Derivando el Diseño a Partir de Especificaciones de Requisitos Basadas en Casos de Uso

Luis Roqué Fourcade, Liliana Arakaki

Dpto. de Informática – Facultad de Ciencias Físico Matemáticas y Naturales
Universidad Nacional de San Luis
Ejército de Los Andes 950, CP D5700HHW, San Luis, Argentina, +54 (0266) 4424027
Email: araroq@unsl.edu.ar, liliana.arakaki@yahoo.com

Abstract. Although approaches based on use cases proved to be very effective in requirements capture activity, its lack of formality leaves them far away from the expressions of solution designs for the modeled cases. This lack of formality in the use cases expressions makes the task of deriving solution designs in an ambiguous one, left solely to the wisdom of the analyst/designer. This paper presents a proposal that tries to support the process of deriving solution designs for requirements specifications based on use cases, without resigning the advantages obtained from the informal, close to business expressions that distinguish these approaches.

Keywords: caso de uso, diseño, derivación de diseño, OOD, UML, modelado, MDA, QVT, XSLT

1 Introducción

La falla para alcanzar un grado de satisfacción aceptable en los destinatarios de un producto de software, conjuntamente con la incapacidad para demostrar el grado de satisfacción o insatisfacción alcanzado, constituye el eje de una polémica que ha existido siempre y que perdura aún hoy en ámbitos de la ingeniería de software.

Por este motivo, entre otras cosas, la ingeniería de software se ha ocupado de producir y evolucionar métodos y técnicas destinadas a descubrir y especificar de manera ordenada y sistemática un conjunto de requisitos, especificar el diseño de un producto de software que satisfaga dichos requisitos y, finalmente, construir el producto que implemente tal especificación de diseño y administrar su ciclo de vida.

Sin embargo, a pesar de que "especificar el diseño de un producto de software que satisfaga dichos requisitos", no se refiere en absoluto a una tarea trivial, ninguno de los métodos actuales se ocupa de proveer un soporte que asista de manera efectiva a los desarrolladores en la mencionada tarea.

En particular, los enfoques basados en casos de uso han demostrado ser muy efectivos en la actividad de captura de requisitos, sin embargo, no han acompañado ese éxito con las técnicas necesarias que permitan al desarrollador derivar el diseño de solución correspondiente de manera estándar y segura.

En este trabajo presentamos una propuesta, basada en modelos o patrones que abstraen características útiles como fundamento o guía en decisiones de diseño, con el objetivo de proveer soporte al proceso de derivación de diseños arquitecturales de solución para especificaciones de requisitos basadas en casos de uso.

Es decir, proponemos sustituir el estado actual en el cual el desarrollador ejecuta un trabajo de interpretación y diseño a partir de especificaciones informales y ambiguas de requisitos, por otro en el cual el desarrollador ejecute el mencionado proceso a partir de especificaciones que identifiquen y especifiquen tipos de conocimiento en los requisitos útiles al proceso y asistido por técnicas formales y/o semi-formales y herramientas de desarrollo que las implementen.

El objetivo de la propuesta abarca dos áreas bien definidas:

- Especificación de técnicas para derivación de diseños de solución a partir de especificaciones de requisitos basadas en casos de uso.
- Provisión de bases para la automatización de las técnicas especificadas siguiendo el enfoque MDA (*Model Driven Architecture*) [1].

Hemos organizado el trabajo según las siguientes secciones:

- Fundamentos: En esta sección, en diferentes subsecciones, presentamos una introducción y descripción de diferentes áreas (aunque no exhaustivas) que proveen fundamento a potenciales soportes al proceso de derivación.
- Derivación de diseños arquitecturales: En esta sección, basándonos en los fundamentos presentados, desarrollamos la propuesta mediante dos subsecciones: una dedicada a la propuesta en sí misma y otra dedicada a presentar un bosquejo de una posible técnica de derivación sustentada en la propuesta.
- Resultados y trabajos futuros: Finalmente, en esta sección describimos el estado en que se encuentra la evolución en las dos áreas comprendidas en el trabajo y mencionamos pasos a seguir en el corto y mediano plazo.

2 Fundamentos

En esta sección presentamos áreas de conocimiento que sirven de fundamento a la propuesta descripta en el trabajo, organizadas en las diferentes subsecciones.

Estas áreas de conocimiento, aunque no son exhaustivas, forman parte de los fundamentos del trabajo y las desarrollamos con suficiente nivel de detalle para simplificar la siguiente sección que expone el desarrollo del trabajo.

2.1 Especificaciones de Requisitos Basadas en Casos de Uso

Un sistema no existe de manera aislada, sino que interactúa con elementos externos, humanos o mecánicos, que lo utilizan para lograr algún objetivo. Un caso de uso especifica el comportamiento de un sistema o parte del mismo mediante la descripción escrita de las interacciones que se llevan a cabo con los elementos externos, o acto-

res[1], en respuesta a un evento iniciado por uno de ellos, el actor principal. El conjunto de interacciones es descripto de manera textual y en lenguaje natural, como una secuencia de acciones ejecutadas por el sistema, destacando una secuencia o flujo normal y luego todas las secuencias alternativas posibles.

La técnica de casos de uso, por su capacidad expresiva y facilidad de uso, es muy valiosa en la captura de requisitos; cada caso de uso describe en términos de los usuarios un requisito funcional del sistema sin comprometerlo con criterios formales de solución o aspectos tecnológicos. Pero tal flexibilidad descriptiva resulta en especificaciones informales que dificultan el proceso de derivación de la solución y de validación de los requisitos.

Por ese motivo, el espacio entre la expresión informal de una especificación de requisitos basada en casos de uso y la correspondiente expresión formal de un modelo de diseño ha sido direccionado por numerosas propuestas de trabajo. El objetivo de estas propuestas consiste en expresar el conocimiento con ciertos niveles de formalidad de manera tal que vuelvan más controlable el proceso de derivación de diseños de solución sin resignar la riqueza del vocabulario del dominio ni la facilidad de uso, propios de la técnica de casos de uso.

Entre estas propuestas podemos mencionar, a modo de ejemplo, 'Uso de plantillas genéricas para la descripción de casos de uso' [2] y 'Uso de esquemas de operación como suplemento a los casos de uso' [3].

El uso de plantillas genéricas para la descripción de casos de uso [2], intenta organizar el conocimiento de manera sistemática y categorizada. Permite "plantear soluciones genéricas, estandarizadas y validadas a problemas similares, en las etapas de captura de requisitos, análisis y diseño" [2]. Este enfoque constituye un recurso valioso porque necesitamos recursos que nos permitan establecer un vínculo riguroso entre especificaciones 'cercanas al negocio' y especificaciones formales (o semi-formales) del conocimiento, aptas para tareas de diseño de solución (subsección 2.2).

El uso de esquemas de operación como suplemento a los casos de uso [3], propone describir declarativamente los efectos de una operación de sistema[2] por medio de una representación abstracta del estado del sistema antes y después de ejecutar la operación y de todos los eventos enviados al exterior.

Los estados inicial y final son especificados mediante pre y pos-condiciones expresados en OCL (*Object Constraint Language*) [4], que especifican los estados en términos de objetos, atributos y asociaciones del modelo de clases del dominio.

Claramente es posible, a partir de un esquema de operación derivar, una especificación de diseño expresada como un modelo de colaboración. Así, la posibilidad de identificar operaciones de sistema, especificarlas mediante esquemas de operación y derivar el correspondiente modelo de colaboración, las convierte en un enfoque valioso para el objetivo planteado en este trabajo.

---

[1] Estas interacciones constituyen una de las claves en la identificación de aspectos útiles al proceso de derivación de diseño [3]

[2] Una operación de sistema es una acción particular del sistema que se ejecuta atómicamente y que conjuntamente con otras acciones integran una secuencia de un caso de uso [3] [9].

2.2 Modelo de Procesamiento de Requerimientos[3]

A partir de la definición de caso de uso proporcionada por OMG[4], podemos tratar un caso de uso como una especificación de un tipo de requerimiento que un actor hace a un sistema[5]. De este modo, las especificaciones de posibles secuencias de acciones especifican posibles tratamientos del mencionado requerimiento, es decir, el comportamiento esperado del sistema.

Teniendo en cuenta estas consideraciones, las especificaciones de requisitos de software deberían incluir al menos dos partes: una que preserve las características benéficas, típicas de las descripciones de casos de uso, para especificar requisitos externos, y otra que considere características más adecuadas para especificaciones de la funcionalidad ofrecida, como por ejemplo, correctitud, ausencia de ambigüedad, completitud, consistencia y trazabilidad, entre otras [5] .

Es decir que, podemos adoptar dos puntos de vista distintos para la especificación de requisitos de un producto de software y definir rigurosamente las correspondencias entre los tipos de especificaciones resultantes.

Especificaciones enfocadas desde el punto de vista del actor: con el poder expresivo y la naturaleza textual que caracteriza a las especificaciones actuales de requisitos.

Este tipo de especificación además, se caracteriza por ser funcional es decir, al ser enfocado desde el punto de vista del actor, el requisito es visto como un servicio que el sistema le brinda al actor, lo que equivale a una mirada funcional del sistema.
Especificaciones de funcionalidad ofrecida: libres de ambigüedades y por lo tanto con requisitos de formalidad imprescindibles para especificaciones internas de requisitos.

En este trabajo incluimos una abstracción que observa cada requerimiento desde el punto de vista de su tratamiento, identificando categorías de responsabilidades involucradas, y plasmándolas en un patrón (figura 1).

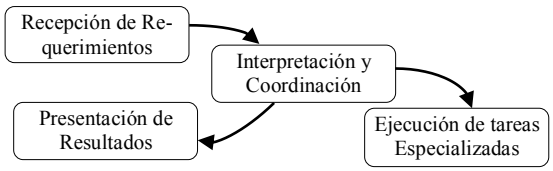

Fig. 1. Patrón de tratamiento de requerimientos

El objetivo consiste en adoptar una forma de especificación de requisitos que permita especificarlos a la luz de este modelo o patrón de manera tal que la información aportada por el mismo pueda ser tomada por una técnica que permita derivar y/o validar posibles diseños arquitecturales de soluciones para el requisito en cuestión.
En el patrón (figura 1) se representan las siguientes categorías de responsabilidades:

[3] Utilizamos aquí *requerimiento* en lugar de r*equisito* para distinguir 'lo que se espera cumplir' (requisito) de 'lo que se ha dado y es necesario procesar' (requerimiento) [11]

[4] Casos de uso pueden ser usados tanto para la especificación de requisitos (externos) al sistema como para la especificación de la funcionalidad ofrecida por éste [10].

[5] Requerimiento: Acción y efecto de requerir, requisito: Circunstancia o condición necesaria para algo. [11]

Recepción de Requerimientos: presentación de formularios, asistencia inteligente en la entrada del requerimiento, validaciones, conversiones, envío del requerimiento en términos de los requisitos para los mismos, etc.
Interpretación y coordinación de tareas: Interpretación y selección de plan de acción, coordinación y control de tareas especializadas, integración de resultados parciales y elaboración de resultado final, despacho de resultado, etc.
Ejecución de tareas especializadas: Ejecución de diferentes niveles de tareas con alto contenido de conocimiento especializado y reusable. Tareas atómicas, como por ejemplo representación de objetos de negocio o gestión de ciclo de vida de los mismos, o colaboraciones, como modelado de distintos tipos de procesos de negocio.
Presentación de Resultados: Elaboración de presentaciones de resultados, asistencia inteligente en la interpretación de los resultados, etc.

Derivar el diseño de una arquitectura de solución para el tipo de requerimiento en cuestión, requiere un cambio en nuestro punto de vista: de uno "centrado en el actor" a otro "centrado en el sistema" y de uno funcional a otro orientado a objetos

En este punto es posible reorganizar y/o formalizar algunos aspectos de la especificación del caso de uso en cuestión, de modo que resulte menos 'brusco' y menos librado al azar el paso a una primera versión de una especificación de diseño.

El posible patrón bosquejado (figura 1) se caracteriza por su simplicidad pero arroja una luz importante al proceso de derivación. La organización del conocimiento sugerida por el patrón puede actuar como una guía efectiva a la hora de descubrir un conjunto de objetos que, asignándoles tales responsabilidades, sean capaces de colaborar para satisfacer el requerimiento. Por ejemplo, a partir de esta simple tipificación de responsabilidades es posible inferir una colaboración en la que estarán presentes mínimamente, los siguientes componentes:

- el actor que formula el requerimiento,
- un gestor que posee el conocimiento acerca de cómo gestionarlo,
- un conjunto de objetos de negocio involucrados en la gestión y
- gestores de ciclo de vida de los objetos de negocio involucrados

Así, la construcción de un modelo de colaboración y/o de secuencia a partir de esta información resulta prácticamente trivial.

2.3 Patrones Arquitecturales y de Diseño

Si aspiramos a contar con un proceso de derivación del diseño de solución que sea controlado y guiado por técnicas precisas, que garanticen la calidad del resultado obtenido, podemos asegurar que el uso de patrones arquitecturales y de diseño constituye un recurso esencial en cualquier técnica que pretenda dar soporte al proceso.

Los patrones arquitecturales expresan esquemas de organización estructurales fundamentales de los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen normas y directrices para organizar las relaciones entre los mismos [6].

Los patrones arquitecturales desempeñan un rol de gran importancia en el proceso de desarrollo de software desde diferentes ángulos: estandarización de diseños arqui-

tecturales, garantía de características deseables en aspectos direccionados por el patrón y asistencia al proceso de diseño arquitectural.

En particular, en cuanto a la "asistencia al proceso de diseño arquitectural", intuitivamente es fácil concluir que un patrón arquitectural puede actuar como guía y como marco contra el cual contrastar decisiones en el proceso de derivación. Esta conclusión puede ser inferida también si se homologa el proceso de derivación con otro dominio en el cual esta conclusión haya sido convalidada. Un dominio de esta naturaleza, y con el cual dicha homologación se puede realizar con relativa simplicidad, es el de Tecnología Orientada a Objetos. Considerando este dominio, se pueden establecer correspondencias casi directas entre patrón arquitectural y clase y entre diseño arquitectural e instancia. Es decir que estas correspondencias nos habilitan para considerar a un patrón arquitectural como una clase y a un diseño arquitectural como una instancia de una clase, el patrón arquitectural.

Así, estamos en condiciones de asegurar que el proceso de derivar un diseño arquitectural a partir de un patrón arquitectural equivale al proceso de instanciación por el cual una instancia hereda características definidas en la clase y ajusta (o asigna) valores a lo que define su propia identidad.

En el trabajo tomamos en consideración dos patrones arquitecturales los cuales abstraen características que además de ser complementarias, de manera conjunta conforman un importante marco de referencia para el proceso en cuestión.

Patrón Arquitectural *Model-View-Controller* (MVC). Este patrón especifica un modelo arquitectural que separa claramente la presentación, las acciones basadas en las entradas del usuario y el modelado del dominio, definiendo con precisión los tres componentes arquitecturales y sus relaciones [7].

Como se puede apreciar (figura 2), el modelo no posee ninguna dependencia respecto de los otros componentes (vista y controlador) dado que, como lo indica la navegabilidad de las relaciones, tanto la vista como el controlador, tienen conocimiento del modelo, pero no ocurre la inversa.

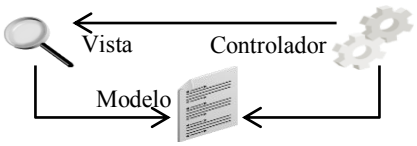

Fig. 2. MVC – Modelo estructural

Del mismo modo (figura 2), podemos observar también que la vista es independiente del controlador ya que la navegabilidad de la relación indica que sólo el controlador tiene conocimiento de la vista y no a la inversa.

Así, una de la principales virtudes de este patrón arquitectural, radica en la flexibilidad de los diseños arquitecturales resultantes: claramente el modelo es altamente reusable, el controlador es el único componente capaz de recibir, interpretar y procesar entrada de usuario y es responsable tanto de operar sobre el modelo como de seleccionar las vistas para interactuar con el usuario y, la vista resulta en un componente

completamente independiente del controlador y, por lo tanto, intercambiable sin afectar al resto de la arquitectura.

Como se observa (figura 4), MVC define formalmente el significado de las relaciones entre los componentes arquitecturales utilizando el patrón de diseño *Observer*, el cual provee un mecanismo para alertar a otros objetos acerca de cambios de estado sin introducir dependencias sobre ellos [8].

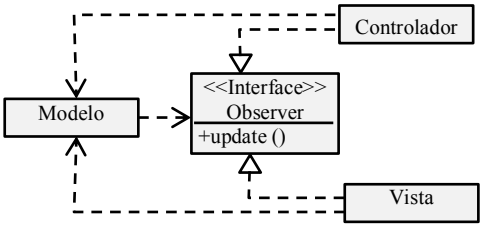

Fig. 3. Patrón de diseño observer

Steve Burbeck en [7], describe dos modelos: pasivo y activo. En el modelo pasivo, dado que el controlador es responsable de recibir, interpretar y procesar entrada del usuario y de operar sobre el modelo, cuando el estado del modelo cambia como consecuencia de esta operación, el controlador alerta a la vista para que ésta se refresque consultando el modelo. En el modelo activo, cuando el estado del modelo cambia por otra vía (actúan sobre él otras fuentes fuera del control del controlador), el modelo debe alertar a la vista. Sin embargo, esta situación no significa que exista una dependencia del modelo respecto de las vistas, porque el modelo mantiene un único objeto que será notificado y, este objeto, mantiene una lista de todas las vistas (y eventualmente controladores) suscriptas a la notificación e itera por ellas notificándolas para que se refresquen consultando el modelo.

Estas características que han hecho tan popular a este patrón, son heredadas por cualquier diseño arquitectural que pueda ser considerado instancia del patrón.

Otra característica destacada de MVC, quizás no tan popular, pero que se desprende de su definición, radica en su "cercanía" al conocimiento de negocio. Es decir que, sus componentes arquitecturales inducen una partición en el conocimiento de negocio que hace relativamente fácil "mapear" este conocimiento a sus componentes arquitecturales. Esta característica es muy importante para nuestro propósito porque, como veremos más adelante en el trabajo, sumada a una tipificación del conocimiento de negocio nos permite rápidamente interpretarlo en términos del patrón y establecer correspondencias esenciales para la técnica de derivación.

Patrón Arquitectural *Java Enterprise Edition* (JEE). JEE define un patrón arquitectural que interpreta el estado del arte en el aspecto arquitectural. Un estado que ha evolucionado desde los primeros enfoques monolíticos, centralizados y homogéneos hasta arribar a uno de múltiples capas, distribuido, orientado a componentes y desplegable en un ambiente heterogéneo como internet.

Pero JEE, va un poco más allá y especializa esa concepción definiendo un modelo arquitectural que incluye como mínimo tres capas con funcionalidades bien definidas

(cliente, web y negocio) y tipos de componentes que se desempeñan en ellas con funcionalidades y responsabilidades específicas en el marco de las mismas.

Tanto las capas como los componentes que se despeñan en las mismas, están formalmente definidos como así también las relaciones e interfaces entre los mismos.

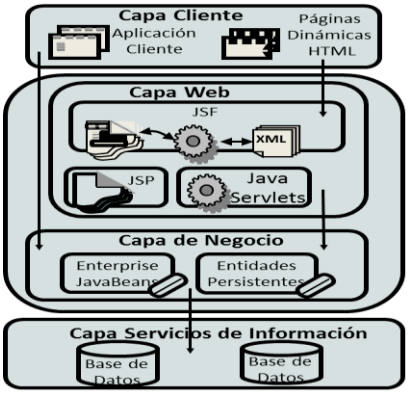

Fig. 4. JEE – Modelo estructural[6]

En la capa cliente se desempeña el cliente que puede ser de dos tipos: cliente web, que consiste de un navegador exhibiendo páginas dinámicas generadas por componentes que se desempeñan en la capa web, y aplicación cliente, un cliente autónomo (*standalone*) que se responsabiliza de todas las funciones que de otro modo recaerían en componentes que se desempeñan en la capa web.

La capa web hospeda componentes con responsabilidades como elaborar presentaciones, recibir y procesar requerimiento y otras tareas complementarias.

La capa de negocio hospeda componentes que modelan conocimiento de negocio especificados de manera tal que garantizan su reusabilidad y portabilidad y requieren exclusivamente código inherente al conocimiento que modelan. Además están tipificados con características y servicios particulares que los hacen adecuados para modelar diferentes tipos de conocimiento de negocio.

Cada capa, especificada como un contenedor, coopera con el servidor JEE para proveer de manera declarativa y transparente un amplio conjunto de servicios, inherentes a la funcionalidad y responsabilidades del tipo de los componentes, como por ejemplo manejo transaccional, seguridad, intercambio de mensajes, etc. [9]

Al igual que en el caso de MVC, las características descriptas son heredadas por cualquier diseño arquitectural que pueda ser considerado instancia del patrón.

También, al igual que en el caso de MVC, JEE tiene otra característica destacada que nos será de utilidad en el trabajo: su "cercanía" a la implementación. Es decir que si es posible especificar el diseño arquitectural en términos de las capas, componentes y especificaciones JEE, el camino para obtener un producto de software que satisfaga la especificación de requisitos, será relativamente fácil de recorrer.

[6] Adaptación de figuras similares incluidas en el tutorial Java EE5 Tutorial [13]

A modo de resumen podemos afirmar que ambos patrones especializan otro más abstracto: patrón arquitectural de múltiples capas, distribuido, orientado a componentes y heterogéneo, que abstrae el estado del arte en la evolución arquitectural.

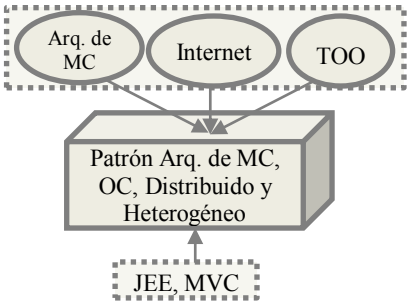

Fig. 5. Especializaciones MVC y JEE

Dado el propósito del trabajo, resulta de interés la definición de un patrón arquitectural más especializado aún, que herede características de ambos patrones. Desde el punto de vista de la técnica para soportar el proceso de derivación, estamos particularmente interesados en la "cercanía al negocio" que caracteriza a MVC y la "cercanía a la implementación" que caracteriza a JEE.

Otro aspecto a considerar en este marco es patrones de diseño. Éstos proveen un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describen una estructura recurrente de componentes comunicándose que resuelven problemas generales en contextos particulares [8].

Algunos patrones de diseño proveen estructuras para descomponer componentes o servicios más complejos. Otros direccionan la cooperación efectiva entre ellos [6].

Es decir que en una eventual técnica de derivación del diseño arquitectural de solución, patrones arquitecturales y patrones de diseño pueden colaborar para proveer un soporte comprensivo a la técnica. En un posible paso de refinamiento, en el que se pretende delegar responsabilidades de un objeto, un análisis del contexto y del problema particular puede aconsejar el empleo de diferentes patrones de diseño, por ejemplo un patrón de fachada para ocultar la complejidad del control y coordinación de un conjunto de objetos que cooperan para proveer una serie de servicios.

Claramente, los diseños arquitecturales obtenidos como consecuencia de cada paso de refinamiento mantendrán la propiedad de ser instancia del patrón arquitectural y, por lo tanto, el objetivo de asistencia automatizada al proceso de derivación es mucho más factible. Siguiendo el enfoque MDA (*Model Driven Architecture*) [1], es posible extender los meta-modelos de casos de uso y de diseño, definir meta-modelos de las técnicas sugeridas y definir transformaciones QVT entre éstos de manera que sean fundamento de posibles soportes automatizados al proceso de derivación.

2.4 Clasificación del Conocimiento de Negocio

La asistencia al proceso de derivación de diseños de solución constituye un objetivo verdaderamente ambicioso y ya hemos comentado acerca de la importancia de obser-

var y analizar el conocimiento representado por el caso de uso a la luz de patrones. Si logramos además establecer una tipificación del conocimiento, estaremos en mejor posición para tomar decisiones arquitecturales y de diseño contrastando el conocimiento tipificado contra las especificaciones de los patrones. Por ejemplo tipificar el conocimiento de negocio clasificándolo en procesos y objetos de negocio, aportaría una herramienta fundamental a la hora de derivar una primera versión de diseño. Ante un requerimiento hipotético podríamos mínimamente concluir que al menos habrá un único proceso de negocio responsable de satisfacerlo y que deberá y coordinar un conjunto de objetos de negocio para ese objetivo.

Hay abundante material acerca de conocimiento de negocio, su representación y diferentes clasificaciones. Nosotros, a los efectos de las características del presente trabajo, introducimos una clasificación bastante simple y al único efecto de lo que requerimos, omitiendo algunos casos que no nos resultan de utilidad.

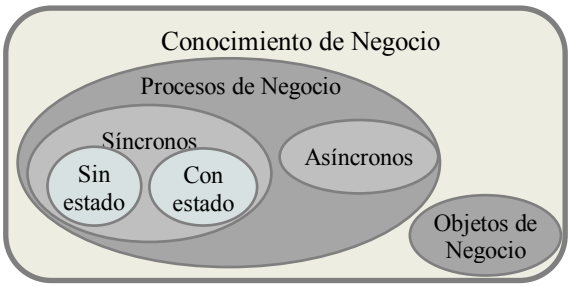

Fig. 6. Modelo de conocimiento de negocio

El modelo presenta dos tipos o clases de conocimiento de negocio (figura 6):

Procesos de Negocio: son procesos que la organización lleva a cabo para cumplir con sus objetivos, los cuales pueden ser de naturaleza síncrona o asíncrona.

Los de naturaleza síncrona se refieren a procesos que requieren que las interacciones con sus clientes se produzcan de manera explícita y síncrona. Cuando cada interacción con un cliente dado está definida por un conjunto de invocaciones dependientes entre sí, tipificamos al proceso como síncrono con requerimientos de estado. En cambio, cuando cada interacción involucra una única invocación, lo tipificamos como síncrono sin requerimientos de estado.

Los de naturaleza asíncrona en cambio, se caracterizan porque su ejecución se produce generalmente por la satisfacción de algún hecho, como podría ser 'el arribo de un momento en el tiempo'. De esta manera, estos procesos no pueden ser invocados de manera síncrona sino que, en el negocio, se verifican ciertos hechos, los cuales pueden ser determinantes para la ejecución asíncrona de algún proceso de este tipo. No consideraremos tipificaciones dependientes de requerimientos de estado para esta clase de procesos por no aportar información de utilidad a este trabajo en virtud de los patrones considerados en el mismo.

Objetos de Negocio: representan entidades de interés para el negocio sobre las cuales puede hacer o decir algo, particularmente a través de sus procesos de negocio. Una característica distintiva de éstos radica en su ciclo de vida, el cual es ortogonal al de

los procesos de negocio. Los objetos de negocio persisten más allá del alcance de los procesos de negocio y, generalmente, poseen requerimientos de persistencia.

3 Derivación de Diseños Arquitecturales

En esta sección, a la luz de lo expuesto en la sección *Fundamentos*, presentamos el desarrollo de una posible técnica para soportar el proceso de derivación de diseños arquitecturales para especificaciones de requisitos basadas en casos de uso.

Para este efecto formulamos un patrón que resulta de integrar los patrones y modelos descriptos. En particular, no tendremos en cuenta las áreas de esquemas de operación y patrones de diseño, por cuestiones relativas a la extensión del trabajo y a aspectos de ambas que se encuentran en procesos de estabilización.

3.1 Integrando Patrones y Modelos

En esta sección presentamos los resultados de integrar los diferentes patrones y modelos descriptos y apreciaciones acerca de los mismos.

MVC→JEE

La integración MVC → JEE es bastante directa. De acuerdo a lo introducido en la sección Fundamentos para estos patrones arquitecturales resulta casi obvio concluir las correspondencias (figura 7).

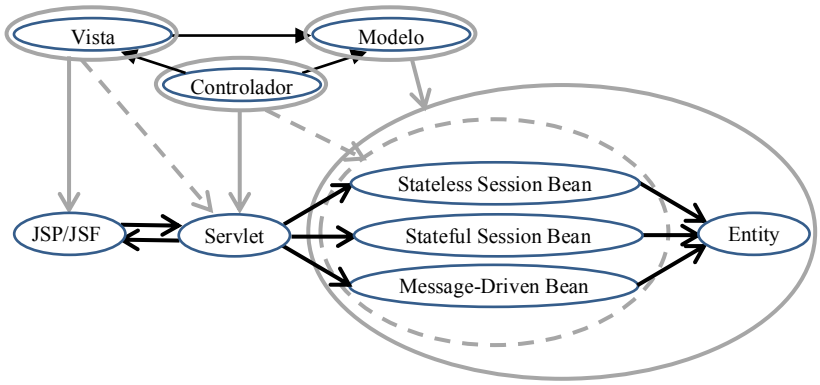

Fig. 7. Integración MVC→JEE

La integración incluye las siguientes correspondencias (figura 7):

Vista →JSP/JSF: La vista es responsable de las presentaciones. En JEE, los tipos de componentes que mejor se adecuan a este tipo de responsabilidades son JSP y JSF [9].
Vista - ->Servlet: Si bien los *servlets* pueden asumir estas responsabilidades, no es lo más recomendable por la incomodidad de generar contenido desde sentencias Java.
Controlador →Servlet: El controlador tiene la responsabilidad de controlar componentes del modelo. En JEE componentes de tipo Servlet son los que mejor se adecuan

por sus capacidades para el manejo de complejidad sumadas a servicios provistos por el contenedor web del servidor JEE para tal fin [9].
Controlador - -→ EJB: Componentes de tipo EJB en JEE también se adecuan bien a este tipo de responsabilidades. Sin embargo, las características de alta reusabilidad sumadas a capacidades provistas por el contenedor de negocio implican un costo que debe justificarse por el tipo de conocimiento a modelar, el cual debe ser especializado, es decir reusable y con necesidades de servicios como los mencionados [9].
Modelo →EJB + Entidades Persistentes: El modelo comprende responsabilidades referidas a lógica y reglas de negocio y, por lo tanto, componentes de tipo EJB en JEE se adecuan bien para modelarlas. Cuando el tipo de conocimiento además posee requerimientos de persistencia, el tipo de componente que mejor se adecua es *entity* [9].

Conocimiento de Negocio → (MVC → JEE)
El conocimiento de negocio, por tratarse de conocimiento especializado, es altamente reusable y requiere de servicios del tipo de los provistos por el servidor JEE [9]. Por este motivo determina responsabilidades categorizadas como parte del modelo en MVC y asignables a componentes JEE del tipo EJB, en el caso de procesos de negocio y *entity* (entidad persistente), en el caso de objetos de negocio.

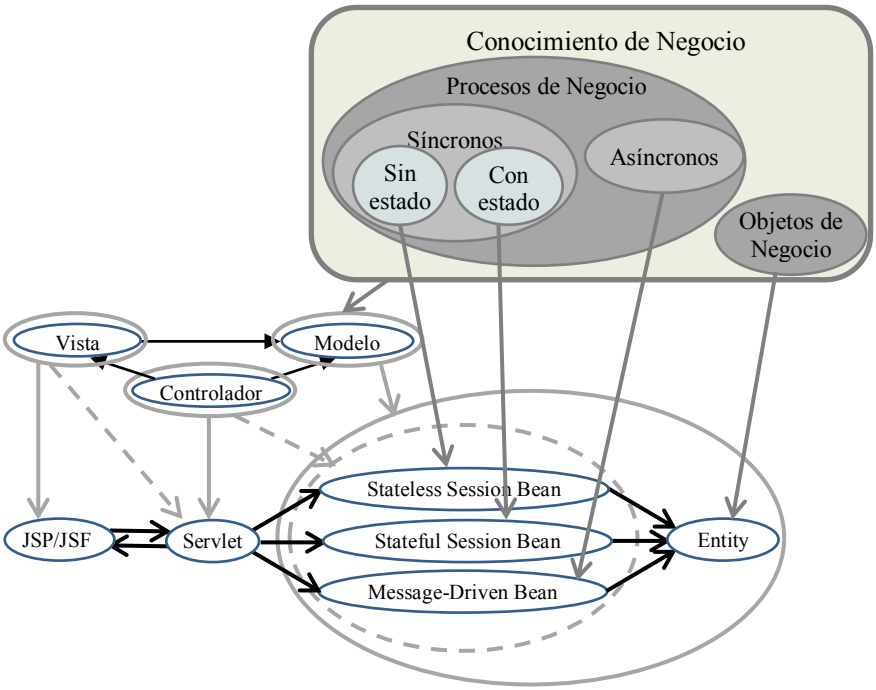

Fig. 8. Integración conocimiento de negocio → (MVC→JEE)

En la integración se pueden observar las siguientes correspondencias (figura 8):

- Conocimiento de Negocio → Modelo (MVC)

- Proceso de Negocio Síncrono
  - con requerimientos de estado → *Stateful Session Bean* (JEE)
  - sin requerimientos de estado → *Stateless Session Bean* (JEE)
- Proceso de Negocio Asíncrono → *Message-Driven Bean* (JEE)
- Objeto de Negocio → *Entity* (JEE)

Modelo de Procesamiento de Requerimientos → (MVC → JEE).
El patrón de tratamiento de requerimientos esbozado modela una abstracción de estrategias para el tratamiento de requerimientos. Como en el caso de MVC, es fácilmente asimilable al negocio. Por lo tanto, es conveniente especificar las correspondencias con componentes arquitecturales de MVC y luego aprovechar las que ya están definidas entre MVC y JEE para establecer las correspondencias entre responsabilidades del patrón de tratamiento de requerimientos y componentes de JEE.

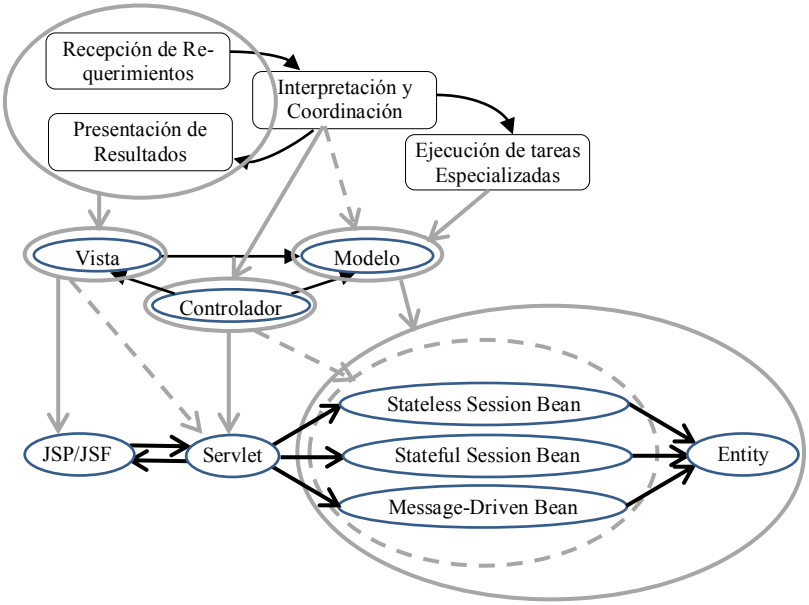

Fig. 9. Integración modelo de procesamiento de requerimientos → (MVC→JEE)

En la integración se pueden observar las siguientes correspondencias (figura 9):

- Recepción de Requerimientos/ Presentación de Resultados → Vista → JSP/JSF
- Interpretación y coordinación → Controlador/Modelo → Servlet/EJB
- Ejecución de Tareas Especializadas → Modelo → EJB

3.2 Modelo Integrado Resultante

La estrategia que hemos seguido en el proceso de integración se ha basado en el objetivo de tratar de explotar características benéficas de cada patrón para un eventual proceso de derivación de diseño arquitectural (figura 10).

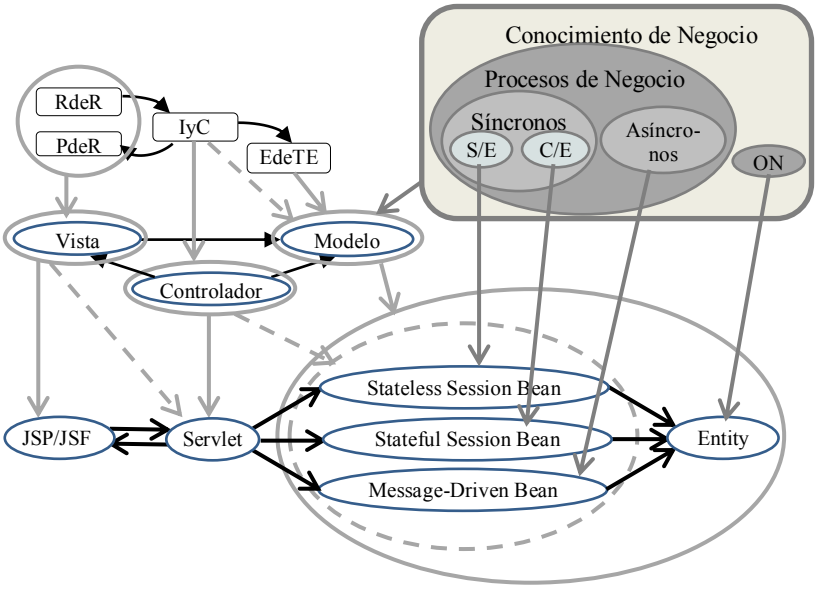

Fig. 10. Modelo integrado de patrones[7]

La cercanía al negocio de modelos de tratamiento de requerimientos y de conocimiento de negocio permite una interpretación relativamente fácil del conocimiento contenido en un caso de uso para proceder posteriormente, apoyados en las correspondencias, a derivar una especificación de diseño muy cercana a la implementación heredando decisiones y características benéficas de los patrones MVC y JEE.

3.3 Derivando el Diseño Arquitectural

Presentamos aquí un bosquejo de una posible técnica de derivación de un diseño arquitectural a partir de una especificación de requisitos basada en casos de uso.

- Identificación de responsabilidades en patrón de tratamiento de requerimientos
  - Actividades de ejecución de tareas especializadas: comenzar por el conocimiento obvio, los objetos de negocio involucrados en el caso de uso.
  - Actividades de interpretación y coordinación: inicialmente asumir un único objeto, el cual posee el conocimiento completo acerca de cómo satisfacer el requerimiento valiéndose de los objetos de negocio involucrados.
  - Actividades de recepción de requerimientos y presentación de resultados: asignar interacciones de envío del requerimiento y presentación del resultado entre el actor y el único objeto controlador.
- Aplicación de técnicas de delegación y abstracción de comportamiento hasta alcanzar un diseño satisfactorio[8].

[7] Hemos optado por hacer reducciones que comprimen nombres de elementos. Para su correcta interpretación, referirse a los modelos originales de los patrones.

- Asignación de categorías MVC a los componentes
  - Asignar objetos que modelan objetos de negocio a categoría modelo.
  - Asignar objetos que modelan conocimiento de cómo tratar el requerimiento a categoría controlador.
  - Asignar objetos que modelan actividades de recepción de requerimientos y presentación de resultados a categoría vista.
- Asignación de componente JEE que mejor se adecua a la clase de cada objeto
  - Asignar tipo *entity* a objetos que modelan objetos de negocio.
  - Asignar tipo *servlet* a objetos que modelan conocimiento de tratamiento del requerimiento con conocimiento de coordinación y control no especializado.
  - Asignar tipo EJB, s*ession bean* (*stateless* o *stateful*) o *message-driven bean*, cuando se trate de conocimiento de coordinación y control especializado[9].
  - Asignar tipo JSP/JSF a objetos que modelan actividades de recepción de requerimientos y presentación de resultados
- Derivar especificación de comportamiento y de estado de cada objeto
  - Derivar especificaciones de métodos: método a partir de intercambio de mensajes, tipo de resultado a partir de respuesta de los mensajes y parámetros a partir de necesidades del objeto para procesar el mensaje, provistas por el emisor.
- Derivar estado del objeto a partir de necesidades para procesar cada mensaje no resueltas con parámetros ni con preguntas a otros objetos.

4 Resultados y Trabajos Futuros

En lo referido a técnicas de derivación, se han formulado hipótesis de trabajo:

- A partir de un modelo de tratamiento de requerimientos, derivación de primera versión de un modelo de secuencia asignando responsabilidades del modelo.
- Aplicación de pasos de refinamiento aplicando técnicas de delegación apoyadas en patrones de diseño simples como composición, fachada, comandos, etc.
- Estructuración arquitectural del diseño de solución obtenido basada en parámetros resultantes de la integración de patrones.

En base a estas hipótesis de trabajo se han formulado casos experimentales (algunos ya terminados y otros aún en curso) en el ámbito académico y privado.

En lo referido a extensión de meta-modelos existentes y definición de meta-modelos de técnicas de derivación, se encuentra en desarrollo el estudio de meta-modelos UML involucrados en el alcance, y la especificación de necesidades de extensión, en función de resultados obtenidos a partir de casos experimentales. Esta actividad es parte del desarrollo de tesis de maestría correspondientes a la Maestría en Ingeniería de Software de la Universidad Nacional de San Luis y del proyecto de investigación en que se enmarca el trabajo.

---

[8] Aquí intervienen áreas como patrones de diseño y esquemas de operación, fuera del alcance del trabajo por los motivos esgrimidos previamente.

[9] Al momento de estas asignaciones, el diseño estará en una versión refinada, con una abundante riqueza de objetos y mucha simplicidad en la tarea de asignación.

También en el mismo marco de tesis de maestría y proyecto, se están analizando y especificando casos de transformaciones utilizando QVT y XSLT.

El objetivo a mediano plazo consiste en alcanzar un punto en el cual sea posible especificar formalmente algunas técnicas de derivación junto con las especificaciones de meta-modelo y posibles transformaciones.

También, un grupo se encuentra trabajando en recopilación y formalización de información de patrones de diseño e investigación de aplicaciones en casos de refinamiento de diseños arquitecturales.

Referencias

[1] O. Object Management Group, «OMG Model Driven Architecture», 2012. En línea: http://www.omg.org/mda/.

[2] M. Daniele y D. Riesco, «Transformación de modelos aplicada a la definición genérica de Casos de Uso utilizando QVT (Query/View/Transformation) y RTG (Reglas de Transformación de Grafos)», *WICC*, Rosario, Arg., 2011.

[3] S. Sendall y A. Strohmeier, «From use cases to system operation specifications», de *Proceedings of the 3rd international conference on The unified modeling language: advancing the standard*, York, UK, 2000.

[4] J. Warmer y A. Kleppe, «The Object Constraint Language: Precise Modeling With UML», Addison-Wesley, 1998.

[5] Software Engineering Standards Committee, «830-1998 - IEEE Recommended Practice for Software Requirements Specifications», 1998. En línea: http://standards.ieee.org/findstds/standard/830-1998.html.

[6] Buschmann, et al., «Pattern Oriented Software Architecture – A System of Patterns», Wiley, 2001.

[7] S. Burbeck, «Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)», 1992. En línea: st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

[8] Gamma, et al., «Design Patterns: Elements of Reusable Object-Oriented Software», Addison Wesley, 1995.

[9] Oracle, «Java EE at a glance», 2012. En línea: http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[10] I. Jacobson, M. Griss y P. Jonsson, «Software Reuse: Architecture Process and Organization for Business Success», Addison-Wesley, 1997.

[11] O. Object Management Group, «OMG Unified Modeling Language Superstructure Specification», Mayo 2011. En línea: http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF.

[12] Real Academia Española, «Real Academia Española», 2010. En línea: http://buscon.rae.es/draeI/.

[13] Oracle, «Java EE 5 Tutorial», June 2010. En línea: http://docs.oracle.com/javaee/5/tutorial/doc/gexaj.html.