MODELACIÓN DE LA ORIENTACIÓN A ASPECTOS

Francisca Losavio[1], Alfredo Matteo[1], Patricia Morantes[2]

[1]Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación.
Caracas. Venezuela. e-mail: francislosavio@gmail.com
[2]Universidad Nacional Experimental Francisco de Miranda. Coro 4101. Venezuela

Recibido: octubre de 2008 Recibido en forma final revisado: agosto de 2010

RESUMEN

El desarrollo de software orientado a aspectos (DSOA) representa un nuevo enfoque dentro de la Ingeniería del software. Está basado en la programación orientada a aspectos (POA) y centrado en la separación de incumbencias transversales o *crosscuting concerns*. Muchos conceptos y/o elementos de modelación se utilizan durante las diferentes etapas del DSOA, sin embargo, se presentan ambigüedades en cuanto a su semántica, no es claro cuál de ellos debe ser utilizado en cuál etapa. Este trabajo presenta un núcleo o Core UML (lenguaje de modelado unificado), para el DSOA, que engloba diferentes elementos de modelación definidos en la literatura, focalizándose particularmente en el documento de ontología DSOA de la comunidad europea y en diferentes extensiones UML. En el Core UML presentado, cada notación es identificada, clarificada, presentada por autor y asociada a una etapa del desarrollo. Los resultados contribuyen al establecimiento de los estándares para una terminología unificada en el DSOA, favoreciendo el entendimiento, la comunicación y facilitando el diseño arquitectónico orientado a aspecto.

Palabras clave: Modelado de software, Aspectos tempranos, POA, DSOA, UML.

ASPECT ORIENTED MODELING

ABSTRACT

Aspect Oriented Software Development (AOSD) is an emerging discipline in Software Engineering based on the Aspect Oriented Programming (AOP) paradigm, and focused on the separation of tangled and scattered concerns (crosscutting concerns). Many concepts and mechanisms have been proposed to handle properly the crosscutting concerns; however, terms are in general semantically slightly different according to the development phase in which they have been defined, causing misunderstanding and confusion. This paper presents an AOSD UML Core (Unified Modeling Language) to integrate different modeling elements defined in the literature, focusing in particular on the AOSD ontology of the European Community and some UML profiles proposed by other authors. In this UML Core, each notation is identified, clarified, presented by author and related to a development phase. This result, on the one hand contributes to the establishment of standards for a unified AOSD terminology, favoring understanding and communication; on the other hand, it facilitates aspect-oriented architectural design.

Keywords: Software modeling, Early aspects, AOP, AOSD, UML.

INTRODUCCIÓN

El DSOA (Sutton *et al*. 2002) es un nuevo paradigma en el desarrollo de software, originado de los trabajos previos sobre la POA. La POA se fundamenta en las metodologías existentes, tanto estructuradas como de objetos, permitiendo nuevos niveles de modularidad a nivel de implementación (Kiczales *et al*. 1997). Actualmente existe un acuerdo general sobre los requisitos funcionales y no funcionales del sistema de software, los cuales deben obedecer a las metas de calidad que el sistema debe lograr, a menudo introduciendo nuevos componentes o mecanismos para satisfacer dichas metas. Sin embargo, muchas de estas incumbencias se consideran o se introducen tarde en el proceso de desarrollo, y entrecruzan las funcionalidades del sistema, haciendo el mantenimiento difícil, contradiciendo el para-

digma orientado a objeto. La idea central del DSOA como disciplina que emerge de la tecnología del post-objeto, es proporcionar una ayuda fuerte en manejar la dispersión o enmarañamiento de incumbencias en cada etapa del desarrollo de software, introduciendo una nueva unidad modular de encapsulación para facilitar la extensibilidad, el mantenimiento y la reutilización.

En el contexto de la ingeniería de software una incumbencia se define como una propiedad o punto de interés de un sistema (Sutton *et al*. 2002). Algunas incumbencias se pueden encapsular fácilmente dentro de clases o de módulos, según el lenguaje de programación elegido; sin embargo, otros requisitos, que generalmente no provienen directamente del usuario, sino que aparecen implícitamente como funcionalidades, pueden afectar varios módulos, y son llamadas incumbencias transversales y no son fáciles de separarse. La meta de la POA es encapsular estos requisitos en una unidad modular llamada aspecto en el nivel de implementación. Las recientes investigaciones proponen utilizar las nociones y mecanismos de la POA en las primeras etapas de desarrollo del software para reducir costos de desarrollo (Rashid *et al*. 2002). Estamos interesados en el enfoque denominado aspectos tempranos, el cual consiste en la identificación de las incumbencias transversales durante las disciplinas de requisitos, análisis y diseño para facilitar el diseño arquitectónico.

El UML (Unified Modeling Language) (Booch *et al*. 1999; OMG, 2003), es un lenguaje gráfico usado ampliamente como un estándar, para apoyar los procesos de desarrollo de software orientado a objeto y basado en componente. UML proporciona mecanismos de extensión para definir nuevos elementos de modelación (Booch *et al*. 1999), se han propuesto extensiones de UML para el DSOA para modelar en etapas tempranas de desarrollo.

Este trabajo presenta un núcleo o Core UML para el DSOA en disciplinas tempranas de desarrollo, recolectando diferentes elementos de modelación encontrados en la literatura (Zakaria *et al*. 2002; Aldawud *et al*. 2003; Brito *et al*. 2003; Moreira *et al*. 2002; Bash *et al*. 2003; Zhang, 2005; Kaewkasi *et al*. 2003; Krechetov *et al*. 2006; entres otros). Los resultados contribuyen a unificar un conjunto de términos usados en DSOA, que permiten mejorar el entendimiento y la comunicación entre los miembros que pertenecen a la comunidad de la ingeniería del software. Por otra parte facilitan el diseño arquitectónico orientado a aspecto.

Nótese que se han dejado algunos términos en inglés, usuales en la literatura sobre este tema, para así facilitar la lectura de las notaciones graficas UML y en particular del Core UML.

La estructura del trabajo es la siguiente: la sección 1 presenta los aspectos básicos del modelado para el DSOA, la sección 2 muestra las extensiones de UML para el DSOA, adicionalmente se define el Core UML para el DSOA basado en las extensiones y la ontología. La sección 3 describe un caso de estudio, un Portal para Cadena de Cines para mostrar el modelado conceptual el cual es modelada utilizando los elementos del Core. Finalmente la sección 4 presenta la conclusión y el trabajo futuro y, la sección 5 el agradecimiento.

MODELADO PARA EL DSOA

El uso de UML, como estándar, implica la comunicación sin ambigüedad. Sin embargo, si un nuevo elemento de modelación se introduce con una nueva semántica, UML se puede ampliar de dos maneras, perfiles y metamodelos, a fin de preservar el estándar. Un perfil expresa los conceptos específicos para cierto dominio de aplicación. Su definición ha evolucionado con las diferentes versiones de UML. Un perfil utiliza la misma notación que un paquete de UML y la palabra clave «profile» (Fuentes *et al*. 2004).

Cuando se utiliza la modalidad de perfil, se usan los mecanismos de extensión, es decir, el estereotipo, valor etiquetado y las restricciones. Un estereotipo es una clase que define cómo las metaclases ya estereotipadas deben ser extendidas, permitiendo el uso de la terminología y/o notación de una plataforma o dominio de una aplicación particular. Algunos estereotipos se predefinen en UML, otros pueden ser definidos por el usuario (OMG, 2003). Una restricción impone condiciones ante los elementos de modelación que se han estereotipado (OMG, 2003). Un valor etiquetado es un meta-atributo adicional asociado a la metaclase del metamodelo extendido por un perfil. Cada valor etiquetado es un par (etiqueta, valor) que se puede utilizar para asignar la información a cualquier elemento o instancia del modelo (OMG, 2003; Fuentes *et al*. 2004).

Un metamodelo es un modelo del lenguaje de modelado, que consiste en un conjunto de conceptos básicos y reglas, permitiendo la construcción de modelos conceptuales en un dominio dado; define las entidades del dominio y sus relaciones.

Recientemente, los elementos de modelación para el DSOA se han definido con frecuencia usando estereotipos de UML. Este trabajo se centra en el estudio de los perfiles para el DSOA. Varios autores (Zakaria *et al*. 2002; Aldawud *et al*. 2003; Brito *et al*. 2003; Moreira *et al*. 2002; Bash *et al*. 2003; Zhang, 2005; Kaewkasi *et al*. 2003; Krechetov *et al*. 2006; entres otros), han hecho esfuerzos en el modelado de elementos de software en etapas tempranas usando el para-

digma de la orientación a aspecto. En este trabajo se obtiene un núcleo o Core UML para DSOA que considera extensiones UML para etapas tempranas de desarrollo. En la figura 1, O representa el conjunto de todos los conceptos del documento "AOSD Ontology" de la Comunidad Europea (Berg *et al*. 2005). CO es un subconjunto de O que incluye todos los conceptos que tienen asociado elementos de modelación para el DSOA; (complemento) es el conjunto de elementos no presentes en CO que han sido definidos por varios autores a través de estereotipos para el DSOA. Por otra parte, la mayoría de estos conceptos tienen estereotipos asociados. El Core DSOA con las extensiones UML está conformado por los conceptos de CO requeridos para modelar DSOA para el cual existen los estereotipos, más los de de la ontología de la Comunidad Europea. En filosofía, la ontología es el estudio de conceptos de la realidad y de la naturaleza de ser. En informática, una ontología es un documento que define formalmente términos y sus relaciones; en un dominio particular proporciona un vocabulario común permitiendo una interpretación uniforme de la terminología, facilitando la comunicación entre los equipos de trabajo, la reutilización y compartir la información.

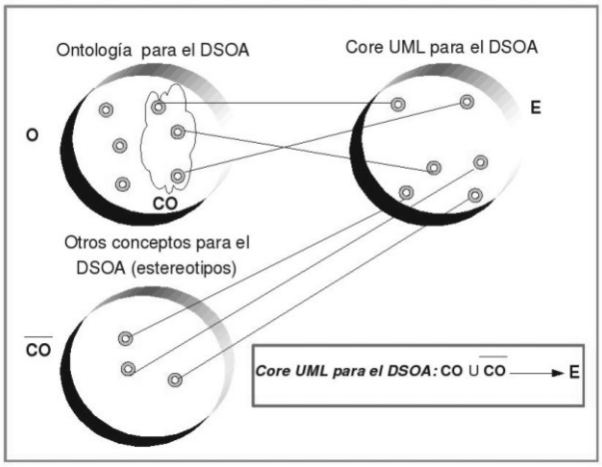


Figura 1. Composición del Core UML para el DSOA.

EXTENSIONES UML PARA EL DSOA

Los conceptos para el modelado orientado a aspecto presentados a continuación se extraen del documento "AOSD Ontology" de la Comunidad Europea (Berg *et al*. 2005), y se comparan con las extensiones orientadas a aspecto (perfiles) presentadas por diversos autores que no se incluyen en la ontología (Zakaria *et al*. 2002; Aldawud *et al*. 2003; Brito *et al*. 2003; Moreira *et al*. 2002; Bash *et al*. 2003; Zhang, 2005; Kaewkasi *et al*. 2003; Krechetov *et al*. 2006; entres otros). Se identifican conceptos que no están en la ontología y aparecen en los trabajos de estos autores, los cuales se incluyen en nuestro Core (figura 1). Estos trabajos fueron seleccionados de la bibliografía de Filman (2005) sobre el DSOA. En lo que sigue, hemos indicado los términos DSOA en castellano y los originales en inglés; sin embargo, podrán ser utilizados indistintamente en el texto, para facilitar la presentación. En lo que sigue, se conservó la terminología original en inglés para los elementos de modelación DSOA.

- *Incumbencia*: es un tema de interés, una parte del problema a resolver, o una característica o propiedad que se debe considerar. Este concepto se utiliza para referirse tanto a propiedades funcionales como a las no funcionales de un sistema. Por ejemplo, un sistema administrador de hotel implica la realización de cliente, reservaciones, control de las entradas y salidas del hotel, autorización de acceso a servicios, etc. Además el proyecto de software requiere de otras incumbencias como capacidad de mantenimiento, comprensibilidad y fácil evolución. Las incumbencias, como cliente y reservación, se conocen como incumbencias primarias porque capturan la funcionalidad central del sistema, mientras que las demás incumbencias se conocen como secundarias porque están compartidas entre las incumbencias primarias, como lo es la persistencia y la autorización de acceso.

- *Incumbencia transversal*: es una incumbencia donde la implementación se dispersa a través del resto del sistema de software. En muchas oportunidades, un nuevo caso de uso es sumado al diagrama de casos de uso por cada incumbencia transversal posible, y se llama aspecto candidato y es representado por un óvalo. También se puede decir que, las incumbencias transversales son conceptos diseminados en el código atravesando partes del sistema no relacionados en el modelo. Ejemplos: *Logging* o registro de la actividad en el sistema, acceso a base de datos, temas relacionados con la seguridad, concurrencia; estas características comúnmente se solapan en otros componentes del sistema funcional base.

- *Aspecto*: es una unidad para modularizar una incumbencia transversal, evitando los problemas de dispersión y enmarañamiento. La notación gráfica del componente es utilizada como representación visual requerida para el desarrollo temprano de aspectos.

- *Composición*: reúne elementos del software creados separadamente. La composición a nivel de modelado de los aspectos tempranos, encapsula aspectos en su propio paquete.

- *Tejido*: es el proceso de componer los módulos de funcionalidades base con aspectos. Se representa por el estereotipo de componente tejedor. Tiene la responsabilidad de procesar el lenguaje base y el lenguaje aspecto

y componerlos con el objetivo de obtener la implementación del sistema.

- *Punto de unión*: indica los lugares bien definidos donde un comportamiento adicional es agregado en la estructura de ejecución o flujo de un programa. Los puntos de unión representan el concepto central de la Programación Orientada a Aspectos y el de su principal lenguaje de implementación AspectJ, se definen como puntos en la ejecución dinámica de un programa interceptados por los aspectos, y en los cuales secciones adicionales de código denominadas recomendación pueden ser ejecutadas. De manera elemental, un punto de corte es cualquier punto de ejecución identificable en un programa o sistema, la llamada a un método es un punto de unión y la ejecución de un método también lo es; una asignación a una variable, la construcción de un objeto, una comparación, un manejador de excepciones entre otros, son ejemplos que permiten identificar tal concepto.

- *Recomendación*: es el comportamiento a ejecutarse en un punto de unión. Una recomendación ayuda a definir "qué hacer", es un mecanismo muy similar a un método, usado para declarar el código que debe ejecutarse en cada punto de unión que es capturado por el punto de corte. Existen tres tipos de recomendaciones: *before advice* (se ejecuta previo al punto de unión), *after advice* (se ejecuta siguiendo al punto de unión) y *around advice* (encierra la ejecución del punto de unión); este último tiene la habilidad de desviar la ejecución, continuar la ejecución normal o causar la ejecución con un contexto alterado.

- *Punto de corte*: es un predicado que se ensambla con los puntos de unión. Es también un conjunto de *puntos de unión*; también se definen como una estructura diseñada para identificar y seleccionar los puntos de unión dentro de un programa en AspectJ.

Los siguientes conceptos no están presentes en la ontología citada, pero han sido presentados por reconocidos autores especialistas en el área. Lo consideramos porque este trabajo se centra en las etapas tempranas del desarrollo de software.

- *Clase base*: es identificada como una unidad principal que encapsula algunas funcionalidades del sistema.

- *Punto de decisión*: es una abstracción del concepto join point en Aspect (Brito *et al*. 2003; Moreira *et al*. 2002). No se ha definido ninguna representación estereotipada ni gráfica para este concepto. Por eso en la tabla 1 el punto de decisión no tiene representación gráfica.

- *Relación clase-aspecto*: el estereotipo «crosscut» se utiliza para modelar las relaciones transversales o entrecruzadas.

- Relación incumbencia-Aspecto candidato: estos afectan a las incumbencias que tienen que ver con solapamiento: overlapping, overriding y wrapping. En los diagramas de casos de uso, se usa la relación del «include» clásico.

En la tabla 1 seleccionamos los elementos del Core UML para el DSOA, agrupados por las disciplinas del proceso del desarrollo de requisitos, análisis y diseño. Note que los elementos de modelación seleccionados son particularmente útiles en el diseño de la arquitectura. En este sentido la noción de componente, que representa incumbencia, aspecto y tejido, se utiliza en el diagrama de componentes, y serán tratados en la sección siguiente.

CASO DE ESTUDIO: MODELANDO UNA ARQUITECTURA INICIAL PARA EL PORTAL CADENA DE CINES

El Core UML para el DSOA será utilizado para construir una arquitectura inicial del problema en estudio. Se desarrollarán la vista de despliegue, los modelos de caso de uso y componentes en la disciplina de análisis.

El problema

Una cadena de cines solicita un sistema que proporcione el uso masivo de funcionalidades para el acceso público. Debe proporcionar también la facilidad para promover los diversos cines de la cadena. Principalmente se deben considerar en el sistema: reservación de boleto, pago, consulta y cancelación en línea, registro del usuario, transacciones estadísticas.

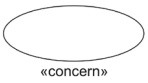
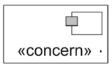
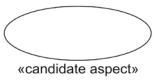
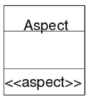
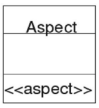
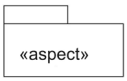
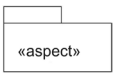
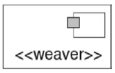
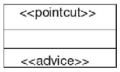
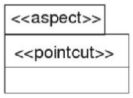
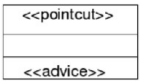
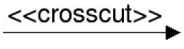
La solución

Se propone un portal, con un navegador comercial y facilidad de acceso a internet.

Análisis del dominio para la aplicación portal cadena de cines

Un portal es una aplicación web principalmente transaccional con servicios de portal. En este tipo de aplicación, se identifican en general las incumbencias no funcionales (requisitos de calidad). Por otra parte, los principales requisitos funcionales y el estilo arquitectónico en aplicación web de tipo transaccional y portal son conocidos (ver por ejemplo, Losavio *et al*. 2008). Los requisitos de calidad para

Tabla 1. Core UML para el DSOA.

Conceptos	Disciplinas			
	Requisitos	**Análisis**		**Diseño**
Concern	Diagrama de casos de uso	Diagrama de componentes		
				
Candidate Aspect (crosscuting concern)	Diagrama de casos de uso			
				
Aspect		Diagrama de componentes	Diagrama de clases de análisis	Diagrama de colaboración	Diagrama de clases de diseño
					
Composition		Diagrama de Paquetes			Diagrama de Paquetes
					
Weaving		Diagrama de componentes			
					
Join point	«match point»				Diagrama de clases, secuencia y estado
					
Advice		Diagrama de Paquetes			Diagrama de clases de diseño
					
Pointcut	Diagrama de casos de uso	Diagrama de Paquetes			Diagrama de clases de diseño
					
core class					Diagrama de clases de diseño
					
relations concern–candidate aspect	Diagrama de casos de uso				
relations class-aspect					Diagrama de clases de diseño
					

aplicaciones transaccionales son: funcionalidad (seguridad, precisión), reutilización (disponibilidad), eficiencia (tiempo de respuesta, uso de recurso). Para aplicaciones tipo portal son: Portabilidad (adaptabilidad, escalabilidad), Eficiencia (tiempo de respuesta, uso de recursos). El estilo es 3 capas, bajo modelo cliente- servidor.

Requisitos funcionales

Las principales funcionalidades de la aplica*ción portal cadena de cines* son observadas de la descripción del problema: intercambio de datos, control de acceso y encriptamiento, ya que en el portal las transacciones, consultas y el acceso son necesarias.

Arquitectura inicial

Una solución arquitectónica para nuestra aplicación es de 3 capas, el modelo cliente servidor, se considerará para garantizar la separación entre la Interfaz de usuario (cliente web) y la Base de datos (servidor de base de datos). La capa lógica de la aplicación (servidor de aplicaciones) media entre el cliente y la capa del servidor. La comunicación es asegurada por el protocolo TCP/IP.

En conclusión, la información obtenida del análisis del dominio es la siguiente: requisitos funcionales, requisitos de calidad y la arquitectura inicial.

Luego de plantear el problema, las siguientes incumbencias que corresponden a las funcionalidades del problema pueden ser identificadas: consultar páginas web del portal (operaciones de consulta, acceso), registrar usuario en línea (intercambio de datos, control de acceso, encriptamiento), reservar en línea el boleto (intercambio de datos y control de acceso), pago en línea del boleto (intercambio de datos, control de acceso y encriptamiento). Para construir el modelo de caso de uso, presentamos el diagrama de caso de uso usando el estereotipo de relación «*include*» para indicar la relación transversal (*crosscut*) (figura 2), incluyendo los requisitos de calidad como el estereotipo «*candidate aspect*» (Brito *et al*. 2003).

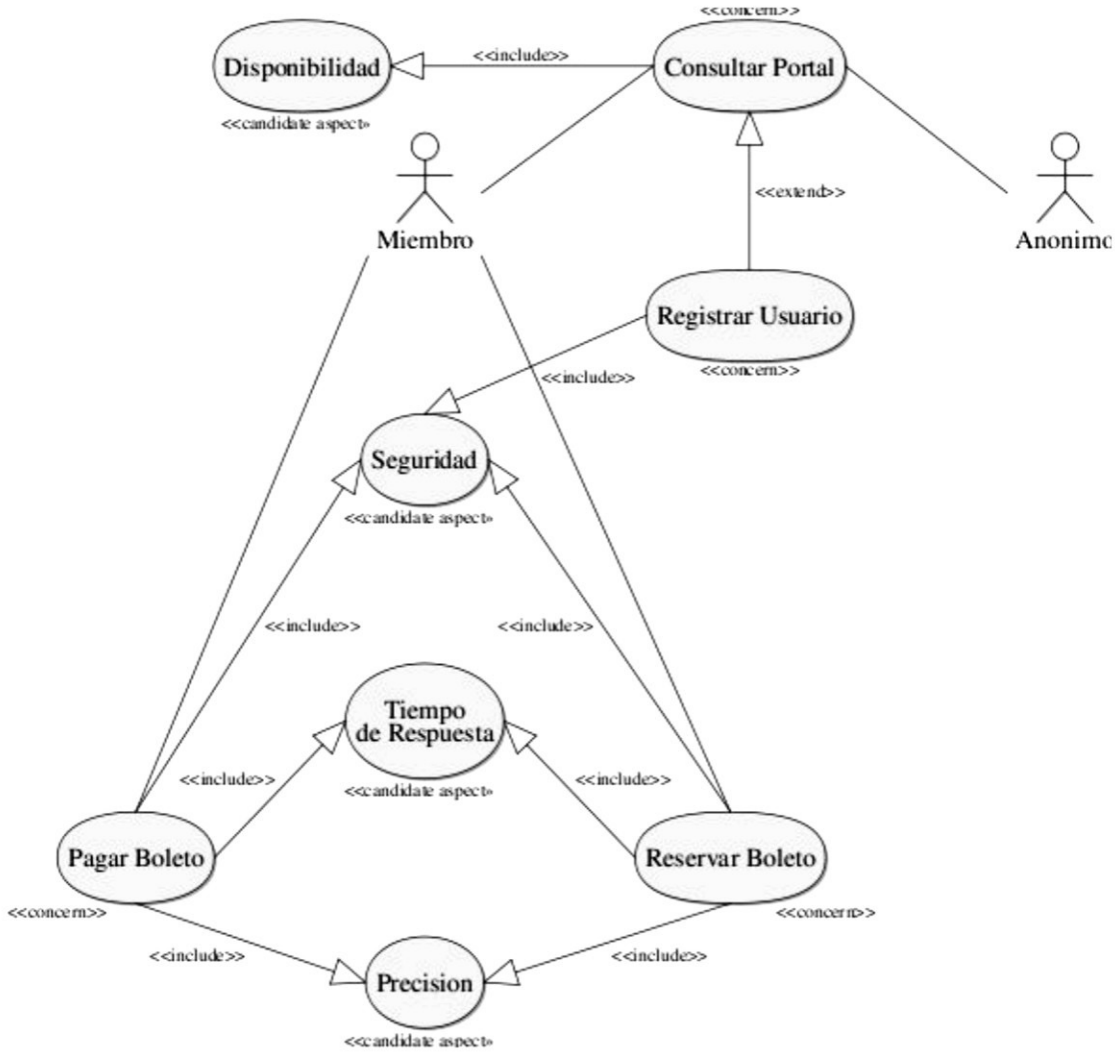


Figura 2. Diagrama de caso de uso para el portal cadena de cines.

Arquitectura para el portal cadena de cines

Ahora procedemos a construir una arquitectura inicial para la aplicación *portal cadena de cines*, en base a la información proporcionada por el análisis del dominio, del diagrama de caso de uso y la tabla de composición.

Configuración de la arquitectura inicial

Se obtiene directamente de la arquitectura inicial del análisis del dominio (figura 3). El servidor de aplicaciones soporta el componente lógica de la aplicación, donde se realiza el mayor esfuerzo de desarrollo y es modelado por el diagrama de caso de uso.

Refinamiento de la arquitectura

El componente Lógica de la aplicación se refina asignando un componente a cada caso de uso del diagrama de caso de uso (figura 2). Note que los componentes «concern» requieren componentes «aspect», según la notación del estándar de UML 2.0 para expresar el diagrama de componente (figura 4). De la tabla de composición (tabla 2), observamos que la "disponibilidad" es un aspecto candidato que corta (crosscuts) solamente la funcionalidad "consultar portal", por lo tanto se considera un componente «concern».

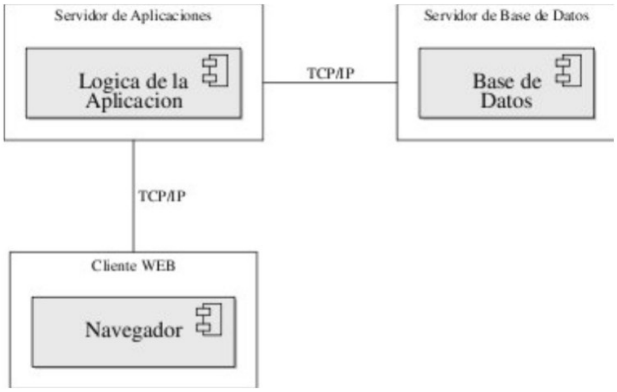


Figura 3. Arquitectura inicial de la aplicación portal cadena de cines: diagrama de despliegue.

Tabla 2. Identificación de puntos de composición.

Aspecto candidato	Consultar portal	Registrar usuario	Reservar boleto	Pagar boleto
Tiempo de respuesta			X	X
Precisión			X	X
Seguridad		X	X	X
Disponibilidad	X			

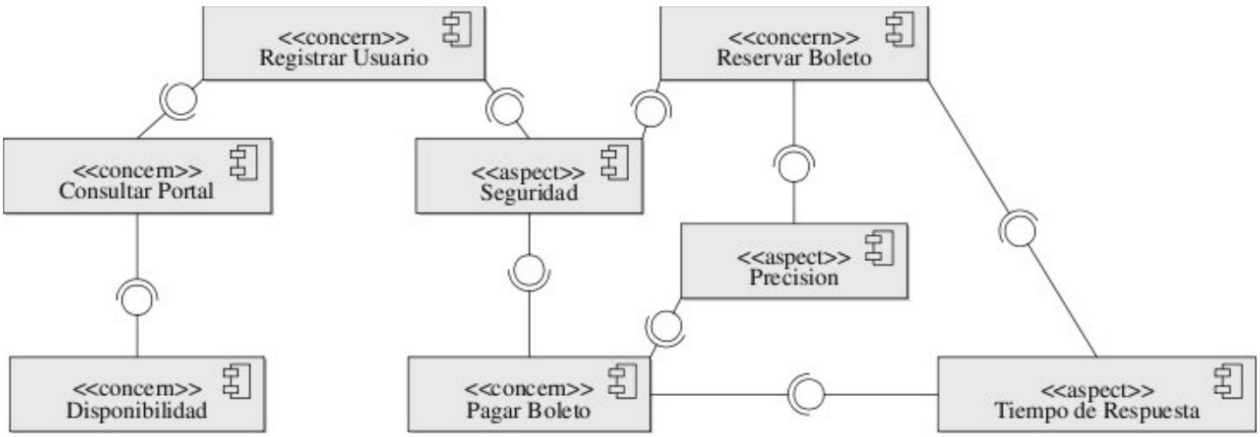


Figura 4. Refinamiento de la arquitectura, componente lógico: diagrama de componentes con incumbencias y aspectos.

Tejido de componentes

En la misma configuración, el proceso de tejido es representado usando el componente «weaver» (figura 5). Hemos elegido representar a un tejido para cada incumbencia transversal. Otra opción habría podido ser representar solamente un componente tejido, compuesto por los tres «weaver» mencionados.

Refinamientos futuros

En vista de que las incumbencias transversales se han identificado, podemos proceder a mirar con más detalles cada componente concern, usando técnicas de diseño arquitectónico conocidas. Es claro que podemos utilizar todos los estereotipos propuestos en el Core UML para el DSOA, con el fin de conseguir un diseño arquitectónico orientado a aspecto más detallado durante las disciplinas de análisis y diseño.

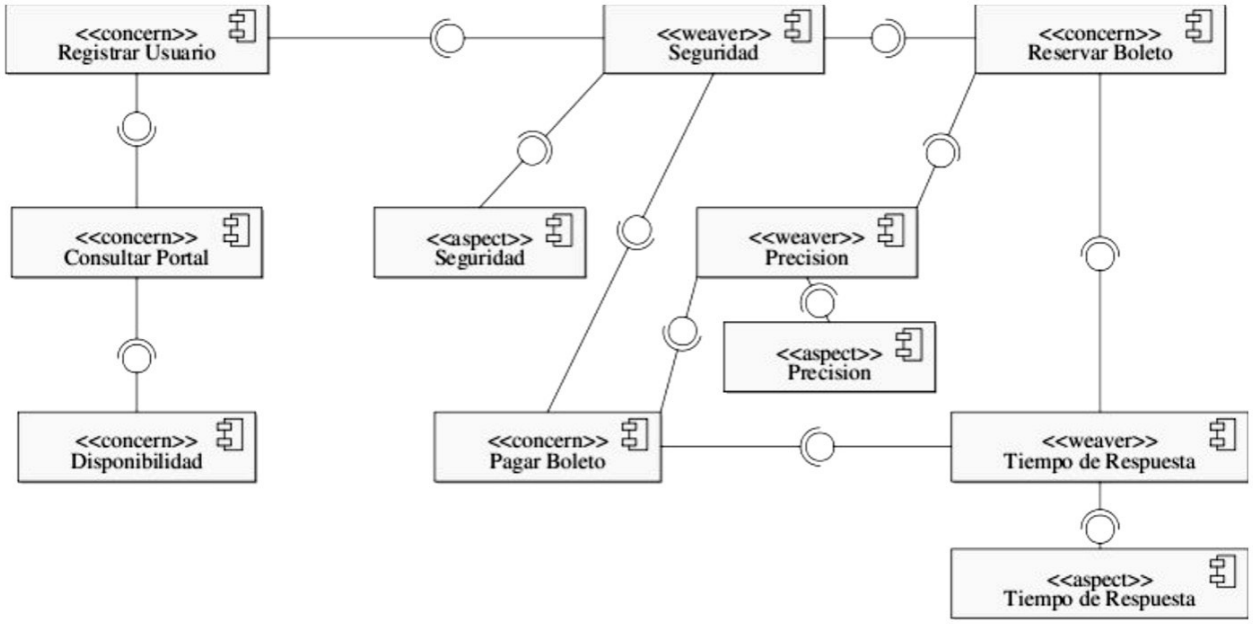


Figura 5. Refinamiento de la arquitectura, componente lógico: diagrama de tejido de componentes.

CONCLUSIÓN

El DSOA es un nuevo paradigma en la ingeniería del software, ahora considerado como una alternativa conveniente a la evolución del software y a la mejora de procesos de desarrollo de sistemas complejos modernos. Este trabajo ha estudiado los conceptos del documento "AOSD Ontology", de la Comunidad Europea, además de las numerosas extensiones UML para la orientación a aspectos propuestas por diversos autores. De este estudio, se obtiene un Core UML para el DSOA en etapas tempranos del desarrollo (disciplinas de requisitos, análisis y diseño), conformado por los elementos de modelación y sus estereotipos. Estos resultados contribuyen en general al establecimiento de los estándares para una terminología unificada en el DSOA, favoreciendo la usabilidad de las técnicas de diseño arquitectónico. El caso de estudio ilustra la aplicación del Core, para definir la arquitectura inicial de una aplicación web. Además este Core será utilizado en la definición de los modelos y transformaciones de modelos en el desarrollo de software orientado a aspecto. Estos resultados se han aplicado y adaptado en diversos proyectos académicos. Los trabajos futuros son revisar y mejorar este enfoque para definir un paso inicial que pueda ser finalmente incluido en un proceso de diseño arquitectónico.

AGRADECIMIENTO

Este trabajo es soportado por el Consejo de Desarrollo Científico y Humanístico (CDCH) de la Universidad Central de Venezuela, ADIRE Proyecto No. PG-03-7310-2008/1.

REFERENCIAS

Aldawud, O., Elrad, T., Bader, A. (2003). A uml profile for aspect oriented software development. Third International Workshop on AO Modeling Proceeding. s/n.

Bash, M. & Sánchez, A. (2003). Incorporating aspects into the uml. International Conference on Aspect-Oriented Software Development. s/n.

Berg, K., Conejero, J., Chitchyan, R. (2005). AOSD ontology 1.0 public ontology of aspect orientation. Fourth International Conference on Aspect – Oriented Software Development. p. 90. Recuperado el 4 de mayo de 2008 en: http://doc.utwente.nl/64104/.

Booch, G.., Rumbaugh, J., Jacobson, I. (1999). El Lenguaje Unificado de Modelado. Addison Wesley Iberoamericana, Madrid. s/n.

Brito, I. & Moreira, A. (2003). Towards a composition process for aspect-oriented requirements. Early Aspects: aspect-Oriented Requirements Engineering and Architecture Design, workshop of the AOSD. s/n.

Filman, R. (2005). A bibliography of aosd version 2.0. Recuperado el 2 de Julio de 2008 en: http://www.docstoc.com/docs/20096173/A-Bibliography-of-Aspect-Oriented-Software-Development-Version-20

Fuentes, L., Vallecillo, A. (2004). Una introducción a los perfiles de UML, Novática. 168; pp. 6-11.

Kaewkasi, C., Rivepiboon, W. (2003). Aspect-oriented extension for capturing requirements in use-case model. CAISE: The 15th Conference on Advanced Information Systems Engineering. CEUR. s/n.

Kiczales, G.., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J. (1997). Aspect-oriented programming. ECOOP'97 Proceedings LNCS 1241; pp. 220-242.

Krechetov, I., Tekinerdogan, B., García, A. (2006). Towards an integrated aspect-oriented modeling approach for software architecture design. Eng, I.T.S., ed.: DSOA 2006 13; pp. 336-355.

Losavio, F., Matteo, A., Rahamut, R. (2008). Characterization of web services domain based on quality standards. En Second European Conference, ECSA. Computer Science Springer LNCS. 5292; pp. 354-358.

Moreira, A. & Brito, J. (2002). Crosscutting quality attributes for requirements engineering. 14th International Conference on Software Engineering and Knowledge Engineering ACM Press: 167–174.

OMG (2003). UML 2.0 Infrastructure Specification. OMG document ptc/03-09-15.

Rashid, A., Sawyer, P., Moreira, A., Araujo, J. (2002). Early aspects: a model for aspect-oriented requirement engineering. IEEE Joint Conference on Requirements Engineering Proceedings. pp. 199-202.

Sutton, S. & Tarr, P. (2002). Aspect-oriented design needs concern modeling. Aspect Oriented Design workshop in conjunction with AOSD. s/n.

Zakaria, A., Hosny, H., Zeid, A. (2002). A uml extension for modeling aspect-oriented systems. UML Workshop on Aspect Oriented Modeling. s/n.

Zhang, G. (2005). Toward aspect-oriented class diagram. 12th Asia-Pacific Software Engineering Conf. (APSEC'05). pp. 763-768.