XI Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)

ASIGNACIÓN SISTEMÁTICA DE RESPONSABILIDADES EN UNA ARQUITECTURA DE TRES CAPAS[†]

Xavier Franch[1]*, Jordi Pradel[1,2], y Jose Raya[1,2]

1: Grup d'Enginyeria del Software per als Sistemas d'Informació (GESSI)
Universitat Politècnica de Catalunya (UPC)
UPC – Campus Nord, edifici Omega, c/Jordi Girona 1-3 08034 Barcelona
e-mail: franch@lsi.upc.edu, web: http://www.lsi.upc.es/~webgessi/index.html

2: Agilogy SL
Avda. Meridiana, 519 1°, 08016 Barcelona
e-mail: {jordi.pradel,jose.raya}@agilogy.com, web: http://www.agilogy.com

Palabras clave: Arquitectura en tres capas, asignación de responsabilidades, UML.

Resumen. *La arquitectura en tres capas es probablemente el patrón arquitectónico más utilizado en el diseño de sistemas de información. Su aplicación se basa en una correcta distribución de las responsabilidades que se derivan de la especificación del sistema entre las tres capas que conforman la arquitectura (presentación, dominio y gestión de datos). En este artículo presentamos una metodología que asiste en la asignación de responsabilidades a las tres capas de un sistema de información partiendo de la especificación en UML de dicho sistema. La metodología se basa en la definición y tipificación precisa del concepto de responsabilidad, así como de sus posibles tratamientos, y en la aplicación reiterada de patrones de asignación de responsabilidades a capas para determinar a qué capa o capas asignamos cada responsabilidad y qué elementos de diseño serán necesarios para su tratamiento.*

1. INTRODUCCIÓN

Las *arquitecturas software* estructuran las aplicaciones en subsistemas o componentes interconectados. Podemos encontrar en la literatura un gran número de estilos arquitectónicos [1], que frecuentemente toman la forma de *patrones arquitectónicos* [2]. Uno de los patrones arquitectónicos más utilizados es el *patrón arquitectónico en capas*. Su característica principal radica en la definición de diversos niveles de abstracción que permiten estructurar adecuadamente los servicios del sistema, favoreciendo características tales como la modificabilidad, la portabilidad y la reusabilidad.

[†] Este trabajo cuenta con el apoyo del proyecto TIN2004-07461-C02-01.

En el contexto del desarrollo de sistemas de información, el patrón en capas está especialmente extendido. Si bien en sus inicios se definieron tan sólo dos capas, actualmente existe un amplio consenso en la definición de tres capas: la *capa de presentación*, que gestiona los aspectos de comunicación del sistema con el usuario; la *capa de dominio*, que implementa la lógica de la aplicación; y la *capa de gestión de datos*, cuya misión consiste en interaccionar con el soporte de persistencia definido para el sistema (generalmente, un sistema gestor de bases de datos relacional, SGBDR).

Aplicar correctamente el patrón arquitectónico en capas no es una tarea sencilla. Buschmann *et al.* [2, pp.38-43] definen 10 pasos para implementar el patrón en un sistema determinado. En el contexto que nos ocupa, los tres primeros pasos (definir el criterio de abstracción; determinar el número de niveles de abstracción; nombrar los niveles y asignarles tareas) se pueden considerar bien resueltos, resultando en las capas presentadas arriba. El paso 4 exige especificar los servicios de cada capa, y ahí nos encontramos con las primeras dificultades, pues generalmente los servicios se especifican de manera *ad hoc*.

Por otro lado, en el marco de la tecnología de procesos basada en la existencia de un modelo de especificación previo al diseño (generalmente en el contexto de procesos iterativos e incrementales), como es el caso del Proceso Unificado (UP) [3], los servicios deben necesariamente derivarse de la especificación del sistema, que supondremos en adelante que está expresada en UML [4]. No obstante, incluso en trabajos tan detallados como puede ser el clásico libro de Larman [5], la transición de las responsabilidades establecidas en la especificación, a los servicios de la arquitectura en capas, se deja al arbitrio del diseñador, que se supone que va a aplicar su experiencia o el conocimiento de ciertos patrones como los propuestos por Fowler [6]. Puede decirse por ello que existe un cierto desacoplamiento entre las propuestas existentes para el análisis orientado a objetos y la implementación de una arquitectura en capas.

Por último, es innegable que la asignación de servicios a una arquitectura en capas es especialmente sensible a las innovaciones tecnológicas que continuamente aparecen en el mercado. Cada nuevo producto (o versión) ofrece ciertas facilidades para distinguirse de sus competidores que deben ser aprovechadas en una asignación óptima de servicios. Es más, periódicamente aparecen ya no productos sino nuevos paradigmas con un fuerte impacto en esta tarea; un ejemplo reciente sería la orientación a aspectos [7].

Fig. 1. Contextualización de la metodología ASIRCA.

En este artículo se propone un enfoque sistemático a la asignación de servicios a las capas de una arquitectura en capas (v. fig. 1). La metodología propuesta, ASIRCA, está basada en el concepto de responsabilidad inherente a toda especificación orientada a objetos. Proponemos la utilización de patrones de asignación de responsabilidades a capas, y definimos un modelo de referencia para la definición de tales patrones.

2. CONSIDERACIONES PREVIAS

Una primera pregunta que surge al considerar la asignación de responsabilidades a capas es si las responsabilidades deben asignarse en un orden determinado o no. Para ello, debemos reconocer en primer lugar que la situación de partida puede ser diferente en cada caso concreto. Por ejemplo:

- En el contexto de desarrollo de aplicaciones basado en UP, existirá una especificación UML de partida del dominio de la aplicación. En este caso, lo más natural será asignar por omisión todas las responsabilidades a la capa de dominio e identificar cuáles de ellas pueden trasladarse a alguna de las otras dos capas (v. fig. 2, a).
- En cambio, si aplicamos alguna metodología ágil de desarrollo [8], no existirá una especificación detallada y formal; en su lugar podríamos tener un versión avanzada de la interacción usuario-sistema. En tal caso, podemos considerar una asignación descendente desde la capa de presentación (v. fig. 2, b).
- Por otro lado, el diseño de una aplicación puede partir de una base de datos ya existente que no puede modificarse. La base de datos determina pues las responsabilidades asignadas a la capa de gestión de datos, y a partir de ésta se pueden asignar de forma ascendente el resto de responsabilidades (v. fig. 2, c).
- Finalmente, un equipo de desarrollo de amplia experiencia dispondrá de una base (explícita o no) de conocimiento que le permitirá asignar rápidamente las responsabilidades. En este caso, se puede considerar que se diseñan las tres capas simultáneamente (v. fig. 2, d).

Un objetivo de nuestra propuesta es que la metodología propuesta se adapte a las situaciones descritas anteriormente, para no perder aplicabilidad.

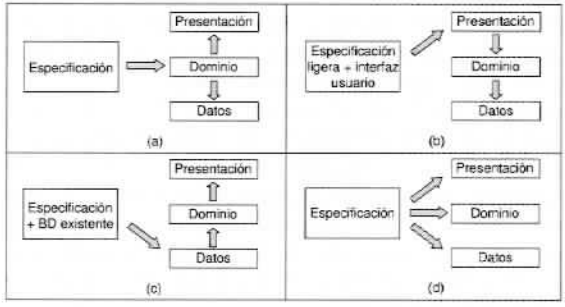


Fig. 2. Cuatro ejemplos de transición de la especificación del sistema a la especificación de las capas.

3. TIPOS DE RESPONSABILIDADES Y SU TRATAMIENTO

En las secciones anteriores, hemos asimilado los conceptos de *responsabilidad*, propio de la comunidad de diseño por patrones, y *servicio*, según la perspectiva de la arquitectura en capas. Por ello, para realizar una correcta asignación de servicios a capas, debemos primero determinar cuáles son los tipos de responsabilidad que pueden aparecer en una especificación orientada a objetos. En consonancia con las metodologías basadas en UP, tomamos como punto de partida una especificación orientada a objetos constituida por: modelo conceptual de datos (diagrama de clases con anotaciones OCL o textuales), modelo de casos de uso (diagramas de casos de uso y descripción individual mediante diagramas de secuencia), modelo del comportamiento (contratos pre/post para las operaciones que aparecen en los diagramas de secuencia) y modelo de estados (diagramas de estados para las jerarquías dinámicas de objetos). Para simplificar la discusión, y sin pérdida de generalidad, vamos a suponer que el modelo de casos de uso (y en consecuencia el del comportamiento) es concordante con el diseño de interfaz de la aplicación (es decir, los casos de uso son concretos en vez de esenciales).

Para facilitar la definición posterior de la metodología, clasificamos las responsabilidades en dos clases:

- *Responsabilidades permanentes*. El sistema ha de asegurar su cumplimiento en todo momento. Asimilables al concepto de invariante del sistema. Aparecen en el modelo conceptual de datos ya sea: 1) de forma gráfica, e.g.: restricciones propias de las clases (multiplicidad, etc.) o de los roles (multiplicidad, ordenación, etc.), restricciones relativas a jerarquías (completitud, solapamiento, etc.), dependencias entre asociaciones, etc.; 2) mediante anotaciones, que serán restricciones de integridad de todo tipo (de clave, de intervalo, etc.) y reglas de cálculo de información derivada (atributos y asociaciones), principalmente.
- *Responsabilidades eventuales*. El sistema ha de asegurar su cumplimiento tan sólo cuando se produce un evento que las involucra. Aparecen en los contratos de las operaciones (verificación de las precondiciones y ejecución de la postcondiciones) y también en los diagramas de estado (verificación del estado de partida y alcance del estado destino en el tratamiento de un evento-transición).

Como veremos en la sección siguiente, las responsabilidades permanentes pueden transformarse en eventuales durante la asignación a capas, si se considera adecuado.

Asimismo, distinguimos entre dos clases de tratamientos:

- *Tratamientos declarativos*. El tratamiento de la responsabilidad se declara en alguna parte del sistema, en cualquiera de las capas. Si bien la capa de gestión de datos es la que ofrece más tratamientos declarativos (e.g., al definir el esquema de la base de datos –declaración de claves, *checks*, etc.–, disparadores, vistas, etc.), éstos también son posibles a nivel de dominio (e.g. mediante Hibernate Validator [9]).
- *Tratamientos procedimentales*. El tratamiento de la responsabilidad se describe mediante un comportamiento procedimental (un diagrama de secuencia, un fragmento de código, una sentencia sql, etc.).

4. ASIGNACIÓN SISTEMÁTICA DE RESPONSABILIDADES A CAPAS

En esta sección presentamos ASIRCA, el método de **A**signación **SI**stemática de **R**esponsabilidades a **CA**pas que proponemos.

4.1. Modelo de referencia

En la fig. 3 se presenta el modelo de referencia para ASIRCA. Cada responsabilidad se clasifica en un determinado tipo, que formará parte de una taxonomía de tipos. Por ejemplo, una determinada restricción puede pertenecer al tipo "Restricción de Clave" que es subtipo de "Restricción de Integridad". A cada tipo de responsabilidad se puede aplicar uno o más tipos de tratamientos (que, a su vez, formarán parte de una taxonomía). La restricción de integridad principal de este modelo asegura que el tratamiento aplicado a una responsabilidad es de un tipo aplicable (directa o indirectamente) al tipo de la responsabilidad.

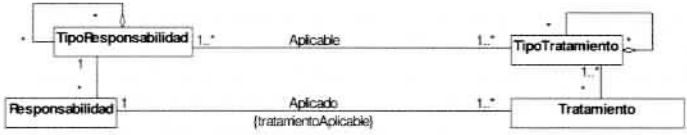


Fig. 3. Modelo de referencia para la asignación de responsabilidades a capas.

4.2. Etapas en la asignación de responsabilidades

Sean RP_0 y RE_0 los conjuntos de responsabilidades permanentes y eventuales de un sistema de información (v. fig. 4). La metodología ASIRCA consta de tres fases.

Fase 1: Tras el análisis de las responsabilidades de partida, y basándonos en la información contenida en el catálogo de tratamientos, se decide aplicar un conjunto de tratamientos declarativos a algunas responsabilidades de RP_0 y RE_0. Como resultado, se obtiene un conjunto de servicios declarativos asignados a capas y los conjuntos RP_1 y RE_1 de responsabilidades aún no tratadas.

Fase 2: Las responsabilidades permanentes no tratadas (RP_1) deberán ser tratadas de manera procedimental. Para ello, son transformadas en responsabilidades eventuales y añadidas a RE_1 para obtener el nuevo conjunto de responsabilidades eventuales RE_2. Esto nos llevará a modificar los contratos del sistema para incluir en cada operación aquellas pre y postcondiciones que aseguren el cumplimiento de cada responsabilidad permanente.

Fase 3: Guiados de nuevo por el catálogo, se aplican los tratamientos procedimentales adecuados a las responsabilidades de RE_2 para obtener los servicios procedimentales de cada capa.

El resultado de este proceso es la definición, para cada capa, de un conjunto de servicios declarativos y otro de servicios procedimentales. Es posible, aunque queda fuera del alcance de este documento, formalizar estos servicios generando las interfaces de cada capa y los contratos de sus operaciones.

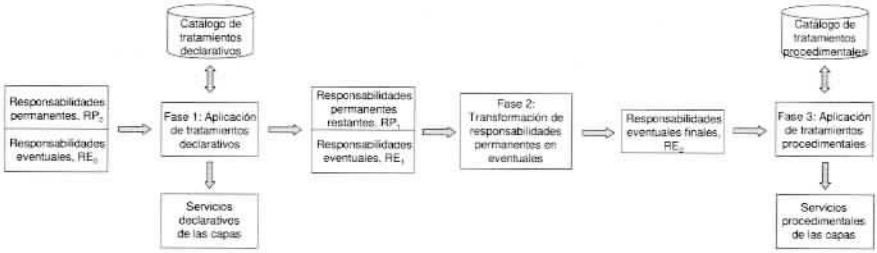


Fig. 4. Etapas en la asignación de responsabilidades a capas.

5. PATRONES DE ASIGNACIÓN DE RESPONSABILIDADES A CAPAS

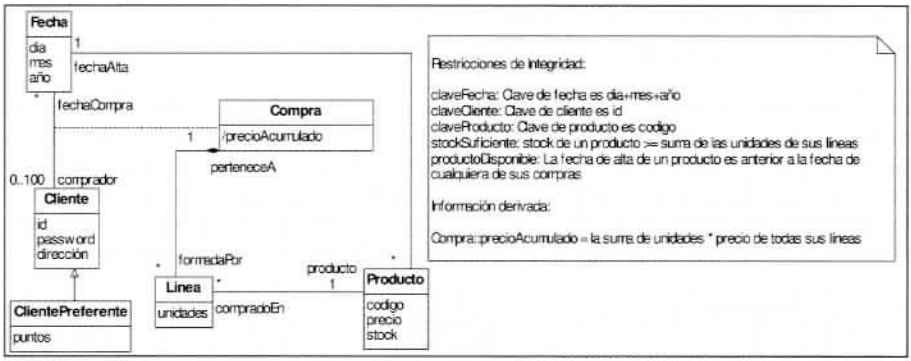
La aplicación sistemática de tratamientos a responsabilidades nos sugiere el uso de patrones de asignación de responsabilidades a capas. Cada uno de dichos patrones nos indicará, para un tipo de tratamiento, a qué tipos de responsabilidades es aplicable, cuáles son las consecuencias de su aplicación y nos dará indicaciones de diseño para elaborar el tratamiento concreto para dar soporte a una responsabilidad en una o más capas. El estudio y documentación de un catálogo completo de patrones queda fuera del alcance de este artículo por motivos de espacio. A modo de ejemplo, en la fig. 5 presentamos el detalle de uno de dichos patrones.

Patrón: *Materialización de atributo mediante disparador en SGBDR*
Tipo de responsabilidad: Atributo Derivado (**subtipo de** Información Derivada)
Tipo de tratamiento: Disparador en SGBDR (**subtipo de** Materialización de Información Derivada)
Contexto: El coste del cálculo del atributo derivado es considerable y/o el atributo tiene una alta frecuencia de consulta y una baja frecuencia de actualización.
Solución: Asignar la responsabilidad exclusivamente a la capa de datos mediante el uso de un disparador en el SGBDR.
Diseño del tratamiento: Materializar el atributo derivado en el esquema de base de datos (creando una columna para dicho atributo en la tabla correspondiente) y crear los disparadores necesarios para que, cuando se modifique la información original, se actualice dicha columna.
Consecuencias:
- La regla de activación del disparador y la de cálculo de la información derivada se expresan al nivel de abstracción del modelo relacional, lo cual puede ser un inconveniente en determinados casos.
- La capa de dominio queda libre de responsabilidades en cuanto al atributo derivado, pero solo lo podrá consultar para aquellos datos que sean persistentes.
- Si la base de datos es compartida, la solución es válida para (y afecta a) cualquier cliente de la base de datos, no sólo el sistema en estudio.
- Puesto que la información se recalcula en el SGBD, no hay coste añadido de transporte de datos entre capas para su cálculo.
- Es necesario evitar las actualizaciones directas (no causadas por el disparador) de la información materializada para evitar inconsistencias.

Fig. 5. Ejemplo de patrón de asignación de responsabilidades a capas.

6. EJEMPLO: CESTO DE LA COMPRA EN UN E-SUPERMERCADO

En esta sección mostramos un ejemplo de aplicación de la metodología ASIRCA sobre un sistema sencillo que pretende implementar el cesto de la compra en un e-supermercado. Por motivos de brevedad, sólo se mostrará el resultado final de cada etapa. La fig. 6 muestra la especificación de partida.



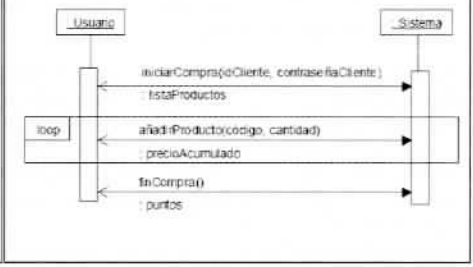
Caso de uso concreto: Comprar productos

Flujo de eventos principal:

1. El cliente inicia una compra en el sistema indicando su identificador y contraseña.
2. El sistema verifica los datos introducidos y muestra una lista de productos que el cliente puede comprar.
3. El cliente selecciona un producto e introduce el número de unidades a comprar
4. El sistema muestra el precio acumulado de la compra hasta el momento

Se repiten los pasos 3 y 4 hasta que el cliente no quiere comprar más productos.

5. El sistema registra la compra del cliente
6. El sistema muestra los puntos acumulados por el cliente si es un cliente preferente o 0 en caso contrario.



context iniciarCompra(idCliente: String, contraseñaCliente: String): listaProductos
pre: *clienteExiste*: Existe un cliente C identificado por idCliente
pre: *contraseñaCorrecta*: C.password = contraseñaCliente
post: *compraCreada*: compraActual = nueva compra entre C y la fecha actual
post: *resultado*: result = lista de código y precio de todos los productos
context añadirProducto(codigo: String, cantidad: Int): Int
pre: *productoExiste*: Existe un producto P identificado por código
pre: *unidadesPositivo*: cantidad > 0
post: *lineaCreada*: Se ha creado una línea de compra asociada a compraActual para P con unidades = cantidad
post: *resultado*: result = compraActual.precioAcumulado
context finCompra(): Int
post: *puntosCliente*: si el cliente era preferente se ha sumado un punto a sus puntos y el sistema ha registrado la nueva puntuación
post: *resultado*: result = C.puntos o 0 si no es cliente preferente
post: *compraAlmacenada*: Se ha registrado compraActual y sus líneas

Fig. 6. Especificación del ejemplo del e-supermercado: modelo conceptual; caso de uso concreto; diagrama de secuencia correspondiente al caso de uso; contratos de las operaciones.

Aplicando la metodología, partiríamos de los siguientes conjuntos de responsabilidades:

RP_0 = *uniqFechaAlta* (cardinalidad uno del rol Producto::fechaAlta), *uniqProducto*, *uniqPerteneceA*, *maxComprador* (cardinalidad máxima del rol Fecha::comprador), *compraAsociativa* (no puede haber dos instancias de Compra entre el mismo cliente y la misma fecha) y todas las restricciones textuales y reglas de derivación.

RE_0 = Conjunto de las pre y postcondiciones de los contratos de las operaciones.

En la tabla 1 se muestra el resultado de la fase 1 de ASIRCA. Se seleccionan tres patrones que aplican tratamientos declarativos sobre las responsabilidades de clave establecidas en las restricciones textuales y algunas de cardinalidad. Tras la primera fase, RP_1 estará formado por *stockSuficiente*, *productoDisponible*, *Compra::precioAcumulado* y *maxComprador* mientras que RE_1 será igual a RE_0. En la tabla 2 se muestra el resultado de la fase 2, en la que se obtienen las responsabilidades eventuales RE_2 que dan lugar a los contratos mostrados en la fig. 7 (se muestran en cursiva las pre y postcondiciones procedentes de las responsabilidades permanentes, y en gris las que ya se conocían).

Responsabilidad	Tipo	Patrón	Capa
claveFecha, claveCliente, claveProducto	Clave (**subtipo de** RI)	Clave Primaria en BD	Datos
compraAsociativa	Clave (**subtipo de** RI)	Clave Primaria en BD	Datos
uniqFechaAlta, uniqProducto, uniqPerteneceA	Multiplicidad uno (**subtipo de** RI)	Foreign Key Not Null en BD	Datos

Tabla 1. Resultado de la aplicación de tratamientos declarativos (RI: Restricción de Integridad)

Responsabilidad	Tipo	Operaciones afectadas
stockSuficiente	RI Multiclase (**subtipo de** RI)	*añadirProducto, finCompra*[*]
productoDisponible	RI Multiclase (**subtipo de** RI)	*añadirProducto*
Compra::precioAcumulado	Atributo Derivado (**subtipo de** Información Derivada)	*añadirProducto*
maxComprador	Multiplicidad máxima (**subtipo de** RI)	*iniciarCompra y finCompra*[*]

Tabla 2. Identificación de las operaciones afectadas por las responsabilidades de RP_1

context iniciarCompra(idCliente: String, contraseñaCliente: String): listaProductos
pre: *clienteExiste, contraseñaCorrecta*
pre: *maxComprador: La fecha actual tiene asociados menos de 100 usuarios*
post: *compraCreada, resultado*

context añadirProducto(codigo: String, cantidad: Int): Int
pre: *productoExiste, unidadesPositivo*
pre: *stockSuficiente: P.stock >= unidades + suma de las unidades de lineas actuales de P*
pre: *productoDisponible: P.fechaAlta anterior a compraActual.fechaCompra*
pre: *precioAcumulado: se ha incrementado compraActual.precioAcumulado en cantidad * P.precio*
post: *lineaCreada, resultado*

context finCompra(): Int
pre: *maxComprador, stockSuficiente* -- v. operaciones *iniciarCompra* y *productoDisponible*
post: *puntosCliente, resultado, compraAlmacenada*

Fig. 7. Contratos obtenidos tras la aplicación de la fase 2 de ASIRCA sobre el ejemplo.

[*] Estas responsabilidades afectan a *finCompra* si es posible el acceso concurrente al sistema.

Finalmente, en la fase 3, asignaremos cada una de las responsabilidades de RE_2 a una o más capas mediante la aplicación sistemática de patrones del catálogo (para lo cual nos basamos en las condiciones de aplicación descritas por el patrón; v. tabla 3): EOBD (Comprobación de que existe un objeto mediante acceso a la base de datos), RD (Comprobación o cálculo efectuado en la capa de dominio), ECD (Declaración de atributos en la capa de dominio para mantener el estado del caso de uso), CE (Consulta escalar en SQL), EL (Presentación en menú de diversas opciones evitando así ciertos tipos de errores de inexistencia), VV (Validación del rango de un valor en la capa de presentación), AP (Actualización de la base de datos).

Patrón	Capa	Responsabilidades sobre las que se aplica
EOBD	Dominio+Datos	iniciarCompra::clienteExiste
RD	Dominio+Datos	iniciarCompra::maxComprador, contraseñaCorrecta añadirProducto::stockSuficiente finCompra::maxComprador, stockSuficiente
ECD	Dominio	iniciarCompra::compraCreada añadirProducto::precioAcumulado, lineaCreada, resultado finCompra::resultado
CE	Datos	iniciarCompra::resultado
EL	Presentación	añadirProducto::productoExiste, productoDisponible
VV	Presentación	añadirProducto::unidadesPositivo
AP	Datos	finCompra::compraAlmacenada
RD+AP	Dominio+Datos	finCompra::puntosCliente

Tabla 3. Relación de patrones aplicados y capas a las que se asignan las responsabilidades de RE_2

7. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo hemos estudiado la asignación de servicios en el contexto de la arquitectura en tres capas. Las aportaciones más relevantes de nuestra propuesta son:

- Definición de un marco metodológico sistemático para llevar a cabo esta asignación. El marco se basa en la identificación de los conceptos de responsabilidad y servicio. Hemos clasificado los tipos de responsabilidad, los tratamientos asociados, y hemos introducido un modelo de referencia que define estos conceptos rigurosamente.
- Transición más suave entre las etapas de análisis y diseño. Hemos partido de un modelo de especificación clásico, representativo de los procesos de desarrollo basados en UP, y hemos conectado esta especificación a los conceptos consolidados del mundo de la arquitectura en tres capas. La transición de una etapa a la siguiente queda facilitada y se consigue una buena trazabilidad.
- Definición de un lenguaje de patrones arquitectónicos para la asignación de responsabilidades a capas. Hemos definido catálogos jerarquizados de tipos de responsabilidades y de tratamientos, y los hemos estructurado en forma de patrones. De esta manera, podemos trazar un símil entre nuestra propuesta y el proceso de asignación de responsabilidades a objetos promulgado por Larman [6] en base al catálogo seminal de Gamma *et al.* [10].
- Flexibilidad respecto el contexto de la aplicación. Como hemos argumentado en la sección 2, ASIRCA puede utilizarse en diversas situaciones de partida posibles.
- Robustez respecto la evolución tecnológica. La aparición de nuevas tecnologías se va a

traducir en la formulación de nuevos patrones para tratar los tipos de responsabilidades identificados. El impacto en la adopción de nuevas tecnologías queda minimizado, mientras que por otro lado la portabilidad de las arquitecturas en capas entre diferentes tecnologías se ve facilitada por la trazabilidad propia de la propuesta.

Ya hemos comentado los beneficios de nuestra propuesta en el marco de la asignación manual de responsabilidades a capas. Cabe citar otro ámbito de trabajos relacionados con nuestra propuesta, aquéllos dirigidos a la generación automática de código a partir de una especificación (e.g. Pastor et al. [11]), generalmente en el ámbito MDA. Uno de nuestros objetivos es que ASIRCA sea aplicable tanto a procesos manuales de diseño como a procesos de generación automática de código guiados.

Como trabajo futuro, tenemos tres líneas prioritarias de acción. Por una parte, construir una herramienta de soporte para ASIRCA; nuestra intención es utilizar la herramienta AndroMDA [12] para generar los tratamientos a partir de anotaciones en el modelo que indiquen el patrón a aplicar para cada responsabilidad. Segundo, nos proponemos definir un catálogo exhaustivo de patrones que incluya patrones tecnológicos para algunos de los productos actualmente más difundidos. Por último, queremos usar la metodología en proyectos industriales, lo que es viable debido a la tipología de proyectos desarrollados en el seno de la empresa a la que pertenecen dos de los autores del trabajo.

REFERENCIAS

[1] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, (1996).

[2] F. Buschmann, R. Meunier, H. Rohnert and P. Sommerlad, *Pattern Oriented Software Arquitecture*, Vol. 1, John Wiley, (1996).

[3] G. Booch, I. Jacobson and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, (1999)

[4] G. Booch, I. Jacobson and J. Rumbaugh, *The Unified Modeling Language Reference Manual*, 2ª edición, Addison-Wesley, (2004)

[5] C. Larman, *Applying UML and Patterns, 3rd edition*, Prentice Hall, (2005).

[6] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, (2003).

[7] R.E. Filman, T. Elrad, S. Clarke and M. Aksit, *Aspect-Oriented Software Development*, Addison-Wesley, (2004).

[8] A. Cockburn, *Agile Software Development*, Addison-Wesley, (2002).

[9] www.hibernate.org. Última visita: Mayo 2006

[10] E. Gamma, R. Helm, R. Johnson and J.M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, (1995).

[11] O. Pastor, J. Gómez, E. Insfrán and V. Pelechano, *The OO-method Approach for Information Systems Modeling: from Object-Oriented Conceptual Modeling to Automated Programming*, Information Systems, Vol. 26, n. 7, (2001).

[12] www.andromda.org. Última visita: Mayo 2006.