la reunión genera un acta. Antes se ha podido aplicar: ▪ COORDINADOR$_{(2.3.1)}$: Planificar reuniones suele ser una de las tareas de este rol. ▪ SECRETARIO: Convocar reuniones suele ser una de las tareas de este rol.
Anexo Figuras (v. pág. siguiente)	

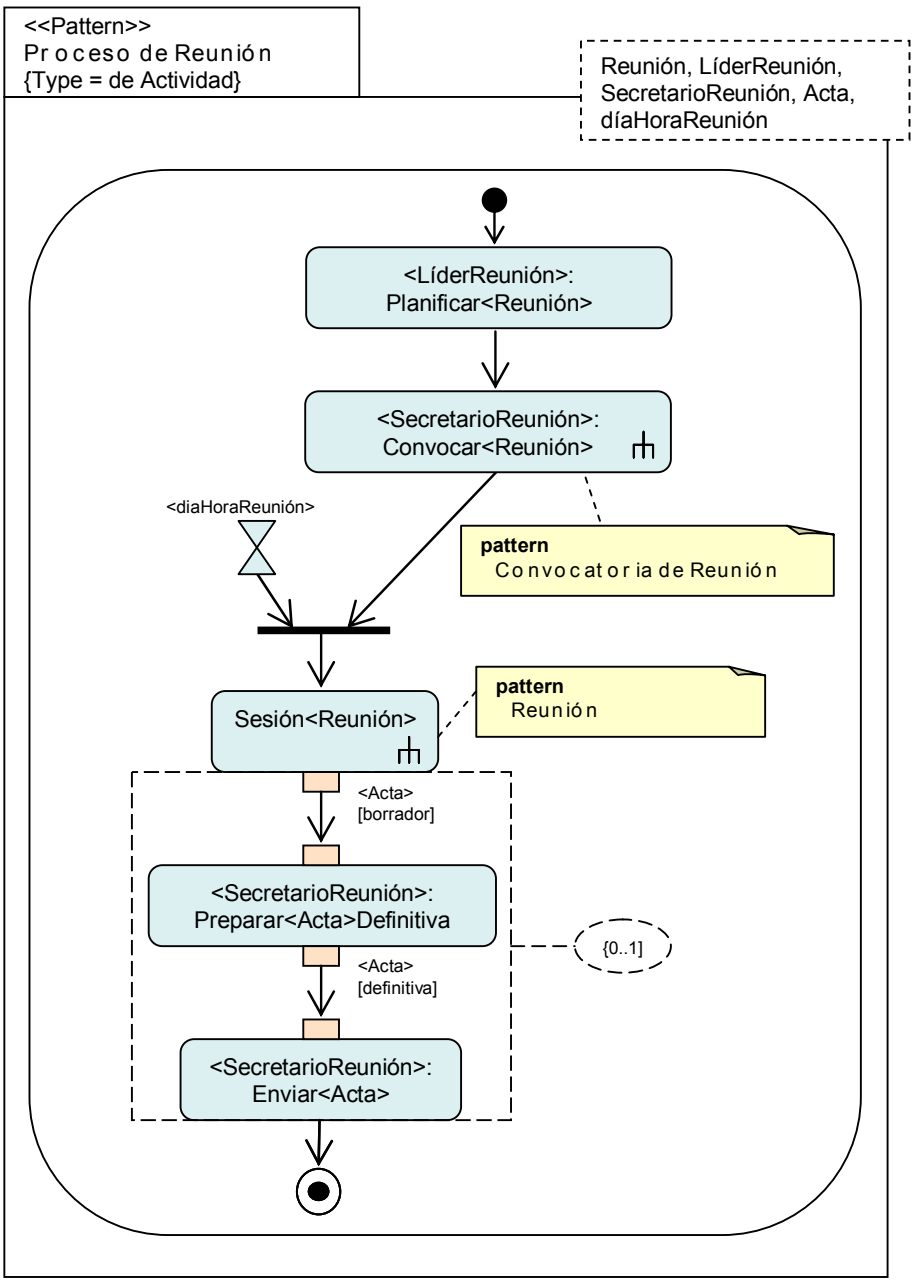


Fig. B.9: Modelado del patrón PROCESO DE REUNIÓN

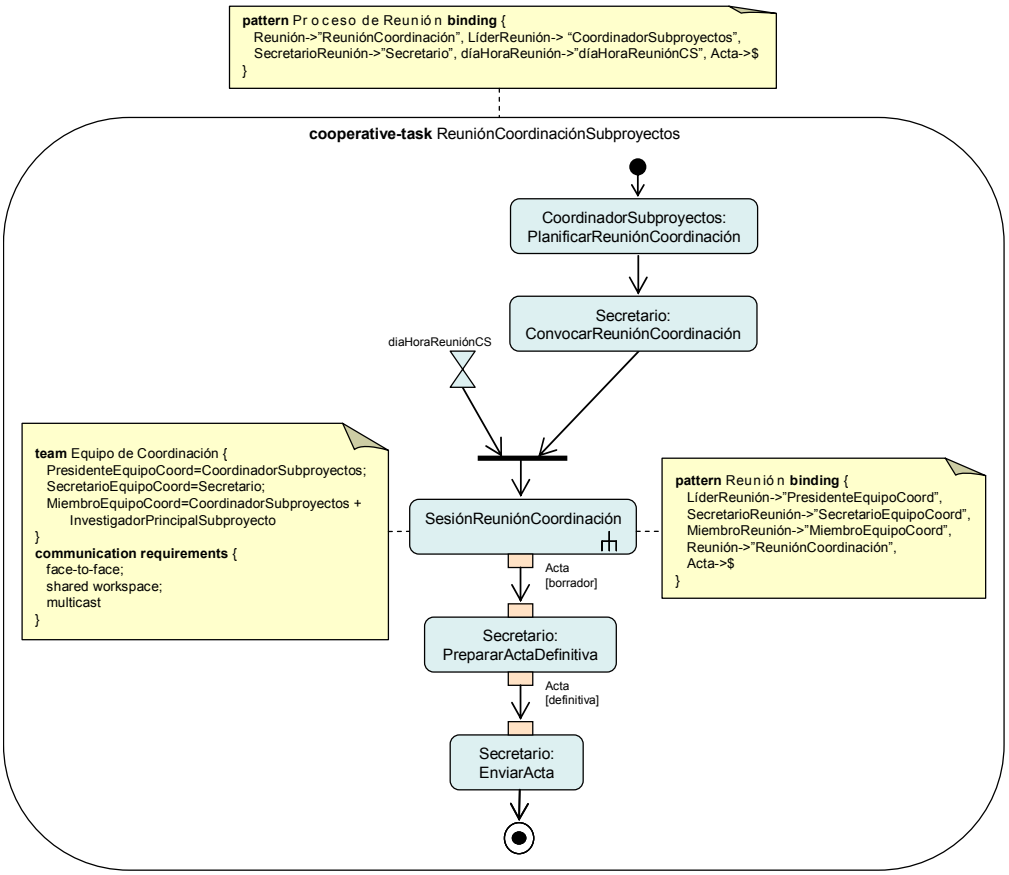


Fig. B.10: Aplicación del patrón PROCESO DE REUNIÓN

2.4.2. Patrón REUNIÓN

Nombre/alias	REUNIÓN
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Tareas. ▪ Tipo: Patrón de Actividad.
Intención	Modelar el flujo general de trabajo que comprende una sesión de reunión, en la que existe un líder que la conduce y una serie de miembros que debaten y, llegado el caso, votan los distintos puntos del orden del día. Los aspectos más relevantes de la sesión pueden ser registrados en un acta por alguien que desempeña el papel de secretario.
Contexto	▪ Hay un grupo que se reúne formalmente para tratar una serie de temas de interés común. ▪ Existe alguien que dirige la reunión, actuando como líder de la reunión. ▪ El líder se encarga de comenzar y finalizar la reunión. También abre y cierra los puntos que aparecen en el orden del día de la convocatoria. ▪ Cada vez que se abre un punto se debate y, si es necesario, se votan las decisiones a tomar. ▪ Puede haber un actor que se encargue de crear un borrador de acta que registra los eventos más importantes acontecidos en la sesión de reunión.
Solución	Véase Figura B.11
Explicación	Este patrón representa un diagrama de actividades genérico que puede ser instanciado para modelar el flujo de control y de objetos

correspondiente a una sesión de reunión concreta.

Inicialmente hay dos flujos de control concurrentes, aunque dependiendo de la instancia, uno de ellos puede utilizarse o no (v. la restricción de multiplicidad de ligadura indicando que el área encerrada dentro del rectángulo punteado es opcional, es decir, su multiplicidad es `{0..1}`).

El flujo obligatorio transcurre como sigue:

El actor que hace de líder en la reunión (rol `<LíderReunión>`) da comienzo a ésta (acción `Iniciar<Reunión>`). Seguidamente, este actor lee el primer punto y lo desarrolla (acción `AbrirPrimerPunto`). A continuación, se abre un debate (actividad `DebatirPunto`) entre todos los miembros de la reunión (rol `MiembroReunión`) y, si es necesario (condición `votar`), las decisiones relacionadas con el punto se someten a votación (actividad `VotarPunto`) con la participación de todos los que intervienen en la reunión (rol `all`). Después, el líder de la reunión da por terminado el punto que acaba de tratarse (acción `CerrarPunto`) y si hay más puntos que tratar, éste abre el siguiente punto (acción `AbrirSiguientePunto`), creándose un nuevo ciclo con su debate y votación, si procede. Este flujo termina con la finalización de la reunión por parte del líder (acción `Finalizar<Reunión>`).

Como se puede apreciar en el diagrama, el patrón informa de que la acción `DebatirPunto` puede ser modelada a partir del patrón DEBATE MODERADO(2.6.1) o del patrón DEBATE NO MODERADO(2.6.2). Así, podríamos elegir entre usar uno u otro, o bien especificar una ligadura dinámica con los dos patrones, indicando de esta forma que es el propio grupo el que elige su patrón de trabajo en tiempo real. Del mismo modo, el patrón informa de que la acción `VotarPunto` puede ser modelada a partir del patrón VOTACIÓN(2.4.3).

Respecto al flujo optativo concurrente al anterior:

Un actor, desempeñando el rol de secretario de la reunión (rol `<SecretarioReunión>`), puede abrir un acta (acción `Abrir<Acta>`) para, posteriormente, registrar en ésta los

	acontecimientos que suceden (acción `TomarNotas`). La primera acción genera un objeto de tipo `<Acta>` con el estado compuesto `[borrador, abierta]` que fluye hasta la segunda acción, la cual tiene un pin que representa la salida del objeto `<Acta>` con el estado compuesto `[borrador, anotada]`. Terminados ambos flujos paralelos, opcionalmente y siempre que el flujo optativo anterior aparezca, el actor que actúa como secretario (rol `<SecretarioReunión>`) cierra el acta (acción `Cerrar<Acta>`) y se devuelve ésta en el estado `[borrador, cerrada]` a la acción que invocó la actividad de reunión, a través de su parámetro de salida. Los parámetros del patrón (expresados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo para especificar un escenario de reunión concreto, son los siguientes: `<LíderReunión>`, `<SecretarioReunión>`, `<MiembroReunión>`, `<Reunión>` y `<Acta>`. Hay que tener en cuenta que, según la restricción de multiplicidad de ligadura especificada en el patrón, los parámetros `<Acta>` y `<SecretarioReunión>` son optativos y éstos, si se ligan, se ligan conjuntamente.
Ejemplo	Podemos usar este patrón para facilitar el modelado de una sesión de reunión concreta, como por ejemplo, la que se celebra entre los investigadores principales de los distintos subproyectos que componen un proyecto de investigación coordinado (actividad `SesiónReuniónCoordinación`). En las reuniones suele haber un líder, normalmente el coordinador de subproyectos, que actúa como presidente del equipo de coordinación (rol `PresidenteEquipoCoord`) y alguien que hace de secretario (rol `SecretarioEquipoCoord`). La Figura B.12 muestra cómo es posible construir y describir este modelo a partir del patrón REUNIÓN. En este caso, la instancia refleja la ligadura de los parámetros opcionales `<Acta>` y `<Secretario>`, dejando claro que las decisiones serán reflejadas

	en un acta. Además, se puede ver que la actividad `DebatirPunto` se describe mediante una ligadura dinámica concreta de los patrones DEBATE MODERADO(2.6.1) y DEBATE NO MODERADO(2.6.2). También la actividad `VotarPunto` se define a partir de una ligadura específica del patrón VOTACIÓN(2.4.3). Obsérvese que el parámetro `Acta` recibe el valor `$` para indicar que la instancia mantiene el mismo nombre que posee ese elemento en el patrón. No se le asigna un nombre específico en la instancia.
Patrones relacionados	Hace uso o referencia a: ▪ VOTACIÓN(2.4.3): Para modelar la actividad `VotarPunto`. ▪ DEBATE NO MODERADO(2.6.2): Para modelar la actividad `DebatirPunto`. ▪ DEBATE MODERADO(2.6.1): Para modelar la actividad `DebatirPunto`. En ocasiones se aplica después: ▪ ACTA DE REUNIÓN: Si la reunión genera un acta. ▪ TURNO DE PALABRA: Si alguna actividad (por ejemplo `DebatirPunto`) necesita que haya un moderador para la asignación del turno de palabra a los que deseen intervenir. Antes se ha podido aplicar: ▪ TORMENTA DE IDEAS: Si se han tenido que generar abundantes ideas sobre un determinado asunto que va a ser tratado en la reunión. ▪ CÍRCULO DE CALIDAD(2.2.1): Si en la sesión de reunión participa un equipo de este tipo. ▪ CONVOCATORIA DE REUNIÓN: Si se ha modelado el flujo de trabajo que define la convocatoria de la reunión ▪ PROCESO DE REUNIÓN(2.4.1): Si se ha descrito el proceso que conlleva la planificación y celebración de la reunión, así como la generación de actas.
Anexo Figuras (v. pág. siguiente)	

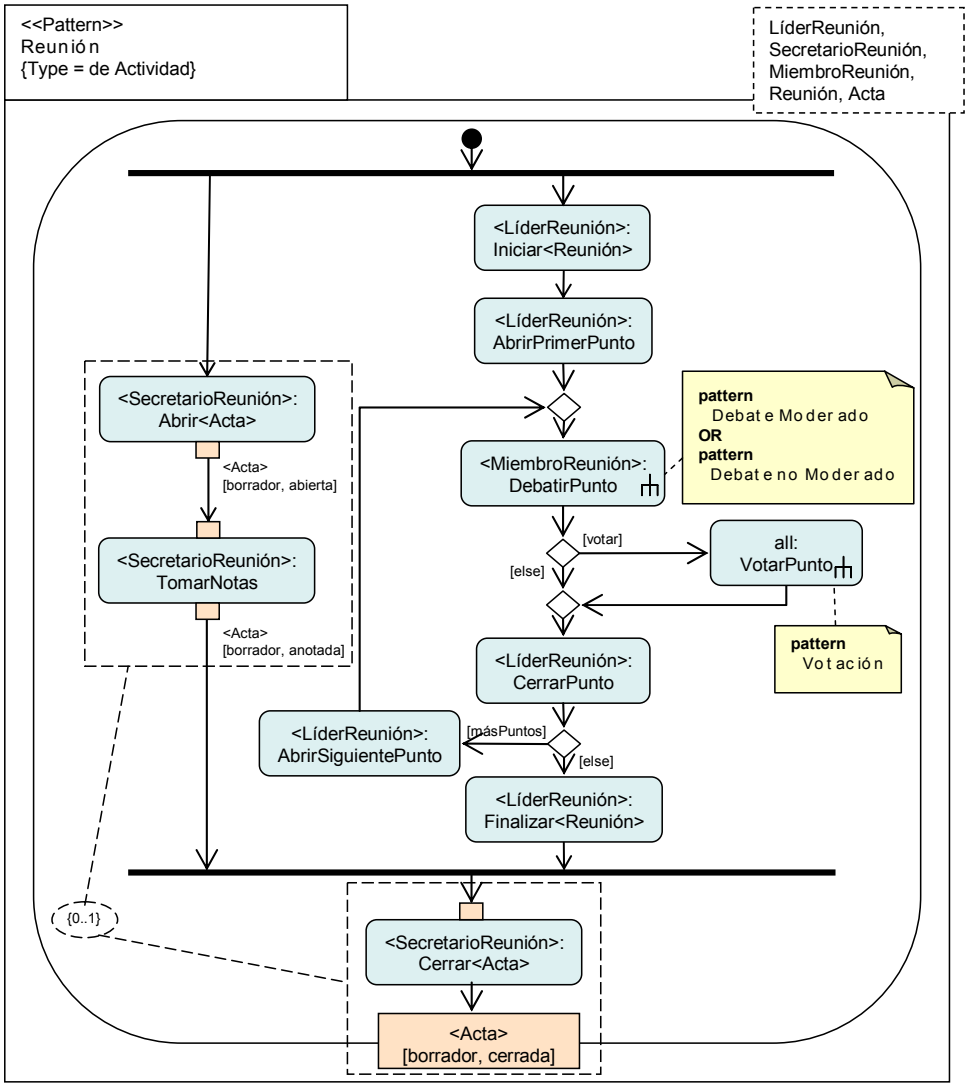


Fig. B.11: Modelado del patrón REUNIÓN

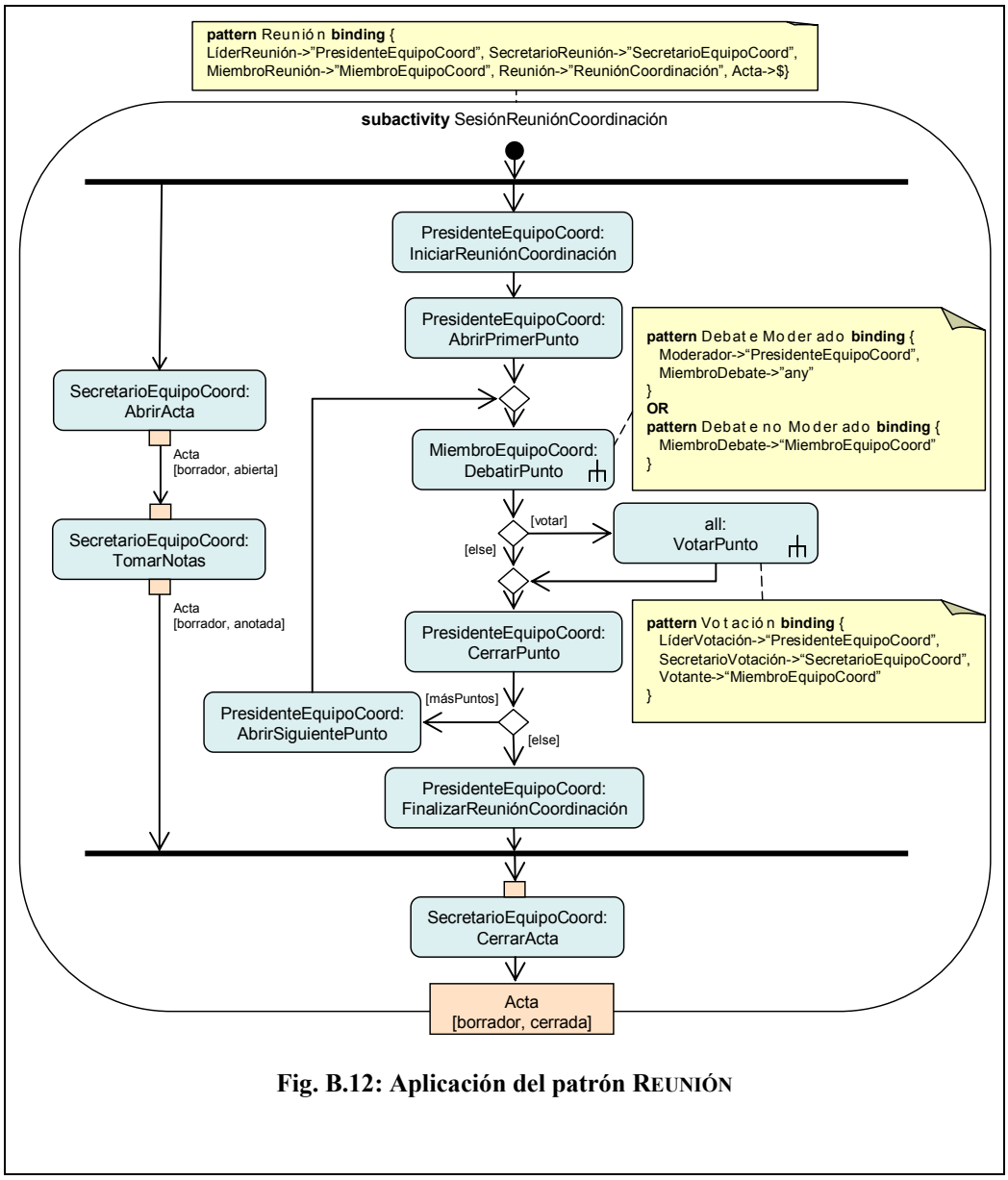


Fig. B.12: Aplicación del patrón REUNIÓN

2.4.3. Patrón VOTACIÓN

Nombre/alias	VOTACIÓN
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Tareas. ▪ Tipo: Patrón de Actividad.
Intención	Modelar el flujo general de trabajo que comporta una sesión de votación, desde que alguien, actuando como líder o secretario, informa de las alternativas posibles de voto, hasta que éste informa del resultado obtenido en la votación.
Contexto	▪ Hay un grupo que necesita votar para tomar una decisión. ▪ Existe alguien que dirige la votación, actuando como líder de la votación. ▪ El líder se encarga de comenzar y finalizar la votación. También puede informar de las opciones de voto y del resultado de la votación. ▪ Hay un actor que se encarga de solicitar los votos a los participantes, registrar los votos y contabilizarlos. Este actor, que asume el rol de secretario, también puede informar de las opciones de voto y del resultado de la votación. ▪ La petición de voto y su respuesta se puede hacer votante por votante o de manera conjunta.
Solución	Véase Figura B.13
Explicación	Este patrón representa un diagrama de actividades genérico que puede ser instanciado para modelar el flujo de control y de objetos correspondiente a una sesión de votación. El flujo transcurre como sigue:

El actor que hace de líder en la votación (rol `<LíderVotación>`) da comienzo a ésta (acción `IniciarVotación`). Seguidamente, este actor o el que hace de secretario (rol `<SecretarioVotación>`) informa a los votantes acerca de las distintas opciones a votar (actividad `InformarOpcionesVotación`). Como se puede comprobar, esta actividad queda definida por medio del patrón EXPOSICIÓN$_{(2.6.5)}$, especificando que el rol de `Ponente` en el patrón puede ser desempeñado exclusivamente por el líder o el secretario (v. en expresión de ligadura `Ponente -> "<SecretarioVotación> x <LíderVotación>"`) y el parámetro `Tema` recibe el valor `"OpcionesVotación"` (v. en expresión de ligadura `Tema -> "OpcionesVotación"`).

A continuación, para cada una de las opciones a votar y dependiendo de si se va a solicitar el voto uno a uno (condición `[solicitarVotoIndividualmente]`) o si se va a hacer conjuntamente (condición `[solicitarVotoConjuntamente]`), en el primer caso se realiza la actividad `EmitirvotoIndividuo` por cada uno de los votantes (v. condición `[másVotantes]`), y en el segundo, se ejecuta una sola vez la actividad `EmitirVotoGrupo`. Como se puede ver, la primera actividad es una instancia del patrón PETICIÓN-RESPUESTA SIMPLE$_{(2.6.3)}$ (v. expresión de ligadura en etiqueta conectada) y la segunda es una instancia del patrón PETICIÓN-RESPUESTA MÚLTIPLE$_{(2.6.4)}$ (v. expresión de ligadura en etiqueta conectada). Es preciso advertir que estas expresiones están también parametrizadas de modo que, cuando se instancie el patrón VOTACIÓN, éstas poseerán los valores necesarios para que dichas expresiones de ligadura estén bien construidas y definan las instancias deseadas.

Cuando ya no hay más propuestas que votar, el que hace de líder (rol `<LíderVotación>`) o de secretario (rol `<SecretarioVotación>`) informa a los votantes acerca de los resultados de la votación (actividad `InformarResultadoVotación`). Esta actividad también queda definida por medio del patrón EXPOSICIÓN$_{(2.6.5)}$, especificando que el rol de `Ponente` en el patrón puede ser desempeñado

	exclusivamente por el líder o el secretario (v. en expresión de ligadura `Ponente -> "<SecretarioVotación> x <LíderVotación>"`) y el parámetro `Tema` recibe el valor `"ResultadoVotación"` (v. en expresión de ligadura `Tema -> "OpcionesVotación"`). Los parámetros del patrón (expresados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), las cuales permitirán instanciar este modelo para especificar un escenario de votación concreto, son los siguientes: `<LíderVotación>`, `<SecretarioVotación>` y `<Votante>`.
Ejemplo	Podemos usar este patrón para facilitar el modelado de una sesión de votación concreta, como por ejemplo, la que permite especificar la actividad `VotarPunto` que aparece en la Figura B.12., dentro de la descripción del patrón REUNIÓN(2.4.2). La Figura B.14 muestra dicha actividad construida y descrita a partir del patrón. No obstante, no es necesario desplegar dicha actividad, ya que el patrón define totalmente el diagrama. Bastaría simplemente con conectar la etiqueta de ligadura directamente con el símbolo que referencia dicha actividad. Esta práctica permite reducir considerablemente la complejidad de los modelos.
Patrones relacionados	Hace uso o referencia a: ▪ EXPOSICIÓN(2.6.5): Para modelar las actividades `InformarOpcionesVotación` e `InformarResultado Votación`. ▪ PETICIÓN-RESPUESTA SIMPLE(2.6.3): Para modelar la actividad `EmitirVotoIndividuo`. ▪ PETICIÓN-RESPUESTA MÚLTIPLE(2.6.4): Para modelar la actividad `EmitirVotoGrupo`. En ocasiones se aplica después: ▪ ACTA DE REUNIÓN: Si el resultado de la votación se almacena en un documento.

Antes se ha podido aplicar:

- TORMENTA DE IDEAS: Si se han generado abundantes ideas sobre un determinado asunto y hay que someter a votación algunas de éstas.

Anexo Figuras

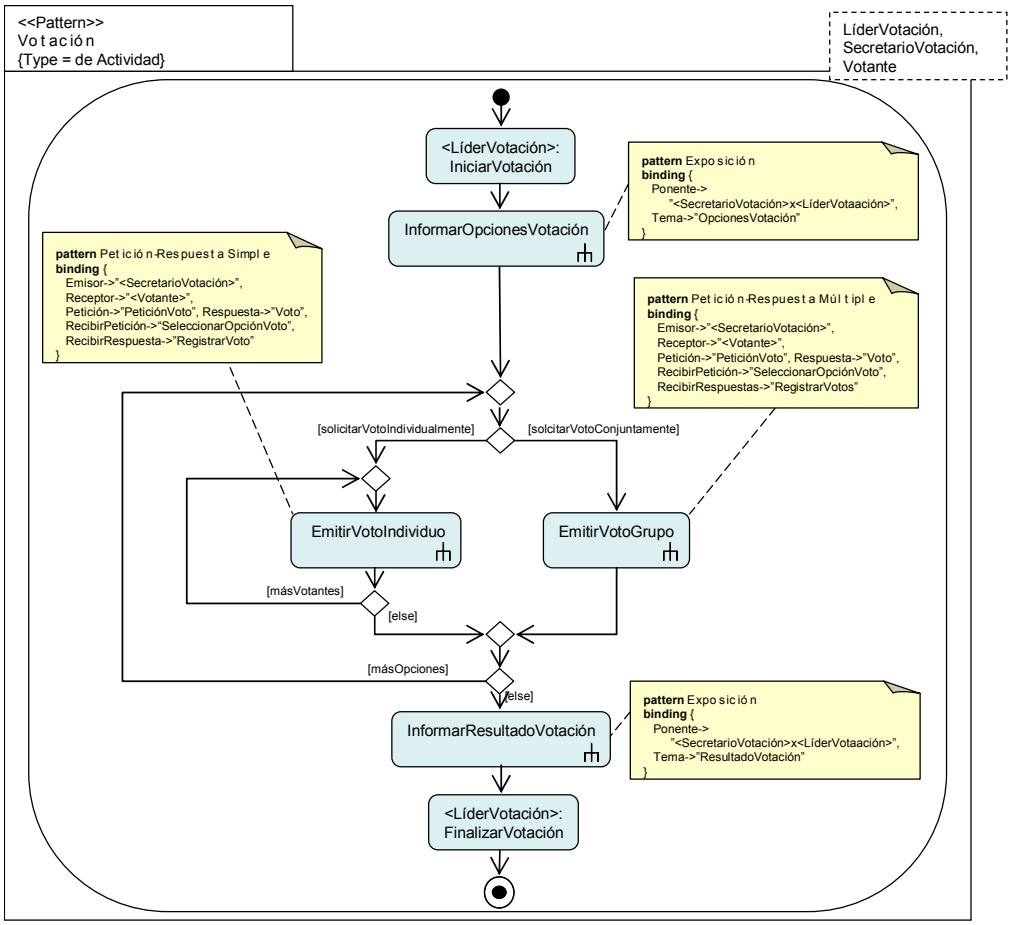


Fig. B.13: Modelado del patrón VOTACIÓN

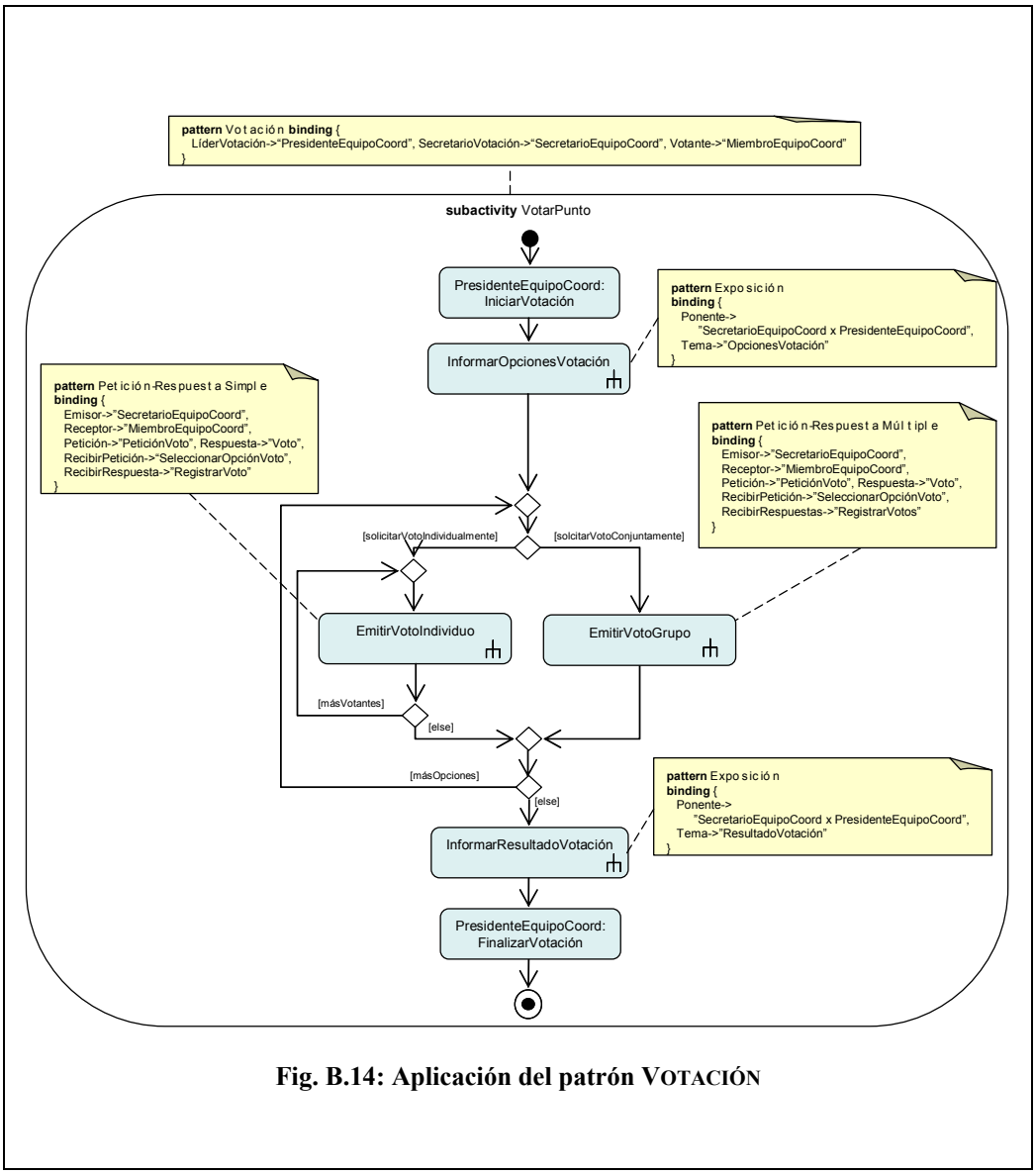


Fig. B.14: Aplicación del patrón VOTACIÓN

2.4.4. Patrón NEGOCIACIÓN NO MODERADA

Nombre/alias	NEGOCIACIÓN NO MODERADA
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Tareas. ▪ Tipo: Patrón de Actividad.
Intención	Modelar el flujo general de trabajo que define una negociación no moderada, es decir, no existe un moderador responsable de coordinar las intervenciones. Se modela desde que los negociadores plantean sus necesidades, hasta que entre éstos se acuerda una propuesta o se cierra la negociación sin éxito.
Contexto	▪ Hay un conflicto de intereses entre dos o más personas. ▪ La negociación la emprende cualquiera de los interesados. ▪ Cada negociador plantea inicialmente sus necesidades para que, en consecuencia, puedan emitirse las propuestas/contraofertas por las distintas partes que intervienen. ▪ Cualquiera puede hacer una propuesta. ▪ Las propuestas se valoran por parte de todos los negociadores y se aceptan o se rechazan. ▪ Puede llegar un momento en el que ninguna propuesta ha sido satisfactoria, cerrándose la negociación.
Solución	Véase Figura B.15
Explicación	Este patrón representa un diagrama de actividades genérico que puede ser instanciado para modelar el flujo de control y de objetos correspondiente a una sesión de negociación.

	El flujo transcurre como sigue: Cualquiera de los actores participantes (rol `any`) puede iniciar la actividad de negociación (acción `AbrirNegociación`). Abierta la negociación, todos y cada uno de los negociadores (rol `<Negociador>`) plantean sus necesidades (acción `PlantearNecesidades`). Después, cualquiera (rol `any`) puede hacer una propuesta (acción `Hacer<Propuesta>`), sometiéndose ésta a evaluación (acción `Evaluar<Propuesta>`) por parte de todos (rol `all`). Para ello, la propuesta (objeto `<Propuesta>` en el estado `[borrador]`) fluye hasta el extremo de entrada del nodo que representa la acción de evaluación. La acción `Evaluar<Propuesta>` tiene dos parámetros de salida alternativos, devolviendo la propuesta aceptada (pin de salida etiquetado como `<Propuesta> [aceptada]`) o la propuesta rechazada (pin de salida etiquetado como `<Propuesta> [rechazada]`). En caso de que la propuesta sea aceptada (condición `[propuesta aceptada]`), ésta será acordada (acción `Acordar<Propuesta>`) por todos (rol `all`). En otro caso, si se desea que continúe la negociación (condición [`seguir negociando]`) se volverán a hacer nuevas propuestas y se repetirá el ciclo. Si no hay más propuestas o no se quiere seguir negociando cualquiera de los presentes (rol `any`) puede cerrar la negociación (acción `CerrarNegociación`). Si hay más propuestas se repetirá el ciclo. Los parámetros del patrón (expresados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo para especificar un escenario de negociación concreto, son los siguientes: `<Negociador>` y `<Propuesta>`.
Ejemplo	Podemos usar este patrón para facilitar el modelado de una sesión de negociación concreta, por ejemplo, la que usan los alumnos que participan en una estrategia de aprendizaje cooperativo, denominada Jigsaw (rompecabezas), para repartirse las subtareas en las que se divide la tarea que deberán afrontar conjuntamente.

	La Figura B.16 muestra dicha actividad construida y descrita a partir del patrón. No obstante, no es necesario desplegar dicha actividad, ya que el patrón define totalmente el diagrama. Bastaría simplemente con conectar la etiqueta de ligadura directamente con el símbolo que referencia dicha actividad. Esta práctica permite reducir considerablemente la complejidad de los modelos.
Patrones relacionados	En ocasiones se aplica después: ▪ ACTA DE REUNIÓN: Si el resultado de la negociación se almacena en un documento. ▪ DEBATE NO MODERADO$_{(2.6.2)}$: Puede ser necesario si desplegamos la actividad `EvaluarElecciónSubtareas`. Antes se ha podido aplicar: ▪ TORMENTA DE IDEAS: Si se han generado abundantes ideas sobre un determinado asunto y hay que negociar algunas de éstas.
Anexo Figuras (v. pág. siguiente)	

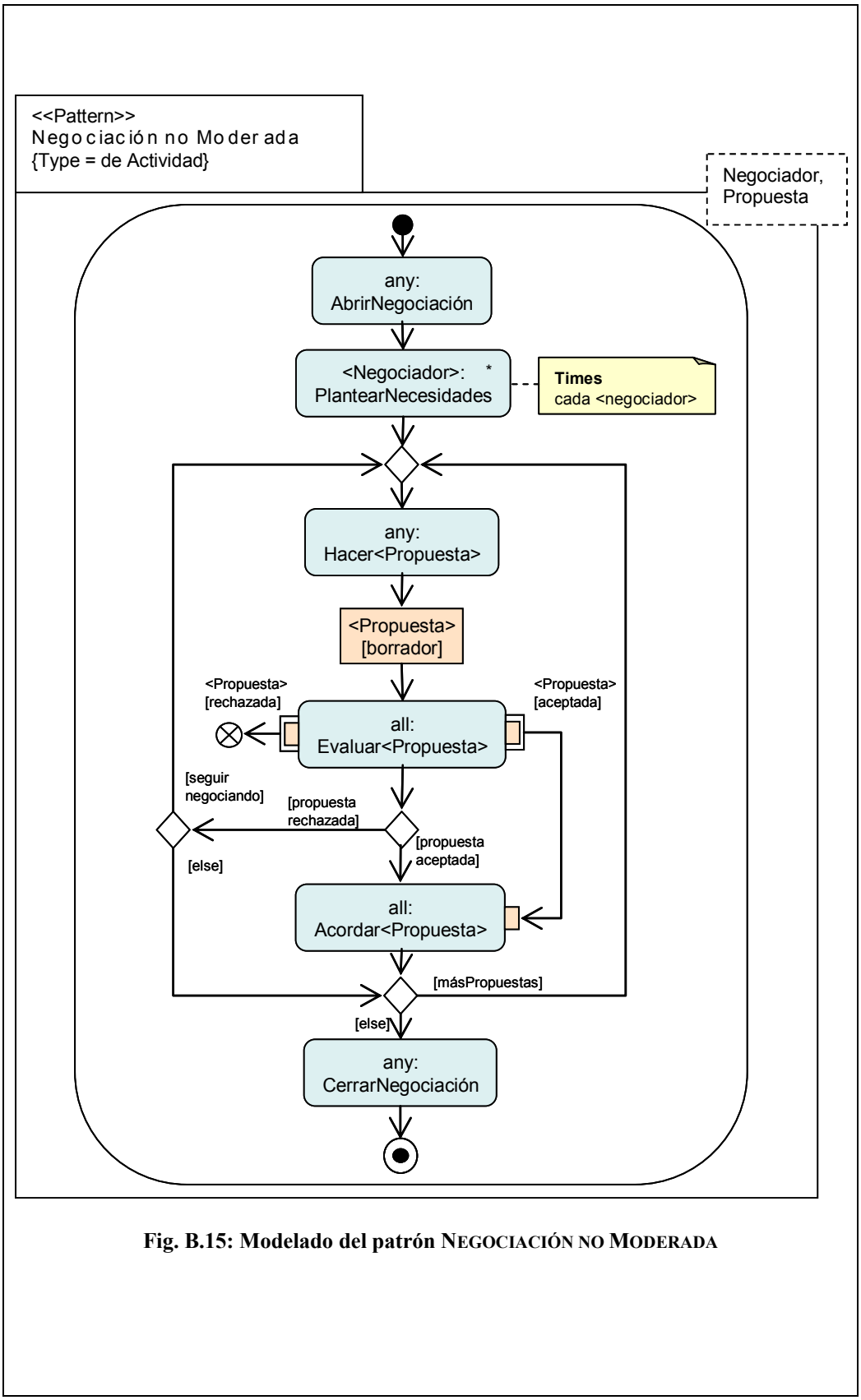


Fig. B.15: Modelado del patrón NEGOCIACIÓN NO MODERADA

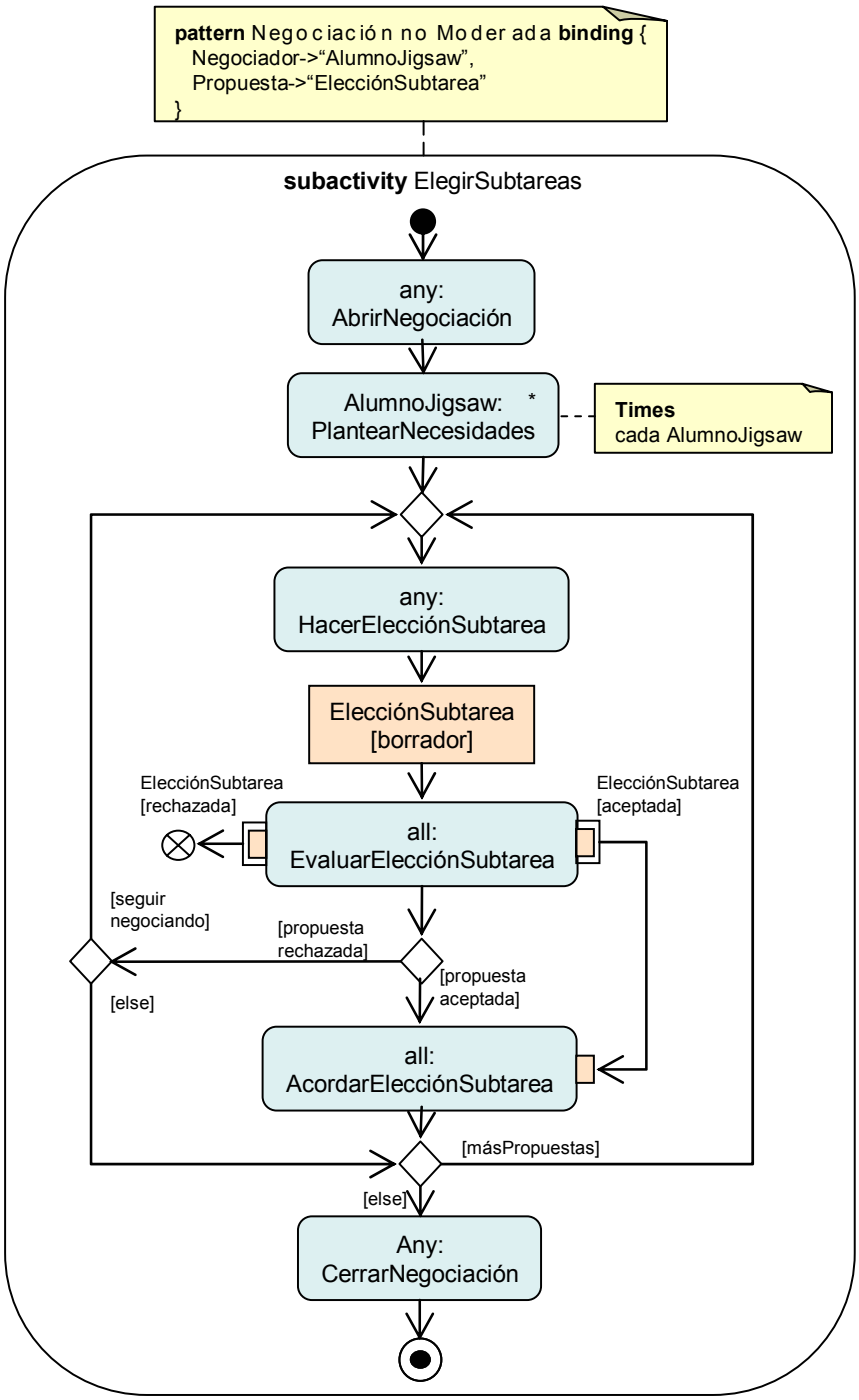


Fig. B.16: Aplicación del patrón NEGOCIACIÓN NO MODERADA

2.5. Patrones de coordinación

2.5.1. Patrón PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO

Nombre/alias	PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Tareas, Especificación de Roles. ▪ Tipo: Patrón de Coordinación.
Intención	Modelar una estructura de coordinación en la que existe un actor (productor) que genera un producto y un actor (consumidor) que lo consume. La acción de producir o consumir debe terminar antes de volver a producir o consumir, respectivamente, otro producto.
Contexto	▪ Existe un actor (el productor) que va generando productos y otro actor (el consumidor) que va haciendo uso de éstos. ▪ El ritmo de producción puede ser diferente al de consumición, con lo que el productor puede estar parado mientras que el consumidor está trabajando o viceversa. ▪ Los productos generados por el productor se van depositando en un lugar accesible al consumidor para su consumición. ▪ Tanto el productor como el consumidor pueden, respectivamente, dejar de producir o consumir cuando quieran. ▪ La producción de un nuevo producto no comienza hasta que no se haya terminado de producir el anterior. ▪ La consumición de un nuevo producto no comienza hasta que no se haya terminado de consumir el anterior.
Solución	Véase Figura B.17

Explicación

Este patrón se corresponde con el fragmento genérico de un diagrama de actividades que permite modelar, dentro de un escenario de trabajo cooperativo concreto, el mecanismo de coordinación especificado.

La coordinación se lleva a cabo como sigue:

Cuando el actor productor (rol `<Productor>`) ha terminado la generación de un producto (acción `<Producir><Producto>`) se envían dos tokens, uno por cada extremo de salida. Por un lado, un token de objeto, el cual representa al producto, es enviado y almacenado en el nodo de objeto `<Producto>`, a la espera de que el consumidor lo consuma. Por otro lado, un token de control fluye hasta el nodo de control (fork), duplicándolo y permitiendo que el productor vuelva a producir otro producto si lo desea (condición `[<producir>Más]`), al mismo tiempo que, si es la primera vez (condición `[primeraVez]`), se pase el control al actor consumidor (rol `<Consumidor>`) para que consuma el producto (acción `<Consumir><Producto>`). La acción de consumición requiere que haya un token de objeto en el nodo `<Producto>` y un token de control en el otro extremo de entrada. Después de consumir el producto, el consumidor puede volver a consumir otro siempre que quiera consumir más (condición `[<consumir>Más]`) y exista algún otro producto que consumir, es decir, algún token en el nodo de objeto `<Producto>`. En dicho nodo podría haber más de un producto esperando para ser consumido, ya que los nodos de objeto tienen semántica de buffer, es decir, en ellos pueden residir tokens de objeto mientras esperan a ser aceptados por otros nodos.

En el diagrama también aparecen elementos de tipo `<<UncertainElement>>` (representados con un hexágono punteado incluyendo un símbolo de interrogación). Como explicamos en el profile PMP (v. Cap. IV), en ocasiones es necesario simbolizar aquellos elementos indeterminados del contexto, fronterizos al patrón, que se conectarán con las instancias para que estén bien formadas.

Los parámetros del patrón (expresados entre ángulos en el modelo

	y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo para especificar esta estructura de coordinación dentro de un escenario de trabajo concreto, son los siguientes: `<Productor>`, `<Consumidor>`, `<Producto>`, `<Producir>` y `<Consumir>`.
Ejemplo	Podemos usar este patrón para facilitar la construcción del modelo que permite reflejar la coordinación que se establece entre un/a profesor/a (rol `Profesor`) que va cumplimentando actas (acción `EscribirActa`) y a un/a director/a (rol `Director`) que las va validando (acción `ValidarActa`). El/la profesor/a no comienza la escritura de un nuevo acta hasta que no ha terminado con la anterior y el/la director/a no comienza a validar un nuevo acta sin haber terminado la validación de la anterior. La Figura B.18 muestra la ligadura concreta que permite modelar y describir dicha coordinación. Obsérvese también cómo los elementos `<<UncertainElement>>` conectan el diagrama de la instancia con ciertos elementos de su entorno.
Patrones relacionados	Los siguientes patrones son similares (resuelven problemas semejantes, pero sus contextos de aplicación son diferentes): ▪ PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO$_{(2.5.2)}$: Aplicable cuando la producción y la consumición entre el productor y el consumidor es continua, no termina. ▪ PRODUCTOR-CONSUMIDOR MÚLTIPLE DISCONTINUO: Aplicable cuando existen varios productores y varios consumidores. Los actores no comienzan la producción o consumición de un nuevo producto antes de terminar de producir o consumir, respectivamente, el anterior producto. ▪ PRODUCTOR-CONSUMIDOR MÚLTIPLE CONTINUO: Aplicable cuando existen varios productores y varios consumidores. La producción y la consumición son continuas, no terminan.
Anexo Figuras (v. pág. siguiente)	

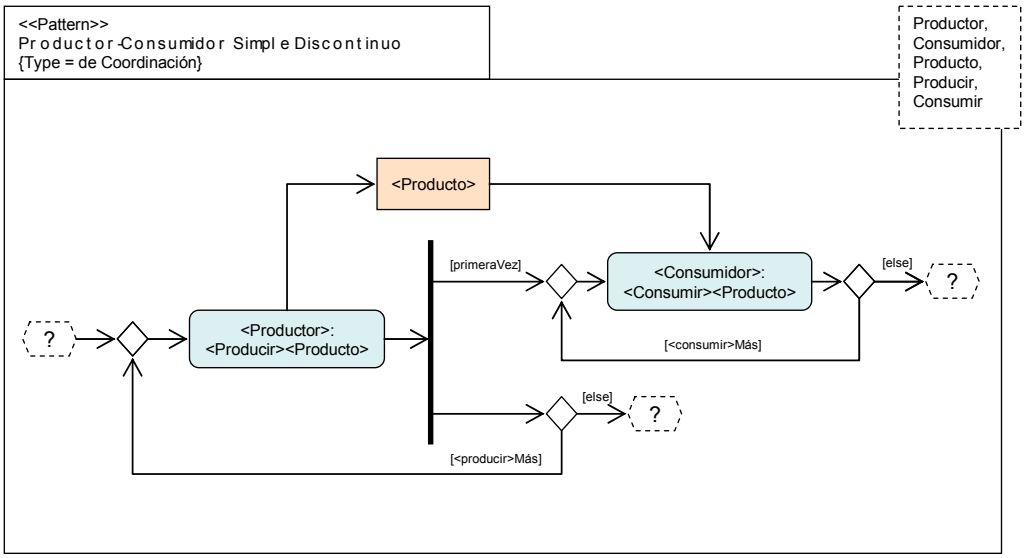


Fig. B.17: Modelado del patrón PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO

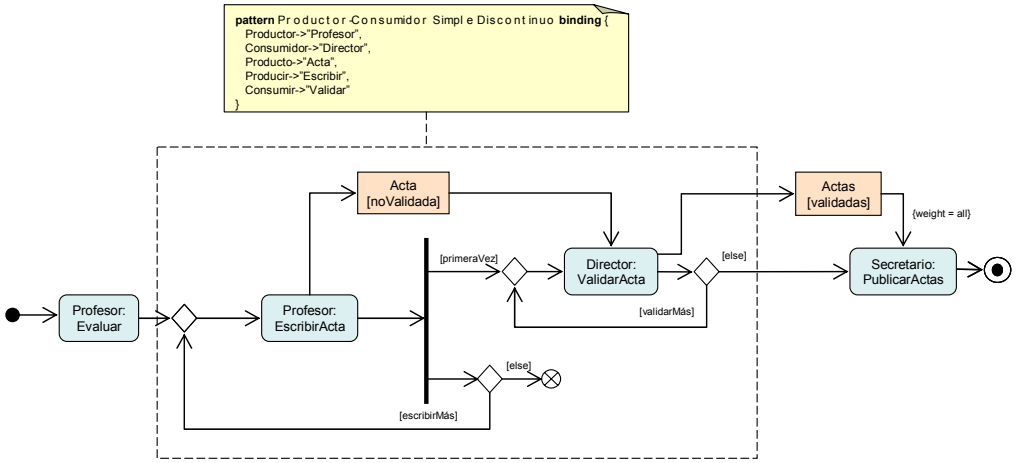


Fig. B.18: Aplicación del patrón PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO

2.5.2. Patrón PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO

Nombre/alias	PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Tareas, Especificación de Roles. ▪ Tipo: Patrón de Coordinación.
Intención	Modelar una estructura de coordinación en la que existe un actor (productor) que genera un producto y un actor (consumidor) que lo consume. La producción y la consumición es un flujo continuo. No es necesario que termine la producción de un producto para comenzar la producción del siguiente, al igual que no es necesario que finalice la consumición de un producto para empezar a consumir el siguiente.
Contexto	▪ Existe un actor (el productor) que va generando productos y otro actor (el consumidor) que los va consumiendo. ▪ El ritmo de producción puede ser diferente al de consumición. ▪ Los productos generados por el productor se van depositando en un lugar accesible al consumidor para su consumición. ▪ El productor y el consumidor están produciendo y consumiendo continuamente. Inician la producción o consumición de un nuevo producto sin haber acabado de producir o consumir el anterior.
Solución	Véase Figura B.19
Explicación	Este patrón se corresponde con el fragmento genérico de un diagrama de actividades que permite modelar, dentro de un escenario de trabajo cooperativo concreto, el mecanismo de coordinación especificado.

	El patrón ofrece dos modelos alternativos y equivalentes (v. opciones A y B en el diagrama del patrón). La diferencia reside únicamente en la notación utilizada. El modelador podrá usar cualquiera de las dos opciones para generar sus instancias. Ambos comparten los mismos parámetros. La capacidad de expresión de UML 2 permite modelar fácilmente este estilo de coordinación. En el diagrama aparece la acción de producción (acción `<Producir><Producto>`), realizada por el actor que desempeña el papel de `<Productor>`, y la acción de consumición (acción `<Consumir><Producto>`), realizada por el actor que desempeña el papel de `<Consumidor>`. El pin de salida de la acción de producción, se conecta a través de un flujo continuo de objeto (stream) con el pin de entrada de la acción de consumición. Este tipo de flujo se puede especificar mediante una restricción `{stream}` junto al nodo de salida o rellenando de negro los pins que intervienen. El streaming permite que la ejecución de una acción tome entradas y provea salidas mientras se está ejecutando, pudiendo aceptar y producir múltiples tokens durante una misma ejecución. En el diagrama también aparecen elementos de tipo `<<UncertainElement>>` (representados con un hexágono punteado incluyendo un símbolo de interrogación). Como explicamos en el profile PMP (v. Cap. IV), en ocasiones es necesario simbolizar aquellos elementos indeterminados del contexto, fronterizos al patrón, que se conectarán con las instancias para que estén bien formadas. Los parámetros del patrón (expresados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo para especificar esta estructura de coordinación dentro de un escenario de trabajo concreto, son los siguientes: `<Productor>`, `<Consumidor>`, `<Producto>`, `<Producir>` y `<Consumir>`.
Ejemplo	Podemos usar este patrón para facilitar la construcción del modelo que permite reflejar la coordinación que se establece entre un/a alumno/a de doctorado (rol `Doctorando`), el cual puede haber

	empezado a escribir nuevos capítulos de su tesis (acción `EscribirCapítuloTesis`) sin haber enviado a revisar los anteriores, y un director de tesis (rol `DirectorTesis`) que puede aceptar nuevos capítulos para revisión (acción `RevisarCapítuloTesis`) sin haber terminado la revisión de los anteriores. La Figura B.20 muestra la ligadura concreta que permite modelar y describir dicha coordinación.
Patrones relacionados	Los siguientes patrones son similares (resuelven problemas semejantes, pero sus contextos de aplicación son diferentes): ▪ PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO$_{(2.5.1)}$: Aplicable cuando la acción de producir o consumir debe terminar antes de volver a producir o consumir, respectivamente, otro producto. ▪ PRODUCTOR-CONSUMIDOR MÚLTIPLE DISCONTINUO: Aplicable cuando existen varios productores y varios consumidores. Los actores no comienzan la producción o consumición de un nuevo producto antes de terminar de producir o consumir, respectivamente, el anterior producto. ▪ PRODUCTOR-CONSUMIDOR MÚLTIPLE CONTINUO: Aplicable cuando existen varios productores y varios consumidores. La producción y la consumición son continuas, no terminan.
Anexo Figuras (v. pág. siguiente)	

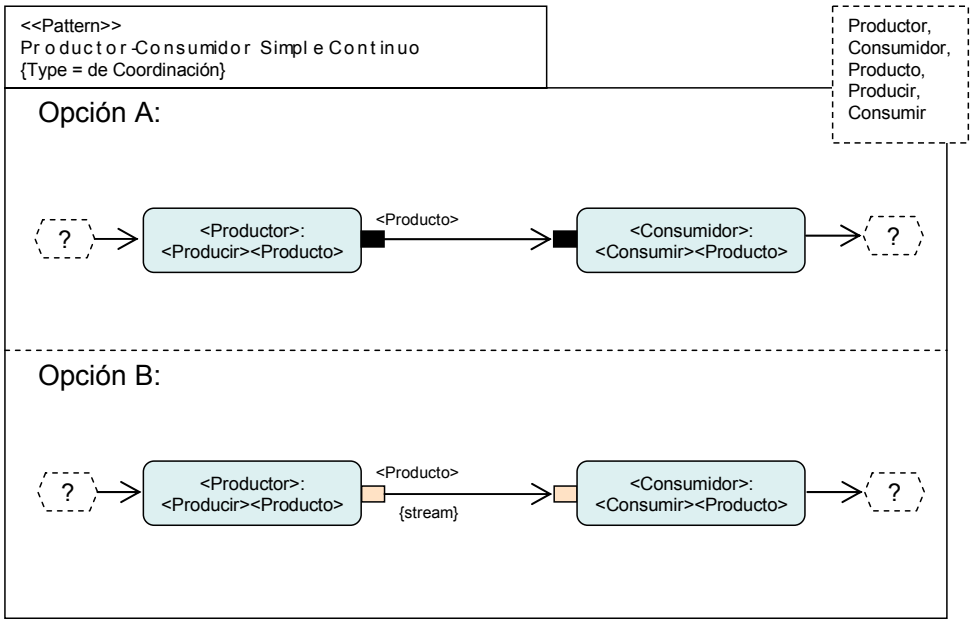


Fig. B.19: Modelado del patrón PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO

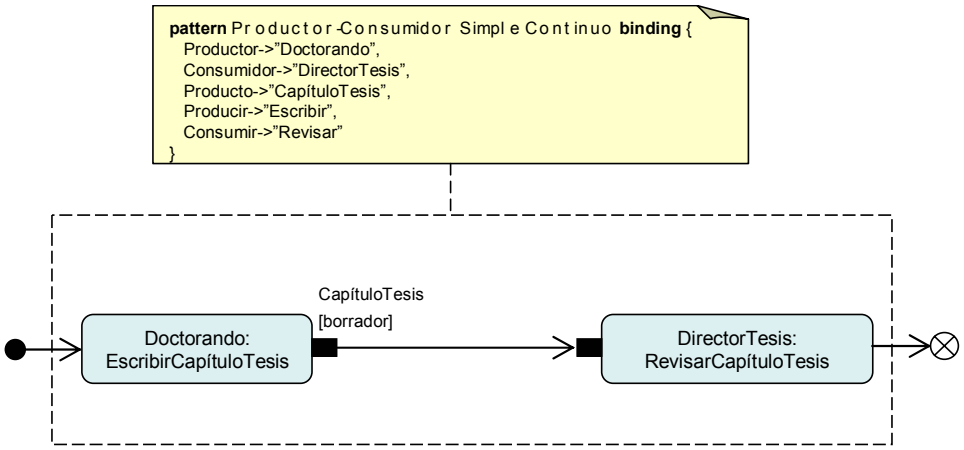


Fig. B.20: Aplicación del patrón PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO

2.5.3. Patrón SALVAVIDAS

Nombre/alias	SALVAVIDAS / PROTECTOR
Clasificación	▪ Vista: Cognitiva. ▪ Fase: Especificación de Tareas, Especificación de Roles. ▪ Tipo: Patrón de Coordinación.
Intención	Modelar una estructura de coordinación en la que si surge un problema mientras un actor está realizando una determinada acción (acción protegida), existe otro actor específicamente encargado de darle solución, pudiendo continuar el primero con la acción que estaba realizando.
Contexto	▪ Un actor realiza una acción que es necesario proteger ante una excepción que pudiera producirse. ▪ Existe un actor cuyo objetivo es resolver dicha excepción y devolver el control al actor que realizaba la acción protegida para que continúe con su trabajo.
Solución	Véase Figura B.21
Explicación	Este patrón se corresponde con el fragmento genérico de un diagrama de actividades que permite modelar, dentro de un escenario de trabajo cooperativo concreto, la estructura de coordinación especificada. La capacidad de expresión de UML 2 permite modelar fácilmente este estilo de coordinación. En el diagrama aparece la acción que es necesario proteger (acción `<AcciónProtegida>`), realizada por el actor que desempeña el papel de `<Protegido>`, y la acción encargada de manejar la excepción (acción `<TratamientoExcepción>`), realizada por el actor que desempeña el papel de `<ManejadorExcepción>`. De la acción

	protegida sale un extremo de interrupción, etiquetado con el tipo de excepción (etiqueta `<excepción>`), que conecta con el pin de entrada a la acción que maneja la excepción. Esto permite expresar que cuando se produce dicha `<excepción>` durante la ejecución de la acción `<AcciónProtegida>` la acción `<ManejadorExcepción>` trata dicho error y devuelve el control a la `<AcciónProtegida>` una vez tratado el problema. En el diagrama también se representan elementos de tipo `<<UncertainElement>>` (simbolizados con un hexágono punteado incluyendo un símbolo de interrogación). Como explicamos en el profile PMP (v. Cap. IV), en ocasiones es necesario simbolizar aquellos elementos indeterminados del contexto, fronterizos al patrón, que se conectarán con las instancias para que estén bien formadas. Los parámetros del patrón (expresados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo para especificar esta estructura de coordinación dentro de un escenario de trabajo concreto, son los siguientes: `<Protegido>`, `<AcciónProtegida>`,`<excepción>`,`<ManejadorExcepción>` y `<TratamientoExcepción>`.
Ejemplo	Podemos usar este patrón para facilitar la construcción del modelo que permite reflejar cuándo un profesor debe actuar mientras un grupo de estudiantes está realizando un ejercicio individualmente (acción `RealizarEjercicio`). Si alguno de los alumnos necesita ayuda (excepción `ayuda`), el profesor tiene la obligación de ayudarle (acción `Ayudar`) para que pueda continuar con el ejercicio. La Figura B.22 muestra la ligadura concreta que permite modelar y describir dicha coordinación.
Patrones relacionados	Antes se ha podido aplicar: ▪ ESTRUCTURA EN 5: Este tipo de estructura organizativa es muy proclive a que suceda esta clase de coordinación entre los

distintos niveles.

- MANDO-SUBMANDO: A menudo el actor que hace de submando tiene por objetivo resolver ciertos problemas que pueden surgir durante las actividades del mando, y viceversa.

Anexo Figuras

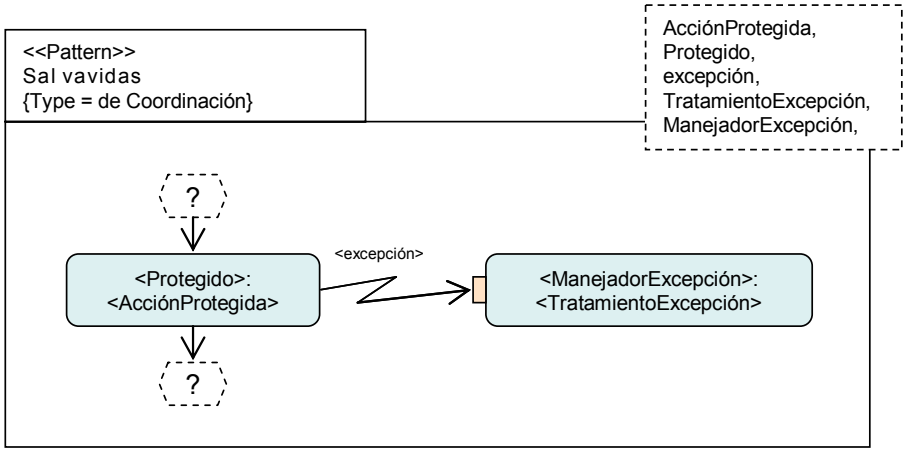


Fig. B.21: Modelado del patrón SALVAVIDAS

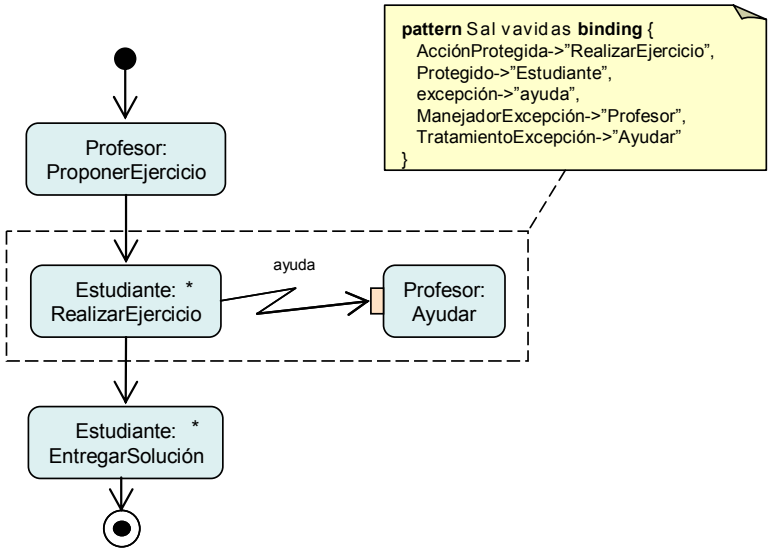


Fig. B.22: Aplicación del patrón SALVAVIDAS

2.6. Patrones de comunicación

2.6.1. Patrón DEBATE MODERADO

Nombre/alias	DEBATE MODERADO
Clasificación	▪ Vista: Interacción. ▪ Fase: Especificación de Tareas, Especificación de Protocolos de Interacción. ▪ Tipo: Patrón de Comunicación.
Intención	Modelar una conversación grupal moderada. Los actores solicitan la palabra a un moderador, quien registra el turno y decide el momento en el que debe intervenir cada cual.
Contexto	▪ Hay un grupo de personas que necesitan mantener una conversación sobre algún tema. ▪ Existe un actor (moderador) encargado de moderar las intervenciones de los interlocutores. ▪ Cuando alguien quiere intervenir debe obligatoriamente solicitar su turno al moderador, quien lo registra para, posteriormente, asignarlo cuando corresponda. ▪ Sólo puede intervenir aquella persona que tiene el turno de palabra asignado. Dicha intervención termina cuando lo decide el actor que está interviniendo o, si es necesario, cuando el moderador le quita el turno de palabra. ▪ El debate termina por decisión del moderador.
Solución	Véase Figura B.23
Explicación	Este patrón facilita el modelado del flujo de control y de objetos que sucede durante una conversación grupal moderada.

El diagrama muestra varios nodos de acción de tipo aceptación de eventos (`decidePedirTurno`, `peticiónTurno`, `decideAsignarTurno`, `asignaciónTurno`, `decideFinalizarTurno` y `decideFinalizarDebate`) que, como no disponen de extremos de entrada, se activan inmediatamente cuando comienza la actividad en la que están contenidos, permaneciendo activos mientras dicha actividad no termine. Cuando sucede dicho evento se pasa el control al siguiente nodo.

Así pues, podemos ver que:

- En el momento en que alguno de los miembros participantes, excepto el que hace de moderador (v. expresión de rol <`MiembroDebate`>-<`Moderador`>), decide solicitar turno de palabra (evento `decidePedirTurno`), éste envía una señal de petición de turno (señal `peticiónTurno`).
- Cuando el actor que modera el debate (rol <`Moderador`>) recibe una solicitud de petición de turno (evento `peticiónTurno`) éste registra el turno (acción `RegistrarTurno`) para llevar un control y poder asignarlo posteriormente cuando corresponda.
- Cuando el moderador, a la vista de la situación en la que se encuentre el debate y dependiendo los turnos que hay pendientes de asignación, considera oportuno asignar un turno de palabra (evento `decideAsignarTurno`), éste envía una señal de asignación de turno (señal `asignaciónTurno`).
- Una vez que un uno de los miembros del debate (rol <`MiembroDebate`>) recibe la asignación de turno de palabra (evento `asignaciónTurno`), éste interviene (acción `Intervenir`) y lo hace mientras no le indique el moderador que su turno ha finalizado (evento `finTurno`) o decida terminar el interviniente a iniciativa propia.
- Si el moderador toma la decisión de finalizar el turno de palabra en vigor (evento `decideFinalizarTurno`) éste envía una señal de finalización de turno de palabra (señal `finTurno`).

	▪ El debate termina cuando el moderador así lo decide (evento `decideFinalizarDebate`). Cada uno de los flujos de control termina con un nodo que finaliza dicho flujo, mientras el resto continúan con su ejecución. No obstante, como los nodos de acción de aceptación de evento no tienen extremo de entrada, éstos continúan activos y pueden seguir aceptando eventos en todo momento, lo que permite mantener la dinámica del debate. Los parámetros del patrón (representados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permitirán instanciar este modelo dentro de un escenario concreto de trabajo, son los siguientes: `<Moderador>` y `<MiembroDebate>`.
Ejemplo	Podemos usar este patrón para facilitar la construcción de un modelo que representa un debate moderado (actividad `DebatirPunto`) entre los miembros de un equipo de coordinación de proyectos de investigación. La Figura B.24 muestra la ligadura concreta que permite modelar y describir dicha comunicación. No obstante, no es necesario desplegar dicha actividad, ya que el patrón define totalmente el diagrama. Bastaría simplemente con conectar la etiqueta de ligadura directamente con el símbolo que referencia dicha actividad. Esta práctica permite reducir considerablemente la complejidad de los modelos.
Patrones relacionados	Antes se ha podido aplicar: ▪ REUNIÓN$_{(2.4.2)}$: Los debates suelen suceder durante el transcurso de las sesiones de reunión. ▪ TORMENTA DE IDEAS: Una vez generadas abundantes ideas sobre un determinado asunto, a menudo es necesario debatirlas después para seleccionar aquellas que más interesan. En ocasiones se aplica después:

- VOTACIÓN(2.4.3): La dinámica del debate puede llevar a la necesidad de tomar alguna decisión mediante la realización de una votación.
- NEGOCIACIÓN NO MODERADA(2.4.4): La dinámica del debate puede llevar a la necesidad de realizar algún tipo de negociación sobre los asuntos que han sido debatidos.

Hace uso o referencia a:

- TURNO DE PALABRA: El moderador coordina las intervenciones a través de la asignación de turnos de palabra.

El siguiente patrón es similar, aunque su contexto de aplicación varía:

- DEBATE NO MODERADO(2.6.2): Aplicable en situaciones de debate no asistidas por un moderador.

Anexo Figuras (v. pág. siguiente)

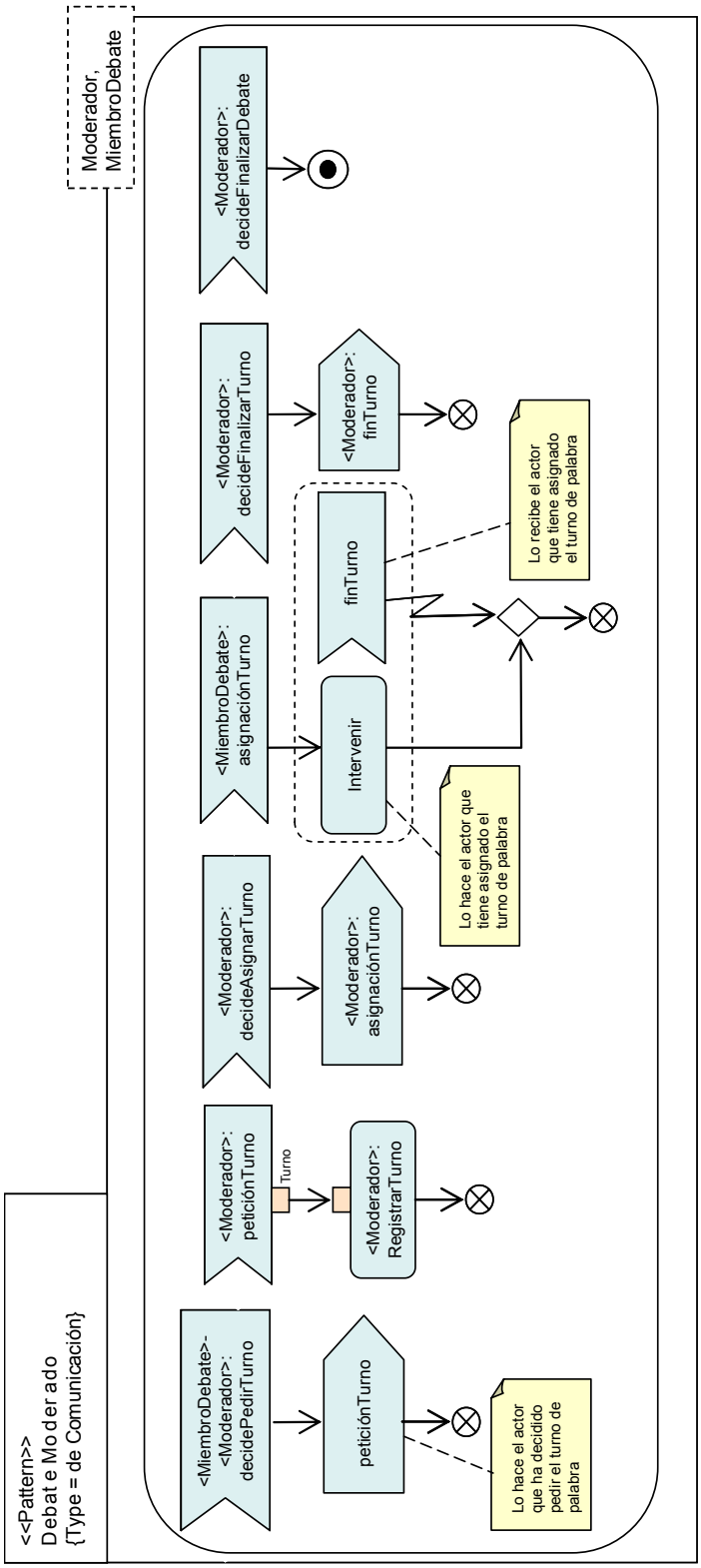


Fig. B.23: Modelado del patrón DEBATE MODERADO

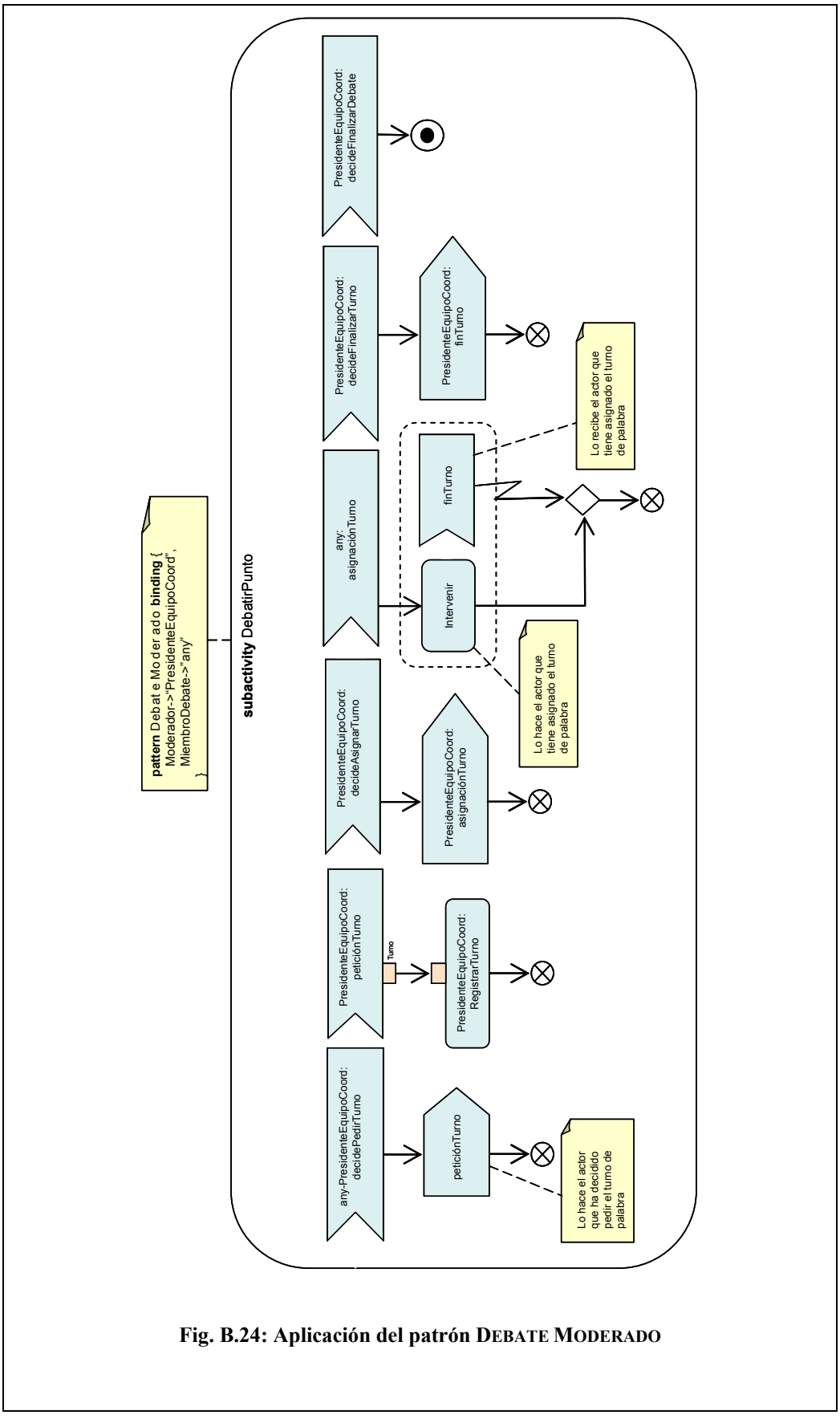


Fig. B.24: Aplicación del patrón DEBATE MODERADO

2.6.2. Patrón DEBATE NO MODERADO

Nombre/alias	DEBATE NO MODERADO
Clasificación	▪ Vista: Interacción. ▪ Fase: Especificación de Tareas, Especificación de Protocolos de Interacción. ▪ Tipo: Patrón de Comunicación.
Intención	Modelar una conversación libre entre varios participantes.
Contexto	▪ Hay un grupo de personas que necesitan mantener una conversación sobre algún tema. ▪ Los participantes, en cualquier momento, pueden interrumpir la persona que está interviniendo y solicitar la atención de los demás para intervenir. ▪ Una persona puede intervenir en el debate cuando recibe la atención de los demás. Dicha intervención termina cuando lo decide la propia persona que interviene o cuando ésta es interrumpida por alguno de los otros participantes. ▪ El debate termina cuando no hay más peticiones para intervenir.
Solución	Véase Figura B.25
Explicación	Este patrón facilita el modelado del flujo de control que sucede durante una conversación grupal no moderada. El diagrama muestra varios nodos de acción de tipo aceptación de eventos (`decideIntervenir`, `decideInterrumpir` y `recibeAtención`) que, como no disponen de extremos de entrada, se activan inmediatamente cuando comienza la actividad en la que están contenidos, permaneciendo activos mientras dicha actividad no termine. Cuando sucede dicho evento se pasa el

	control al siguiente nodo. Así pues, podemos ver que: ▪ En el momento en que alguno de los miembros participantes (v. rol `<MiembroDebate>`) decide intervenir (evento `decideIntervenir`) éste solicita la atención de los demás emitiendo una señal (señal `solicitaAtención`). ▪ Cuando alguien decide interrumpir a la persona que está interviniendo (evento `decideInterrumpir`) emite una señal de interrupción (señal `interrupción`). ▪ Una vez que uno de los miembros del debate recibe la atención de los demás (evento `recibeAtención`), éste interviene (acción `Intervenir`) y lo hace mientras no lo interrumpa otro miembro (evento `interrupción`) o éste decida terminar por iniciativa propia. ▪ El debate termina cuando nadie más solicita atención para intervenir (v. condición `[finDebate]`). Cada uno de los flujos de control termina con un nodo que finaliza dicho flujo, mientras el resto continúan con su ejecución. No obstante, como los nodos de acción de aceptación de evento no tienen extremo de entrada, éstos continúan activos y pueden seguir aceptando eventos en todo momento, lo que permite mantener la dinámica del debate. El parámetro del patrón (representado entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), el cual permite instanciar este modelo dentro de un escenario concreto de trabajo, es el siguiente: `<MiembroDebate>`.
Ejemplo	Podemos usar este patrón para facilitar la construcción de un modelo que representa un debate no moderado (actividad `DebatirPunto`) entre los miembros de un equipo de coordinación de proyectos de investigación. La Figura B.26 muestra la ligadura concreta que permite modelar y describir dicha comunicación. No obstante, no es necesario desplegar dicha actividad, ya que el patrón define totalmente el

	diagrama. Bastaría simplemente con conectar la etiqueta de ligadura directamente con el símbolo que referencia dicha actividad. Esta práctica permite reducir considerablemente la complejidad de los modelos.
Patrones relacionados	Antes se ha podido aplicar: ▪ REUNIÓN(2.4.2): Los debates suelen suceder durante el transcurso de las sesiones de reunión. ▪ TORMENTA DE IDEAS: Una vez generadas abundantes ideas sobre un determinado asunto, a menudo es necesario debatirlas después para seleccionar aquellas que más interesan. En ocasiones se aplica después: ▪ VOTACIÓN(2.4.3): La dinámica del debate puede llevar a la necesidad de tomar alguna decisión mediante la realización de una votación. ▪ NEGOCIACIÓN NO MODERADA(2.4.4): La dinámica del debate puede llevar a la necesidad de realizar algún tipo de negociación sobre los asuntos que han sido debatidos. El siguiente patrón es similar, aunque su contexto de aplicación es diferente: ▪ DEBATE MODERADO(2.6.1): Aplicable en situaciones de debate no asistidas por un moderador.
Anexo Figuras (v. pág. siguiente)	

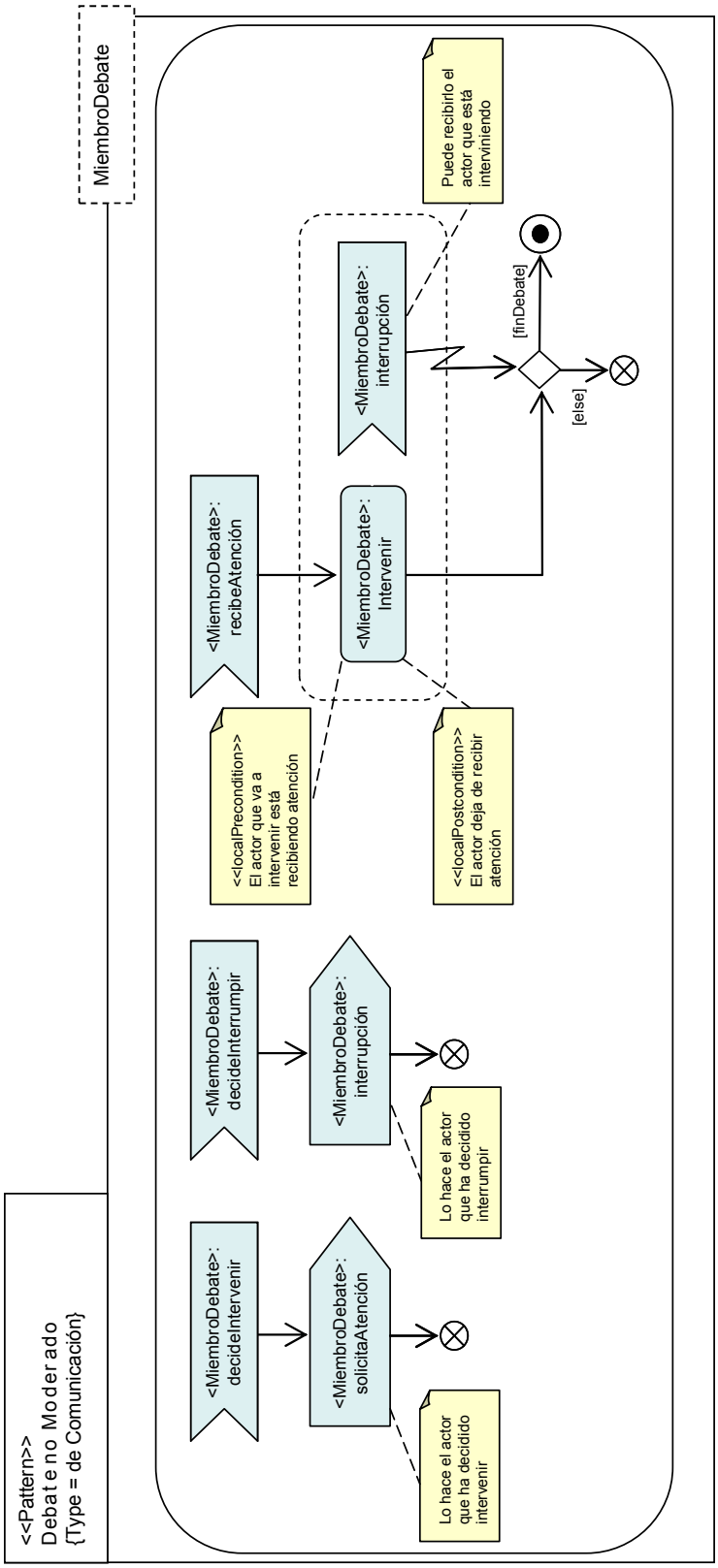


Fig. B.25: Modelado del patrón DEBATE NO MODERADO

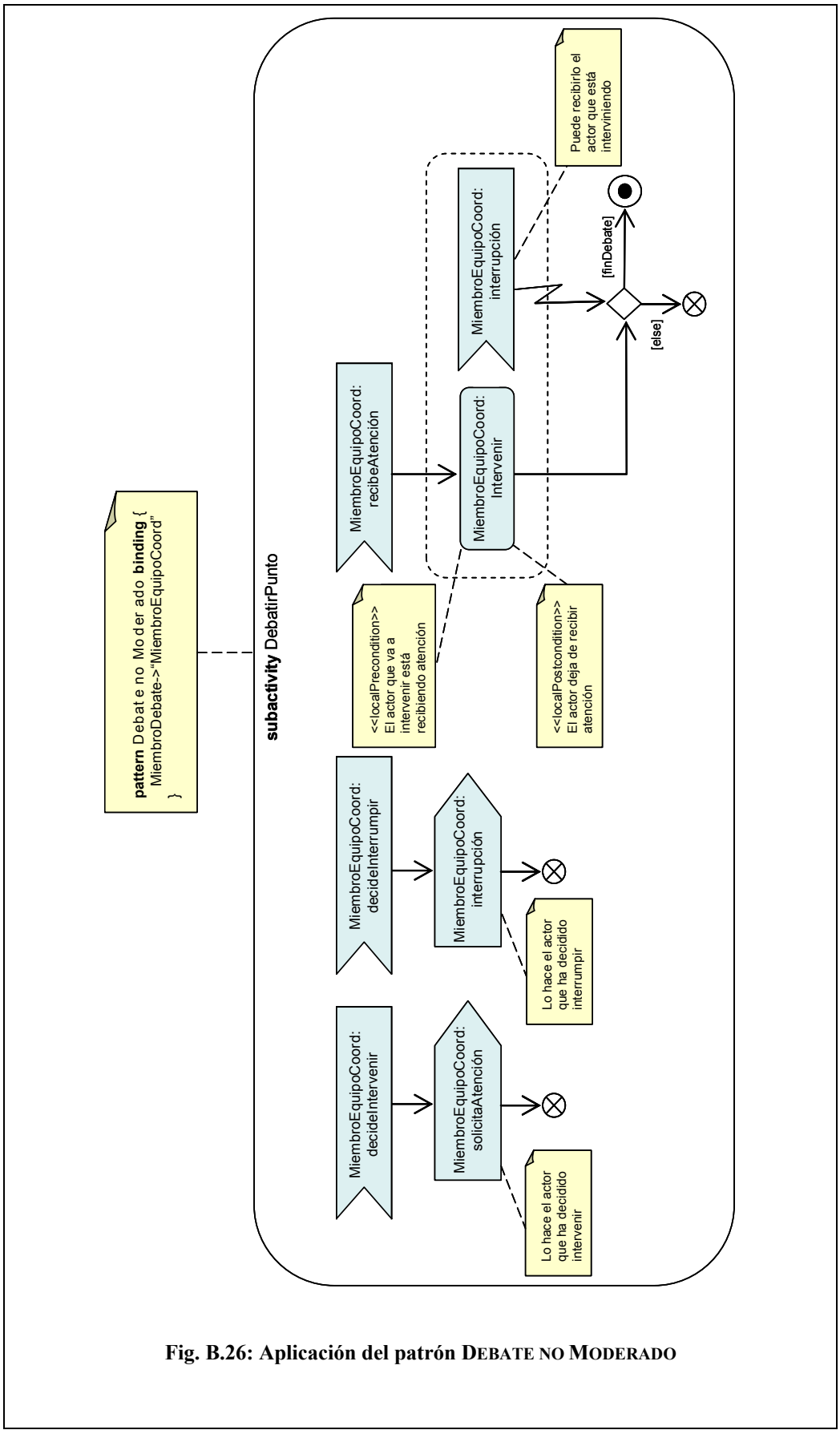


Fig. B.26: Aplicación del patrón DEBATE NO MODERADO

2.6.3. Patrón PETICIÓN-RESPUESTA SIMPLE

Nombre/alias	PETICIÓN-RESPUESTA SIMPLE
Clasificación	▪ Vista: Interacción. ▪ Fase: Especificación de Tareas, Especificación de Protocolos de Interacción. ▪ Tipo: Patrón de Comunicación.
Intención	Modelar una forma de comunicación sincrónica donde un participante emite una petición para, a continuación, recibir una respuesta por parte del receptor, la cual es esperada por el emisor antes de continuar.
Contexto	▪ Dos actores (emisor y receptor) necesitan comunicarse. ▪ La comunicación consiste en que cuando el emisor hace una petición, éste espera hasta obtener respuesta por parte del receptor. ▪ El receptor procesa la petición recibida antes de enviar una respuesta. ▪ El emisor procesa la respuesta recibida.
Solución	Véase Figura B.27
Explicación	Este patrón facilita el modelado del flujo de control y de objetos que representa esta forma de comunicación. En concreto, el flujo transcurre como sigue: En primer lugar, el actor que juega el papel de emisor dentro del patrón (rol `<Emisor>`) realiza el envío de una petición (acción `Enviar<Petición>`). Una vez realizada esta acción, se envía un token de control por el extremo de salida que conecta con el nodo de control de tipo fork, a la vez que un token de objeto fluye por el extremo de salida que conecta con el nodo de objeto que

	representa la petición (nodo `<Petición>`). El flujo de control se divide de forma que, mientras el emisor espera la confirmación de que el receptor ha enviado una respuesta (acción de aceptación de evento `Confirmación`), el receptor (rol `<Receptor>`) procesa la petición (acción `<RecibirPetición>`) en el momento en que recibe ésta (v. extremo de salida del nodo de objeto `<Petición>` que conecta con dicha acción). A continuación, una vez procesada la petición, el receptor envía la respuesta (acción `Enviar<Respuesta>`) y emite la señal (acción de envío de señal `Confirmación`) que está esperando el emisor para poder continuar (acción de aceptación de evento `Confirmación`). Ejecutadas estas dos últimas acciones, el emisor trata la respuesta (acción `<RecibirRespuesta>`) siempre y cuando el token que contiene ésta se encuentre en su otro extremo de entrada (v. nodo `<Respuesta>`). Procesada la respuesta, la actividad termina. Los parámetros del patrón (representados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permiten instanciar este modelo dentro de un escenario concreto de trabajo, son los siguientes: `<Emisor>`, `<Receptor>`, `<Petición>`, `<Respuesta>`, `<RecibirPetición>` y `<RecibirRespuesta>`.
Ejemplo	Podemos usar este patrón para la construcción de un modelo que representa la petición de voto por parte del secretario de un equipo de coordinación (rol `SecretarioEquipoCoord`) y la emisión de éste por parte de un miembro de dicho equipo (rol `MiembroEquipoCoord`). Cuando el votante recibe la petición (objeto `PeticiónVoto`), selecciona la opción que quiere votar (acción `SeleccionarOpciónVoto`) y envía el voto (objeto `Voto`) al secretario para que lo registre (acción `RegistrarVoto`). La Figura B.28 muestra la ligadura concreta que permite modelar y describir este tipo de comunicación en la actividad `EmitirVotoIndividuo`. No obstante, no es necesario desplegar dicha actividad, ya que el patrón define totalmente el diagrama. Bastaría simplemente con conectar la etiqueta de ligadura directamente con el símbolo que referencia dicha actividad. Esta

	práctica permite reducir considerablemente la complejidad de los modelos.
Patrones relacionados	Antes se ha podido aplicar: ▪ VOTACIÓN$_{(2.4.3)}$: Usa el patrón PETICIÓN-RESPUESTA SIMPLE para modelar la comunicación necesaria para la emisión de voto de un individuo. ▪ EXPOSICIÓN$_{(2.6.5)}$: Usa el patrón PETICIÓN-RESPUESTA SIMPLE para modelar la comunicación que representa el planteamiento y resolución de alguna duda. El siguiente patrón es similar, aunque su contexto de aplicación es diferente: ▪ PETICIÓN-RESPUESTA MÚLTIPLE$_{(2.6.4)}$: En este caso la respuesta no tiene porqué ser única. Existen múltiples receptores que pueden enviar alguna respuesta.
Anexo Figuras (v. pág. siguiente)	

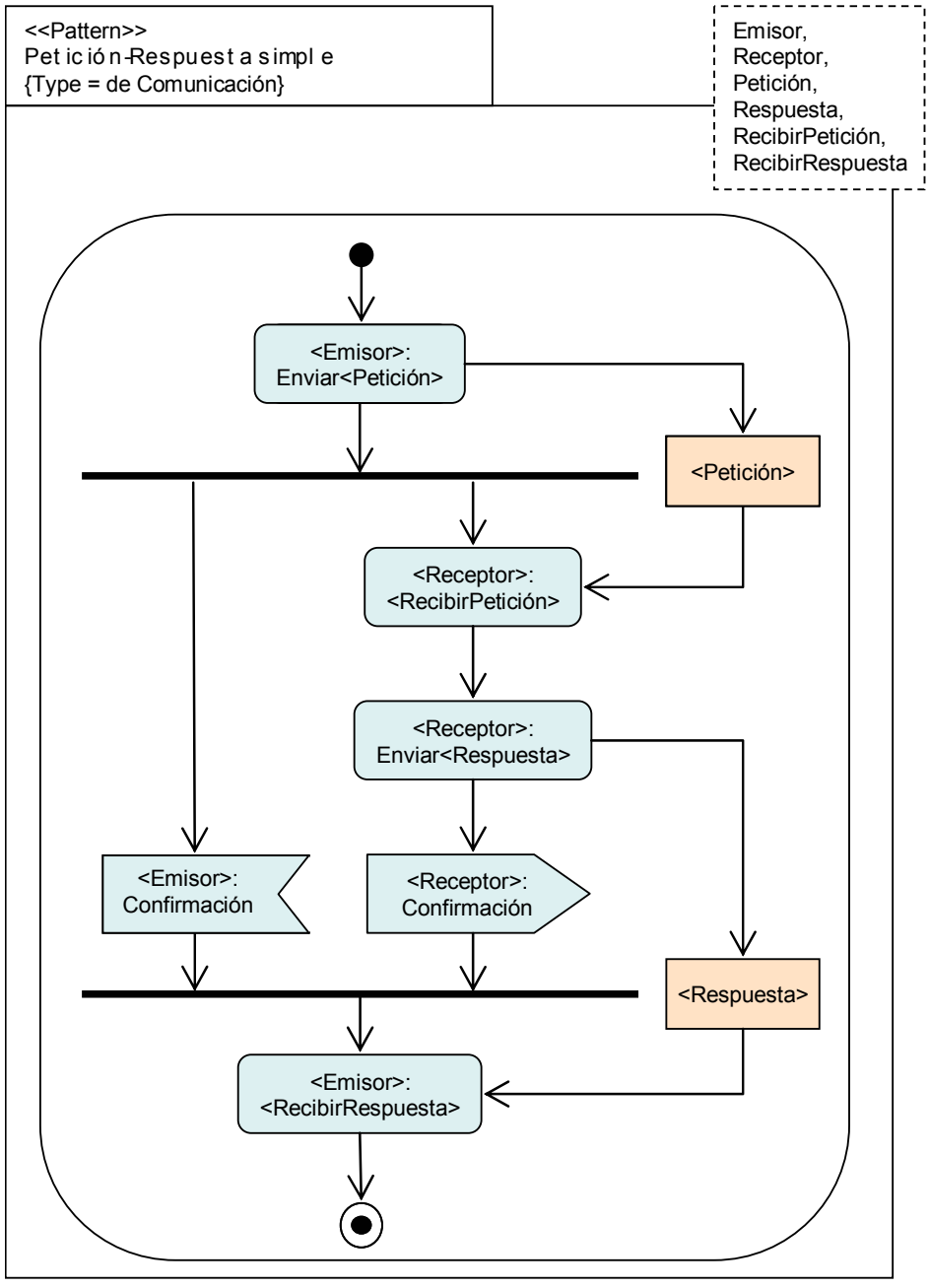


Fig. B.27: Modelado del patrón PETICIÓN-RESPUESTA SIMPLE

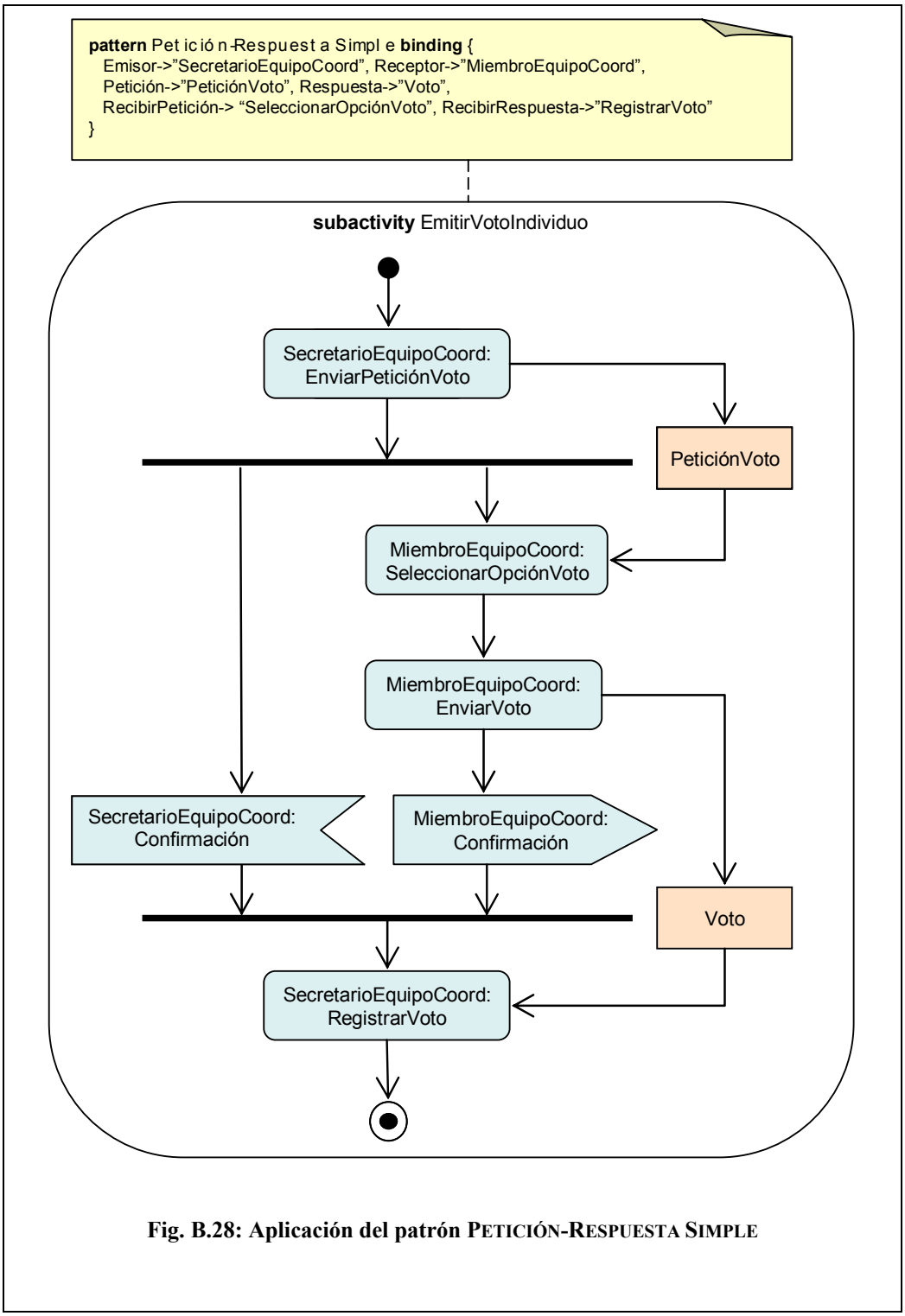


Fig. B.28: Aplicación del patrón PETICIÓN-RESPUESTA SIMPLE

2.6.4. Patrón PETICIÓN-RESPUESTA MÚLTIPLE

Nombre/alias	PETICIÓN-RESPUESTA MÚLTIPLE
Clasificación	▪ Vista: Interacción. ▪ Fase: Especificación de Tareas, Especificación de Protocolos de Interacción. ▪ Tipo: Patrón de Comunicación.
Intención	Modelar una forma de comunicación sincrónica donde un participante emite una petición para, a continuación, recibir respuesta por parte de un grupo de receptores. El emisor espera la llegada de suficientes respuestas antes de continuar.
Contexto	▪ Un actor (emisor) se comunica con varios actores (receptores). ▪ La comunicación consiste en que el emisor, una vez hecha una petición dirigida al grupo de receptores, espera hasta obtener una cantidad suficiente de respuestas por parte de los distintos receptores. ▪ Cada receptor procesa la petición recibida antes de enviar una respuesta. ▪ El emisor procesa las respuestas recibidas.
Solución	Véase Figura B.29
Explicación	Este patrón facilita el modelado del flujo de control y de objetos que representa esta forma de comunicación. En concreto, el flujo transcurre como sigue: En primer lugar, el actor que juega el papel de emisor dentro del patrón (rol `<Emisor>`) realiza el envío de una petición (acción `Enviar<Petición>`). Una vez realizada esta acción, se envía un token de control por el extremo de salida que conecta con el nodo de control de tipo fork, a la vez que un token de objeto fluye por el

	extremo de salida que conecta con el nodo de objeto que representa la petición (nodo <Petición>). El flujo de control se divide de forma que, mientras el emisor espera que haya suficientes respuestas emitidas por los receptores (acción de aceptación de evento Suficientes<Respuesta>s), cada receptor (rol <Receptor>) procesa la petición (acción <RecibirPetición>) en el momento en que recibe ésta (v. extremo de salida del nodo de objeto <Petición> estereotipado como <<multicast>> y que conecta con dicha acción). A continuación, una vez procesada la petición, múltiples receptores (v. estereotipo <<multireceive>>) envían su respuesta (acción Enviar<Respuesta>), quedando almacenadas las respuestas recibidas en el nodo de objeto ConjuntoDe<Respuesta>s. Cuando el emisor considera que tiene suficientes respuestas (acción de aceptación de evento Suficientes<Respuesta>s), todas las que hay almacenadas en ese momento (weight = all) pasan del nodo ConjuntoDe<Respuesta>s a la acción correspondiente al tratamiento que hace el receptor de todas ellas (acción <RecibirRespuestas>). Procesadas las respuestas, la actividad termina. Los parámetros del patrón (representados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permiten instanciar este modelo dentro de un escenario concreto de trabajo, son los siguientes: <Emisor>, <Receptor>, <Petición>, <Respuesta>, <RecibirPetición> y <RecibirRespuestas>.
Ejemplo	Podemos usar este patrón para la construcción de un modelo que representa la petición de voto por parte del secretario de un equipo de coordinación (rol SecretarioEquipoCoord) y la emisión paralela de éstos por parte del conjunto de miembros de dicho equipo (rol MiembroEquipoCoord). Cuando los votantes reciben la petición (objeto PeticiónVoto), seleccionan la opción por la que quieren votar (acción SeleccionarOpciónVoto) y los votos (objeto ConjuntoDeVotos) son enviados al secretario para que los registre (acción RegistrarVotos).

	La Figura B.30 muestra la ligadura concreta que permite modelar y describir este tipo de comunicación en la actividad `EmitirVotoGrupo`. No obstante, no es necesario desplegar dicha actividad, ya que el patrón define totalmente el diagrama. Bastaría simplemente con conectar la etiqueta de ligadura directamente con el símbolo que referencia dicha actividad. Esta práctica permite reducir considerablemente la complejidad de los modelos.
Patrones relacionados	Antes se ha podido aplicar: ▪ VOTACIÓN$_{(2.4.3)}$: Usa el patrón PETICIÓN-RESPUESTA MÚLTIPLE en el modelado de la comunicación necesaria para la emisión de voto de forma conjunta. ▪ LLAMAMIENTO PARA PROPUESTAS: Este patrón hace uso del patrón PETICIÓN-RESPUESTA MÚLTIPLE para hacer el llamamiento y recibir las múltiples propuestas. ▪ TORMENTA DE IDEAS: Dentro de esta dinámica se suele hacer una petición de ideas a todos los miembros del grupo. El siguiente patrón es similar, aunque su contexto de aplicación es diferente: ▪ PETICIÓN-RESPUESTA SIMPLE$_{(2.6.3)}$: En este caso existe un único receptor que envía una sola respuesta.
Anexo Figuras (v. pág. siguiente)	

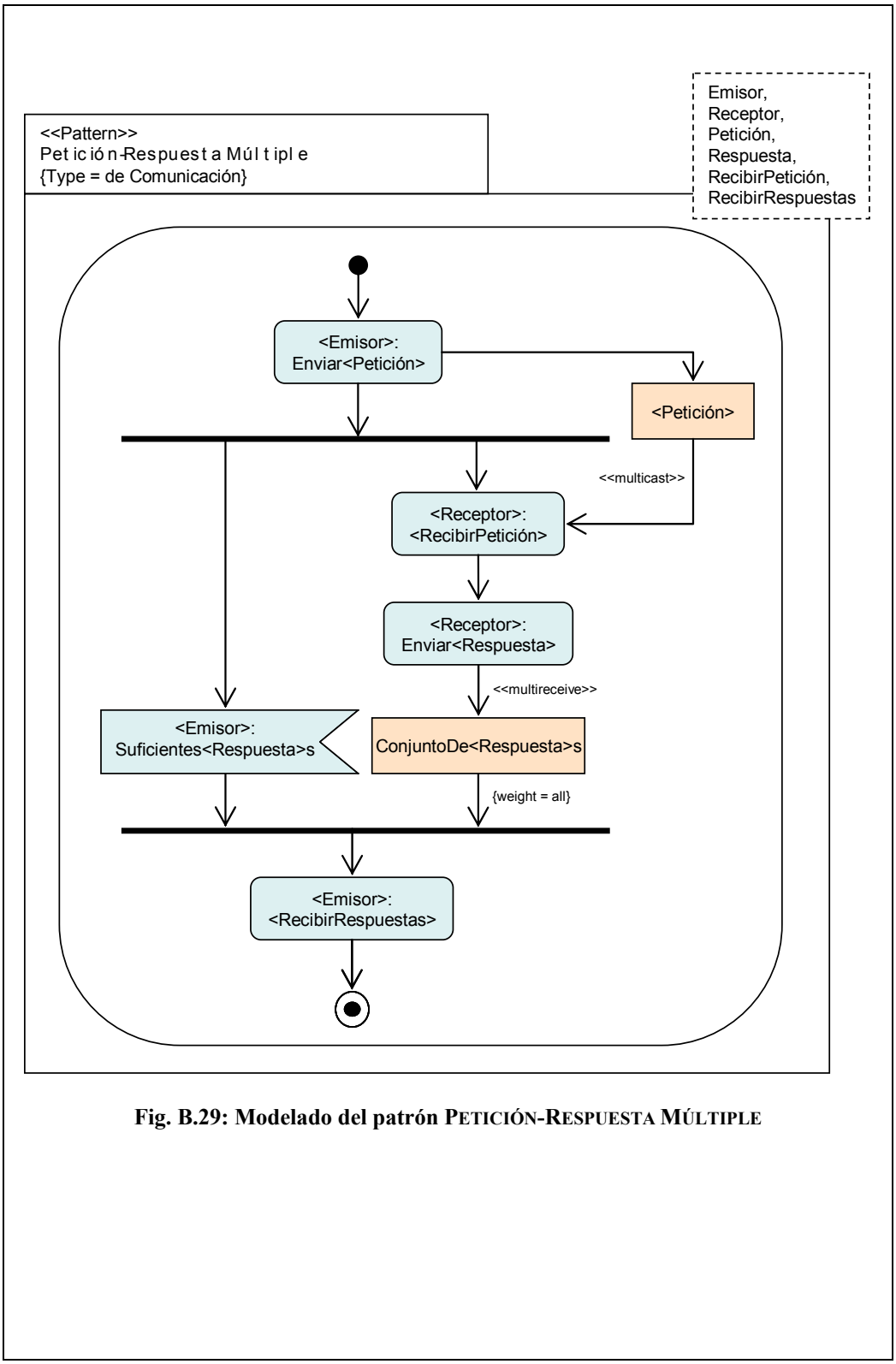


Fig. B.29: Modelado del patrón PETICIÓN-RESPUESTA MÚLTIPLE

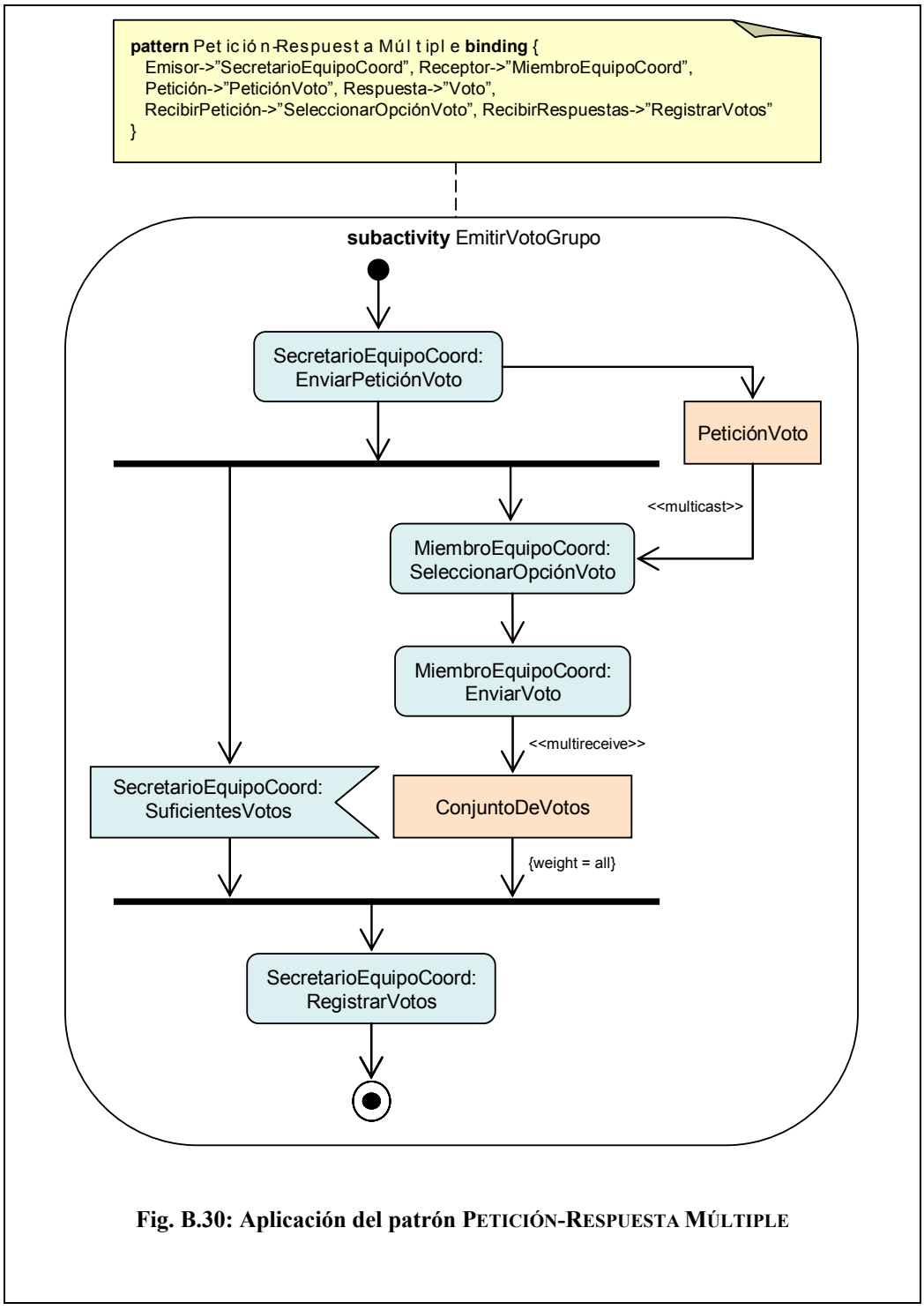


Fig. B.30: Aplicación del patrón PETICIÓN-RESPUESTA MÚLTIPLE

2.6.5. Patrón EXPOSICIÓN

Nombre/alias	EXPOSICIÓN
Clasificación	▪ Vista: Interacción. ▪ Fase: Especificación de Tareas, Especificación de Protocolos de Interacción. ▪ Tipo: Patrón de Comunicación.
Intención	Modelar una forma de comunicación en la que un actor (ponente) expone un tema a un grupo de participantes y resuelve las dudas que pudieran plantearle.
Contexto	▪ Un actor (ponente) informa a un grupo de actores sobre algún tema. ▪ Durante y después de la exposición, el ponente resuelve las dudas que le surja a la audiencia. ▪ La actividad termina cuando el ponente ha terminado de informar a la audiencia y no existen más dudas por resolver o se abandonan las que hubiere.
Solución	Véase Figura B.31
Explicación	Este patrón facilita el modelado del flujo de control y de objetos que representa esta forma de comunicación. En concreto, el flujo transcurre como sigue: En primer lugar, el actor que juega el papel de ponente dentro del patrón (rol `<Ponente>`) da inicio a la exposición (acción `InicioExposición`). A continuación se crean dos flujos de control paralelos de forma que, durante o después de la exposición del ponente (acción `Exponer<Tema>`), éste va respondiendo a cualquier pregunta que los participantes pudieran plantearle (acción `ResolverDuda`). Esta última acción se realiza sólo cuando existe

	alguna duda que resolver (condición `existeDuda`), formando parte de un ciclo que terminará sólo cuando el ponente haya finalizado su exposición y no existan más dudas que resolver o, si éstas existen, se hayan abandonado (condición `[(noExisteDuda or abandonoDuda) and finExposición]`). Cuando esta última condición no se satisface, el ciclo termina y el ponente da por finalizada la actividad (acción `FinExposición`). Como se puede advertir, la actividad `ResolverDuda` se define a partir de la ligadura dinámica del patrón PETICIÓN-RESPUESTA SIMPLE$_{(2.6.3)}$, donde el `Ponente` unas veces hace de `Receptor` y otras de `Emisor`. Los parámetros del patrón (representados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permiten instanciar este modelo dentro de un escenario concreto de trabajo, son los siguientes: `<Ponente>` y `<Tema>`.
Ejemplo	Podemos usar este patrón para facilitar la construcción de un modelo que representa la exposición de una tarea (`Tema -> "Tarea"`) por parte del portavoz de un grupo de aprendizaje según una estrategia de Jigsaw (`Ponente->"PortavozJigsaw"`). La Figura B.32 muestra la ligadura concreta que permite modelar y describir este tipo de comunicación en la actividad `PresentarTareaPúblicamente`. No obstante, no es necesario desplegar dicha actividad, ya que el patrón define totalmente el diagrama. Bastaría simplemente con conectar la etiqueta de ligadura directamente con el símbolo que referencia dicha actividad. Esta práctica permite reducir considerablemente la complejidad de los modelos.
Patrones relacionados	Antes se ha podido aplicar: ▪ VOTACIÓN$_{(2.4.3)}$: Usa el patrón EXPOSICIÓN en la especificación de la acción usada para informar del resultado de la votación. ▪ CREAR PLAN DE TRABAJO$_{(2.4.3)}$: A menudo requiere su posterior

exposición a los miembros del grupo de trabajo.

Hace uso o referencia a:

- PETICIÓN-RESPUESTA SIMPLE$_{(2.6.3)}$: Para especificar la acción que representa la resolución de dudas.

Anexo Figuras

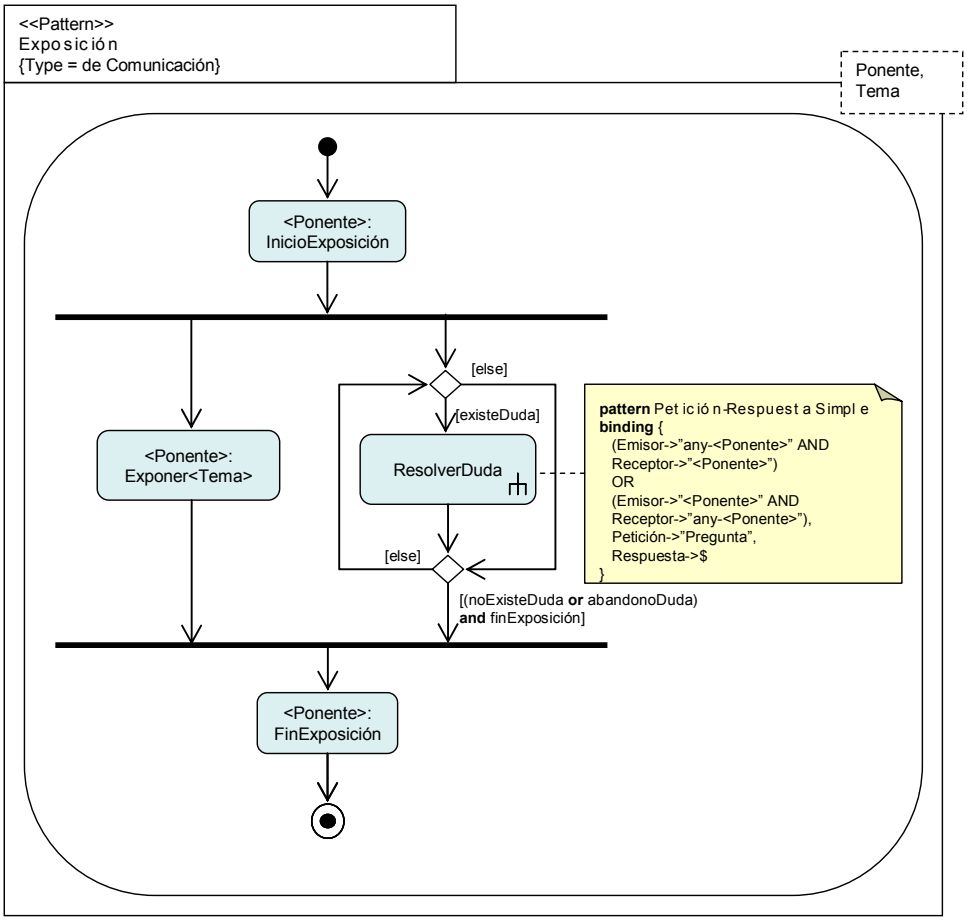


Fig. B.31: Modelado del patrón EXPOSICIÓN

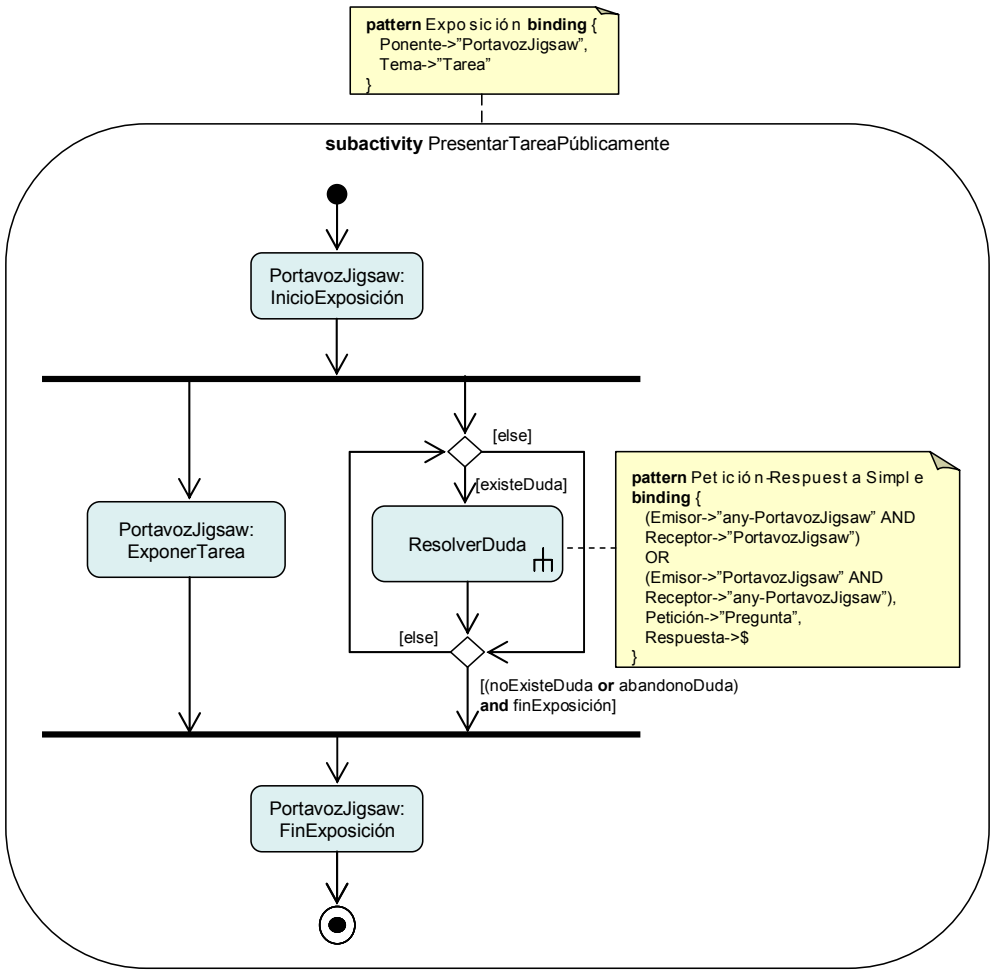


Fig. B.32: Aplicación del patrón EXPOSICIÓN

2.7. Patrones de estructura

2.7.1. Patrón ACTA DE REUNIÓN

Nombre/alias	ACTA DE REUNIÓN
Clasificación	▪ Vista: Información. ▪ Fase: Especificación del Modelo Conceptual de Datos. ▪ Tipo: Patrón de Estructura.
Intención	Modelar la estructura conceptual correspondiente al acta de una reunión.
Contexto	▪ Se celebran reuniones cuyo desarrollo se registra en un acta por alguien que desempeña las labores de secretario.
Solución	Véase Figura B.33
Explicación	Este patrón facilita el modelado de la estructura conceptual correspondiente al acta de una reunión. Un acta (v. clase `<Acta>`) se compone de: ▪ `Identificación`. Está formada por un número de acta opcional (`N° Acta`), el tipo de reunión mantenida (`Tipo Reunión`, que puede ser `Ordinaria` o `Extraordinaria`), el órgano o grupo que se reúne (`Grupo Reunido`) y el momento de la reunión (`Fecha/Hora`). ▪ `Asistentes`. Lista con los asistentes a la reunión y de aquellos que han justificado su ausencia. ▪ `Apertura`. Es un texto que introduce y sitúa el contexto de la reunión (p. ej., “Siendo las 13:45 h. del día 16-5-07 se reúnen los arriba citados en sesión ordinaria de la Junta Directiva de

	Mercury S.A. para tratar el siguiente orden del día:") ▪ `Orden del Día`. Compuesto por uno o más puntos (`Punto Orden del Día`) ordenados según el momento en el que van a ser tratados dentro de la reunión (v. restricción `{ordered}`). ▪ `Desarrollo`. Incluye cada uno de los puntos del orden del día desarrollados (`Desarrollo Punto`). ▪ `Cierre`. Es el texto que cierra la reunión y que incluye la hora en la que termina ésta (`Hora`). Por ejemplo "No habiendo más asuntos que tratar, se levanta la sesión siendo las 14:15 h. del día citado, de todo lo cual doy fe como Secretario y firmo la presente con el VºBº del Presidente." Un acta siempre tiene un `Autor`, normalmente el actor que actúa como secretario en la reunión. Un acta, hasta que no es aprobada (`Fecha Aprobación`), puede ser modificada en varias ocasiones (`Fecha Modificación`). El parámetro del patrón (representado entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), el cual permite instanciar este modelo en un contexto determinado, es `<Acta>`.
Ejemplo	Podemos usar este patrón para facilitar el modelado conceptual de este tipo de entidad. Así, por ejemplo, es posible ligar el patrón a un paquete para indicar que su contenido es el modelo conceptual de datos de un acta. La Figura B.34 muestra una ligadura concreta que permite modelar y describir estructuralmente este tipo de concepto.
Patrones relacionados	Antes se ha podido aplicar: ▪ REUNIÓN$_{(2.4.2)}$: Los aspectos más relevantes de la sesión de reunión pueden ser registrados en un acta por alguien que desempeña el papel de secretario. ▪ PROCESO DE REUNIÓN$_{(2.4.1)}$: La preparación del acta definitiva, si la hay, y su envío a los miembros del grupo una vez realizada

la sesión de reunión forma parte de este patrón.

Anexo Figuras

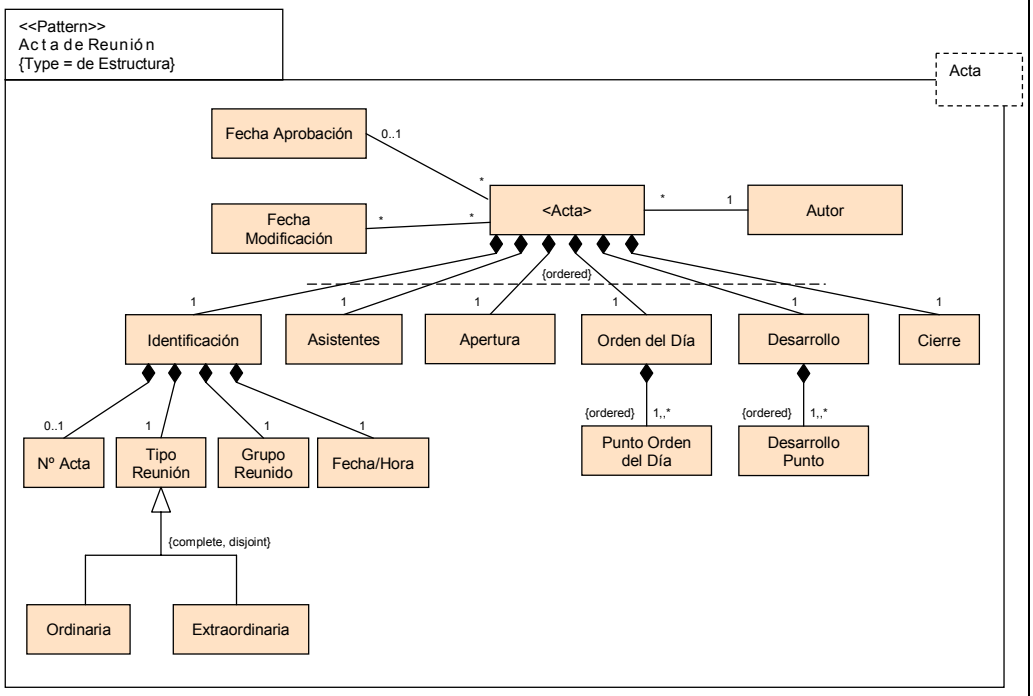


Fig. B.33: Modelado del patrón ACTA DE REUNIÓN

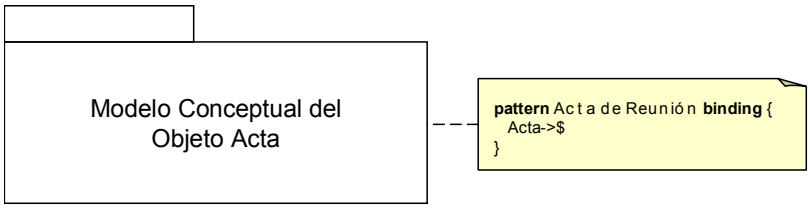


Fig. B.34: Aplicación del patrón ACTA DE REUNIÓN

2.8. Patrones de acceso

2.8.1. Patrón AUTORIZADO

Nombre/alias	AUTORIZADO
Clasificación	▪ Vista: Información. ▪ Fase: Especificación de Tareas, Especificación de Roles. ▪ Tipo: Patrón de Acceso.
Intención	Modelar un mecanismo de coordinación en el que un actor o, lo que es lo mismo, la acción que realiza, podrá acceder a un determinado recurso compartido si dispone de los privilegios necesarios para ello y el recurso está disponible.
Contexto	▪ Hay un recurso que puede ser usado por varios actores. ▪ El recurso podrá ser utilizado si éste se encuentra disponible. ▪ Para poder acceder al recurso compartido es necesario que el actor disponga de la autorización necesaria.
Solución	Véase Figura B.35
Explicación	Este patrón facilita el modelado de la restricción de acceso necesaria para que un actor o, lo que es lo mismo, la acción que realiza, pueda acceder a un determinado recurso compartido si dispone de la autorización necesaria para ello y el recurso está disponible. El patrón muestra el recurso, representado como un nodo de objeto (`<Recurso>`), cuyo extremo de salida conecta con la acción a través de la cual se pretende acceder al recurso (`<AcciónDeAcceso>`). Si hay un token en dicho nodo significa que el recurso está disponible, pudiendo pasar el token al siguiente nodo a través de su extremo de salida. Sin embargo, como se

	puede comprobar, para que la acción pueda ejecutarse, antes debe cumplirse la condición especificada por medio del estereotipo `<<localPrecondition>>`. En concreto, la precondición es la siguiente: "`El actor que realiza la acción dispone de la autorización necesaria para el acceso a <Recurso> en modo <ModoDeAcceso>`". El tipo de acceso va a depender del recurso y del contexto de trabajo, existiendo plena libertad para su especificación. Así, el modo de acceso puede ser, por ejemplo: "sólo lectura", "escritura", "lectura/escritura", "supervisor", "mantenimiento", etc. Los parámetros del patrón (representados entre ángulos en el modelo y en la esquina superior derecha del paquete que lo contiene), los cuales permiten instanciar este modelo dentro de un escenario concreto de trabajo, son los siguientes: `<Recurso>`, `<AcciónDeAcceso>` y `<ModoDeAcceso>`.
Ejemplo	Podemos usar el patrón para facilitar el modelado de esta forma de acceso cuando un actor, desempeñando el rol de `Coordinador`, necesita actualizar el documento compartido `Plan de Trabajo`. Para ello, la precondición de la acción `Actualizar Plan de Trabajo`, realizada por el `Coordinador`, obliga a que el actor tenga la autorización necesaria para acceder a dicho documento en modo `escritura`. De la misma forma se expresa que un `Operario` podrá acceder a este documento, una vez actualizado por el `Coordinador` (v. estado `[actualizado]`) cuando éste tenga autorización de acceso en modo `lectura`. La Figura B.36 muestra la ligadura concreta que permite modelar y describir este tipo de acceso en la actividad `ModificarPlanDeTrabajo`.
Patrones relacionados	Antes se ha podido aplicar: ▪ PUBLICACIÓN-SUSCRIPCIÓN: Usa el patrón AUTORIZADO para especificar que los suscriptores sólo recibirán los mensajes del tipo al que tenga autorizado su acceso. ▪ PRODUCTOR-CONSUMIDOR SIMPLE DISCONTINUO$_{(2.5.1)}$: La

consumición puede estar restringida a que el consumidor tenga autorización de acceso al objeto producido.

- PRODUCTOR-CONSUMIDOR SIMPLE CONTINUO$_{(2.5.2)}$: La consumición puede estar restringida a que el consumidor tenga autorización de acceso al objeto producido.
- PRODUCTOR-CONSUMIDOR MÚLTIPLE DISCONTINUO: La consumición puede estar restringida a que los consumidores tengan autorización de acceso a los objetos producidos.
- PRODUCTOR-CONSUMIDOR MÚLTIPLE CONTINUO: La consumición puede estar restringida a que los consumidores tengan autorización de acceso a los objetos producidos.
- ENRUTAR FORMULARIO: Esta actividad conlleva la especificación de la autorización y modo de acceso de los distintos actores que pueden usar el formulario.

En ocasiones se aplica después:

- ACTA DE REUNIÓN$_{(2.7.1)}$: El acceso a este tipo de objetos a menudo necesita de un control de autorización.
- CALENDARIO DE EVENTOS: El acceso a este tipo de objetos a menudo necesita de un control de autorización.
- PLAN DE TRABAJO: El acceso a este tipo de objetos a menudo necesita de un control de autorización.

Anexo Figuras (v. pág. siguiente)

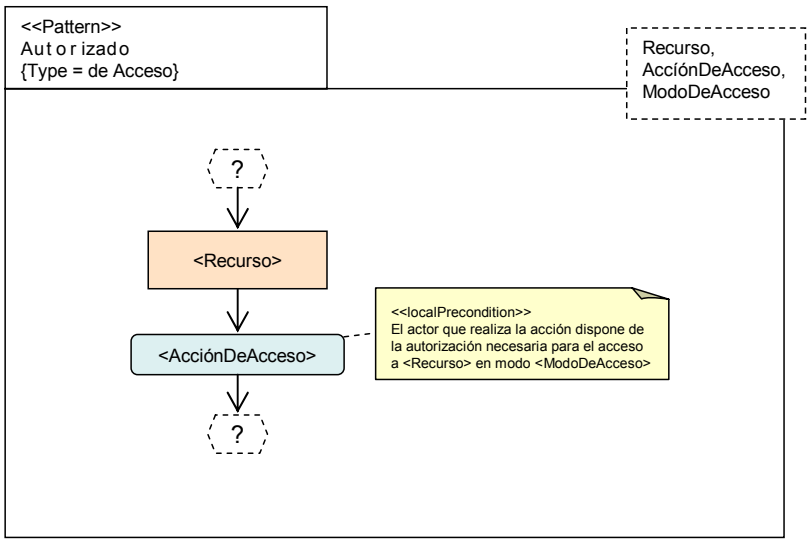


Fig. B.35: Modelado del patrón AUTORIZADO

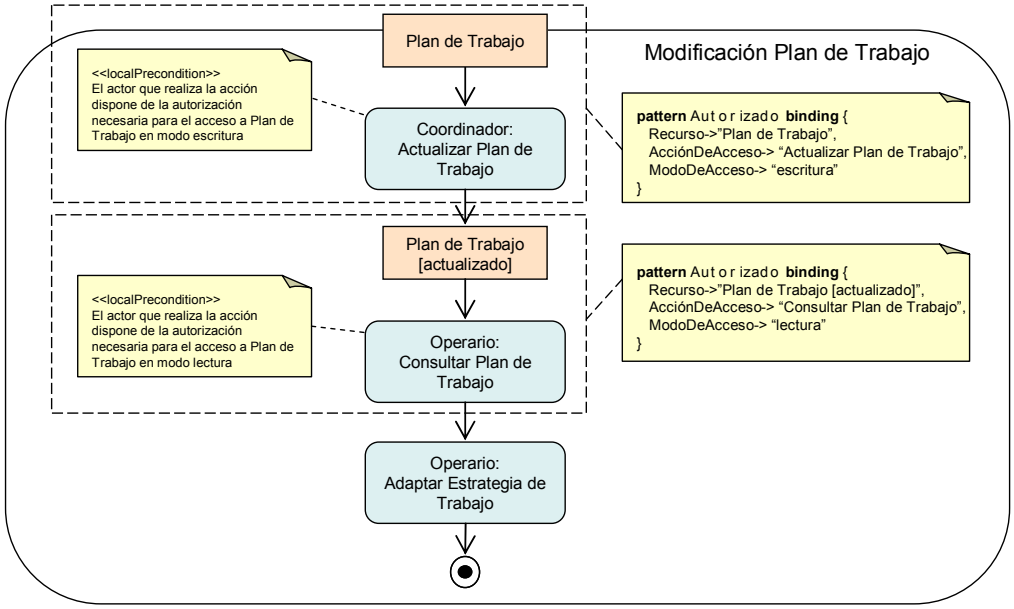


Fig. B.36: Aplicación del patrón AUTORIZADO

-Esta página se encuentra deliberadamente en blanco-

Apéndice C Glosario de Términos

Acción	Unidad básica de trabajo ejecutable atómicamente.
Actor	Miembro (humano, software o hardware) de un grupo de trabajo. Puede desempeñar diferentes roles dependiendo de la dinámica de trabajo.
Artefacto	Dispositivo (hardware y/o software) utilizado para llevar a cabo ciertas acciones.
Capacidad	Habilidad o responsabilidad asociada a un actor que le permite desempeñar roles y llevar a cabo tareas, subactividades o acciones.
Catálogo de patrones	Colección de patrones dentro de un dominio específico. Debe estructurarse de manera que se facilite la selección de éstos durante el desarrollo de software.
Colaboración	Interacción entre varios participantes para la consecución de un objetivo común (creación compartida de algo, toma de decisión, etc.). Aunque desde un punto de vista tecnológico en esta tesis no hacemos distinción alguna entre sistemas cooperativos y colaborativos, reconocemos ciertas diferencias entre los términos *cooperación* y *colaboración*. En el caso de la *cooperación* la tarea se divide en subtareas independientes asignadas a diferentes participantes, y la coordinación es necesaria sólo cuando hay que juntar los resultados parciales. En cambio, cuando hablamos de *colaboración*, en un principio no está clara la responsabilidad de cada uno de los participantes, existiendo un compromiso mutuo para la resolución del problema. Cada uno de los participantes interviene en todas y cada una de las partes en que se puede dividir el problema o proyecto común, teniendo que coordinarse con los demás para resolver cada parte.
Comunicación	Transmisión de información entre varios participantes

	usando un código común.
Conciencia de grupo	Entendimiento y conocimiento de las actividades de los otros miembros del grupo, proporcionando el contexto para llevar a cabo la propia actividad [Dourish y Bellotti, 1992].
Contexto compartido	Conjunto de objetos donde éstos y las acciones que se pueden realizar sobre ellos son visibles a un conjunto de usuarios [Ellis et al., 1991].
Cooperación	v. *colaboración.*
Coordinación	Actividad encaminada a gestionar las dependencias entre actividades realizadas en grupo para la consecución de un objetivo común [Malone y Crowston, 1990]. La coordinación permite que cada unidad o parte de un todo sepa cómo y cuándo actuar para conseguir un objetivo mayor.
CSCW (Computer-Supported Cooperative Work)	Area de investigación interdisciplinar que tiene por objeto estudiar cómo las personas trabajan en grupo y cómo la tecnología puede ayudarles [Greif, 1988].
Equipo	Grupo de personas organizadas para una investigación o servicio determinado [Real Academia Española, 2003].
Evento	Ocurrencia de algún hecho que tiene una localización tanto en el espacio como en el tiempo.
Grupo	Conjunto de actores desempeñando roles que pertenecen a una misma organización o que participan en la realización de tareas cooperativas.
Groupware	Conjunto de aplicaciones encaminadas a facilitar la comunicación, coordinación y/o colaboración efectiva de grupos durante la realización de tareas compartidas.
Interacción	Acción que se ejerce recíprocamente entre dos o más actores, objetos, etc.
Interacción	Interacción en tiempo real, es decir, las acciones de uno

sincrónica	tienen perfecta correspondencia temporal con las del otro.
Interacción asíncrona	Interacción donde las acciones de uno no tienen correspondencia temporal con las del otro.
IPO (en inglés HCI)	Interacción Persona-Ordenador (en inglés Human-Computer Interaction). Es la disciplina relacionada con el diseño, evaluación e implementación de sistemas informáticos interactivos para el uso por seres humanos, y con el estudio de los fenómenos más importantes con los que se relaciona.
IPOP (en inglés HCHI)	Interacción Persona-Ordenador-Persona (en inglés Human-Computer-Human Interaction).
Lenguaje de patrones	Catálogo de patrones que incluye, además, una red de conexiones entre los patrones. Normalmente, dispone de información (reglas y guías) acerca de cuándo y cómo se pueden componer éstos para la resolución de problemas más grandes y complejos que aquellos que son tratados por cada patrón individualmente. Se podría ver como un vocabulario, formado por los nombres de los patrones, junto a una gramática que indica cómo combinar éstos para construir sentencias más complejas.
Ley	Norma de una organización que restringe su funcionamiento en base a reglas sociales, culturales, capacidades de los actores, etc.
Modelado conceptual	Proceso durante el cual el analista construye uno o varios modelos, denominados modelos conceptuales, que ayudan a entender y simplificar el dominio del problema.
Modelo conceptual	Modelo que describe, usualmente mediante alguna notación gráfica, el conocimiento necesario acerca del sistema para poder abordar su desarrollo. Representan la visión (modelo mental) que los stakeholders y analistas deberían compartir sobre el sistema.
Modelo de análisis	v. *modelo conceptual.*
Negociación	Una forma de toma de decisión que implica a dos o más participantes en un proceso de concesiones para llegar a un acuerdo [Jain y Solomon, 2000].

Objeto de información	Entidad que contiene la información requerida para llevar a cabo acciones, o que se genera como resultado de éstas.
Ontología	Formulación exhaustiva y rigurosa de un esquema conceptual dentro de un dominio dado, con la finalidad de facilitar la comunicación y la compartición de la información entre diferentes sistemas [Wikipedia, 2007]
Organización	Conjunto de roles, y relaciones entre ellos, que se dan en un lugar de trabajo.
Patrón	Modelo que sirve de muestra para sacar otra cosa igual [Real Academia Española, 2003].
Patrón conceptual	Patrón de software aplicable en la construcción del modelo conceptual (o de análisis) de un sistema. Éstos describen conceptos y abstracciones recurrentes en un dominio de aplicación particular. Guían la percepción que se tiene del dominio del problema, ayudando a encontrar, comprender y describir los problemas comunes que aparecen dentro de éste. El objetivo, por tanto, de este tipo de patrones es capturar el conocimiento necesario que nos permita comunicar, entender y reutilizar, durante el proceso de análisis y modelado conceptual, aquellas abstracciones clave que son recurrentes en ciertos dominios.
Patrón de análisis	v. *Patrón conceptual*.
Patrón de arquitectura	Patrón de software que expresa un esquema u organización básica de la estructura de un sistema. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y pautas para organizar las relaciones entre ellos.
Patrón de diseño	Patrón de software aplicable en la construcción del modelo de diseño de un sistema. Habitualmente describen estructuras recurrentes de componentes de diseño (p. ej. clases y objetos) que se relacionan entre sí, dispuestos para resolver un problema de diseño general en un contexto particular.
Patrón de software	Descripción de un problema común (habitualmente de modelado) que aparece durante el desarrollo de software y una solución (normalmente un modelo genérico) efectiva y

	reutilizable en situaciones semejantes.
Proceso de grupo	Tarea bien definida que requiere la participación de un conjunto de usuarios.
Protocolo de grupo	Una forma de interacción entre actores mutuamente acordada.
Protocolo de interacción	v. *protocolo de grupo.*
Protocolo tecnológico	Protocolo de grupo que es soportado por hardware/software.
Protocolo social	Protocolo de grupo donde el control se deja a los propios actores, no siendo forzados por el sistema.
Rol	Conjunto de privilegios y responsabilidades (tareas) atribuidos a un actor. Resuelve la asociación dinámica que existe entre actores y tareas. Puede verse como el estado que posee un actor en un momento dado dentro del grupo, lo que permite describir la evolución de su comportamiento dentro del sistema. Éstos pueden ser asignados formal o informalmente.
Sesión	Período de interacción sincrónica soportada por un sistema groupware [Ellis et al., 1991].
Sistema colaborativo	Sistema pensado para facilitar a un grupo de individuos/organizaciones la realización de tareas compartidas por medio de la comunicación, coordinación y/o colaboración efectiva de sus miembros.
Sistema cooperativo	v. *sistema colaborativo.*
Stakeholder	Cualquier persona que tiene influencia directa o indirecta sobre los requerimientos de un sistema [Sommerville, 2002].
Subactividad	Unidad de trabajo formada por un conjunto de acciones y otras subactividades que permite estructurar tareas.
Swiki (Squeak	Es una wiki escrita en el lenguaje de programación

Wiki)	Squeak [Wikipedia, 2007].
Tarea	Conjunto de subactividades/acciones cuya realización permite alcanzar objetivos. Permiten estructurar y describir el trabajo a llevar a cabo por el grupo en el sistema cooperativo. Poseen un alto nivel de abstracción y se relacionan, o incluso se identifican, directamente con objetivos de los usuarios o del grupo.
Tarea cooperativa	Tarea en la que interviene más de un actor, desempeñando el mismo o distinto rol.
Telepuntero	Cursor que aparece sobre más de una pantalla y que puede ser movido por diferentes usuarios. Cuando es movido en una de las pantallas éste se mueve en todas [Ellis et al., 1991].
Ventana de grupo	Colección de ventanas cuyas instancias aparecen sobre diferentes pantallas, y están conectadas de manera que si se realiza una operación sobre alguna de sus instancias su resultado se ve reflejado en el resto. Por ejemplo, si se dibuja un círculo en una de las instancias éste aparece en el resto [Ellis et al., 1991].
Vista	Representación (puede haber varias para una misma información) de una parte de un contexto compartido [Ellis et al., 1991].
Wiki	Aplicación web que permite a los usuarios añadir y editar contenido [Wikipedia, 2007].

Apéndice D

Introducción a los Diagramas de Actividad de UML 2

Contenido

1. Introducción

Mientras que los *diagramas de actividad* eran tratados en UML 1 [OMG, 2003] como casos especiales de máquinas de estado, en UML 2 [OMG,

2007b] éstos tienen una semántica inspirada en las Redes de Petri[1], lo que nos reporta una mayor potencia y flexibilidad para el modelado de procesos de negocio distribuidos, como son los flujos de trabajo de un sistema cooperativo.

UML 2.1.1, la versión oficial actual del estándar, dedica buena parte de su especificación a los diagramas de actividad [OMG, 2007b, pp. 295-417]. Además de una nueva semántica, esta especificación proporciona una sintaxis rica en nuevos elementos de modelado. No obstante, la mayor parte de los diagramas los podemos modelar haciendo uso de tan sólo una pequeña parte de la notación propuesta, existiendo elementos, los cuales podríamos considerar avanzados, que serán utilizados en contadas ocasiones. Como va a poder observar el lector, aunque con una semántica diferente, se mantiene una notación similar a la utilizada en UML 1 para muchos de los elementos, lo que facilita bastante la transición de una versión a otra.

Para una mejor comprensión de los diagramas de actividad que aparecen en esta tesis, en este apéndice hacemos una breve recapitulación de la sintaxis y semántica de los elementos gráficos empleados. Aunque la mayor parte de los diagramas son interpretados in situ, creemos adecuado proporcionar los conocimientos básicos necesarios al lector que lo requiera.

2. Nociones fundamentales

Una *actividad* representa un comportamiento que es compuesto por elementos individuales, los cuales pueden ser acciones y/o otras actividades (subactividades).

Una *acción* representa un paso simple desde el punto de vista de la actividad que la contiene.

Un *diagrama de actividad* es una red de nodos interconectados por extremos (arcos) que modela el comportamiento de una actividad.

Los *nodos* pueden ser de varios tipos:

[1] Si desea obtener información detallada sobre este formalismo puede consultar http://www.informatik.uni-hamburg.de/TGI/PetriNets/

- De acción: Representan las acciones.
- De control: Controlan el flujo a través de la actividad.
- De objeto: Representan los objetos usados en la actividad.

Los *extremos* pueden ser:

- De control: Representan el flujo de control a través de la actividad.
- De objeto: Representan el flujo de objetos a través de la actividad.

La semántica de las actividades está basada en el flujo de tokens. Por flujo entendemos que la ejecución de un nodo afecta, y es afectado por, la ejecución de otros nodos. Tales dependencias son representadas por los extremos que los conectan.

Un *token* representa un foco de control, un objeto o un dato, el cual siempre está en alguno de los nodos en un momento determinado. El estado del sistema que estamos modelando depende, por tanto, de la disposición de los tokens en cada momento dentro del diagrama. Cada token es distinto de los demás, aún cuando tengan el mismo valor.

Una acción puede tener conjuntos de extremos de entrada y salida que especifican los flujos de datos y de control desde/hacia otros nodos. La secuenciación de las acciones es controlada por los extremos de control y de objeto, los cuales trasportan los tokens de control y de objeto respectivamente.

En general, un nodo puede comenzar su ejecución cuando todas las condiciones[2] especificadas sobre sus tokens de entrada son satisfechas (join implícito). La ejecución de la acción se habilita consumiendo los tokens de control y de objeto que hay en la entrada y borrándolos de sus fuentes. Si hay varios tokens de control en un mismo extremo, es como si se hubiera ofrecido uno en ese momento. Cuando el nodo completa su ejecución, se ofrecen tokens de objeto sobre todos sus extremos de objeto y tokens de control sobre los de control (fork implícito).

[2] Estas condiciones dependerán del tipo de nodo.

Tanto las actividades como las acciones pueden llevar pre y post condiciones asociadas. Para las acciones se especifica mediante los estereotipos `<<localPrecondition>>` y `<<localPostcondition>>` (v. sec. 3.1)) y para las actividades se usa `<<precondition>>` y `<<postcondition>>` (v. sec. 4). Estas condiciones deberán cumplirse antes y después, respectivamente, de la ejecución de la acción o actividad que corresponda.

Si dos acciones no están directa o indirectamente ordenadas por relaciones de flujo pueden ejecutarse concurrentemente, lo que no quiere decir que su ejecución sea necesariamente paralela.

Cuando dos acciones tienen extremos de entrada que parten de un mismo nodo de objeto, ambas acciones compiten por el token que llegue a dicho nodo de objeto, es decir, cuando una lo toma la otra se queda sin él. No se replica el token de objeto.

Como se puede suponer, la semántica de flujo de tokens permite la especificación de escenarios altamente distribuidos, por lo que será necesario tener en cuenta los problemas de tiempo y competencia que pudieran aparecer.

3. Sintaxis y semántica de los distintos tipos de nodos

3.1. Nodos de acción

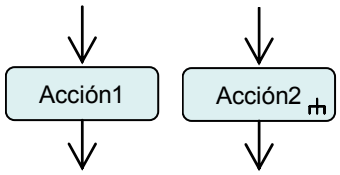
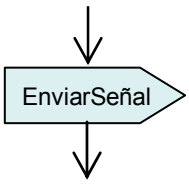
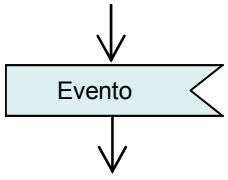
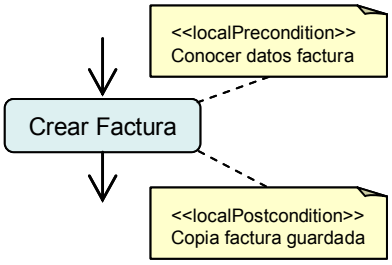
Por lo general, para que comience la ejecución de un nodo de acción es necesario que se cumplan las siguientes dos condiciones:

- Todos y cada uno de sus extremos de entrada poseen tokens, y
- Los tokens satisfacen las precondiciones locales del nodo de acción.

Al terminar su ejecución, se comprueban las postcondiciones locales, si las hay. Si se cumplen, el nodo de acción ofrece simultáneamente tokens en todos sus extremos de salida.

Veamos la sintaxis y semántica asociada con los nodos de acción en la tabla D.1:

Tabla D.1: Sintaxis y semántica relacionada con los nodos de acción

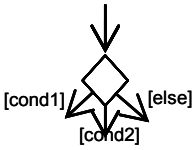
Sintaxis	Semántica
Acción de Llamada 	Invoca una operación, comportamiento o actividad. Para indicar que una acción referencia una actividad que hemos definido incluimos un símbolo de tridente o rastrillo (véase Acción2).
Acción de envío de señal 	Envía una señal asincrónicamente, con lo que el emisor no espera la confirmación de la recepción.
Acción de aceptación de evento 	Espera el evento indicado. Se activa si hay un token en su extremo de entrada. En caso de que no exista tal extremo, se activa cuando comienza la actividad que lo contiene y permanece activo en todo momento mientras que dicha actividad no termine. Cuando se recibe el evento especificado se pasa el control al siguiente nodo.
Acción de aceptación de evento de tiempo 	Igual que el anterior, pero en este caso el evento se define a través de una expresión de tiempo.
Precondición y postcondición local 	Un nodo de acción puede llevar asociada una precondición y/o una postcondición para su ejecución. En la figura anexa podemos observar cómo para realizar la acción `Crear Factura` es necesario conocer antes los datos de la factura y, una vez ejecutada la acción, debe existir una

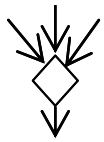
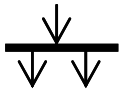
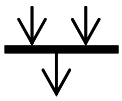
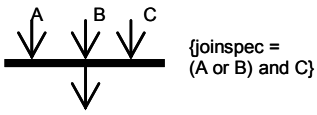
Sintaxis	Semántica
	copia guardada de la factura.

3.2. Nodos de control

En la tabla D.2 mostramos la sintaxis y semántica de los diferentes nodos de control:

Tabla D.2: Sintaxis y semántica relacionada con los nodos de control

Sintaxis	Semántica
Nodo inicial 	Representa el lugar donde comienza el flujo cuando se invoca una actividad. Puede haber más de un nodo inicial, comenzando el flujo simultáneamente en varios lugares. No es obligatorio ya que una actividad puede iniciarse también con una acción de aceptación de evento o a partir de un nodo parámetro de actividad.
Nodo final de actividad 	Finaliza una actividad, es decir, termina todos los flujos dentro de una actividad. Si una actividad tiene varios nodos de este tipo, el primero en activarse finaliza la actividad.
Nodo final de flujo 	Termina ese flujo concreto dentro de la actividad, el resto de flujos continúan.
Nodo de decisión 	El token que llegue al extremo de entrada se ofrecerá a aquel extremo de salida cuya condición se evalúe como verdadera. Sólo una de las condiciones debería ser verdadera. Puede usarse la palabra clave `else` si ninguna

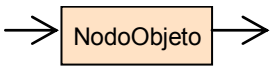
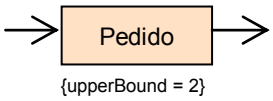
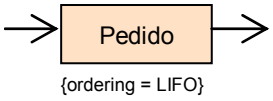
Sintaxis	Semántica
	de las condiciones es verdadera.
Nodo de fusión 	Los tokens ofrecidos en los extremos entrantes se ofrecen al extremo de salida.
Nodo fork 	Divide un simple flujo en múltiples flujos concurrentes. Los tokens que llegan al extremo entrante se duplican y se ofrecen en todos los extremos salientes simultáneamente. Cada extremo saliente puede llevar asociada una condición, pudiendo atravesar el token solamente aquel extremo cuya condición se evalúe como verdad.
Nodo join  	Sincroniza múltiples flujos concurrentes. Se ofrece un token en su único extremo de salida, en principio, cuando existe un token en todos sus extremos de entrada. Sin embargo, podemos realizar opcionalmente una especificación de sincronización para modificar esta semántica. Por ejemplo, la segunda figura muestra sus extremos de entrada etiquetados (`A`, `B` y `C`) y una especificación de join indicando que se ofrecerá un token de salida cuando haya un token en el extremo `C` y además en alguno de los otros dos (`A or B`).

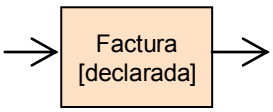
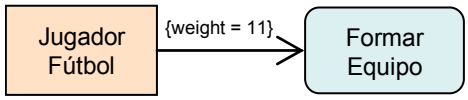
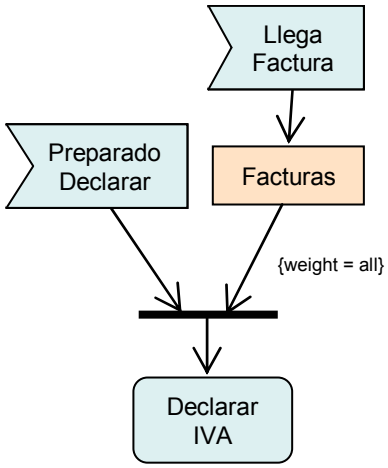
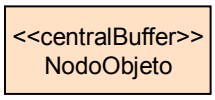
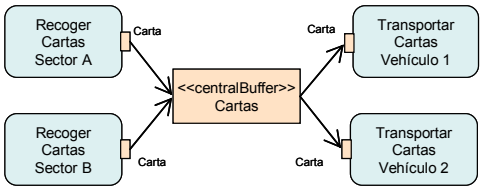
Los extremos que llegan o salen de un nodo de decisión, merge o fork deben ser todos de objeto o todos de control. En el caso de un join, si un extremo de entrada es de objeto entonces su extremo de salida será también de objeto.

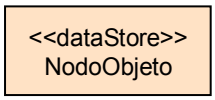
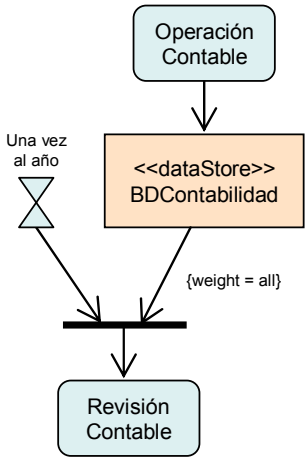
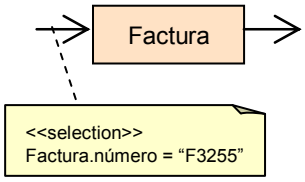
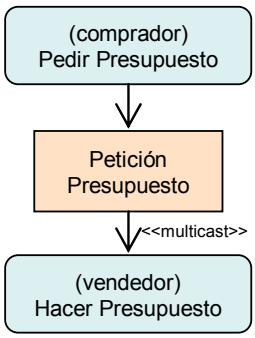
3.3. Nodos de objeto

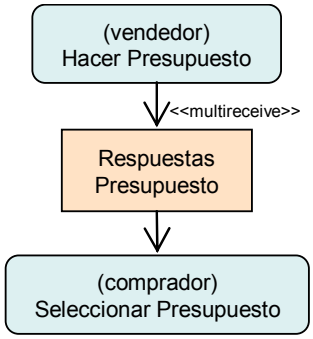
La tabla D.3 muestra la sintaxis y semántica correspondiente a los distintos tipos de nodos y flujos de objeto:

Tabla D.3: Sintaxis y semántica relacionada con los nodos de objeto

Sintaxis	Semántica
Nodo de objeto 	Representan puntos específicos de la actividad en los que puede haber objetos disponibles. Se identifican con el nombre del clasificador cuyos objetos son instancias de éste. Los nodos de objeto proveen y aceptan tokens de objeto, esto es, los extremos de entrada y de salida son flujos de objeto. Los tokens de objeto se crean y consumen por los nodos de acción. Tiene semántica de buffer, es decir, en ellos pueden residir tokens de objeto mientras esperan a ser aceptados por otros nodos.
Limite superior 	Por defecto, todo nodo de objeto puede almacenar un número infinito de tokens de objeto. No obstante, es posible especificar el número máximo de tokens que puede albergar y a partir del cual no aceptará más tokens. En el ejemplo, el límite máximo de tokens de objeto de tipo pedido que es posible albergar es 2.
Ordenación 	Indica el orden en el que salen los tokens de objeto. En el ejemplo, el último token de objeto en entrar es el primero en salir. El modo por defecto es FIFO, es decir, el primero en entrar es el primero en salir.
Estado	Muestra el estado de los tokens de objeto en

Sintaxis	Semántica
	ese nodo de objeto. En el ejemplo, la `factura` posee el estado de `declarada`. Se puede especificar una serie de estados separados por comas.
Peso  	Especifica el número mínimo de tokens que deben atravesar el extremo de salida al mismo tiempo. Cuando se alcanza ese mínimo, todos los tokens en la fuente son ofrecidos al destino al mismo tiempo. En el primer ejemplo, cuando el nodo de objeto dispone de 11 jugadores `{weight = 11}` se pasan en bloque a la siguiente acción (`Formar Equipo`). El peso también puede especificar una condición, por ejemplo `{weight = no_más_jugadores}`. Un peso nulo se indica como `{weight = all}`, lo que permite pasar todos los que hay en ese momento. En el segundo ejemplo, cuando llega el momento de hacer la declaración del IVA, se leen todas las facturas en conjunto y se realiza la acción `Declarar IVA`. El valor por defecto del peso es 1.
Buffer central  	El nodo de objeto acepta tokens de nodos de objeto entrantes y los pasa a los nodos de objeto salientes. Los tokens transitan, siendo la elección de los extremos salientes no determinista. En el ejemplo, aparecen varias acciones que ofrecen objetos de tipo `Carta`, los cuales transitan por el buffer central y son ofrecidos indistintamente a varias acciones.

Sintaxis	Semántica
Almacén de datos  	El nodo de objeto mantiene todos los tokens que entran, mandándose una copia de éstos a su extremo de salida cuando son elegidos. A diferencia del buffer central, los tokens se mantienen y no se mueven. En el ejemplo, las operaciones contables se almacenan en una base de datos y una vez al año se revisa todo su contenido.
Selección 	El nodo de objeto selecciona los tokens de objeto de entrada según algún criterio especificado en una nota estereotipada como `<<selection>>`. En el ejemplo, el nodo de objeto selecciona solo aquellos tokens de objeto Factura cuyo número sea “F3255”.
Multidifusión 	Los tokens de objeto se envían a múltiples receptores. Se estereotipa el flujo de objeto como `<<multicast>>`. El ejemplo representa el envío de peticiones de presupuesto a múltiples vendedores. Nótese que hemos utilizado la notación textual alternativa que UML ofrece para la especificación de las particiones o calles (`comprador` y `vendedor`) de un diagrama de actividad.

Sintaxis	Semántica
	COMO-UML proporciona una notación mucho más rica para estos casos, haciendo uso de las expresiones de rol (v. Apéndice A).
Multirecepción 	Los tokens de objeto se reciben de múltiples emisores. Se estereotipa el flujo de objeto con `<<multireceive>>`. En la figura mostramos la parte que complementa al ejemplo anterior. Se reciben, desde múltiples vendedores, las respuestas al presupuesto solicitado.

4. Actividades

Una actividad especifica un comportamiento subordinado. Mientras que una instancia de una acción es usada sólo una vez en un punto particular de una actividad, una actividad define un comportamiento que puede ser reutilizado en muchos lugares. Esto es así debido a que esa actividad concreta puede ser invocada desde diferentes instancias de una misma acción.

Las actividades pueden llevar parámetros que proporcionan sus entradas y salidas. Éstos se representan como nodos de objeto que solapan el marco de la actividad. Los parámetros pueden añadir soporte para streaming (v. sec. 9), excepciones (v. sec. 7) y conjuntos de parámetros (v. sec. 5). También es posible añadir información sobre el tipo de cada parámetro (v. Figura D.1).

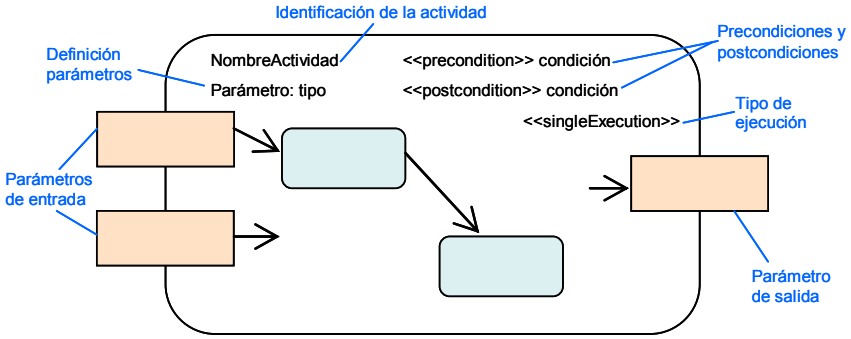


Fig. D.1: Representación de una actividad

Dentro del marco de la actividad podemos indicar la precondición y la postcondición que debería satisfacerse antes y después de su ejecución respectivamente (v. `<<precondition>>` y `<<postcondition>>`).

Así mismo, es posible especificar si la misma ejecución de una actividad maneja los tokens para todas las invocaciones (v. `<<singleExecution>>`), o una ejecución separada de esa actividad es creada para cada invocación (opción por defecto). En el primer caso, como se comparte el mismo espacio, hay que considerar las interacciones entre los múltiples flujos de tokens moviéndose a través de los nodos y extremos. En el segundo caso, los tokens no interactúan.

Cuando todos los nodos de objeto de salida reciben tokens y no queda ningún otro token (de control o de objeto) fluyendo, entonces la actividad termina y devuelve los valores de salida a la acción que la invoca.

5. Pins

Un pin es un nodo de objeto que representa una entrada o una salida de una acción. Los pins de entrada tienen exactamente un extremo de entrada y, de la misma forma, los pins de salida un único extremo de salida. Al ser tan pequeño el símbolo del pin, el identificador del nodo de objeto que representa debe aparecer fuera de éste, pero lo más cerca posible del símbolo (v. Figura D.2).

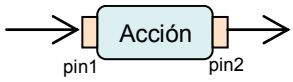


Fig. D.2: Representación de los pins de entrada y pins de salida de una acción

Cuando los objetos de entrada y salida son del mismo tipo caben dos representaciones alternativas. Por ejemplo, los diagramas que aparecen en la Figura D.3 los podemos considerar equivalentes.

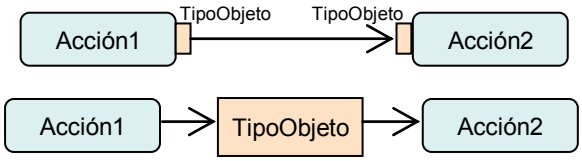


Fig. D.3: Representaciones equivalentes con y sin pins

Es posible agrupar los pins de entrada o salida, lo que permite especificar conjuntos de parámetros de entrada o salida alternativos. Por ejemplo, la Figura D.4 muestra en el diagrama de la izquierda que la acción C se podrá ejecutar cuando sus dos pins de entrada (P1 y P2) reciben tokens (AND lógico), como es lo habitual, mientras que el diagrama de la derecha agrupa los pins para que dicha acción comience su ejecución con la llegada de un token a P1 ó, alternativamente, a P2 (XOR lógico).

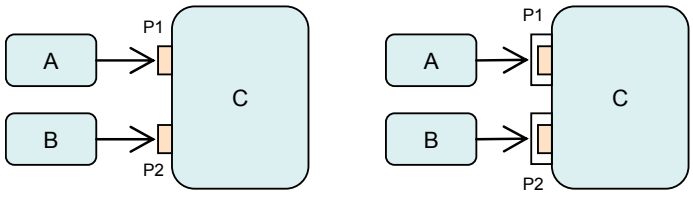


Fig. D.4: Conjuntos de parámetros alternativos

6. Regiones interrumpibles de actividad

Una región interrumpible de actividad es un área, representada como un rectángulo punteado con las esquinas redondeadas, cuya ejecución puede interrumpirse cuando un token la atraviesa por medio de un tipo de extremo

especial, denominado extremo de interrupción. Este tipo de extremo lo simbolizamos como una flecha en zigzag. Por ejemplo, la Figura D.5 representa una región interrumpible de actividad, englobando a los nodos `B`, `C` y `Cancelar`, que será interrumpida cuando la acción de aceptación de evento `Cancelar` reciba el evento correspondiente mientras el flujo de control esté en dicha región.

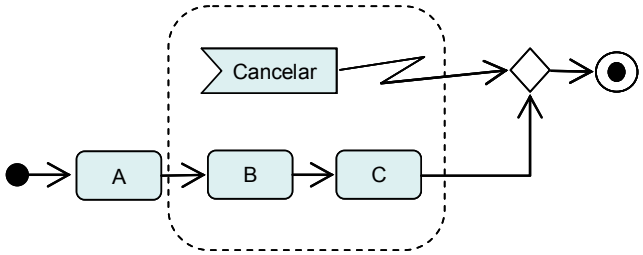


Fig. D.5: Región interrumpible de actividad

7. Gestión de excepciones

A veces es necesario controlar las excepciones que pueden producirse durante la ejecución de alguna acción. Esto lo representamos mediante un extremo de interrupción, etiquetado con el tipo de excepción, que conecta con el pin de un manejador de excepción (actividad que trata el error producido). Por ejemplo, la Figura D.6 muestra que si se produce la excepción `TipoExcepción` (overflow, p. ej.) cuando se está ejecutando `Acción1 Protegida`, se ejecuta la actividad asociada a `Manejador Excepción` y a continuación se realiza `Acción2`.

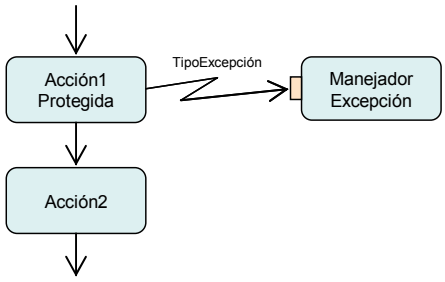


Fig. D.6: Acción protegida y su manejador de excepción

Un parámetro de salida de una actividad, o un pin de salida de una acción, puede ser especificado como de excepción mediante la incorporación de un pequeño triángulo junto a éste (v. Figura D.7). Un token que llega a un parámetro de excepción de una actividad aborta el resto de flujos de esa actividad.

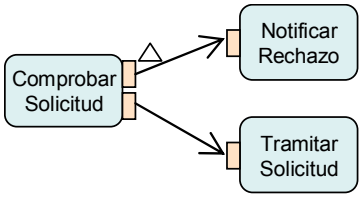


Fig. D.7: Pin de excepción

8. Regiones y nodos de expansión

Una región de expansión es una región de actividad que se ejecuta múltiples veces de acuerdo con los elementos de una colección de entrada.

Un nodo de expansión es un nodo de objeto que representa una colección de objetos que fluyen hacia dentro o hacia fuera de una región de expansión.

La región es ejecutada una vez por cada elemento de la colección. El tipo de la colección de salida debe ser igual al tipo de la colección de entrada. Los tipos de los objetos contenidos en ambos tipos de colecciones deben coincidir.

El número de colecciones de entrada puede variar del número de colecciones de salida. Cada ejecución, si genera un valor de salida en la colección de salida, depositará este valor en la misma posición que ocupa el valor de entrada en la colección de entrada.

Se debe especificar el orden en el que se procesan los elementos de la/s colección/es de entrada, ya que no existe ningún modo por defecto. Existen tres modos posibles que se expresan mediante las siguientes palabras clave:

- *Parallel*. Procesa cada elemento de la colección de entrada en paralelo, es decir, solapado en el tiempo.

- *Iterative*. Procesa cada elemento de la colección de entrada en secuencia, es decir, cuando termina uno pasa el siguiente.

- *Stream*. Procesa cada elemento de la colección de entrada conforme va llegando al nodo de expansión.

Una acción con un asterisco en su interior colocado en la parte superior izquierda es una forma alternativa de representar el modo de procesamiento paralelo.

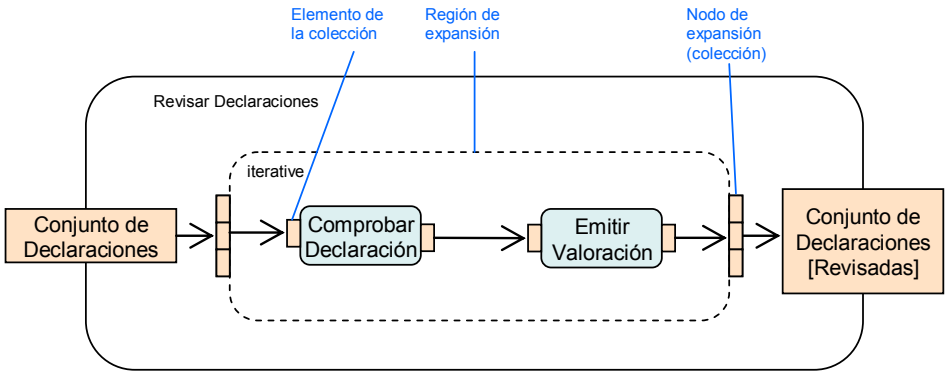


Fig. D.8: Región y nodos de expansión

La Figura D.8 muestra una región de expansión iterativa perteneciente a la actividad `Revisar Declaraciones`, la cual procesa secuencialmente cada uno de los objetos de tipo declaración pertenecientes a la colección incluida en el parámetro de entrada `Conjunto de Declaraciones`. Esta colección es ofrecida al nodo de expansión de entrada a la región y, cada vez que se procesa el elemento de una determinada posición, su resultado se deposita en la misma posición dentro del nodo de expansión de salida de la región. Al finalizar, esta colección se ofrece al nodo que actúa como parámetro de salida, devolviendo la misma colección pero con el estado de `Revisadas`.

9. Streaming

El streaming permite que la ejecución de una acción tome entradas y provea salidas mientras se está ejecutando, pudiendo aceptar y producir múltiples tokens durante una misma ejecución.

Como sabemos, la ejecución de una acción implica, por defecto, coger los tokens de sus extremos de entrada y, una vez terminada su ejecución, ofrecer tokens a sus extremos de salida. Sin embargo, a veces es necesario indicar que la acción acepta y ofrece tokens continuamente.

UML 2 dispone de varias formas alternativas para representar esta misma situación. La Figura D.9 muestra un ejemplo representado mediante cada una de las opciones posibles.

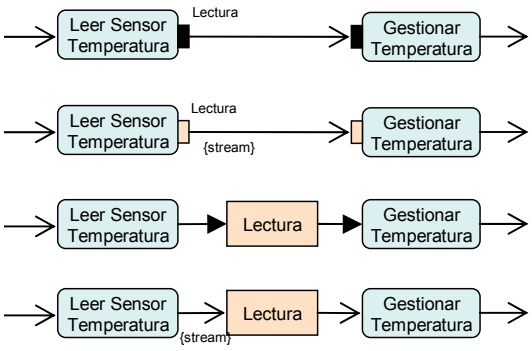


Fig. D.9: Formas de representar el streaming

Los nodos parámetros de una actividad también pueden ser especificados como stream. Para ello, simplemente ponemos la restricción `{stream}` junto a éstos (v. Figura D.10).

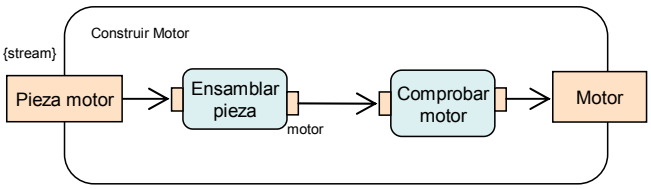


Fig. D.10: Parámetro de actividad con flujo continuo (stream)

-Esta página se encuentra deliberadamente en blanco-

Bibliografía

AARSTEN, A.; MENGA, G.; MOSCONI, L. (1996). "Object-Oriented Design Patterns in Reactive Systems". In Vlissides, Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA: Addison-Wesley, pp. 537-548

ALBIN-AMIOT, H.; GUEHENEUC, Y. G. (2001). "Metamodelling Design Patterns: Application to Pattern Detection and Code Synthesis". In *Proc. of ECOOP Workshop on Automating Object-Oriented Software Development Methods*

ALEXANDER, C.; ISHIKAWA, S.; SILVERSTEIN, M.; JACOBSON, M.; FIKSDAHL-KING, I.; ANGEL, S. (1977). *A pattern Language: Towns, Buildings, Construction*, New York: Oxford University Press

ALEXANDER C. (1979). *The Timeless Way of Building*, Oxford University Press

ANDERSON, B.; COAD, P. (1994). "Patterns workshop". In *OOPSLA'93 Addendum to the Proceedings*, Washington, D.C., January 1994, ACM Press

ANTHONY, Dana L.G. (1996). "Patterns for Classroom Education". In Vlissides, Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA: Addison-Wesley, pp. 391-406

APPELT, W. (1999). "WWW Based Collaboration with the BSCW System". In *Proceedings of SOFSEM'99*, Springer Lecture Notes in Computer Science 1725, p. 66-78

ARONSON, E.; BLANEY, N.; STEPHIN, C.; SIKES, J.; SNAPP, M. (1978). *The jigsaw classroom*, Beverly Hills, CA: Sage Publishing Company

ARONSON, Elliot (2000). *The Jigsaw Classroom: A Cooperative Learning Technique*, (consultada el 11-9-00), `http://www.jigsaw.org`

BAECKER, R. M. (ed.) (1993). *Readings in Groupware and Computer-Supported Cooperative Work*, Morgan Kaufmann, San Mateo, CA

BANNON, L.; SCHMIDT, K. (1989). "CSCW: Four Characters in Search of a Context". *ECSCW´89*, Gatwick, London

BAYLE, E.; BELLAMY, R.; CASADAY, G.; ERICKSON, T. (1998). "Putting it all together: towards a pattern language for interaction design", *SIGCHI Bulletin*, vol. 30 (1), pp. 17-23

BECK, Kent (1988). "Using a Pattern Language for Programming". In *Addendum to the Proceedings of OOPSLA'87*, volume 23(5) of *ACM SIGPLAN Notices*, p. 16

BECK, Kent; COPLIEN, James O. ; CROCKER, Ron; DOMINICK, Lutz; MESZAROS, Gerard; PAULISCH, Frances; VLISSIDES, John (1996). "Industrial Experience with Design Patterns". In *Proceedings of the 18th International Conference on Software Engineering*

BERGHOLTZ, María; JOHANNESSON, Paul (2000). "Validating Conceptual Models - Utilising Analysis Patterns as an Instrument for Explanation Generation", *NLDB 2000*, pp. 325-339

BERNSTEIN, Mark. (1998). "Patterns of hypertext". In Frank Shipman, Elli Mylonas and Kaj Groenback (eds.), *Proceedings of Hypertext'98*, ACM, New York, pp. 21-29

BORCHERS, Jan (2001). *A Pattern Approach to Interaction Design*, Wiley Series in Software Design Patterns

BORCHERS, Jan (2005). *Jan Borcher's HCI patterns page*, (consultada el 28-8-2005), `http://www.hcipatterns.org`

BRADAC, Mark; FLETCHER, Becky (1998). "A Pattern Language for Developing Form Style Windows". In Martin, Riehle and Buschmann (eds.), *Pattern Languages of Program Design 3*, Reading, MA, Addison-Wesley, pp. 347-357

BROWN, Kyle (1996). "Using Patterns in Order Management Systems: A Design Patterns Experience Report", *Object Magazine*, January, 1996

BUSCHMANN, Frank; MEUNIER, Regine; ROHNERT, Hans; SOMMERLAD, Peter; STAL, Michael (1996). *Pattern-Oriented Software Architecture: A System of Patterns,* Chichester, UK, John Wiley and Sons Ltd

CABOT, J.; RAVENTÓS, R. (2004). "Roles as Entity Types: A Conceptual Modelling Pattern", *Conceptual Modeling – ER 2004*, LNCS 3288/2004, Springer-Verlag, pp. 69-82

CAÑAS, J.J.; WAERN, Y. (2001). *Ergonomía cognitiva*. Ed. Panamerica

CAÑETE, J.M.; GALÁN, F.J.; TORO, M. (2004). "Some Problems of Current Modelling Languages that Obstruct to Obtain Models as Instruments". In *Actas de las JISBD 2004,* Málaga, Spain

CARGILL, T. (1996). "Localized Ownership: Managing Dynamic Objects in C++". In Vlissides, Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA, Addison-Wesley, pp. 5-18

CARLSON, Andy; ESTEPP, Sharon; FOWLER, Martin (1999). "Temporal Patterns". In N. Harrison, B. Foote and H. Rohnert (eds.), *Pattern Languages of Program Design 4*, Reading, MA, Addison-Wesley, pp. 241-262

COAD, P.; NORTH, D.; MAYFIELD, M. (1995). *Object Models: Strategies, Patterns and Applications*, Prentice Hall, Englewood Cliffs

COBOS, R.; ALAMÁN, X. (2002). "Cristalización del Conocimiento de una Comunidad de Usuarios". *III Congreso Internacional de Interacción Persona-Ordenador*, Madrid, España, pp. 128-135

COBOS, R.; ALAMAN, X.; GEA, M.; GUTIERREZ, F.L.; GARRIDO, J.L. (2003). "Modelado del sistema para trabajo colaborativo KnowCat mediante AMENITIES". *Actas del IV Congreso Internacional Interacción Persona-Ordenador 2003 (Interacción'03)*, (Vigo, Spain, Mayo 2003)

COPLIEN, J. (1992). *Advanced C++: Programming Styles and Idioms*, Reading, Massachusetts, Addison-Wesley

COPLIEN, James O.; SCHMIDT, Douglas C. (eds.) (1995). *Pattern Languages of Program Design*, Reading, Massachusetts, Addison-Wesley Publishing Company

COPLIEN, James O. (1996). *Software Patterns*, SIGS Books

COPLIEN, James O. (1995). "A Generative Development-Process Pattern Language". In Coplien and Schmidt (eds.), *Pattern Languages of Program Design*, Reading, Massachusetts, Addison-Wesley Publishing Company, pp. 183-238

COPLIEN, James O. (1999). "C++ Idioms". In N. Harrison, B. Foote and H. Rohnert (eds.), *Pattern Languages of Program Design 4*, Reading, MA, Addison-Wesley, pp. 167-198

CRABTREE, Andy; RODDEN, Tom (2002). "Patterns: Problem and Solutions?", *CSCW Workshop on Socio-Technical Pattern Languages*, (New Orleans, November 16-20, 2002)

DAS NEVES, Fernando; GARRIDO, Alejandra (1998). "BodyGuard". In Martin, Riehle and Buschmann (eds.), *Pattern Languages of Program Design 3*, Reading, MA, Addison-Wesley, pp. 231-244

DAVID, B.; DELOTTE, O.; CHALON, R.; TARPIN-BERNARD, F.; SAIKALI, K. (2003). "Patterns in Collaborative System Design, Development and Use", *2nd Workshop on Software and Usability Cross-Pollination: The Role of Usability Patterns*, (September 1-2, 2003, Zürich, Switzerland)

DEARDEN, A.; FINLAY, J.; ALLGAR, E.; McMANUS, B. (2002). "Using Pattern Languages in Participatory Design", *Proceedings PDC 2002*, (23-25 June, 2002, Malmö, SWE), pp. 104-113

DEBNATH, N. C.; GARIS, A.; RIESCO, D.; MONTEJANO, G. (2006). "Defining Patterns Using UML Profiles", *IEEE International Conference on Computer Systems and Applications 2006.*, pp. 1147-1150

DEBRULER, Dennis L. (1995). "A Generative Pattern Language for Distributed Processing". In Coplien and Schmidt (eds.), *Pattern Languages of Program Design*, Reading, Massachusetts, Addison-Wesley Publishing Company, pp. 69-90

DESANCTIS, G. L.; GALLUPE, R. B. (1987). "A Foundation for the Study of Group Decision Support Systems". *Management Science*, 33(5), pp. 589-609

DILLENBOURG, P.; BAKER, M.; BLAYE A.; O'MALLEY, C. (1995). "The Evolution of Research on Collaborative Learning". In P. Reimann & H. Spada (Eds). *Learning in Humans and Machines. Towards an Interdisciplinary Learning Science*, London: Pergamon, pp. 189-211

DISCENZA, A.; GARZOTTO, F. (1999). "Design Patterns for Museum Web Sites". In *Proceedings MW'99 – 3rd International Conference on Museums and the Web*, New Orleans, USA, March 1999, pp. 144-153

DIX, A.; FINLAY, J.; ABOWD, G.; BEALE, R. (1998). *Human Computer Interaction, second edition*. Prentice Hall

DONG, J.; YANG, S. (2003). "Visualizing Design Patterns with a UML Profile". In *Proc. of the IEEE Symposium on Visual/Multimedia Languages (VL), (*Auckland, New Zealand, October, 2003), pp. 123-125

DOURISH, P.; BELLOTTI, V. (1992). "Awareness and Coordination in Shared Workspaces". In *Proceedings of ACM CSCW'92 Conference on Cumputer Supported Cooperative Work*, Toronto, Canada, pp. 107-114

DURÁN, Amador (2000). *Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información*. Tesis Doctoral, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla

EDEN, A. H. (2000). *Precise Specification of Design Patterns and Tool Support in their Application.* PhD diss., Department of Computer Science, Tel Aviv University

ELLIS, C.A.; GIBBS, S.J.; REIN, G.L. (1991). "Groupware: Some Issues and Experiences". *Communications of the ACM*, 34(1):38-58

ERICKSON, Thomas (2000a). "Lingua francas for design: sacred places and pattern languages". In *Proceedings of DIS 2000* (Brooklyn, NY, August 17-19, 2000), New York, ACM Press, pp. 357-368

ERICKSON, Thomas (2000b). "Supporting interdisciplinary design: towards pattern languages for workplaces". *Workplace Studies* (eds. Luff, P.; Hindmarsh, J.; Heath, C.), pp. 252-261, Cambridge University Press.

ERICKSON, Thomas (2004). *The Interaction Design Patterns Page*, (consultada el 9-3-2007), http://www.visi.com/~snowfall/InteractionPatterns.html

FAYAD, M. E.; ALTMAN, A. (2001). "Introduction to Software Stability", *Comm. of the ACM*, Vol. 44, No. 9, Sept. 2001

FERNANDEZ, Eduardo B; YUAN, X. (1999). "An analysis pattern for reservation and use of entities". In *Procs.of PLoP99*, http://st-www.cs.uiuc.edu/~plop/plop99

FERNANDEZ, E. B. (2000). "Stock manager: An analysis pattern for inventories". In *Proceedings of PloP 2000*.

FERNANDEZ, E. B.; YUAN, X.; BREY, S. (2000). "Analysis Patterns for the Order and Shipment of a Product". In *Proceedings of PLoP 2000*.

FERNÁNDEZ, Eduardo B.; YUAN, Xiaohong (2000). "Semantic Analysis Patterns". *In Procs. of 19th Int. Conf. on Conceptual Modeling (ER2000)*, pp. 183-195

FINCHER, S. (2000). "HCI Pattern-Form Gallery", (consultada el 27-3-2007), http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html

FOWLER, Martin (1997). *Analysis Patterns: Reusable Object Models*. Booch, G., Jacobson, I. and Rumbaugh, J. (eds.), Object Technology Series, Reading, MA, Addison-Wesley Publishing Company

FOWLER, Martin (1996). "Accountability and Organizational Structures". In Vlissides, Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA, Addison-Wesley, pp. 353-370

FUXMAN, A.; GIORGINI, P.; KOLP, M; MYLOPOULOS J. (2001). "Information Systems as Social Structures", In *Proceedings of the 2nd International Conference on Formal Ontologies for Information Systems (FOIS'01)*, Ogunquit, USA

FRANCE, Robert B.; KIM, Dae-Kyoo; GHOSH, Sudipto; SONG, Eunjee (2004). "A UML-Based Pattern Specification Technique". *IEEE Transactions on Software Engineering*, Vol. 30, Nº 3, Marzo 2004

GAMMA, Erich (1991). *Object-Oriented Software Development based on ET++*. PhD thesis, University of Zurich, Institut für Informatik

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John (1993). "Design Patterns: Abstraction and Reuse of Object-Oriented Design". In O. Nierstrasz (ed.), *European Conference on Object-Oriented Programming (ECOOP)*, (Kaiserslautern, Germany, July, 1993), Springer Verlag, LNCS 707

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, MA, Addison Wesley Professional Computing Series

GARRIDO, J. L.; GEA, M.; GUTIÉRREZ, F. L.; PADILLA, N. (2000). "Designing Cooperative Systems for Human Collaboration". In Dieng, R., Giboin, A., Karsenty, L., De Michelis, G. (eds.), *Designing Cooperative Systems – The Use of Theories and Models*, IOS Press-Ohmsha, pp. 399-412

GARRIDO, José Luis; GEA, Miguel (2001). "Modelling Dynamic Group Behaviours". In Johnson, C. (ed.) *Interactive Systems – Design, Specification and Verification*, LNCS 2220, Springer, pp. 128-143

GARRIDO, José Luis; GEA, Miguel (2002). "A Coloured Petri Net Formalisation for a UML-based Notation Applied to Cooperative System Modelling". In *Proceedings of IX Workshop on Design Specification and Verification of Interactive System*

GARRIDO, J. L.; GEA, M.; PADILLA, N.; CAÑAS, J. J.; WAERN, Y. (2002). "AMENITIES: Modelado de Entornos Cooperativos". En Aedo, I., Díaz, P., Fernández, C. (eds.), *Actas del III Congreso Internacional Interacción Persona-Ordenador 2002 (Interacción'02)*, (Madrid, Spain, mayo 2002), pp. 97-104

GARRIDO, José Luis (2003). *AMENITIES: Una Metodología para el Desarrollo de Sistemas Cooperativos Basada en Modelos de Comportamiento y Tareas*. Tesis Doctoral, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Granada

GARRIDO, J. L.; GEA, M.; NOGUERA, M.; GONZÁLEZ, M.; IBAÑEZ, J. A. (2004). "Una Propuesta Arquitectónica para el Desarrollo de

Aplicaciones Colaborativas". *Actas del V Congreso de Interacción Persona-Ordenador*, pp. 164-171

GARRIDO, J.L.; GEA, M.; RODRÍGUEZ, M.L. (2005a). "Requirements Engineering in Cooperative Systems". *Requirements Engineering for Sociotechnical Systems*, Chapter XIV, IDEA GROUP, Inc. USA, pp. 226-244

GARRIDO, J.L.; PADEREWSKI, P.; RODRÍGUEZ, M.L.; HORNOS, M.; NOGUERA, M. (2005b). "A Software Architecture Intended to Design High Quality Groupware Applications". In *Proc. of The 2005 International Conference on Software Engineering Research and Practice*, CSREA Press, pp. 59-65

GARRIDO, J.L.; NOGUERA, M.; GONZÁLEZ, M.; GEA, M.; HURTADO, M. V. (2006). "Leveraging the Linda Coordination Model for a Groupware Architecture Implementation". *Groupware: Design, Implementation and Use*, LNCS, Vol. 4154, Springer, pp. 286–301

GARRIDO, J. L.; NOGUERA, M.; GONZÁLEZ, M.; HURTADO, M. V.; RODRÍGUEZ, M. L. (2007). "Definition and Use of Computation Independent Models in an MDA-based Groupware Development Process", *Science of Computer Programming*, vol. 66, nº 1, 2007, pp. 25–43

GARZOTO, Franca; PAOLINI, Paolo; BOLCHINI, Davide; VALENTI, Sara. (1999). "Modeling-by-Patterns of Web Applications". In P.P. Chen, D.W. Embley, J. Kouloumdjian, S.W. Liddle and J.F. Roddick (eds). *Advances in Conceptual Modeling, ER'99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling*. Paris, France, 13-18 November 1999, Springer-Verlag, Berlin-Heidelberg

GEA, M.; GUTIÉRREZ, F.L. (2001). "Aplicación de Técnicas Formales en el Diseño de Sistemas Cooperativos". En Abascal, J. et al. (eds.) *II Congreso Internacional de Interacción Persona-Ordenador (Interacción 2001)*, Salamanca, Spain

GEA, M.; GUTIÉRREZ, F.L.; GARRIDO, J.L.; CAÑAS J.J. (2003). "Teorías y Modelos Conceptuales para un Diseño basado en Grupos". *Actas del*

IV Congreso Internacional Interacción Persona-Ordenador 2003 (Interacción'03), (Vigo, Spain, Mayo 2003)

GERMÁN, D. M.; COWAN, D. D. (2000). "Towards a Unified Catalog of Hypermedia Design Patterns". In *Proceedings of the 33rd Hawaii International Conference on System Sciences*

GEYER-SCHULZ, Andreas; HAHSLER, Michael (2002). "Software reuse with analysis patterns". In *Proceedings of AMCIS 2002* (Dallas, TX, August 2002)

GONZÁLEZ, P.; GRANOLLERS, T.; GUTIÉRREZ, F. L.; ISLA, J. L.; MONTERO, F. (2007). "De los Patrones de Organización e Interacción al Diseño de Interfaces de Usuario Colaborativas". En *Actas del 5º Taller en Sistemas Hipermedia Colaborativos y Adaptativos (SHCA 2007)* dentro de las Jornadas de Ingeniería del Software y Bases de Datos (JISBD07), (Zaragoza, Spain, 11 de Septiembre de 2007), pp. 9-16

GRANLUND, Å.; LAFRENIÈRE, D.; CARR, D. (2001). "A pattern-supported approach to the user interface design process". In *Proceedings of HCI International 2001, 9th International Conference on Human-Computer Interaction* (New Orleans, LA, 5-10 August 2001), vol. 1, pp. 282-286

GREIF, I. (1988). *Computer-Supported Cooperative Work: A Book of Readings*. Morgan Kaufmann, San Mateo, California

GROENBAEK, K.; HEM, J.; MADSEN, O.; SLOTH, L. (1994). "Cooperative Hypermedia Systems: A Dexter-Based Architecture". In *Communications of the ACM (CACM)* Vol. 37, Nº 2, Feb. 1994, pp. 64-74

GRUDIN, J. (1993). "Groupware and Cooperative Work: Problems and Prospects". Reprinted in Baecker, R.M. (ed.), *Readings in Groupware and Computer Supported Cooperative Work*, San Mateo, CA, Morgan Kaufman Publishers, pp. 97-105

GRUDIN, Jonathan; POLTROCK, Steven E. (1997). "Computer-Supported Cooperative Work and Groupware". In M. Zelkowitz (Ed.), *Advances in Computers*, Vol. 45, Orlando: Academic Press, pp. 269-320

GUERRERO, Luis A.; FULLER, David A. (1999). "Design Patterns for Collaborative Systems". In *Proceedings of the Fifth International Workshop on Groupware (CRIWG)*

GUTIÉRREZ, F. L.; GEA, M.; GARRIDO, J. L.; PADILLA, N. (2002). "Desarrollo de sistemas interactivos en base a modelos de usuario". *Revista Iberoamericana de Inteligencia Artificial*, 16(1):71-82

GUTIÉRREZ, F. L.; ISLA, J. L.; GEA, M. (2003). "Modelando Patrones de Organización". En *Actas del II Taller de Sistemas Hipermedia Colaborativos y Adaptativos*, Jornadas de Ingeniería del Software y Bases de Datos (JISBD03), Alicante, España

GUTIÉRREZ, F. L.; ISLA, J. L.; PADEREWSKI, P.; SÁNCHEZ, M. (2006a). "Organization Modelling to Support Access Control for Collaborative Systems". In *Proceedings of The 2006 International Conference on Software Engineering Research and Practice*, CSREA Press, Las Vegas, Nevada, USA, pp. 757-763

GUTIÉRREZ, F. L.; PENICHET, V. M. R.; ISLA, J. L.; MONTERO, F.; LOZANO, M. D.; GALLUD, J. A.; RODRÍGUEZ, M. L. (2006b). "Un Marco Conceptual para el Modelado de Sistemas Colaborativos Empresariales". En *Actas del VII Congreso Internacional de Interacción Persona-Ordenador (Interacción 2006)*, Puertollano (Ciudad Real), España, 13-17 de noviembre de 2006

GUTIÉRREZ, F. L.; ISLA, J. L.; PADEREWSKI, P.; SÁNCHEZ, M.; JIMÉNEZ, B. (2007). "An Architecture for Access Control Management in Collaborative Enterprise Systems Based on Organization Models". *Science of Computer Programming*, vol. 66, nº 1, 2007, pp. 44–59

HAMZA, H. (2002) "A foundation for building stable analysis patterns", M.S. thesis, Dept. Computer Science, University of Nebraska-Lincoln, Lincoln, NE

HAMZA, H.; FAYAD, M. E. (2002a). "Model-based Software Reuse Using Stable Analysis Patterns", *ECOOP 2002*, Workshop on Model-based Software Reuse, (Malaga, Spain, June 2002)

HAMZA, H.; FAYAD, M. E. (2002b). "A Pattern Language for Building Stable Analysis Patterns", *9th Conference on Pattern Language of Programs (PLoP 02)*, (Illinois, USA, September 2002)

HARRISON, Neil (1996). "Organizational Patterns for Teams". In Vlissides, Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA, Addison-Wesley, pp. 345-352

HARRISON, Neil; FOOTE, Brian; ROHNERT, Hans (eds.) (1999). *Pattern Languages of Program Design 4*, Reading, MA, Addison-Wesley

HAY, D. C. (1996). *Data Model Patterns: Conventions of Thought*, Dorset House Publishing, New York, NY

HERRMANN, T.; LOSER, K-U (1999). "Vagueness in models of socio-technical systems", *Behaviour and Information Technology*, Vol. 18, nº 5, pp. 313-323

HERRMANN, T.; JAHNKE, I.; KUNAU, G.; SCHNEIDER, H. (2002). "Patterns of Socio-Technical Systems – Characteristics and Requirements in the Field of CSCW-Applications", *CSCW Workshop on Socio-Technical Pattern Languages*, (New Orleans, November 16-20, 2002)

HERRMANN, T.; HOFFMANN, M.; JAHNKE, I.; KIENLE, A.; KUNAU, G.; LOSER, K.; MENOLD, N. (2003). "Concepts for Usable Patterns of Groupware Applications", In Mark Pendergast, Kjeld Schmidt, Carla Simone, Marilyn Tremaine (Eds.): *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*, pp. 349-358

HILL, W.; STEAD, L.; ROSENTEIN, M.; FURNAS, G. (1995). "Recommending and Evaluating Choices in a Virtual Community of Use". *Proceedings of the Computer Human Interaction 1995* (CHI95), ACM Press, Denver, CO, USA, pp. 194-201

HILLSIDE Group (2007a). *Home of the Patterns Library*, (consultada el 15-2-07), `http://hillside.net`

HILLSIDE Group (2007b). *Pattern Languages*, (consultada el 15-2-07), `http://hillside.net/patterns/patternslanguages.htm`

HILLSIDE Group (2007c). *Tools*, (consultada el 15-2-07), http://hillside.net/patterns/tools/

HILLSIDE Group (2007d). *Conferences*, (consultada el 15-2-07), http://hillside.net/conferences

HILLSIDE Group (2007e). *Online Pattern Catalog*, (consultada el 15-2-07), http://hillside.net/patterns/onlinepatterncatalog.htm

HUGHES, J.A.; O'BRIEN, J.; RODDEN, T.; ROUNCEFIELD, M. (2000). "Ethnography, communication and support for design", *Workplace Studies* (eds. Luff, P.; Hindmarsh, J.; Heath, C.), Cambridge University Press, pp. 187-214.

ISLA, J. L.; ORTEGA, F. D. (1999). "Las Nuevas Tecnologías en la Comunicación Humana: La Videoconferencia en la Educación". *Actas del Congreso de Nuevas Tecnologías de la Información y la Comunicación para la Educación (EDUTEC99*), 14-17 septiembre, Sevilla

ISLA, J. L. (2001). *Un Modelo para la Especificación Estructural de Patrones y su Aplicación a UML*, Tesina, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Granada

ISLA, J. L.; ORTEGA, F. D. (2001). "Consideraciones para la implantación de la videoconferencia en el aula". *Revista PIXEL-BIT*, Secretariado de Recursos Audiovisuales y Nuevas Tecnologías de la Universidad de Sevilla, Vol. 17, junio 2001, pp. 23-31

ISLA, J. L. (2002). "A New Approach for Modelling Design Patterns with UML". *The 12th PHDOOS workshop (ECOOP2002)* (Málaga, Spain, 10-14 june 2002). Abstract in Hernández and Moreira (eds.), *ECOOP 2002 Workshops and Posters*, Springer-Verlag, LNCS 2548, http://www.softlab.ece.ntua.gr/facilities/public/AD/phdoos02/jose.ps

ISLA, J. L.; GUTIÉRREZ, F. L. (2002). "Diseño en Base a Patrones. Aplicación a Sistemas Hipermedia Colaborativos". En *Actas del I Taller en Sistemas Hipermedia Colaborativos y Adaptativos*, VII Jornadas de Ingeniería del Software y Bases de Datos, (El Escorial, Madrid, 18 al 22 de Noviembre de 2002)

ISLA, J. L.; GUTIÉRREZ, F. L. (2003). "Structural Modeling of Design Patterns: REP Diagrams". In Hanmer and Andrade (eds.), *Proceedings of the SugarLoafPLoP 2003* (Porto de Galinhas, Pernambuco, Brazil, August 12-15, 2003), pp. 301-309

ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M. (2004a). "Patrones de Organización. Integración en un Proceso de Desarrollo Centrado en el Grupo", *Actas del Congreso Internacional Interacción 2004*, Lorés y Navarro (eds.), Lleida, 2004, pp. 172-179

ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M.; GARRIDO, J. L. (2004b). "Descripción de Patrones de Organización y su Modelado con AMENITIES", *Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'04)*, Dieste y Moreno (eds.), Madrid, pp. 3-14

ISLA, J. L.; GUTIÉRREZ, F. L.; PADEREWSKI, P. (2005a). "Un Profile para el Modelado de Patrones de Software". En *Actas de las X Jornadas en Ingeniería del Software y Bases de Datos*, Thomson Paraninfo, pp. 265-270

ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M. (2005b). "Organization Modelling of the Collaborative Process: A Pattern-Based Approach". In *Proceedings of The 9th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2005)*, vol. 2, Coventry, UK, pp. 944-949

ISLA, J. L.; GUTIÉRREZ, F. L.; PADEREWSKI, P. (2006a). "Una Aproximación Basada en Patrones para el Modelado Conceptual de Sistemas Cooperativos". En Riquelme, J. y Botella, P. (eds.), *Actas de las XI Jornadas de Ingeniería del Software y Bases de Datos*, CIMNE, Barcelona, pp. 305-314

ISLA, J. L.; GUTIÉRREZ, F. L.; GEA, M. (2006b). "Supporting Social Organization Modelling in Cooperative Work Using Patterns". In Shen, W. et al. (eds.), *Computer Supported Cooperative Work in Design II*, LNCS 3865, Springer, pp. 112-121

ISLA, J. L.; GUTIÉRREZ, F. L.; GARRIDO, J. L.; HURTADO, M. V.; HORNOS, M. J. (2006c). "Integration of Organisational Patterns into a Group-

Centred Methodology". *HCI Related Papers of Interacción 2004*, Springer-Verlag, Dordrecht, Netherlands, pp. 137-146

ISLA, J. L.; GUTIÉRREZ, F. L.; PADEREWSKI, P. (2007). "Una Aproximación Basada en Patrones para el Modelado Conceptual de Sistemas Cooperativos". *IEEE Latin America Transactions*, Vol. 5, Nº 4, July 2007, pp. 204-210

JACOBSON, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Professional

JOHANSEN, R. (1989). "User Approaches to Computer-Supported Teams". In M. H. Olson (ed.), *Technological Support for Work Group Collaboration*, Hillsdale, NJ, Lawrence Erlbaum Associates, pp. 1-32

JORDAN, Brigitte (1996). "Ethnographic Workplace Studies and Computer Supported Cooperative Work". In Dan Shapiro, Michael Tauber and Roland Traunmüller (eds.): *The Design of Computer-Supported Cooperative Work and Groupware Systems*, Amsterdam, The Netherlands: North Holland, Elsevier Science, pp. 17-42

KAGAN, S. (1989). *Cooperative learning resources for teachers*, San Juan Capistrano, CA, Resources for Teachers

KENT, Stuart (1997) "Constraint Diagrams: Visualising Invariants in Object-Oriented Models". In *Proceedings of OOPSLA '97*, ACM Press

KIM, D.; FRANCE, R.; GHOSH, S.; SONG, E. (2003). "A UML-Based Metamodeling Language to Specify Design Patterns". In *Proceedings of Workshop in Software Model Engineering (WiSME)* (San Francisco, USA, October, 2003)

KOLP M.; GIORGINI P.; MYLOPOULOS J. (2002). "Information Systems Development Through Social Structures". In *Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering*, Italy

KOLP M.; GIORGINI P.; MYLOPOULOS J. (2003). "Organizational Patterns for Early Requirements Analysis". In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, Velden, Austria

KONRAD, Sascha; CHENG, Betty H.C.; CAMPBELL, Laura A. (2004). "Object Analysis Patterns for Embedded Systems", IEEE Transactions on Software Engineering, vol. 30, nº 12, december 2004, pp. 970-992

LANO, K.; GOLDSACK, S.; BICARREGUI, J. (1996). "Formalising Design Patterns". In *1st BCS-FACS Northern Formal Methods Workshop, Electronic Workshops in Computer Science,* Springer-Verlag

LAUDER, A.; KENT, S. (1998). "Precise Visual Specification of Design Patterns". In *Proceedings of ECOOP'98*, Springer-Verlag

LE GUENNEC, A. ; SUNYE, G.; JEZEQUEL, J. (2000). "Precise Modeling of Design Patterns". In *Proceedings of UML' 2000*, Springer Verlag, LNCS 1939

LISBOA, J.; IOCHPE, C.; BEARD, K. (1998). "Applying Analysis Patterns in the GIS Domain". In *Proceedings of the Spatial Information Research Centre's 10th Colloquium* (University of Otago, New Zealand, 16-19 November, 1998), pp. 181-188

LORÉS, J. (Ed.) (2001). *Introducción a la Interacción Persona-Ordenador*. AIPO, Lleida, `http://griho.udl.es/ipo/libroe.html`

LOZANO, M. (2001). *Entorno Metodológico Orientado a Objetos para la Especificación y Desarrollo de Interfaces de Usuario*. Tesis doctoral, Universidad Politécnica de Valencia.

LUKOSCH, Stephan; SCHÜMMER, Till (2004). "Patterns for Managing Shared Objects in Groupware Systems". In *Proceedings of the Ninth European Conference on Pattern Languages and Programs*, Irsee, Germany, pp. 333-378

LUKOSCH, Stephan; SCHÜMMER, Till (2006). "Groupware Development Support with Technology Patterns", *Int. Journal of Human-Computer Studies*, 64, pp. 599-610

LYARDET, Fernando; ROSSI, Gustavo; SCHWABE, Daniel. (1999). "Discovering and Using Design Patterns in the WWW", *Multimedia Tools and Applications* 8, pp. 293-308

LYNCH, K. J.; SNYDER, J. M.; VOGEL, D. R.; McHENRY, W. K. (1990). "The Arizona Analyst Information System: Supporting Collaborative Research on International Technological Trends". In *Proceedings of IFIP WG8.4 Conference on Multi-User Interfaces and Applications*, Crete, Greece

MAK, J.; CHOY, C.; LUN, D. (2004). "Precise Modeling of Design Patterns in UML". In *Proceedings of ICSE´04*

MALONE, T.; CROWSTON, K. (1990). "What is coordination theory and how can it help design cooperative work systems?". In *Proceedings of the Third Conference on Computer-Supported Cooperative Work* (Los Angeles, Calif., Oct. 8-10, 1990), ACM, New York, pp. 357-370

MALONE, T; CROWSTON, K. (1994). "The Interdisciplinary Study of Coordination". *ACM Computing Surveys*, Vol. 26, Nº 1, March, 1994, pp. 87-119

MANOLESCU, Dragos; VOELTER, Markus; NOBLE, James (eds.) (2006). *Pattern Languages of Program Design 5 (Software Patterns Series)*, Reading, MA, Addison-Wesley Professional

MAPELSDEN, D.; HOSKING, J.; GRUNDY, J. (2002). "Design Pattern Modelling and Instantiation using DPML", *40th International Conference on Technology of Object-Oriented Languages and Systems*, Sydney, Australia, *Conferences in Research and Practice in Information Technology*, v.10

MARTIN, David; RODDEN, Tom; ROUNCEFIELD, Mark; SOMMERVILLE, Ian; VILLER, Stephen (2001). "Finding Patterns in the Fieldwork". In *Proceedings of ECSCW 2001*, Bonn, Germany, Kluwer Academic Publishers, pp. 39-58

MARTIN, D; ROUNCEFIELD, M.; SOMMERVILLE, I. (2002). "Applying patterns of cooperative interaction to work (re)design". In *Proceedings of CHI 2002*, Minneapolis, ACM Press.

MARTIN, David; RODDEN, Tom; SOMMERVILLE, Ian; ROUNCEFIELD, Mark; HUGHES, John (2005). *PoInter Project (Patterns of Interaction: a Pattern Language for CSCW)*, (consultada el 24-8-05),

http://www.comp.lancs.ac.uk/computing/research/cseg/projects/pointer/pointer.html

MARTIN, Robert C.; RIEHLE, Dirk; BUSCHMANN, Frank (eds.) (1998). *Pattern Languages of Program Design 3*, Reading, MA, Addison-Wesley

MEIJERS, M. (1996). *Tool Support for Object-Oriented Design Patterns*, Master's Thesis, INF-SCR-96-28, Utrecht University

MESZAROS, G. (1996). "A Pattern Language for Improving the Capacity of Reactive Systems". In Vlissides, Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA, Addison-Wesley, pp. 575-591

MESZAROS, Gerard; DOBLE, Jim (1998). "A Pattern Language for Pattern Writing". In Martin, Riehle and Buschmann (eds.), *Pattern Languages of Program Design 3*, Reading, MA, Addison-Wesley, pp. 529-574

MEUNIER, R. (1995). "The Pipes and Filters Architecture". In Coplien and Schmidt (eds.), *Pattern Languages of Program Design*, Reading, Massachusetts, Addison-Wesley Publishing Company, pp. 427-440

MIKKONEN, Tommi (1998). "Formalising Design Patterns". In *Proceedings of the 1998 International Conference on Software Engineering – ICSE'98*, IEEE Computer Society Press, pp. 115-124

MINTZBERG, H. (1992). *Structure in fives: designing effective organizations*. Englewood Cliffs, N.J., Prentice-Hall

MOLINA, P. J. (2003). *Especificación de Interfaz de Usuario: De los Requisitos a la Generación Automática*, Tesis doctoral, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia

MOLINA, Pedro J.; MELIÁ, Santiago; PASTOR, Oscar (2002a). "JUST-UI: A User Interface Specification Model", *Computer-Aided Design of User Interfaces III*, Ch. Kolski & J. Vanderdonckt (eds.), In *Proc. of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002* (Valenciennes, 15-17 May 2002), Kluwer Academics Publisher, Dordrecht, pp. 63-74

MOLINA, Pedro J.; MELIÁ, Santiago; PASTOR, Oscar (2002b). "User Interface Conceptual Patterns", In *Lecture Notes of Computer Science*, Peter Forbrig, Quentin Limbourg, Bodo Urban and Jean Vanderdonckt (eds.), vol. 2545, Springer Verlag, Berlin, Alemania, Diciembre 2002

MONTERO, Francisco; GONZÁLEZ, Pascual; LOZANO, María Dolores (2002). "Patrones de Interacción: Taxonomía y otros Problemas", *Actas de III Congreso Internacional de Interacción Persona-Ordenador (Interacción'2002),* Leganés, 8-10 de mayo de 2002, pp. 226-229

MONTERO, F. (2005). *Integración de Calidad y Experiencia en el Desarrollo de Interfaces de Usuario Dirigido por Modelos*, Tesis doctoral, Universidad de Castilla-La Mancha, 2005

MONTERO, F.; LOZANO, M.; GONZÁLEZ, P. (2005a) "IdealXML: an Experience-Based Environment for User Interface Design and pattern manipulation'', *Technical report DIAB-05-01-4*, Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha. 2005.

MONTERO, F.; LÓPEZ-JAQUERO, V.; RAMÍREZ, Y.; LOZANO, M.; GONZÁLEZ, P. (2005b). "Patrones de Interacción para Usuarios y para Diseñadores", *I Congreso Español de Informática, VI Congreso de Interacción Persona-Ordenador (Interacción'2005),* Granada, del 13 al 16 de septiembre de 2005, pp. 163-170

MORABITO, J.; SACK, I.; BHATE, A. (1999). *Organization Modeling: Innovative Architectures for the 21st Century*, Upper Saddle River, N.J., Prentice Hall PTR

MUMFORD, E. (2000). "A Socio-Technical Approach to Systems Design", *Requirements Engineering*, 5(2), pp. 125-133

NANARD, Jocelyne; NANARD, Marc. (1999). "Toward an Hypermedia Design Patterns Space". In *2nd Workshop in Hypermedia Development: Design Patterns in Hypermedia*

NARDI, Bonnie A. (ed.) (1995). *Context and Consciousness: Activity Theory and Human Computer Interaction.* MIT Press, Cambridge MA

NICOLA, J.; MAYFIELD, M.; ABNEY, M.; ABNEY, M. (2001). *Streamlined Object Modeling: Patterns, Rules, and Implementation*. Prentice Hall PTR, 1st edition, 2001

OMG (2003). *OMG Unified Modeling Language v1.5*, Object Management Group, http://www.omg.org/docs/formal/03-03-01.pdf

OMG (2004). *UML Profile for Patterns, v1.0.*, http://www.omg.org/docs/formal/04-02-04.pdf

OMG (2006). *Object Constraint Language, v.2.*, http://www.omg.org/docs/formal/06-05-01.pdf

OMG (2007a). *UML v2.1.1. Infrastructure Specification*, Object Management Group, http://www.omg.org/docs/formal/07-02-06.pdf

OMG (2007b). *UML v2.1.1. Superstructure Specification*, Object Management Group, http://www.omg.org/docs/formal/07-02-05.pdf

O´REILLY, Tim (2005). "What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software", http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

ORTEGA, F. D.; ISLA, J. L.; PAVÓN, F. (1999). "Las Nuevas Tecnologías en la Comunicación Humana: El IRC en la Educación". *Actas del Congreso de Nuevas Tecnologías de la Información y la Comunicación para la Educación (EDUTEC99*), 14-17 septiembre, Sevilla

ORTEGA, F. D.; ISLA, J. L.; PAVÓN, F. (2000). "El IRC como Herramienta para la Formación Flexible y a Distancia". *Revista PIXEL-BIT*, Secretariado de Recursos Audiovisuales y Nuevas Tecnologías de la Universidad de Sevilla, Vol. 14, enero 2000, pp. 31-41

PADILLA, N. (2002). *Especificación de Sistemas Cooperativos*. Tesis doctoral. Universidad de Almería.

PANTOQUILHO, M., RAMINHOS, R.; ARAÚJO, J. (2003). "Analysis Patterns Specifications: Filling the Gaps". In *proc. of VikingPlop 2003*, Bergen, Norway, 18-21 September 2003

PAOLINI, Paolo; GARZOTO, Franca. (1999). "Design Patterns for WWW hypermedia: problems and proposals". In *Hypertext'99 Workshop on Hypermedia Development: Design Patterns in Hypermedia*.

PATERNÒ, Fabio (1999). *Model-based Design and Evaluation of Interactive Applications*. Springer Verlag

PLOWMAN, L.; ROGERS, Y.; RAMAGE M. (1995). "What are workplace studies for?". In *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work (ECSCW 95)*, pp. 309-324

PREE, Wolfgang (1995). *Design Patterns for Object-Oriented Software Development*, Reading, MA, Addison-Wesley

PURAO, S.; STOREY, V. C.; HAN, T. (2003). "Improving Analysis Pattern Reuse in Conceptual Design: Augmenting Automated Processes with Supervised Learning", *Information Systems Research*, Vol. 14, No. 3, September 2003, pp. 269–290

REAL ACADEMIA ESPAÑOLA (2003). "Diccionario de la Lengua Española". 22ª edición, Madrid, Espasa Calpe

RIEHLE, D. (1996). "Describing and Composing Patterns Using Role Diagrams". *WOON '96 (1st International Conference on Object-Orientation in Russia), Conference Proceedings*, St. Petersburg Electrotechnical University, 1996. Reprinted in K.-U. Mätzel and Hans-Peter Frei (eds.), *Proceedings of The Ubilab Conference '96*, Zürich, Germany, Universitätsverlag Konstanz, pp. 137-152

RIEHLE, Dirk; ZÜLLIGHOVEN, Heinz (1996). "Understanding and Using Patterns in Software Development", *Theory and Practice of Object Systems* 2(1): 3-13

ROSSI, G.; GARRIDO, A.; CARVALHO S. (1996a). "Design Patterns for Object-Oriented Hypermedia Applications". In Vlissides, Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA, Addison-Wesley, pp. 177-192

ROSSI, Gustavo.; SCHWABE, Daniel; GARRIDO, Alejandra (1996b). "Towards a Pattern Language for Hypermedia Applications". In *Proceedings of*

the 3rd Annual Conference on Pattern Languages of Programs, Monticello, Illinois, September 1996

ROSSI, Gustavo; SCHWABE, Daniel; GARRIDO, Alejandra (1997). "Design Reuse in Hypermedia Applications Development". In *Proceedings of the ACM International Conference on Hypertext (Hypertext 97)*, ACM Press, 1997, pp. 57-66

ROSSI, Gustavo; SCHWABE, Daniel; LYARDET, Fernando (1999). "Patterns for Designing Navigable Information Spaces". In N. Harrison, B. Foote and H. Rohnert (eds.), *Pattern Languages of Program Design 4*, Reading, MA, Addison-Wesley, pp. 445-460

RUSSELL, N.; TER HOFSTEDE, A.H.M.; EDMOND, D.; VAN DER AALST, W.M.P. (2005). "Workflow Data Patterns: Identification, Representation and Tool Support". In L. Delcambre et al. (eds.), *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, volume 3716 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2005, pp. 353-368

RUSSELL, N.; VAN DER AALST, W. M. P; TER HOFSTEDE, A. H. M.; WOHED, P. (2006). "On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling". In M. Stumptner, S. Hartmann and Y. Kiyoki (eds.), *Proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006)*, volume 53 of CRPIT, Hobart, Australia, 2006, pp. 95-104

SANADA, Y.; ADAMS, R. (2002). "Representing Design Patterns and Frameworks in UML – Towards a Comprehensive Approach". In *Journal of Object Technology*, vol. 1, nº 2, July-August 2002, pp. 143-154

SANCHEZ, M.; JIMÉNEZ, B.; GUTIÉRREZ, F. L.; PADEREWSKI, P.; ISLA, J. L. (2006). "Modelo de Control de Acceso en un Sistema Colaborativo". En *Actas del VII Congreso Internacional de Interacción Persona-Ordenador*, Puertollano (Ciudad Real), pp. 227-237

SANE, A.; CAMPBELL, R. (1996). "Resource Exchanger: A Behavioral Pattern for Low-Overhead Concurrent Resource Management". In Vlissides,

Coplien and Kerth (eds.), *Pattern Languages of Program Design 2*, Reading, MA, Addison-Wesley, pp. 461-473

SCHLICHTER, J.; KOCH, M.; BURGER, M. (1998). "Workspace Awareness for Distributed Teams". In W. Conen and G. Neumann (eds.), *Coordination Technology for Collaborative Applications,* Springer Verlag, Heidelberg

SCHMIDT, Douglas C. (1995a). "Experience Using Design Patterns to Develop Reuseable Object-Oriented Communication Software", *Communications of the ACM* (Special issue on Object-Oriented Experiences), October, 1995

SCHMIDT, Douglas C. (1995b). "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching". In Coplien and Schmidt (eds.), *Pattern Languages of Program Design*, Reading, Massachusetts, Addison-Wesley Publishing Company, pp. 529-545

SCHMIDT, Douglas C.; STEPHENSON, Paul (1995). "Experience Using Design Patterns to Evolve Communication Software Across Diverse OS Platforms". In *Proceedings of the 9th European Conference on Object-Oriented Programming* (Aarhus, Denmark, August 7-11, 1995)

SCHMIDT, D. C.; STAL, M.; ROHNERT, H.; BUSCHMANN, F. (2000). *Patterns for Concurrent and Networked Objects*, Vol. 2 of Pattern-Oriented Software Architecture, Wiley, New York.

SCHULER, D.; NAMIOKA, A. (eds.) (1993). *Participatory Design: Principles and Practices*. Lawrence Erlbaum, Hillsdale, NJ

SCHÜMMER, Jan; SCHUCKMANN, Christian; BIBBÓ, Luis Mariano; ZAPICO, Juan José (1999). "Collaborative Hypermedia Design Patterns in OOHDM", *2nd Workshop in Hypermedia Development: Design Patterns in Hypermedia (HT'99)*

SCHÜMMER, Till (2002). "Constructing a Groupware Pattern Language". *CSCW Workshop on Socio-Technical Pattern Languages,* (New Orleans, November 16-20, 2002)

SCHÜMMER, T. (2004a). "Patterns for Building Communities in Collaborative Systems". In *Proceedings of the Ninth European Conference on Pattern*

Languages of Programs (EuroPLoP'04), UVK, Konstanz, Germany, Irsee, Germany, pp. 379-440

SCHÜMMER, T. (2004b). "The Public Privacy – Patterns for Filtering Personal Information in Collaborative Systems". *CHI 2004 Workshop on Human-Computer-Human-Interaction Patterns*, FernUniversität in Hagen

SCHÜMMER, T. (2004c). "GAMA – A Pattern Language for Computer Supported Dynamic Collaboration". In Henney, K., Schütz, D. (eds.), Proceedings of the Eighth European Conference on Pattern Languages of Programs (EuroPLoP'03). UVK, Konstanz, Germany

SCHÜMMER, T.; SLAGTER, R. (2004). "The Oregon Software Development Process", In J. Eckstein and Baumeister (eds.): XP2004, LNCS 3092, Springer-Verlag, pp. 148-156

SCHWABE, Daniel; ROSSI, Gustavo; BARBOSA, S. (1996). "Systematic Hypermedia Design with OOHDM". In *Proceedings of the ACM International Conference on Hypertext'96*, ACM Press, Washington, March 1996

SHARAN, Shlomo (ed.) (1994). *Handbook of cooperative learning methods*, Greenwood Press

SHAW, M. (1995). "Patterns for Software Architectures". In Coplien and Schmidt (eds.), *Pattern Languages of Program Design*, Reading, Massachusetts, Addison-Wesley Publishing Company, pp. 391-406

SLAVIN, R. E. (1986). *Using student team learning* (3rd ed.), Baltimore, MD, The Johns Hopkins University, Center for Research on Elementary and Middle Schools

SOMMERVILLE, Ian (2002). *Ingeniería de Software*. Pearson Educación, México, 2002

TARPIN-BERNARD, F.; DAVID, B. T.; PRIMET, P. (1998). "Frameworks and Patterns for Synchronous Groupware: AMF-C Approach", *IFIP Working Conference on Engineering for HCI: EHCI'98*, Greece, Kluwer Academia Publishers, pp. 225-241

THOMAS, John; DANIS, Catalina; GREENE, Sharon (2002). "Socio-Technical Pattern Language Proposal", *CSCW Workshop on Socio-Technical Pattern Languages*, (New Orleans, November 16-20, 2002)

TIDWELL, J. (1998). "Interaction Design Patterns". In *Proceedings of PLoP'98*

TIESSEN, E. L.; WARD, D. R. (1997). "Collaboration by Design: Context, Structure and Medium". *Journal of Interactive Learning Research*, 8, pp. 175-197

VAN DER AALST, W.M.P.; TER HOFSTEDE, A.H.M.; KIEPUSZEWSKI, B.; BARROS, A.P. (2003). "Workflow Patterns". *Distributed and Parallel Databases*, 14(1):5–51, 2003

VAN WELIE, M.; VAN DER VEER, G.C.; ELIËNS, A. (1998), "An Ontology for Task World Models", In *Design, Specification and Verification of Interactive System'98*, Springer Computer Science, pp. 57-70

VLISSIDES, John M.; COPLIEN, James O.; KERTH, Norman L. (eds.) (1996). *Pattern Languages of Program Design 2*, Reading, MA, Addison-Wesley

WEIR, Charles (1998). "Patterns for Designing in Teams". In Martin, Riehle and Buschmann (eds.), *Pattern Languages of Program Design 3*, Reading, MA, Addison-Wesley, pp. 487-501

WELLS, David; KURIEN, Ansu (1996). *Groupware & Collaboration Support*, (consultada el 26-1-07), `http://www.objs.com/survey/groupwar.htm`

WHITE, Stephen A. (2004). "Process Modeling Notations and Workflow Patterns". In L. Fischer (ed.), *Workflow Handbook 2004*, Future Strategies Inc., Lighthouse Point, FL, USA, pp. 265–294

WIKIPEDIA (2001). *Wikipedia, The Free Encyclopedia*, `http://en.wikipedia.org/wiki/Main_Page`. También disponible en español, `http://es.wikipedia.org/wiki/Portada`

WILSON, Paul (1991). *Computer Supported Cooperative Work: An Introduction*. Oxford, England Norwell, MA, Intellect; Vendido y distribuido en USA por Kluwer Academic Publishers

WINN, Tiffany; CALDER, Paul (2002). "Is This a Pattern?", *IEEE Software*, January/February, 2002

WOHED, Petia (2000a). "Conceptual Patterns for Reuse in Information Systems Analysis". In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'00)*, Stockholm, pp. 157-175

WOHED, Petia (2000b). "Conceptual Patterns - A Consolidation of Coad's and Wohed's Approaches". In *Proceedings of the 5th International Conference on Applications of Natural Language to Information Systems-Revised Papers*, LNCS 1959, Springer Verlag, pp. 340-351

YODER, Joseph; BARCALOW, Jeffrey (1999). "Architectural Patterns for Enabling Application Security". In N. Harrison, B. Foote and H. Rohnert (eds.), *Pattern Languages of Program Design 4*, Reading, MA, Addison-Wesley, pp. 301-336

ZHEN, L.; SHAO, G. (2002). "Analysis Patterns for Oil Refineries". In *Proc. of PloP2002*, Allerton Park, Monticello, Illinois, USA