Revista Española de Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 4, No. 1, abril, 2008

Web de la editorial: www.ati.es
E-mail: editor-reicis@ati.es
ISSN: 1885-4486

Publicado por la Asociación de Técnicos en Informática

Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS)

Contenidos

REICIS

Editorial

REICIS

En este número de abril de 2008 de REICIS, un nuevo ámbito de la calidad del software es abordado en nuestra sección de contribuciones invitadas: el software en el ámbito de la defensa. Para ello, contamos con el comandante José Gonzalo Delgado De Luque que es el director de proyectos informáticos dentro del CDS (Centro de Desarrollo de Sistemas) del ATI (Área del Tratamiento de la Información) de la SEGENTE (Secretaría General Técnica) del Ministerio de Defensa. En su contribución nos informa de los controles planteados desde el ámbito ministerial para la selección y la calidad de los productos de software que soportarán los distintos ámbitos de las fuerzas armadas.

Por otra parte, la sección regular de artículos se nutre de dos trabajos remitidos directamente por sus autores a la revista. En ellos se tratan importantes aspectos de la ingeniería para el desarrollo de software tanto en su ámbito de gestión de proyectos como en los propios paradigmas de construcción de sistemas En este sentido, la revista continua invitando desde estas líneas a todos los profesionales e investigadores relacionados con el mundo de la Innovación, Calidad e Ingeniería del Software a que utilicen REICIS como el medio para dar a conocer sus trabajos e investigaciones teniendo las máximas garantías de la profesionalidad con que serán tratados sus trabajos. Podrán encontrar todas las instrucciones necesarias para el envío de sus contribuciones en la página web de la revista: www.ati.es/reicis.

Luis Fernández Sanz
Juan J. Cuadrado-Gallego
Editores

Presentación

REICIS

Este primer número de REICIS del año 2008 publica, tras el proceso de revisión de nuestro comité editorial, dos contribuciones directamente recibidas en nuestras revista.

El primero de los trabajos publicados corresponde al titulado "Una ontología para la gestión del conocimiento de proyectos software" cuyos autores son Francisco J. Ruiz Bertol de la Universidad de Zaragoza y Javier Dolado de la Universidad del País Vasco. En este artículo, se presenta una ontología de gestión de proyectos apoyada en las descripciones incluidas en el PMBOK (cuerpo de conocimientos del Project Management Institute) y que trata de facilitar la comprensión de los distintos conceptos y las relaciones entre los mismos.

El segundo trabajo se titula "Orientación a aspectos en UML2 sin extensiones" y ha sido elaborado por Pilar Romay (de la Universidad Europea de Madrid) en colaboración con Carlos Cuesta y Marcos López de la Universidad Rey Juan Carlos de Madrid. Este trabajo aborda el enfoque de desarrollo basado en aspectos y analiza el soporte que la notación UML 2 proporciona para el uso de este paradigma de construcción de sistemas.

Finalmente, en la columna de Actualidad Invitada, es el comandante José Gonzalo Delgado De Luque, quien desde su posición de director de proyectos informáticos dentro del CDS (Centro de Desarrollo de Sistemas) del ATI (Área del Tratamiento de la Información) de la SEGENTE (Secretaría General Técnica) del Ministerio de Defensa, nos desvela el proceso de selección de productos software para las fuerzas armadas de España, con la correspondiente referencia a las distintas instrucciones y disposiciones que regulan el control y la estrategia tecnológica en este ámbito. Así mismo, se menciona el concepto de Arquitectura Técnica Unificada adoptado para el soporte de los procesos de control y selección de software.

Luis Fernández Sanz

Una ontología para la gestión del conocimiento de proyectos software

Francisco Javier Ruiz Bertol
Universidad de Zaragoza
franjr@unizar.es
Javier Dolado
Universidad del País Vasco / Euskal Herriko Unibertsitatea
javier.dolado@ehu.es

Abstract

Project Management has been for years a knowledge area restricted to experts and professionals in management. Currently, this expertise and practice has been recorded in several books and articles, and implemented in software systems and databases. Due to the increasing interest in the Semantic Web, we have sufficient tools, techniques and skills to reflect that information, giving an additional semantic feature, by using knowledge representation. The most adequate way to specify that knowledge is by using domain ontologies that let us to express terms, concepts, properties and relations for a given domain using an ontological language. In this article, we propose the Project Management Ontology (PMO), a set of ontologies that capture and store that knowledge. PMO is constructed in a modular way, so that new ontologies can be joined to PMO to enrich the project management knowledge, and also new ontologies can be built by merging them with other knowledge areas ones.

Resumen

La gestión de proyectos ha sido durante muchos años un área de conocimiento reservada a expertos y profesionales. Sin embargo, dicha experiencia está siendo recopilada en multitud de artículos y libros, e implementada en varios sistemas software a la que cualquier puede acceder. Debido al creciente interés en las tecnologías informáticas para la gestión del conocimiento, actualmente disponemos de suficientes técnicas, herramientas, y habilidades para poder desarrollar una representación del conocimiento de la gestión de proyectos, añadiendo todas las características inherentes a los sistemas basados en el conocimiento. En este artículo, se presenta *Project Management Ontology* (PMO), una ontología de dominio, que recoge tanto la estructura común para la gestión de proyectos, como la información asociada para poblar la ontología. PMO se ha creado de forma modular, permitiendo aplicar la gestión de proyectos a distintas áreas de conocimiento, a través de la unión de ontologías.

Palabras clave: Gestión de proyectos, ontologías de dominio, gestión del conocimiento, ingeniería del software, desarrollo de ontologías.

1. Introducción

En la actualidad existe un creciente interés sobre una gestión del conocimiento adecuada en las distintas áreas de conocimiento. De hecho, en el desarrollo software muy importante gestionar adecuadamente dicho conocimiento, tanto el que se da de manera explícita como el tácito o implícito. El primero se define como "*una herramienta de gestión para aprovechar la manipulación del conocimiento de la organización, groupwares, intranets, servidores de listas, repositorios de conocimiento, gestión de bases de datos y redes de acción del conocimiento permiten compartir la dicha gestión del conocimiento*" [1]. El segundo, más difícil de capturar, se basa más bien en la experiencia, está guiado por el contexto, y por lo general, reside en los individuos.

En los proyectos software, dicho conocimiento es fundamental por su influencia en los distintos componentes: registro histórico, lecciones aprendidas, explotación de datos, toma de decisiones, seguimiento del proyecto, metodologías utilizadas, estimación y planificación, asignación de recursos, etc. Por ello, es necesario capturar y gestionar el conocimiento disponible en un formato y representación adecuados. Esto puede realizarse para la mayoría de áreas de conocimiento por los expertos, donde pueden definirse un conjunto de conceptos, aserciones, reglas e inferencias sobre dicha información. Esta información puede guardarse utilizando alguna de las representaciones de conocimiento existentes.

Las ontologías han sido utilizadas de manera intensiva en el pasado para proporcionar un lenguaje común comprensible por los usuarios para alcanzar un consenso sobre varios temas, incluyendo el conjunto de conceptos y relaciones entre conceptos a manejar, la clasificación de entidades, y que proporcionen una abstracción del mundo real. Pero las ontologías también proporcionan un lenguaje comprensible por los computadores para la representación de dichos conceptos, entidades, relaciones y abstracciones para facilitar la interoperabilidad y el intercambio de información sobre el conocimiento [2].

Para determinar dicho conocimiento, las ontologías, que proporcionan "una especificación formal y explícita de una conceptualización compartida" [3], aportan una representación declarativa de conceptos, estructuras de datos, relaciones, aserciones, reglas y restricciones que representan un modelo abstracto y simplificado de la realidad. Las ontologías se expresan comúnmente utilizando ontologías específicas de dominios. Estas

ontologías tienen dos características: proporcionan una representación explícita de un modelo conceptual sobre un dominio determinado; y dicho modelo establece una representación compartida y consensuada sobre el conocimiento para dicho dominio.

En la actualidad se han creado varias ontologías de dominio y sistemas basados en el conocimiento para representar el conocimiento en un dominio dado, y que abarcan un amplio conjunto de áreas de conocimiento en diversos dominios.

Una de las ontologías que más se acerca al dominio de este artículo es Onto-SWEBOK [4], basada en los trabajos de Abran *et al.* [5], cuyo objetivo es facilitar una meta-descripción de los conceptos expresados en el SWEBOK [6] utilizando una estructura ontológica por capas expresada mediante los conceptos descritos en la base de conocimiento *OpenCyc* [7].

Otros trabajos de interés sobre el desarrollo de ontologías pueden encontrarse en varios repositorios de ontologías [8][9][10]. Sin embargo, llama la atención la ausencia de representaciones formales y explícitas para la gestión de proyectos, independientemente del dominio o área de conocimiento.

En este artículo se presenta *Project Management Ontology* (PMO), una ontología de dominio que representa un modelo formal de los procesos, actividades, herramientas y técnicas específicas de la gestión de proyectos. PMO proporciona una descripción completa de los términos fundamentales y características inherentes al manejo de la información asociada a la gestión, seguimiento, control y dirección de los proyectos, así como de los procesos, relaciones, restricciones y aserciones sobre los datos de proyectos.

El artículo está dividido de la siguiente manera: la sección 2 describe *Project Management Ontology* (PMO), detallando las principales características de esta ontología, el proceso de desarrollo y sus componentes. En la sección 3 se explica cómo PMO puede ser integrada con otras áreas de conocimiento utilizando técnicas de mapeado y fusión. En la sección 4 se exponen las conclusiones y el trabajo futuro.

2. *Project Management Ontology* (PMO)

La gestión de proyectos se define como el conjunto de herramientas, técnicas, conocimiento y habilidades aplicadas a un proyecto para cumplir un conjunto de requisitos, estándares, especificaciones y objetivos que llevan a completar dicho proyecto. La gestión por

proyectos se llevan a cabo en varios ámbitos, incluyendo entre ellos a su aplicación en arquitectura e ingeniería, industria química, desarrollo software o en el ámbito de la investigación. En todos estos ámbitos, es de vital importancia la gestión de proyectos, ya que proporciona una documentación exhaustiva en el área de la gestión de proyectos que ha sido desarrollada por expertos en el área y en áreas multidisciplinares adyacentes.

Para gestionar los proyectos, se dispone de varias herramientas centradas en capturar una parte de la información del proyecto. Sin embargo, estas herramientas se utilizan de una manera aislada (válida únicamente para las personas que utilizan dichas herramientas), o como un conjunto integrado de aplicaciones. Por lo tanto, es complicado compartir este conocimiento con otras organizaciones o incluso, dentro de una misma organización que utilice distintas aplicaciones para la gestión de proyectos, incluso si existe la posibilidad de importar/exportar la información. Por ello, es necesario definir una representación del proyecto que pueda ser utilizada por dichas aplicaciones, pero que también permita la interoperabilidad entre éstas. En este sentido, se considera necesario establecer una representación del conocimiento que permita capturar y gestionar adecuadamente la información del proyecto. Para este propósito, lo más adecuado es la utilización de ontologías de dominio.

Para desarrollar esta ontología, ha sido necesario plantearse las distintas opciones para capturar dicho conocimiento: (i) hablar con expertos en la gestión de proyectos, (ii) capturar el conocimiento directamente de las aplicaciones de gestión de proyectos, o bien, (iii) obtener de alguna fuente de información un modelo descriptivo y consensuado del conocimiento. La primera opción es difícil de lograr, ya que parte de dicho conocimiento es tácito y por lo tanto, no hay forma de modelarlo adecuadamente. En la segunda opción, el modelo de desarrollo seguido por los desarrolladores está centrado en la aplicación, por lo que el modelo de datos únicamente tiene los parámetros suficientes para que la aplicación funcione, sin que esté definido el conocimiento para la gestión de proyectos de una manera completa. Finalmente, para la tercera opción, existe una gran cantidad de literatura, tanto en forma de libros como en forma de artículos de investigación. Entre todos ellos, se ha seleccionado *Project Management Body of Knowledge*® (PMBOK) [11], ya que proporciona una documentación exhaustiva en el área de la gestión de proyectos que ha sido desarrollado por expertos en el área y en áreas adyacentes.

Para capturar y gestionar dicho conocimiento, se ha desarrollado *Project Management Ontology* (PMO), un conjunto de ontologías que abarcan los principales procesos, conceptos y relaciones de la gestión de proyectos. Este sistema basado en el conocimiento proporciona la primera representación formal de conocimiento en el dominio de la gestión de proyectos. Para su desarrollo, ha sido fundamental desarrollarlo de una manera modular y estructurada, de forma que otras ontologías o sistemas basados en el conocimiento puedan unirse a PMO para crear una ontología específica de la gestión de proyectos en el dominio aplicado. PMO está compuesta de:

- Una taxonomía que define la estructura de un proyecto. Esta taxonomía proporciona una jerarquía basándose en la estructuración de los proyectos, así como en la definición de los principales términos. Esta taxonomía está formada principalmente por relaciones del tipo *isA* (subclases) o *has* (composición), que define esencialmente las partes en que se puede dividir y estructurar un proyecto y sus componentes.
- Un completo vocabulario que define conceptos específicos de la gestión de proyectos, y que pueden ser aplicables a la mayoría de áreas de conocimiento. Este vocabulario ha sido obtenido del glosario de términos del PMBOK [11].
- Un sistema basado en el conocimiento (KBS) que contiene el conocimiento experto de cómo gestionar un proyecto de manera adecuada. Esto incluye la estructura, el contenido semántico, y los procesos e instancias necesarias que proporcionan una guía para la dirección.
- Un conjunto de *slots* o propiedades asociadas a uno o varios de los conceptos presentes en la taxonomía o el vocabulario.
- El conjunto de relaciones entre conceptos definidos tanto a nivel de taxonomía, como entre los distintos componentes de PMO (utilizando las propiedades *owl:equivalentClass* y *owl:sameAs*). Estas relaciones están basadas en la propia definición de los conceptos.

Se puede observar la estructura básica de PMO en la Figura 1. Cada uno de los componentes mostrados en esta figura representa una ontología. En PMO actualmente se

han definido cinco ontologías: *PM-Cost*, *PM-Process*, *PM-Planning*, *PM-Organization*, y la ontología núcleo *PM-Core*.

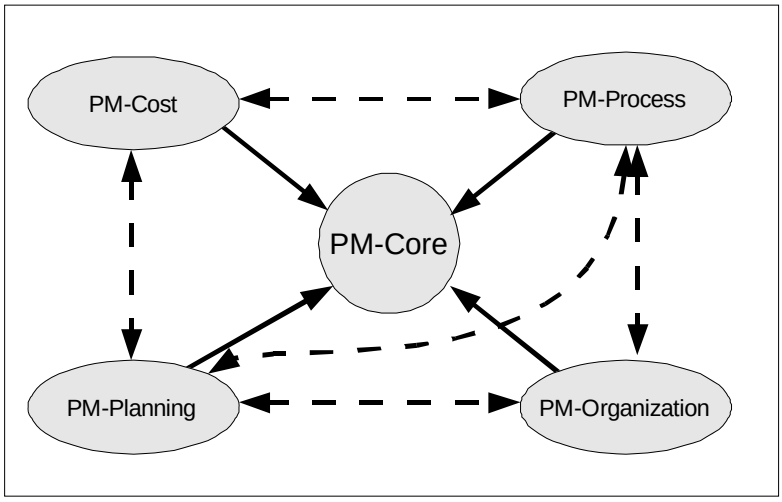

Figura 1. Ontologías componentes de *Project Management Ontology*.

En la Figura 1, se pueden diferenciar dos tipos de relaciones: las indicadas con línea continua, y las indicadas con punteado. Las flechas continuas indican una correspondencia directa entre clases que representan un mismo concepto utilizando la propiedad de OWL *owl:sameAs*, como por ejemplo los conceptos que representan *Actividad* y *Paquete de Trabajo*. Las flechas punteadas representan conceptos relacionados entre sí a través de los conceptos definidos en *PM-Core*.

Para el proceso de desarrollo se ha seguido la metodología recomendada por Noy y McGuiness [12]. Este proceso de desarrollo consiste en un conjunto de pasos propuestos por los autores para la creación de ontologías de dominio: (i) determinación del dominio y ámbito de la ontología; (ii) consideración de reutilización de ontologías ya existentes; (iii) enumeración de los conceptos y términos clave; (iv) definición de la estructura de clases y atributos de cada una de las clases; (v) definición de las restricciones sobre los atributos; y (vi) completar la ontología, poblándola con instancias o individuos.

Para crear una vista gestionable del dominio de gestión de proyecto, se ha dividido PMO en varios componentes u ontologías, cada uno de ellos proporcionando una visión parcial de una parte del conocimiento en la gestión de proyectos. Esta estructura se ha definido en base a las distintas partes diferenciadas en las que se compone el área de

conocimiento (coste, planificación, riesgos, requisitos, aseguración de la calidad, etc.). PMO tiene los siguientes componentes:

- *PM-Core*. El conjunto de conceptos, relaciones, axiomas y atributos que forman la base para la gestión de proyectos. En esta ontología se incluyen conceptos como proyecto, fase, entregables, productos, o actividades.
- *PM-Process*. Esta ontología representa el conjunto de procesos y grupos de procesos de recomendada aplicación para guiar un proyecto. Esta ontología proporciona principalmente el conocimiento recogido en el PMBOK [11].
- *PM-Organization*. Esta ontología proporciona la estructura organizativa del proyecto, definiendo los conceptos de equipo, persona, atribuciones, habilidades, asignaciones, etc. El enfoque tomado para el desarrollo de esta ontología ha sido la división desde la perspectiva de la gestión para la organización que desarrolla el proyecto, el desarrollo de los equipos y los actores que forman parte del mismo.
- *PM-Cost*. Esta ontología incluye todos aquellos conceptos, atributos y relaciones asociados a la gestión de costes, tanto monetarios como de esfuerzo, así como conceptos relacionados con estimaciones y presupuestos.
- *PM-Planning*. Finalmente, esta ontología desarrolla toda la parte de gestión de la planificación, calendario y seguimiento de un proyecto.

Aún están en proceso de desarrollo ontologías adicionales que completen las partes de conocimiento no incluidas en este primer desarrollo de PMO, ya que se consideraron en un primer momento como secundarias en la definición del domino. Entre estas partes del conocimiento no desarrolladas están los riesgos, los requisitos, la aseguración de la calidad y la gestión de las comunicaciones. Una vez desarrolladas este conjunto secundario de ontologías pasarán a formar parte de PMO, utilizando la unión de ontologías, con el objetivo de completar el conocimiento del dominio.

2.1. Ontología núcleo: *PM-Core*

PM-Core es la principal ontología de PMO. Todas las demás ontologías en PMO están directamente relacionadas con *PM-Core*, como se puede observar en la Figura 1. *PM-Core* contiene el vocabulario y la estructura básica para la definición de un proyecto genérico, en

cualquier ámbito de aplicación. El principal concepto definido en la ontología es el *Proyecto*, que tiene asociado una serie de atributos inherentes, como su nombre, descripción, procesos a utilizar, criterios de calidad o hitos. Así mismo, un proyecto puede ser parte de un *Programa* o un *Portafolio*. En el proceso de división hacia artefactos más manejables, se pueden definir distintos tipos de *Componentes de proyecto* (clase abstracta): *Fases* (o subfases), *Entregables*, o *Paquetes de trabajo*, siguiendo las recomendaciones de estructuración del trabajo del proyecto [13]. Cada uno de estos componentes puede combinarse entre sí mediante las relaciones definidas en la propia ontología, para formar la *Estructura de Descomposición del Trabajo*.

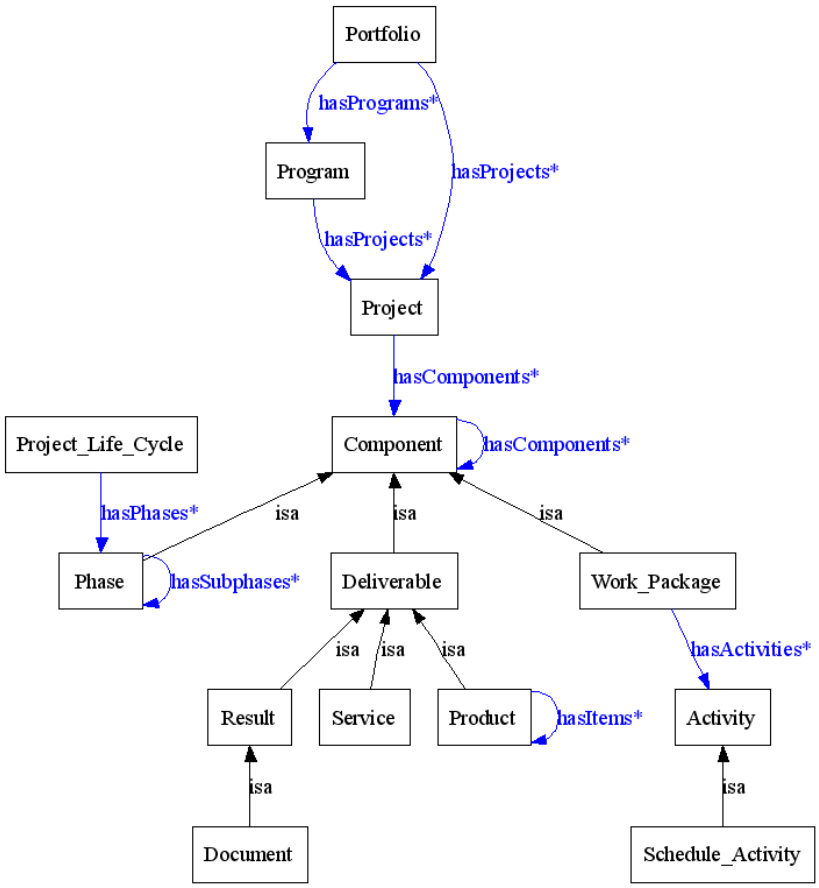

Figura 2. Vista simplificada de la jerarquía de clases de *PM-Core*.

En la Figura 2 se puede observar la jerarquía de clases definida en *PM-Core*. Según la definición [13], "*una Estructura de Descomposición del Trabajo es una descomposición jerárquica orientada a ser entregada del trabajo a ser ejecutado por el equipo del proyecto para cumplir los objetivos del proyecto y crear los entregables solicitados*". Los elementos

terminales son principalmente entregables o paquetes de trabajo, que se dividen a su vez en actividades. Las actividades representan los elementos gestionables más pequeños que pueden asignarse directamente a un equipo del proyecto o una persona.

2.2. Ontología de procesos de gestión: *PM-Process*

PM-Process es la ontología que contiene el modelo de procesos de gestión del PMBOK [11]. En esta ontología se incluyen aquellos grupos de procesos en los que está dividido el cuerpo del conocimiento en la gestión del proyecto: procesos de iniciación, planificación, ejecución, control y monitorización, y finalización.

Estos grupos de procesos son necesarios para cualquier proyecto, ya que proporciona un proceso guiado, siendo necesario que se ejecuten siguiendo la misma secuencia para todos los proyectos, independientemente del ciclo de vida utilizado o el proceso de desarrollo software seleccionado. Cada uno de estos grupos de procesos contiene una serie de procesos de gestión interrelacionados entre sí, tal y como se define en el PMBOK. Esto incluye una descripción completa del proceso de gestión, así como las entradas, salidas, herramientas y técnicas necesarias para llevar a cabo dicho proceso. El equipo de gestión del proyecto será el encargado de seleccionar el subconjunto adecuado de estos procesos para llevar a cabo el conjunto de actividades del proyecto.

Sin embargo, *PM-Process* no está restringido a la definición de una serie de clases, relaciones y propiedades para almacenar el conocimiento sobre los procesos de gestión, sino que también proporciona un sistema basado en el conocimiento, ya que la ontología se ha completado con un conjunto de instancias que definen los procesos con la información obtenida del PMBOK [11].

Este conocimiento tiene el objetivo de proporcionar una guía a la hora de definir el conjunto de actividades presentes en *PM-Core*, pero también de adecuar la gestión de estas actividades al proyecto específico. Esta ontología representa únicamente los procesos de gestión, por lo que no incluye otros procesos que no son específicamente procesos de gestión, por ejemplo, procesos software, que podrían estar definidos en otra ontología.

2.3. Ontología de organización: *PM-Organization*

En un sentido estricto, la organización y los recursos humanos no es una parte constituyente de la gestión de proyectos, aunque sí que es una pieza fundamental para el éxito del

proyecto. De hecho, algunos procesos de gestión están directa o indirectamente relacionados a la organización (por ejemplo, todos aquellos procesos de la Gestión de Recursos Humanos del Proyecto presentes en el PMBOK).

Durante el desarrollo de esta ontología se ha utilizado parte el trabajo ya desarrollado en las ontologías de organización disponibles en [14], que describen las principales características de una organización, incluyendo metas, jerarquía, roles, puestos desempeñados, personas, equipos, etc. La utilización del conocimiento disponible en estas ontologías ha sido de gran utilidad para crear y adaptarlas en *PM-Organization* a un enfoque de gestión de proyectos.

En *PM-Organization* se han definido los conceptos básicos de una organización, como organización, persona, empleado, puesto o habilidades, y se ha extendido añadiendo conceptos más relacionados a la gestión del proyectos, como equipo de proyecto, miembro del equipo, gestor/director o equipo virtual. Finalmente se han redefinido o agregado nuevos términos conducentes a su adaptación a las actividades de gestión, como por ejemplo, habilidades de gestión, responsabilidades de gestión, competencias o comunicaciones.

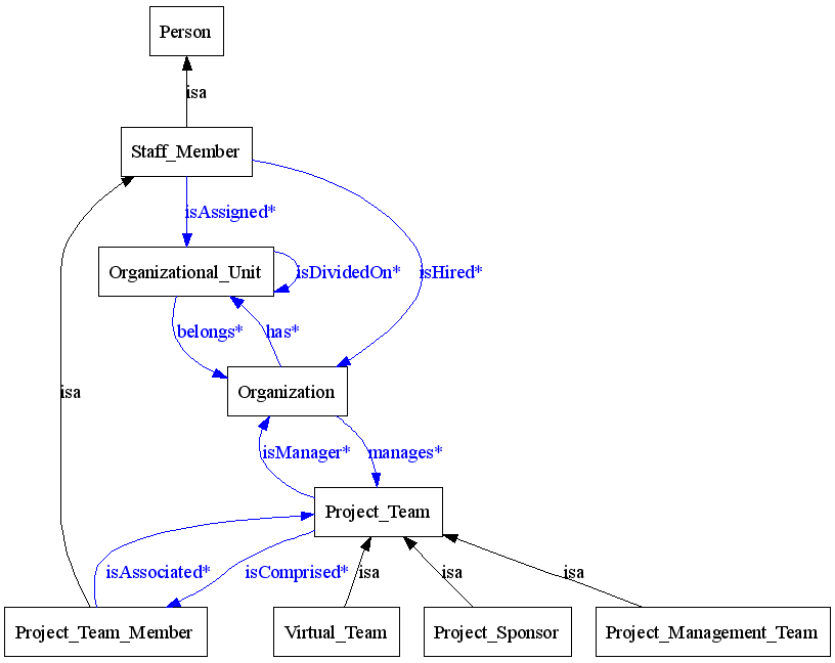

Figura 3. Estructura de la jerarquía de clases de *PM-Organization*.

Esta ontología es una base del conocimiento que puede unirse a *PM-Core* para proporcionar una definición completa del proyecto, tanto desde el punto de vista estructural como desde el organizativo. La estructura de la taxonomía asociada a *PM-Organization* puede observarse en la Figura 3.

2.4. Ontología de planificación: *PM-Planning*

La ontología *PM-Planning* contiene el conocimiento necesario para la representación de la planificación, calendario, estimaciones, asignación, seguimiento, control, gestión de cambios y finalización de las actividades del proyecto.

La unidad utilizada en esta ontología es principalmente el tiempo, aunque también están consideradas otras unidades de medida, como los recursos y el esfuerzo. Debido a que ya existen representaciones del tiempo en forma de ontologías, se ha considerado conveniente la utilización de *OWL Time Ontology* [15] para ayudar al desarrollo de la ontología *PM-Planning*. De hecho, todas las medidas de duración e intervalos de tiempo han sido definidas en función de esta ontología. En *PM-Planning* se pueden diferenciar dos tipos de medidas:

- Las propiedades que están directamente relacionadas con la planificación. Entre estas propiedades se incluyen aquellos valores necesarios para las estimaciones y seguimiento del proyecto, como por ejemplo, *Fecha de Inicio Temprana*, *Fecha de Inicio Tardía*, *Fecha de Finalización Temprana* y *Fecha de Finalización Tardía*, utilizados en el Método del Camino Crítico [16], o simplemente otras propiedades directamente asociadas al cumplimiento de plazos de las actividades (por ejemplo, *Fecha Actual*, *Fecha de Finalización Actual*, *Fecha de Finalización Estimada*, etc.).
- Otros conceptos que definen estructuras de planificación y estimación más complejas como las Líneas de Base.

En *PM-Planning* se ha decidido incluir la definición temporal para distintos husos horarios y calendarios, ya que los proyectos, especialmente aquellos que trabajan con equipos virtuales, son desarrollados generalmente en distintas localizaciones. *PM-Planning* está muy relacionada con *PM-Cost* a través de *PM-Core*, ya que ambas comparten varios

de los términos de alto nivel, como por ejemplo, la *Línea de Base* o la *Secuencia del Camino Crítico*, entre otros.

2.4. Ontología de estimación de costes: *PM-Planning*

El componente *PM-Cost* representa el conjunto de conceptos relativos al presupuesto, planificación, estimación y control de los costes. Se ha diferenciado *PM-Cost* de *PM-Planning* ya que ambos componentes representan diferentes términos: el coste del proyecto determina generalmente un valor monetario del proyecto, mientras que el calendario engloba los aspectos temporales del proyecto, comúnmente más utilizados para medir la progresión del proyecto. De hecho, las organizaciones generalmente tienen dos aplicaciones separadas para las estimaciones de coste/presupuesto y el calendario. Sin embargo, ambas ontologías determinan una información fundamental en la gestión de proyectos.

De hecho, determinar buenas estimaciones del presupuesto y del calendario son factores fundamentales para el éxito del proyecto [17].

La ontología *PM-Cost* está principalmente expresada en términos de coste monetario dada una cierta moneda (por ejemplo, si el proyecto se está realizando conjuntamente entre Europa y EE.UU., los equipos del proyecto trabajarán tanto en Euros como en Dólares).

Debido a que el proyecto generalmente está supeditado a un presupuesto definido, las unidades a utilizar para expresar el coste no deberían cambiar durante dicho proyecto, o al menos no deberían estar sujetas a los cambios monetarios que se produzcan. Este hecho está capturado en la ontología, donde se establecen distintas unidades monetarias, y un tipo de datos con funciones embebidas para hacer transparente el coste del proyecto.

3. Integración de *Project Management Ontology*

En la actualidad, una de las principales características de las ontologías es facilitar la interoperabilidad con otra información, ontologías o aplicaciones. Durante el proceso de desarrollo de PMO, se ha decidido desarrollar una definición genérica del área de conocimiento de la gestión de proyectos, de tal manera que pueda hacerse extensible a otras áreas de conocimiento diferentes. De hecho, se puede afirmar que esta característica es fundamental para la integración del conocimiento de la gestión de proyectos con otras áreas de conocimiento, como la ingeniería del software o la arquitectura. Así, PMO ha sido

desarrollada utilizando un enfoque modular, para poder facilitar la integración con otras formas de conocimiento (ontologías, bases de datos, sistemas de gestión del conocimiento, aplicaciones, etc.).

Por una parte, todos los componentes de PMO están altamente integrados. Este hecho puede observarse en la Figura 1, donde las dependencias entre todos los componentes de PMO están relacionados de una u otra manera. Por ejemplo, la ontología *PM-Process* puede ser fácilmente definida independientemente de las demás ontologías, pero los conceptos y propiedades presentes en ella están fuertemente relacionados en la manera de distribuir el trabajo al equipo del proyecto (presente en *PM-Organization*). De la misma manera, se pueden encontrar relaciones similares en las demás ontologías de PMO. Actualmente, se está desarrollando nuevos componentes para capturar mayor información en áreas asociadas a la gestión de proyectos, como los riesgos, las comunicaciones o la calidad del proyecto.

Por otra parte, es recomendable integrar este sistema basado en el conocimiento con otras ontologías existentes, para extender y enriquecer el ámbito de actuación de PMO. Esto se puede realizar mediante procesos de unión y mapeado de ontologías [18]. Supongamos que se desea unir PMO con una ontología que modela el conocimiento en Ingeniería del Software. Se puede realizar un mapeado de ambas ontologías siempre que los conceptos estén descritos de similar manera o se puedan encontrar conceptos equivalentes en ambas ontologías. Por ejemplo, en *PM-Process* se ha definido el concepto de *Proceso*, que tiene una subclase denominada *Procesos de Gestión*, que especifica los procesos que se recomiendan para una gestión eficiente de los proyectos. En otra ontología, asociada al conocimiento en la Ingeniería del Software, es altamente probable que nos encontremos con una serie de conceptos que definan los procesos software. Estos procesos software dependerán del ciclo de vida seleccionado, pero también de las políticas aplicadas en la organización, la experiencia del gestor del proyecto, etc. Se puede afirmar que si dicha ontología o representación de conocimiento existe, podría definirse una mapeado entre ambas ontologías para la unión del conocimiento en la gestión de proyectos al desarrollo del proyecto software.

En la Figura 4, dadas dos ontologías, PMO y otra que represente el conocimiento sobre los procesos software, ambas pueden ser unidas utilizando un mapeado del

conocimiento. De hecho, en la actualidad existen varios trabajos abiertos sobre el mapeado de ontologías [18], que incluyen entornos de trabajo para el mapeado, métodos, herramientas, traductores, mediadores y técnicas.

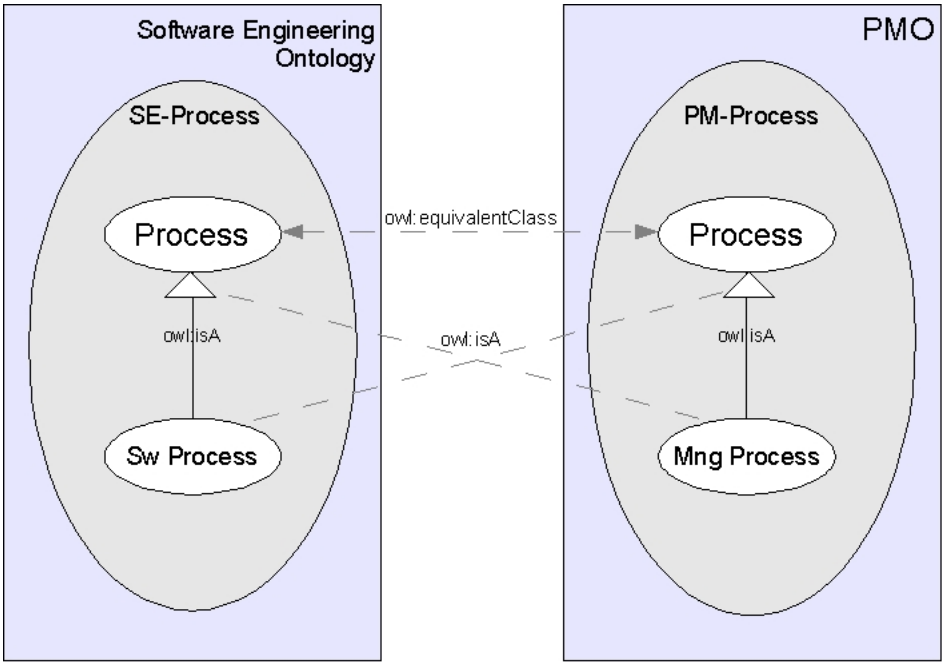

Figura 3. Un ejemplo de mapeado de ontologías para dos ontologías de áreas de conocimiento diferentes.

4. Conclusiones

En este artículo se ha presentado *Project Management Ontology*, una ontología basada en el contenido del PMBOK que presenta los principales conceptos y relaciones. Esta representación del conocimiento proporciona una visión semántica de la información del proyecto utilizando el enfoque de gestión.

Este trabajo propone un conjunto de ontologías básicas que son necesarias para comenzar a capturar y salvar la información de gestión de los proyectos en un formato independiente de las aplicaciones utilizadas, facilitando así mismo el proceso de interoperabilidad e intercambio de información. Aunque el proceso de desarrollo de ontologías es duro y bastante largo, la necesidad de proporcionar y obtener el conocimiento disponible sobre la gestión de proyectos es fundamental para gestionar de una manera cada

vez más eficiente. De hecho, en la actualidad, existe una gran cantidad de grupos de investigación dedicados a la gestión del conocimiento.

Así mismo, en este artículo se han mostrado las principales características de *Project Management Ontology* (PMO). PMO está compuesta de varias ontologías, donde cada una desarrolla una parte del conocimiento de la gestión de proyectos. Esta estructuración por partes de conocimiento, ha permitido trabajar individualmente con cada una de ellas (dividiendo el problema de la representación del conocimiento en partes más pequeñas), lo que ha facilitado su posterior proceso de integración. Así mismo, en la actualidad se está tratando de unir esta ontología con ontologías de otras áreas de conocimiento.

4. Trabajo Futuro

El principal problema para la validación de una ontología aparece cuando dicha ontología no está suficientemente evaluada, no esté consensuada, o que no se haya comunicado de una manera efectiva al colectivo principalmente interesado en la utilización del conocimiento inherente a la ontología. Por ello, es necesario establecer un marco para que el proceso de desarrollo de la ontología proporcione una verificación y evaluación de su contenido. Por ello, la ontología presentada en este artículo necesita de una validación interna y otra externa, basándose en el trabajo de Abran *et al.* [5].

Otros sistemas de validación de la ontología afectan a la propia fuente de obtención de los conceptos de la ontología PMO. Los conceptos de gestión de proyectos definidos en el PMBOK se describen de una manera principalmente narrativa, y por lo tanto, éstos deben ser normalizados en alguno de los sistemas de conocimiento (OpenCyc, SUMO) que proporcionen una definición de los conceptos presentes en la ontología usando conceptos comunes y generales de las *Upper Ontologies*.

Agradecimientos

El desarrollo de la investigación asociada a este artículo ha sido posible gracias a la financiación del proyecto de investigación TIN2004-06689-C03-01 por el Ministerio de Educación y Ciencia.

Referencias

[1] Scarborough, H., Swan, J. y Preston, J., *Knowledge Management: a Literature Review: Issues in People Management*, Institute of Personnel and Development (Londres), 1999.

[2] Gašević, D., Djurić, D. y Devedžić, V., *Model Driven Architecture and Ontology Development*, Springer-Verlag, 2006.

[3] Studer, R., Benjamins, V.R. y Fensel D., "Knowledge Engineering: Principles and Methods", *Data & Knowledge Engineering*, vol. 25, nº 1-2, pp. 161-197, 1998.

[4] Sicilia, M.A., Cuadrado-Gallego, J.J., Rodriguez, D., "Ontologies of software artifacts and activities: Resource annotation and application to learning", en *Proceedings of the 17th Software and Knowledge Engineering Conference*, pp. 145-150. Knowledge Systems Institute, 2005.

[5] Abran, A., Cuadrado-Gallego, J.J., García-Barriocanal, E., Mendes, O., Sánchez-Alonso, S., Sicilia, M.A., "Engineering the ontology for the SWEBOK: Issues and techniques," en *Ontologies for software engineering and software technology*, pp. 103-120. Springer, 2006.

[6] Abran, A., Moore, J.W. (Exec. eds), Bourque, P., Dupuis, R. (eds), Tripp L., "*Guide to the Software Engineering Body of Knowledge*", IEEE Computer Society Press, 2004. http://www.swebok.org/

[7] Lenat, D.B., "Cyc: A Large-Scale investment in Knowledge Infrastructure". *Communications of the ACM*, Vol. 38, nº 11, pp. 33-38, 1995.

[8] Pagels, M. *The DARPA Agent Markup Language Ontology Library* [online], http://www.daml.org/ontologies/. Último acceso: 20/04/2008.

[9] Porter, B. *et al.*, *KKBS / Ontology Projects and Groups Worldwide* [online], http://www.cs.utexas.edu/users/mfkb/related.html. Último acceso: 20/04/2008.

[10] Standford Medical Informatics, *Protégé Wiki: Protégé Ontology Library* [online], http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library. Último acceso: 20/04/2008.

[11] Project Management Institute, *A guide to project management body of knowledge: PMBOK guide. 3rd ed.*, Project Management Institute, 2004.

[12] Noy, N.F., McGuiness, D.L., *Ontology Development 101: A Guide to Creating your First Ontology*, KSL Technical Report KSL-01-05, 2001.

[13] Project Management Institute, *Practice Standard for Work Breakdown Structures. 2nd ed.*, Project Management Institute, 2006.

[14] Heflin, J., Luke, S., *Simple HTML Ontology Extensions (SHOE): Organization Ontology* [online], http://www.cs.umd.edu/projects/plus/SHOE/onts/org1.0.html, Último acceso: 20/04/2008.

[15] Hobbs, J.R., Pan, F., Time Ontology in OWL: W3C Working Draft 27/09/2006 [online], http://www.w3.org/TR/owl-time/, Último acceso: 20/04/2008.

[16] Kerzner, H., *Project Management: A Systems Approach to Planning, Scheduling, and Controlling. 8th ed.*, Willey, 2004.

[17] White, D., Fortune, J., "Current practice in project management - an empirical study", *International Journal of Project Management*, vol. 20, nº 1, pp. 1-11, 2002.

[18] Ehrig, M., Sure, Y., "Ontology Mapping - An Integrated Approach", *First European Semantic Web Symposium, (ESWS 2004)*, Heraklion (Grecia), 2004.

Orientación a aspectos en UML2 sin extensiones

María del Pilar Romay Rodríguez
Depto. de Sistemas Informáticos, Universidad Europea de Madrid
pilar.romay@gmail.com
Carlos E. Cuesta Quintero, Marcos López Sanz
Depto. Lenguajes y Sistemas Informáticos II, Universidad Rey Juan Carlos
carlos.cuesta@urjc.es, marcos.lopez@urjc.es

Abstract

Aspect-oriented software development has evolved far beyond its implementation-level origin and has extended to encompass the entire software lifecycle. Several approaches have tried to express aspect-oriented concepts and principles at the design level, using adequate notations. Most of them have defined some UML profile, or even an extended metamodel, to provide this conceptual framework. However, this article argues that this is actually unnecessary, and UML2 (version 2.1 indeed) provides a native support to describe aspects, once it is carefully considered. The basic idea is to use the new support for *role models* in UML2, and exploit the analogies between roles and aspects to build an aspect-oriented approach. Hence the article explores the evolution of role models in UML, and the specific issues of the current version, where they are strongly tied to the architectural support, defined as the "parts-and-ports" schema. Based on this framework a *dual model for invasive composition* is defined, which is able to include every relevant aspect-oriented concept, and making any specific extension unnecessary.

Resumen

El desarrollo de software orientado a aspectos ha evolucionado más allá de su origen, ligado al nivel de implementación, y se ha extendido por todo el ciclo de vida. Diversos enfoques han tratado de describir los conceptos y principios orientados a aspectos en el nivel de diseño, utilizando notaciones apropiadas. La mayoría han definido algún perfil de UML, o incluso un metamodelo extendido, con el fin de proporcionar el marco conceptual necesario. Este artículo expone, sin embargo, que esto no es necesario, y que UML2 (concretamente la versión 2.1 actual), si se estudia con cuidado, proporciona ya un soporte nativo para la descripción de aspectos. La idea básica es utilizar el nuevo soporte para modelos de roles de UML2, y utilizar las analogías entre roles y aspectos para elaborar un

enfoque orientado a aspectos. Por tanto, el artículo explora la evolución de los modelos de roles en UML, así como los detalles específicos de la versión actual, en la que se encuentran fuertemente ligados al soporte arquitectónico definido mediante el modelo de "partes-y-puertos". Así pues, se define un modelo dual de composición invasiva basado en este esquema, de modo que se incluye todo concepto orientado a aspectos relevante, sin que haya sido necesaria ninguna extensión específica.

Palabras Clave: Orientación a Aspectos, Separación de Asuntos, Modelo de Roles, Colaboración, Clasificador Estructurado, Conectable, Uso de Colaboración, UML2.

1. Introducción

A lo largo de la última década, el enfoque de *orientación a aspectos* [8] ha alcanzado un grado de popularidad significativo. Ligado inicialmente a la orientación a objetos, se presenta como un modelo extendido que pretende superar las limitaciones inherentes a este paradigma. Sin ser el único, ni mucho menos, ha logrado una difusión mayor que la de otras propuestas alternativas y en principio equivalentes; y ha superado los límites de su definición original, extendiéndose a niveles de abstracción más altos e incluso a las etapas iniciales del ciclo de vida (los llamados "aspectos tempranos"). Su mayor mérito, en cualquier caso, consiste en haber llamado la atención sobre los límites del modelo de objetos, presentando toda una serie de técnicas que pretenden superarlos. Se ha de tener en cuenta que bajo el nombre genérico de *aspectos* se engloban ahora tanto la propuesta específica donde se definió ese término [14], como una serie de enfoques muy distintos, pero basados esencialmente en los mismos principios [8][11][13][16].

Considerados a nivel de diseño, lo esencial del modelo de aspectos no es ni la definición de un nuevo tipo de módulo, ni la presencia de un mecanismo de tejido (*weaving*) capaz de combinarlos; tampoco es necesario definir un nuevo marco conceptual que rompa con modelos anteriores. En realidad, el único requisito de un esquema aspectual es que sea capaz de superar las barreras de la modularidad clásica, de modo que proporcione soporte para separar los asuntos sobre un modelo de *composición invasiva* [2].

A lo largo de la última década se han hecho decenas de propuestas de diversos *profiles* de UML que extendían convenientemente su metamodelo para soportar de manera

adecuada los principios y conceptos de la orientación a aspectos, usando típicamente mecanismos ligeros [20][24], pero también los pesados [3]. Desde propuestas iniciales muy primitivas [24] se ha evolucionado hasta propuestas de gran complejidad que, bien desarrollan el modelo JPI hasta sus detalles más concretos [20], o bien describen otras extensiones de la composición invasiva probablemente más adecuadas a nivel de diseño, tanto con un enfoque simétrico [21] como asimétrico [3].

No obstante, este artículo plantea que estas extensiones no son realmente necesarias, y que UML2 "puro" proporciona el soporte necesario para describir un modelo adecuado de composición invasiva utilizando únicamente construcciones exactas. Este enfoque se basa en explotar las analogías y similitudes entre los aspectos y los modelos de roles, y utiliza el soporte definido para estos últimos en la nueva versión de UML, que se basa a su vez en la mayor novedad del mismo, su soporte arquitectónico.

2. El Paradigma de la Orientación a Aspectos

La expresión *orientación a aspectos* se utiliza en un sentido genérico y otro específico. En el segundo, describe a una tecnología concreta, la llamada "programación orientada a aspectos", ligada a ciertas propuestas y al concepto de *aspecto* [14] [18][20]. En el primero, sin embargo, hace referencia a un enfoque integral de desarrollo, el "desarrollo de software orientado a aspectos", que partiendo de un principio genérico llega a definir propuestas muy diversas, incluyendo a la anterior, que aquí recibe el nombre de *modelo de intercepción de puntos de unión* (o *join point interception*, JPI).

La consecuencia última del principio original (la separación de asuntos) es la ruptura de las fronteras modulares clásicas, de modo que la novedad de este enfoque "aspectual" se concreta en la definición de una nueva dimensión de modularidad; de entre los muchos nombres que ha recibido, el de *composición invasiva* [2] es probablemente el que más claramente alude a su naturaleza, por lo que en adelante se usará con preferencia.

2.1. El Principio de Separación de Asuntos

Todo el enfoque se basa en un principio clásico de Ingeniería de Software, el *principio de separación de asuntos*. En este contexto, se denomina *asunto (concern)* a cada uno de los detalles de los que tiene que ocuparse un sistema software, entendido en términos generales (incluido, por ejemplo, un diseño). Aunque "interés" sería probablemente una traducción

más ajustada, no tiene la connotación modular que sí expresa "asunto", por lo que se ha preferido este último.

El término se remonta a pioneros como Dijkstra o Parnas, y de hecho es la justificación clásica que se utilizó para defender los principios de la programación modular hace más de tres décadas. En resumen, se trata de que cada asunto sea considerado por separado, en un módulo independiente. Sin embargo, parece claro que la modularidad clásica no proporciona un soporte adecuado, ya que todavía hoy no hay una correspondencia entre módulos y asuntos que permita una separación real.

Los inconvenientes de una mala separación de asuntos suelen explicarse en términos de dos propiedades, denominadas *enmarañamiento (tangling)* y *dispersión (scattering)*. Se dice que un asunto está *disperso* cuando queda distribuido en múltiples módulos, y se dice que un módulo está *enmarañado* cuando mezcla múltiples asuntos. Aunque resulta sencillo hablar de estas propiedades en términos cualitativos, hacerlo de manera precisa no es trivial. Un reciente trabajo de Conejero *et al.* [5] presenta un marco formal que no sólo proporciona esta definición precisa, sino que también permite definir y comparar modelos de métricas aspectuales. Entre otros detalles, se expone que incluso la noción de *entrecruzamiento (crosscutting)*, fundamental para el modelo y que se desarrolla más adelante, resulta confusa, hasta el punto de demostrar que la conocida definición de Masuhara *et al* [17], que la presenta como la intersección simétrica de enmarañamiento y dispersión, no es la más general, sino que se subsume en otra definición más intuitiva, en la que se requiere que los módulos estén enmarañados, pero sólo es necesario que uno de los asuntos implicados esté además disperso.

En cualquier caso, cuando no existe una correspondencia uno a uno (1:1) entre asuntos y módulos, la complejidad del desarrollo crece exponencialmente. El enfoque clásico se ha centrado en la dimensión funcional, por lo que los módulos se distribuyen de acuerdo con este asunto, pero de modo inadecuado para otros, en particular para los aspectos no funcionales: seguridad, distribución, persistencia, etc. El objetivo último de los modelos aspectuales es mejorar el soporte para la modularización de todos ellos. Pero dado que las fronteras modulares de distintos asuntos no encajan, es preciso que su combinación sea capaz de superarlas: en definitiva, que su composición sea *invasiva*.

2.2. Modelo Asimétrico vs. Modelo Simétrico

El enfoque inicial se basó, entonces, en crear una estructura modular tradicional, esto es, funcional, como esqueleto de partida. A continuación se consideran los demás asuntos, para los que se construyen módulos específicos. Algunos pueden introducirse en el esquema sin problemas; no obstante, cuando se hace necesario alterar alguno de los existentes (hay enmarañamiento), o simultáneamente se afecta a varios (hay dispersión), se dice que el asunto es *transversal (crosscutting*). Es necesario entonces distribuirlo por todo el esquema inicial, combinándolo con sus módulos, rompiendo sus fronteras.

El enfoque inicial y más conocido [14] define entonces los *aspectos* como los módulos que se superponen a los tradicionales, que siguen siendo los objetos. Los aspectos son definiciones parciales, concebidos como extensiones, y se utilizan especialmente para tratar cuestiones no funcionales.

No obstante, varios autores, comenzando por Harrison [11] argumentan que ese enfoque *asimétrico* es inadecuado; una vez que se ha demostrado que el soporte clásico resulta insuficiente, no tiene sentido "complementar" el esquema funcional con un conjunto de extensiones o "parches". En su lugar, proponen que todos los asuntos se consideren por separado, y se describa para cada uno un esquema completo. Después, usando técnicas similares a las del enfoque asimétrico, estos esquemas ortogonales se combinan a partir de sus puntos comunes, en un enfoque *simétrico*. Suele considerarse a éste más general desde un punto de vista teórico, y resulta conceptualmente más simple; no obstante, los enfoques asimétricos son más populares en la práctica.

La distinción entre modelos asimétricos y simétricos de aspectos no resulta esencial para la comprensión de este artículo, por lo que no se profundizará más en estos detalles. Dos comentarios son relevantes, no obstante. En primer lugar, que aunque el enfoque simétrico es más general, en realidad ambos son básicamente equivalentes, si se usan con la disciplina adecuada; y en segundo lugar, que el enfoque final que aquí se propone es esencialmente simétrico, si bien se construye sobre una base funcional (la de UML), por lo que también es fácil verlo desde un punto de vista asimétrico.

2.3. Hipótesis de Interacción y Composición Invasiva

A la hora de implementar la composición invasiva, todos los modelos existentes han adoptado una solución similar, aunque difieren en sus detalles. No se pueden crear las barreras modulares aleatoriamente, ya que éstas vienen dadas por la forma en la que se estructura el problema; los módulos vienen definidos por los propios conceptos, tanto en un enfoque simétrico como en uno asimétrico.

Dado que no se pueden eliminar estas barreras, la única forma de superarlas es basarse en el espacio que existe "entre ellas", es decir, en la *interacción* entre los módulos.

En efecto, la mayoría de los enfoques se basan en la idea de *interceptar* la interacción entre dos módulos, y en este punto alterar la comunicación entre ellos para modificar, complementar, añadir o suprimir el comportamiento del modo adecuado. Por ejemplo, para hacer que un módulo sea seguro, basta con encapsular sus interacciones usando un mecanismo de *tunneling*; igualmente, para proporcionar persistencia, basta con capturar sus interacciones y serializar los datos requeridos en algún almacén.

Esta idea se encarna en la llamada Hipótesis de Interacción, que se enuncia simplemente como: "En todo sistema, se puede introducir un nuevo aspecto mediante la intercepción de la interacción entre sus módulos". Enunciada de manera expresa por Pawlak [20], se puede considerar implícita en el campo incluso con anterioridad.

Por todo ello, la descripción de la orientación a aspectos como "composición invasiva" ayuda, de hecho, a entender el enfoque mucho mejor. No se trata de tener "aspectos" como un nuevo tipo de módulo. Lo que se necesita es disponer de un nuevo tipo de relación, un nuevo tipo de composición. Esto es, un esquema capaz de considerar la interacción de los módulos, e intervenir en ella; en el que todo módulo pueda hablar con cualquier otro, sin que le afecten las barreras de la modularidad. Por el contrario, los módulos habrán de construirse teniendo en cuenta esta interacción, para disponer de la máxima flexibilidad sin perder ninguna de las ventajas tradicionales. En definitiva, se trata simplemente de una nueva dimensión de modularidad [20].

3. Aspectos desde otra Perspectiva

Muchos de los intentos de modelar aspectos a nivel de diseño, y en particular en UML, han partido de la errónea suposición de que era necesario replicar el más conocido de los

modelos de implementación, esto es, el esquema JPI de AspectJ [14]. Sin embargo, existe toda una serie de propuestas basadas en otros enfoques, en algunos casos similares entre sí, que resultan probablemente más apropiados para el diseño. Entre ellos se puede mencionar a las vistas y perspectivas arquitectónicas; los componentes y colaboraciones aspectuales de Lieberherr [16][18]; los aspectos arquitectónicos, sea como *chevrons* [7] o *prismas* [21]; las múltiples dimensiones de Harrison [11], los patrones de composición de Clarke [4], la superposición de Katz [13] o los modelos de roles [8][10]. En todos ellos se plantea una perspectiva de la hipótesis de interacción que se basa en modelar ésta por separado, para combinarla después.

Los *modelos de roles* resultan especialmente interesantes, puesto que, aunque por otros motivos, fueron incorporados a UML, recibiendo un soporte que ha evolucionado de manera radical en las versiones más recientes del lenguaje. Dado que el propósito del artículo es introducir aspectos sin extensiones, y teniendo en cuenta la relación existente entre ambos enfoques, se partirá del estudio del soporte para roles, para plantear más adelante su posible redefinición y reutilización en el contexto de aspectos.

3.1. El Paradigma de Modelos de Roles

Los *modelos de roles* aparecieron en la segunda generación de metodologías orientadas a objetos, aunque por su naturaleza pueden concebirse de manera independiente a este paradigma. Su objetivo es facilitar el modelado de comportamiento, tratado con cierta dificultad en las metodologías inspiradas en el modelado conceptual. En este enfoque, el elemento central es la *interacción* entre objetos, como unidad básica de comportamiento del sistema; en lugar de modelar la funcionalidad abstracta de cada clase por separado, se pretende modelar globalmente cada uno de los comportamientos requeridos para que el sistema realice su actividad; y a partir de estos modelos se deduce con posterioridad la participación de cada objeto o clase en la funcionalidad global. Además de sus obvias ventajas, se suele considerar notable a esta aproximación por su facilidad para acomodar la evolución dinámica de comportamientos.

Existen varias propuestas diferentes para los modelos de roles, si bien son todas ellas más o menos equivalentes. Destacan especialmente las de Kristensen [15] y Reenskaug, ligada al método OOram [22]. La descripción de este apartado se inspira en esta última, ya que su influencia en UML ha sido la más significativa.

La construcción fundamental del modelo de roles es la *interacción*; la idea es que toda la funcionalidad del sistema se desarrolla como un proceso compartido, que implica a varios de sus elementos; y por tanto el comportamiento no se ubica en *uno* de ellos, sino en la *colaboración* entre todos los *participantes* en este proceso. Desde esta perspectiva, cada interacción se modela por separado; el comportamiento global del sistema viene dado por la combinación de todas ellas.

Cuando se describe una interacción, la parte de comportamiento asociada a cada elemento individual se asigna a una entidad llamada *rol*. Ha de tenerse en cuenta que en una primera etapa, estos roles no se identifican ni con objetos ni con clases; ésa será una etapa de *asignación de responsabilidades* posterior en el método. Asimismo, la correspondencia no tendrá que ser necesariamente uno a uno; por ejemplo, varios roles podrán eventualmente asignarse a (es decir, fusionarse en) un mismo objeto.

Un *rol* es, por tanto, un fragmento de comportamiento, en principio sin identidad propia, que representa la participación de un elemento o actor abstracto en una interacción dada. Por supuesto, la complejidad de este comportamiento puede ser tal que se identifique una estructura (externa u observable) para este rol; como mínimo, ha de ser capaz de responder a ciertos mensajes (identificados con operaciones), y ha de ser capaz de emitir o recibir determinados datos (se almacenen o no como atributos). Y, en especial, ha de ser capaz de comunicarse y formar parte de un protocolo potencialmente complejo; es por esto por lo que, necesariamente, define una *interfaz*.

Así pues, un rol define al menos una parte dinámica, que describe su comportamiento observable, y una parte estática, que define una interfaz. Ambas se corresponden, en términos generales, con dos de las vistas definidas en OOram: la vista de *escenario* y la vista de *colaboración*. Como se describe en el apartado siguiente, hay un claro parecido entre estas vistas y los diagramas de secuencia y colaboración de UML 1.x, en especial con los primeros; pero la relación no es tan directa en el segundo caso.

El modelo de Reenskaug es particularmente notable por el hecho de que en él la interfaz se segmenta en puntos de interacción claramente definidos, a los que se denomina *puertos*. Cada rol del modelo se conecta con otro rol al crear un enlace con uno de sus puertos, esto es, con un punto de interacción bien conocido, capaz de *recibir* mensajes; no hay puertos asociados a la emisión. La actividad de un rol siempre se produce en respuesta

a la recepción de un mensaje en uno de sus puertos. En OOram también es posible "invocar" simultáneamente a varios roles a la vez, a través de un único puerto compartido. Esta característica se ha transferido también a UML en la forma de los *multiobjetos* de los dos diagramas de interacción; pero no se entrará en más detalles al respecto, dado que este elemento es perfectamente conocido.

En resumen: un *modelo de roles* describe el comportamiento de una parte del sistema como la colaboración (un protocolo de interacción) entre varios *roles* o participantes, que se conectan entre sí a través de *enlaces* entre los *puertos* en los que se divide su interfaz. Los detalles concretos del protocolo se describen como un escenario, esto es, como algo muy similar a un diagrama de secuencia convencional. Es obvio el parecido de este esquema con las nociones de la arquitectura de software; la sintaxis del modelo de roles, en especial en la formulación de Reenskaug, tiene claros paralelismos con los elementos de un Lenguaje de Descripción de Arquitectura (ADL).

Como ya se ha dicho, la similitud no es estrictamente sintáctica, por lo que se puede aplicar igualmente a otras presentaciones de los modelos de roles; en algunos sentidos, éstas son complementarias. Por ejemplo, la versión de Kristensen [15] no usa puertos de manera explícita, pero en cambio identifica *tipos de rol* (*role types*), un concepto más abstracto, que permite *clasificar* roles similares y desarrollar mejor su claro paralelismo con una instancia, facilitando, de hecho, la comprensión del modelo.

También hay una obvia conexión entre estos roles, descritos como la participación de una entidad en un comportamiento común, y los *sujetos* de Harrison [11], descritos como el punto de vista que tiene una entidad de un contexto compartido, y que son uno de los precedentes más conocidos de los aspectos. No se entrará en más detalles sobre esta relación, ya que ha sido explorada por varios autores [8], y no afecta directamente a la argumentación de este artículo.

3.2. Colaboraciones UML (1.1) vs. Modelos de Roles

El concepto de *colaboración*, ahora considerado esencial al lenguaje, fue incorporado a UML de manera tardía, durante su transición a la versión 1.1, y debido en gran parte a la influencia de métodos como Catalysis y el citado OOram. El objetivo que se perseguía con su incorporación era aclarar el papel, siempre confuso, de los casos de uso. Aunque el método de Jacobson dejaba claro que los escenarios instanciaban casos de uso, y que el

comportamiento del sistema se deducía a partir de estos escenarios, la semántica final no estaba definida. En definitiva, los escenarios especifican el comportamiento externo, mientras que los diagramas de interacción describen el comportamiento interno. Pero esto cambia al introducirse la noción de colaboración como unidad del comportamiento interno, y definir su relación con el caso de uso. Se abre la puerta, además, a utilizarla cono la base de una metodología de segunda generación, como las mencionadas en el apartado anterior, en el que el modelado se centra en la interacción.

En resumen, todo *caso de uso* identificado en la especificación de un sistema se realiza como una *colaboración*, tanto a nivel de análisis como luego a nivel de diseño. Al igual que un caso de uso se concibe como un patrón común que se instancia como un conjunto de *escenarios* concretos, una colaboración se entiende como un patrón (estructural) común a un conjunto de *interacciones* de objetos, entendidas también como ocurrencias de esa colaboración, y que se expresan con los *diagramas de interacción* de UML (esto es, los diagramas de secuencia y los antiguos diagramas de colaboración, renombrados en UML 2.0 como "diagramas de comunicación"). Así pues, los diagramas de interacción (que involucran a objetos, esto es, a instancias), describen *instancias* de una colaboración, y por tanto se corresponden con un escenario, que es a su vez una instancia del caso de uso *realizado* por esa colaboración.

Partiendo de este esquema, a menudo se entiende que una colaboración consiste en un conjunto de clases y algunas de sus asociaciones. Esto es, se consideran todas las interacciones que son instancia de la colaboración; en éstas, se toman todos los objetos implicados, y se reúnen todas las clases de las que a su vez son instancia. El conjunto de clases resultante queda unido por varias asociaciones, que a su vez se corresponden con los mensajes enviados (los enlaces utilizados) en las respectivas interacciones.

Pero en realidad esto no tiene por qué ser así. Los elementos de una colaboración no tienen por qué ser clases, como no lo son, por ejemplo, en Catalysis. En realidad sólo se requiere que sean *clasificadores*, esto es, elementos instanciables, como lo son tanto la propia colaboración como el caso de uso.

Así pues, en UML 1.1, concretamente en su especificación semántica, se incorporaron, junto a la propia colaboración, varios elementos adicionales. Éstos iban a permitir no sólo usarla para definir la realización, sino además expresarla como un *modelo*

de roles, siguiendo las pautas de la *vista de colaboración* de OOram (e, implícitamente, también la de escenario, si bien en este caso no se requerían cambios adicionales). El propio UML proporcionaba ya la mayoría de la estructura; por lo que fue necesario añadir muy pocos elementos. Se puede reducir la lista a tres, de los cuales uno es el concepto central y el resto son las abstracciones de soporte.

El primer elemento es el *clasificador-rol* (*ClassifierRole*), y tiene la misma función que se ha descrito para un rol en el apartado anterior, esto es, encapsular el comportamiento correspondiente a uno de los participantes en una interacción, sin identificarlo todavía con un objeto o clase. Junto a éste se incorporan la *asociación-rol* (*AssociationRole*), que se asimila como un enlace entre roles, y más adelante (en una versión tan posterior como la 1.4) el *extremo-de-asociación-rol* (*AssociationEndRole*), que se puede entender como equivalente a un puerto del modelo de Reenskaug; su función precisa es servir de soporte para ubicar toda la información asociada a la conexión de un rol.

A primera vista, se tiene una identificación directa con los elementos ya descritos del modelo de roles. Desde ese punto de vista, una colaboración se describe como la estructura de interacción formada por un conjunto de clasificadores-rol, conectados entre sí mediante asociaciones-rol, y utilizando los extremos correspondientes para ubicar con claridad esas conexiones. Teóricamente, pues, se podría realizar un caso de uso como una colaboración, y describir ésta como un modelo de roles especificado con estos elementos. Una vez realizadas todas las colaboraciones de este modo, se tendría que decidir cómo se solapan los roles entre sí, para definir la correspondencia con las clases del diagrama de clases. De este modo, se había comenzado el modelado con un método de segunda generación, para terminar con un modelo conceptual completo, que se desarrollaría de la manera habitual en UML (de primera generación).

Ahora bien, hay una inconsistencia en este modelo; y es que en realidad se están usando varios elementos como ellos mismos y *sus propias instancias*. Los roles del modelo de Reenskaug son *instancias*, al nivel de un objeto; sus enlaces son similares a los de los objetos, y se envían los mismos mensajes que los objetos. Esto es coherente con la semántica del diagrama de colaboración, en el que lo que se describe es una interacción de objetos, esto es, una *instancia* de una colaboración. La primera mitad del método, por tanto, asume que los roles y sus interacciones están al nivel de instancia. En cambio, la segunda

mitad del método, en la que se produce la correspondencia conceptual y la fusión de los roles en clases, asume que los modelos están al nivel de clases.

Sin embargo, tanto la semántica como el metamodelo eran claros: el clasificador-rol, la asociación-rol, e incluso la colaboración son todos ellos *clasificadores*, situados en el mismo nivel que la *clase* y el propio caso de uso. En realidad, el esquema resultante es perfectamente consistente dentro del propio UML; de hecho, el único detalle confuso es la definición del papel del diagrama de colaboración. Las restantes dudas se refieren a la correspondencia con los modelos de roles de OOram que, en todo caso, son parte de una especificación ajena al lenguaje.

El uso de estos elementos es, por tanto, ligeramente diferente. En resumen, el método consiste ahora en realizar un caso de uso como una colaboración, que se define como un clasificador. Cada instancia de esta colaboración es una interacción, que se corresponde con un escenario de un caso de uso. Esta interacción se desarrolla mediante el diagrama correspondiente (de secuencia o de colaboración), de modo que cada participante en el protocolo se identifica como un "rol", esto es, una instancia de un clasificador-rol. Estos "roles" (que nunca fueron definidos en el metamodelo de UML 1.x) se comunican entre sí igual que los objetos, mediante el paso de mensaje a través de enlaces. Estos enlaces, sin embargo, serán instancias de asociaciones-rol, no de asociaciones estándar, ya que éstas tan sólo se pueden usar para relacionar clases. A partir de todas las interacciones y escenarios, se puede abstraer ya la estructura de una colaboración, como un conjunto de clasificadores-rol conectados entre sí mediante asociaciones-rol; los primeros clasifican a los "roles" descritos, y los segundos se corresponden con sus enlaces; a partir de los mensajes se deduce además el conjunto de operaciones de cada clasificador-rol. No está claro el diagrama en el que se muestran estos elementos, pero puede asumirse como un fragmento del diagrama de clases. Posteriormente, para terminar, bastará con combinar estos clasificadores-rol, unificando nombres y eliminando redundancias, para obtener como resultado las clases que conformarán el modelo conceptual.

El clasificador-rol, pues, se corresponde con el *tipo-de-rol* de Kristensen, más que con el rol de Reenskaug, que se correspondería por tanto con su instancia. La falta de una definición explícita de este rol-instancia (implícito en todo caso, ya que *ClassifierRole*

especializa a *Classifier*) está en el origen de diversos malos usos de este modelo, y en particular del diagrama de colaboración.

La confusión terminológica también aumenta la complejidad en este caso: los escenarios de UML no son los de OOram; e incluso las colaboraciones, más similares, no son las mismas, ya que tienen más bien el papel de sus clasificadores; a esto hay que sumarle la confusión respecto al nombre de diagrama de colaboración, que parece seguir más a OOram que al propio UML. Puede afirmarse que, en buena medida, la confusión conceptual respecto a los modelos de roles en UML se derivaba, en realidad, de estas simples confusiones terminológicas.

En cualquier caso, y en resumen, en UML 1.x se incorporaron todos los conceptos necesarios para soportar el modelado de roles; bastaba con definirlos al nivel de clases y desarrollar la interacción entre sus instancias para tener, dentro del lenguaje unificado, algo muy similar al método de OOram.

3.3. Transición de los Modelos de Roles en UML (de 1.1 a 2.1)

Los modelos de roles se hicieron (relativamente) populares en UML durante el primer lustro de la presente década, cuando la especificación de referencia evolucionaba entre las versiones 1.1 y 1.4, mientras el lenguaje comenzaba a estabilizarse. Probablemente la causa fue la confusión existente respecto a la realización de los casos de uso en la forma de colaboraciones. No obstante, una vez conocidos, los modelos de roles se revelaron como un método flexible y adaptable para modelar el comportamiento de un modo coherente con el modelado conceptual, compatible además con la evolución del sistema. Su importancia fue creciendo y su uso extendiéndose, hasta el punto de que se esperaba que su importancia en UML creciera en la nueva versión 2.0.

No obstante, esto no fue lo que ocurrió. Por lo contrario, y de un modo casi sorpresivo, los elementos que se habían introducido para soportar modelado de roles desaparecieron de la especificación de UML 2.0. No existe ninguno de estos conceptos en la versión actual: ni el clasificador-rol, ni la asociación-rol, ni ningún concepto equivalente.

La especificación actualmente vigente de UML, concretamente la versión 2.1.2 de la Superestructura [19], dice literalmente: *"El concepto de clasificador-rol es subsumido por un elemento conectable utilizado dentro de una colaboración"*. Es decir, se ha pasado de disponer del concepto de rol como una abstracción propia, un elemento del metamodelo, a

verlo simplemente como un elemento estructural derivado, algo que se obtiene *indirectamente* a partir de la propia arquitectura de modelado. Esta es una transición que dista de ser trivial, ya que su alcance es enorme; y sin embargo, no era una decisión fácilmente previsible.

El verdadero motivo de esta transición hay que encontrarlo en la asimilación, por parte de UML 2.0, de una filosofía de diseño completamente nueva y ajena a la anterior, que ha dado en ser conocida como el modelo de *"partes-y-puertos"* (*parts-and-ports*). Este modelo resume la incorporación en UML de un conjunto de abstracciones básicamente compositivas, que proceden en su gran mayoría de otro estándar: SDL [12].

El motivo resulta evidente: el Proceso Unificado, que se describe a sí mismo como *centrado en la arquitectura*, se basa en la adopción de un lenguaje estándar (UML) que era, de hecho, especialmente inadecuado para la descripción de arquitecturas software, cuando éstas se definen formalmente. Aunque su estructura de múltiples vistas (4+1), encarnada en sus siete diagramas, tenía el potencial adecuado para la especificación de vistas *arquitectónicas*, la estructura conceptual del lenguaje nunca tuvo el nivel de abstracción necesario. UML siempre fue especialmente inadecuado para la descripción de jerarquías de composición, una de las (dos) dimensiones esenciales en la arquitectura de software. Para superar esta limitación, el lenguaje tenía que incorporar un conjunto de conceptos interrelacionados; y éstos se podían tomar de múltiples fuentes. Durante el proceso de estandarización de UML 2.0 se tomó la decisión de basar esta extensión en un estándar ya existente, del que de hecho ya se habían tomado muchos elementos en versiones anteriores de UML: el propio SDL (y otro estándar relacionado, MSC, del que no se hablará aquí por motivos de espacio, pero que proporciona el soporte para los cambios que, por coherencia conceptual, se han tenido que realizar en el lenguaje a nivel de comportamiento, en especial en los diagramas de secuencia). SDL proporciona estructuras jerárquicas, composición de instancias, elementos de conexión (conectores) y, en definitiva, todo lo necesario para soportar la descripción de arquitecturas, con un nivel de detalle muy superior al existente anteriormente. Este enfoque, conocido como el modelo de "partes-y-puertos", ha sido adoptado plenamente en UML 2.0. Ahora bien, la incorporación de este esquema no implica solamente que se dispone de un diagrama o una notación más; por el contrario,

afecta a directamente a conceptos básicos de UML, como los de objeto y clasificador (o clase), y tiene consecuencias en diversos aspectos de la semántica general del lenguaje.

También, en el tema objeto de este artículo. El comentario citado más arriba muestra claramente que el estándar ha decidido adoptar un enfoque de diseño que *no* se basa en los modelos de roles. Pero esto no significa que UML2 haya rechazado el uso de los modelos de roles; de hecho, el lenguaje los soporta con mayor claridad y flexibilidad que antes. Lo que ocurre es que han pasado a ser un concepto *derivado*, ya que al introducir todos los mecanismos para la especificación de estructuras compuestas, en especial los conceptos de *parte* y conector, automáticamente se han introducido todos los elementos necesarios para describir un modelo de roles. Lógicamente, en lugar de mantener un conjunto de conceptos específicos que ahora resultaban redundantes, se ha preferido redefinirlos en función de los nuevos elementos introducidos.

De hecho, el modelo anterior necesitaba claramente una revisión, ya que resultaba claramente confuso. En UML 1.x, el papel del clasificador-rol nunca quedó demasiado claro: por una parte era definido como un clasificador, pero por otra ocupaba el mismo lugar que una instancia en los diagramas de colaboración, lo que hacía confuso tanto al clasificador-rol como al propio diagrama (cuyo nombre era también claramente desafortunado). La propia semántica nunca definió explícitamente la relación entre este elemento (y los demás que lo acompañaban) y el resto del metamodelo.

La frase citada anteriormente es probablemente la declaración más explícita que existe del cambio de enfoque de la especificación; sin embargo, no resulta sencillo localizarla si no se sabe de antemano dónde está o, lo que es lo mismo, los motivos por los que se ha producido este cambio en la misma.

Se encuentra en el capítulo 9 (*"Composite Structures"*) de la Superestructura de UML, una parte de la especificación que se dedica a describir "composiciones de elementos interconectados, que representan a instancias en tiempo de ejecución que colaboran mediante enlaces de comunicación para lograr objetivos comunes" [19]. Se trata de una sección completamente nueva, que describe todos los conceptos "arquitectónicos" que no existían con anterioridad en el lenguaje, y que *no* se han introducido en un bloque único, de manera directa. Al contrario, este nuevo soporte para *estructuras compuestas* implica a elementos y abstracciones de *seis* paquetes distintos del metamodelo, y en particular de tres

de ellos: el de Puertos (*Ports*), el de Colaboraciones (*Collaborations*), y el de Estructuras Internas (*InternalStructures*). Este artículo se centra en especial de los relativos a este último, ya que el primero se limita a proporcionar la base para describir una conexión de comunicación (esencial para la definición de una estructura, pero que no resulta importante en sí mismo), y el segundo se dedica a expresar las consecuencias de estos cambios en el concepto de colaboración existente, y a desarrollar su relación con los elementos más habituales del modelo de objetos y clases.

No se entrará aquí en grandes detalles con respecto a los conceptos incorporados como parte del enfoque de partes-y-puertos, que ya han sido descritos en múltiples ocasiones, incluyendo trabajos previos como [23]. Baste reseñar que se introducen los conceptos de *conector*, como enlace de comunicación entre instancias, y *puerto*, como punto de interacción para el comportamiento de un clasificador; y conceptos de soporte de éstos, como el *clasificador encapsulado* (*EncapsulatedClassifier*), que es simplemente aquél que tiene puertos; el *elemento conectable* (*ConnectableElement*), definido para abstraer a cualquier instancia capaz ser conectada; o *extremo de conector* (*ConnectorEnd)*, que expresa la conexión entre un conector y un elemento conectable.

Se modifica además el concepto de *propiedad*, de modo que además de asumir su matiz original de *atributo*, puede añadir los de *parte* y *puerto* e, indirectamente, el de *rol*. De hecho, en el metamodelo de UML 2.0 no existe el concepto de rol, pero tampoco existe el concepto de parte, y sin embargo es una adición fundamental. La *parte* no es más que una forma de representar una "propiedad" dentro de cierta estructura (precisamente, la compuesta). Pero es esta representación la que describe el enfoque de partes-y-puertos, y por ello supone una de las mayores adiciones de UML2.

Del mismo modo, se modifica también la definición de la *colaboración*, que deja de usar los conceptos de clasificador-rol y asociación-rol. En su lugar, le basta con usar el matiz de *rol* añadido a la redefinida propiedad. Este matiz no es más que un nombre: en la colaboración se llama *rol-de-colaboración* (*collaborationRole*), y en el clasificador estructurado se llamará simplemente *rol* (*role*). Pero en ambos casos no es más que una propiedad convencional, que (eso sí) tiene un tipo de elemento conectable. Lo único que hace la definición de "propiedad" al respecto es, precisamente, indicar que el antiguo rol ha sido eliminado, utilizando para ello la frase citada al principio.

En resumen, la colaboración en UML 2.1 es muy similar a la anterior, aunque con una semántica mucho más clara. Se define simplemente como un conjunto de elementos conectables (por tanto, *instancias*), que se denominan *roles-de-colaboración* desde su perspectiva; éstos son, además, sus únicas propiedades. Sin embargo, al ser conectables, estos roles estarán ligados mediante *conectores*, que no pertenecen propiamente a la colaboración, pero en todo caso relacionan entre sí a sus elementos. En definitiva, lo que se describe es un modelo de roles clásico, completamente isomorfo al descrito en la sección 0, y sin los conflictos semánticos de versiones anteriores. La propia definición indica explícitamente que "sólo se incorporan los aspectos de la realidad que se estimen necesarios (...) Así pues, detalles como la identidad o la clase precisa de las instancias participantes son suprimidos" [19].

Asumida esta base, tan sólo es necesario definir dos elementos más para tener descritos por completo tanto el esquema de partes-y-puertos como el propio modelo de roles; son, además, los que proporcionan más detalles al respecto. Se trata del *uso-de-colaboración* (*Collaborations::CollaborationUse)*, que define en su interior, además, una relación de *ligadura de rol* (*role binding*), y el *clasificador estructurado* (*StructuredClassifier*), que se introduce para definir una estructura compuesta de partes-y-puertos, y se constituye así en el elemento fundamental del paquete *InternalStructures*; pero que es lo bastante versátil como para introducir también el matiz de rol, unificando ambos enfoques.

El más complejo de los dos es el clasificador estructurado, base de todo este enfoque. Se define como "cualquier clasificador cuyo comportamiento se puede describir, en todo o en parte, como la colaboración de las instancias que contiene o referencia" [19]. Tiene dos objetivos, similares y compatibles; en primer lugar, define la composición de partes, siendo la construcción básica del esquema de partes-y-puertos. En segundo lugar, da el soporte para utilizar roles en la composición. Esto es necesario no sólo para su uso en modelado, sino también para el propio esquema de partes.

Un clasificador estructurado tiene propiedades de dos tipos: *atributos* propios y *partes*, que se definen como las propiedades que posee por composición. Éstas son instancias (objetos) de otras clases, contenidas en el clasificador, que constituyen la base de una estructura compuesta. En principio, con esto sería suficiente: pero si bien esto permite

agregar instancias en una estructura, no permite indicar si éstas tienen un papel concreto dentro de la misma. Para ello se hace necesario definir estas partes como *roles*, esto es, referencias a estos papeles, que se corresponden directamente con las partes descritas. De hecho, el clasificador se compone sólo de atributos y roles (denotados como /role:), siendo el concepto de parte, en realidad, derivado [19]. El rol es, además, un elemento conectable, por lo que se comunica con otros roles mediante conectores, y a través de puertos. En definitiva, la estructura compuesta se desarrolla como un modelo de roles completo, si bien su unión es conceptual, no funcional como en la colaboración. Más aún, a diferencia de lo dicho en éstas, el clasificador estructurado contiene (a través de una relación) a los conectores que interconectan a sus partes. Por tanto, el clasificador no sólo se compone de sus partes, sino que equivale a la estructura *completa*.

A primera vista, podría parecer que no hay nada de especial en esta incorporación; en definitiva, las relaciones de agregación y (muy especialmente) composición simple ya expresaban relaciones de composición jerárquica en UML. Sin embargo, existe una diferencia fundamental, que de hecho es el motivo por el que se introduce el esquema de partes-y-puertos: las citadas son relaciones *entre clases*, mientras que aquí se aclara el papel de las *instancias* dentro de la clase contenedora. Por definición, en la composición los elementos contenidos tienen un rol específico; pero antes de este esquema no era posible describir ese rol. Utilizando el ejemplo clásico, en UML 1.x era posible decir que una Bicicleta estaba compuesta de Ruedas, entre otras cosas; pero sólo desde UML2 se puede distinguir a la instancia de la rueda *delantera* de la de la rueda *trasera*. Más allá del cambio de notación, que es claramente más visual en la versión actual, ésta es la verdadera novedad del lenguaje, y la que le dota de nuevas capacidades.

Finalmente, el uso-de-colaboración se presenta como el concepto que encarna la esencia del nuevo modelo, ya que relaciona entre sí todas sus partes. Representa la aplicación del patrón [de comportamiento] descrito por una colaboración a una situación específica en la que se implica a clases o instancias específicas que se corresponden con los roles de la colaboración [19]. Es decir, es una estructura que toma los roles-de-colaboración e indica con qué instancias del modelo se corresponden, sin tener en cuenta las estructuras a las que éstos puedan pertenecer, ni ninguna otra consideración.

El uso-de-colaboración es una función de correspondencia que se define por extensión, esto es, enumerando a todos sus elementos. Toma una colaboración y recorre sus roles uno a uno, asignando a cada uno una *ligadura de rol*, esto es, una dependencia mediante la que cada rol se corresponde con una instancia existente (utilizando el nombre del rol, además, como nombre de la ligadura). De este modo, una instancia "real" ocupa el lugar ocupado por un rol en la colaboración, y así la funcionalidad implementada es realizada por un elemento concreto. Se ha de tener en cuenta, además, que la semántica permite explícitamente que varios roles distintos se correspondan con la misma instancia, de modo que se permite implícitamente su combinación.

Con todo lo dicho, el modo en que se desarrolla un modelo de roles en UML 2.1 resulta evidente, ya que todo elemento del esquema anterior se ha sido sustituido por otro con una semántica más precisa. Dada la relación entre las nociones de rol y aspecto, también es posible intuir una solución en este último marco conceptual.

4. Aspectos en un Clasificador Estructurado

A partir de todo lo expuesto con anterioridad, resulta relativamente sencillo plantear un modo de describir un esquema de composición invasiva (un "modelo de aspectos") que tan sólo utilice construcciones de UML 2.1 "puro", sin necesidad de definir *ninguna* extensión del lenguaje, ni siquiera en la forma de estereotipos. En resumen, basta con describir cada asunto por separado, como una (o varias) colaboraciones; cada uno de los roles será un *aspecto* de este modelo. Se aplica la correspondencia definida en un uso-de-colaboración (o varios) para identificar estos aspectos con instancias de clases. Estas clases pueden definirse de manera independiente (como aspectos *separados*), o usarse para definir *partes* dentro de un clasificador estructurado (siendo en este caso aspectos *integrados*). De hecho, a menudo se mantendrán ambas perspectivas. El mayor mérito de este enfoque es, precisamente, que hace explícitos a los conectores que enlazan a las partes de una estructura: es así uno de los pocos modelos en los que las relaciones entre aspectos se hacen explícitas, y no sólo en relación a uno *dominante*.

Concepto de Orientación a Aspectos	Modelo Dual de Composición Invasiva
Aspecto (Ocurrencia)	Rol de Colaboración / Rol (Instancia)
Aspecto (Tipo) / Clase-Aspecto	Elemento Conectable (Tipo)
Asunto (*Concern*)	Colaboración
Corte Transversal (*Crosscut*)	Colaboración
Punto de Unión (*Join Point*)	Ligadura de Rol (*Role Binding*)
Componente (Tipo)	Clasificador Estructurado
Componente Aspectual	Clasificador Estructurado (de Roles)
Colaboración Aspectual	Clasificador Estructurado (de Roles)
Superposición de Katz	Clasificador Estructurado (de Roles)
Conector (Aspectual)	Uso-de-Colaboración (*CollaborationUse*)
Punto de Corte (*Pointcut*)	Uso-de-Colaboración (*CollaborationUse*)
Función de Fusión (*Merge*)	Uso-de-Colaboración (*CollaborationUse*)
Hiperfragmento (*Hyperslice*)	Clasificador Estructurado (de Roles)
Hipermódulo	Clasificador Estructurado

Tabla 1: Correspondencia con otros Modelos de Aspectos

El modelo es obviamente simétrico, dado que el aspecto funcional no es dominante, si bien sigue siendo más fácil de describir en UML. Su naturaleza es esencialmente *dual*; esto es, se basa en *separar* la modularidad en *dos* estructuras: una de composición, dada por el clasificador estructurado, y una de interacción, expresada por la colaboración. Se desarrollan de manera ortogonal, para combinarse posteriormente. Esta dualidad es casi inevitable, ya que la misma definición de las estructuras compuestas en UML 2.1 tiene una estructura dual. Esto lo diferencia de los esquemas asimétricos, que se basan en aumentar o complementar *una* estructura; pero también de otros esquemas simétricos, que habitualmente se basan en utilizar una estructura unificada para todos los aspectos [21] o para cada aspecto [11].

Esta propuesta proporciona también un proceso para el modelado de aspectos; no se pretende en ningún caso modelarlos de manera aislada, sino como parte de un esquema global de comportamiento. En primer lugar, se parte de una separación de asuntos total; esto es, cada asunto (funcionalidad, persistencia, seguridad, etc.) se considera de manera independiente del resto. Así, cada asunto se modela como una colaboración o, más bien,

como un conjunto de colaboraciones. Cada una de éstas describe una función definida dentro de un asunto (un protocolo de seguridad, por ejemplo), que se desarrolla como un modelo de roles; se puede considerar similar a un corte transversal (*crosscut*), en el sentido de que todos los aspectos relativos a esta función, y sus relaciones, han de ser definidos. Los roles son instancias de elementos conectables, y como tales son tipados; estos tipos de rol se corresponden con los *role types* de los sistemas de Kristensen [15] o Katz [13], y son también los *aspectos*, a nivel de tipo o clase. Dado que la mayoría de los modelos de aspectos, como el JPI [14] tratan con los aspectos a este nivel, más que al nivel de instancia, esta correspondencia resulta significativa. En cualquier caso, estos tipos serían, a todos los efectos, definibles como clases convencionales.

En este punto, los aspectos/roles no tienen por qué corresponderse con clases completas, y pueden verse aún como elementos parciales. Es muy probable, además, que un mismo aspecto aparezca en más de una colaboración, incluso como la misma instancia, aunque no es necesario decidir esa identidad en este punto. No obstante, cuando esta relación es muy directa, puede considerarse conveniente "empaquetar" varios de estos roles en un clasificador estructurado, que también incluiría los conectores relevantes, así como otros que se pudieran definir. En definitiva, esta estructura de roles describe, básicamente, un "aspecto compuesto". Resulta notable porque usa técnicas tradicionales de modularidad para agrupar elementos *aspectuales*, y en sí no es una técnica invasiva; tan sólo agrupa en base a un modelo conceptual. En todo caso, se puede considerar equivalente a los componentes aspectuales de Lieberherr [16] o Caesar [18]; más aún, no tiene por qué circunscribirse a un asunto concreto, por lo su naturaleza puede ser *transversal*. En tal caso, se corresponde con una colaboración aspectual de Lieberherr [16], un *hyperslice* de MDSoC [11], una superposición de Katz [13] o un *chevron* de Cuesta [7]. En todo caso, este elemento no es realmente necesario, y se define sólo si resulta conveniente. Por otra parte, esta etapa puede ser demasiado temprana para decidirlo, por lo que su definición podría demorarse, opcionalmente, incluso hasta el final del proceso.

En todo caso, el proceso de desarrollo transcurre ahora del modo habitual, en el que se identifican los principales elementos conceptuales, típicamente funcionales. Todos los roles descritos en todas las colaboraciones se corresponden, tanto a nivel de tipo como de instancia, con clases concretas. La correspondencia se desarrolla, obviamente, como un

uso-de-colaboración, que juega por tanto el papel de un *pointcut* o una función de *merge*, en que cada ligadura de rol es similar a un punto de unión.

Cada rol se define así como una clase definida para ser independiente, o como parte de una clase contenedora; así pues, se realiza el tejido de aspectos (*weaving*) incluso antes de llegar a la estructura final. Existen dos modos de plantear este tejido. El más sencillo es la correspondencia directa: los aspectos que sólo se corresponden con un elemento se definen simplemente como una clase; en cambio, cuando varios aspectos se relacionan con la misma clase destino, ésta se define como un *clasificador estructurado*, en el que los aspectos se presentan ahora como roles, es decir, *partes* conectables, que mantienen o crean los conectores necesarios. El otro enfoque es algo más complejo: se define una clase para cada rol/aspecto, y éstas se agrupan siguiendo tanto los enlaces ya existentes a nivel de interacción, como mediante fronteras modulares de naturaleza conceptual, a nivel de composición. En todo caso, esta segunda técnica tendrá que utilizarse para construir adecuadamente la jerarquía modular, en la que un clasificador estructurado puede definir las partes de otro contenedor aún mayor.

En este punto, el proceso concluye y toda la estructura se ha definido. Obsérvese que si se parte de la separación de asuntos resulta innecesario, como en todo modelo simétrico, plantearse si una estructura es transversal o no. La definición de *crosscutting* resulta así irrelevante, puesto que el modelo ya cubre las dos relaciones conceptuales básicas; en efecto, el modelo controla tanto el enmarañamiento como la dispersión, ya que dispone de una estructura específica para expresar la correspondencia de la que nacen: el uso-de-colaboración. Un módulo enmarañado es aquí simplemente un clasificador estructurado compuesto por varios roles/aspecto; un aspecto disperso no es más que un *tipo* que se instancia como varios roles desde un principio; cada una de estos roles se corresponde *uno a uno* con un clasificador de destino; por tanto, la complejidad que hacía necesario la composición invasiva ha desaparecido, controlada por las estructuras.

Por motivos de espacio no se realizará una comparación más detallada entre el modelo descrito y otros modelos de aspectos; las correspondencias conceptuales más destacadas se resumen en la Tabla 1. A partir de ésta, resulta fácil hacer la identificación entre esta propuesta y los modelos más conocidos, en particular el JPI de Kiczales *et al*

[14], la MDSoC de Harrison *et al* [11] o las colaboraciones aspectuales de Lieberherr *et al* [16], que resumen las características de muchas otras propuestas similares [18][20].

En resumen, utilizando la base de UML 2.0, este artículo propone desarrollar un *modelo dual de composición invasiva* basado en dos dimensiones de composición ortogonales; se trata por tanto de un modelo simétrico. La primera dimensión tiene una naturaleza arquitectónica (estructural), y utiliza el esquema de partes-y-puertos; tiene como pieza fundamental el clasificador estructurado. Describe un esquema modular jerárquico clásico, puramente compositivo. La segunda dimensión tiene una naturaleza interactiva (de comportamiento), y define un modelo de roles, en el que se tiene como pieza básica la colaboración. En éste la descomposición no es jerárquica, ni siquiera estrictamente conexa; la separación de asuntos no se basa en *dónde* se ubica una entidad, sino en *con quién* se comunica. Ninguna de las dos dimensiones prevalece sobre la otra; ambas se combinan simplemente según la correspondencia definida por los usos-de-colaboración, de igual modo que hacen los *puntos de corte* (*pointcuts*) en el modelo JPI [14] o las funciones de fusión (*merge*) en los *hipermódulos* del modelo MDSOC [11].

Precisamente es esto lo que hace que este modelo sea "orientado a aspectos": en los modelos de composición clásicos, las fronteras de composición limitan el ámbito de la interacción; por eso se habla de asuntos *transversales* cuando se rompen estas fronteras con los modelos de orientación a aspectos (asimétricos). En un modelo de composición invasiva (simétrico), la interacción no queda limitada de este modo en la fase de diseño; simplemente, el modelo de comportamiento se desarrolla como sea necesario. Y sólo después, cuando ambos modelos se han completado, se plantea una correspondencia entre ambos, que únicamente dependerá de sus contenidos. No es el diseñador, sino el propio sistema, el que tendrá que conciliar entonces ambas perspectivas, siguiendo las pautas de esta correspondencia. Este proceso es el equivalente, en este modelo dual, del mecanismo de *tejido de aspectos* en otros esquemas orientados a aspectos.

Por motivos de espacio, no se desarrollan en este artículo los detalles relativos al comportamiento, esto es, al desarrollo del dinamismo dentro de la estructura descrita mediante los diagramas de interacción, y en especial mediante los ampliados diagramas de secuencia de UML 2.0, que reflejan tanto las necesidades del modelo jerárquico de partes-y-puertos como la influencia de MSC. El metamodelo incluye los conceptos de evento

(*InvocationActions::Trigger*) y acción de invocación (*InvocationAction*) para dar la semántica adecuada a los puertos; aun sin entrar en detalles, se usa del modo esperado por la intuición, y de manera consistente con las demás modificaciones.

Los efectos del modelo descrito sobre el comportamiento pueden deducirse fácilmente, y su compatibilidad con los requisitos habituales del paradigma orientado a aspectos, aplicado a nivel de objetos individuales, resulta obvia. Sin entrar en excesivos detalles, pueden considerarse tres de las situaciones más significativas. En primer lugar, el caso de una *introducción*, esto es, cuando considerar un nuevo asunto implica la adición de un nuevo elemento estructural. En realidad, aquí el modelo dual es más flexible que otras propuestas, ya que se puede delimitar la granularidad; aunque en todo caso el nuevo elemento es un rol, éste puede terminar resultando simplemente en un nuevo método (una nueva acción), o incluir además un nuevo puerto (y un nuevo evento); o en el caso más extremo, un nuevo elemento conectable (y sus conectores). El segundo caso es el de un *aspecto* no considerado, en el que se incorporase nuevo comportamiento que antes no era incluido; éste se reduce necesariamente o al caso anterior o al siguiente. El tercer caso es, por fin, aquél en el que se produce la *intercepción* de comportamiento ya existente con anterioridad. En éste se pueden introducir nuevos métodos (o acciones), aparte del interceptado; se alterará el protocolo (secuencia), necesariamente; y también, posiblemente, la asignación de puertos y/o conectores.

5. Conclusiones

Como ha podido verse, el modelo dual de composición invasiva descrito en este artículo cumple las condiciones descritas en un principio. En efecto, todos los detalles relevantes en un modelo de aspectos pueden describirse con este enfoque, de modo que incluso se pueden combinar elementos procedentes de propuestas distintas; no existe ningún matiz que no se haya podido expresar, y en ningún caso la correspondencia conceptual resulta forzada. Por otra parte, sólo se han usado elementos estándar de UML 2.1. No ha sido necesario extender el metamodelo o proporcionar estereotipos. La única función que se podría dar a estas extensiones sería la de meros sinónimos: una hipotética metaclase o estereotipo *Aspecto* no sería más que otro nombre para *ConnectableElement*.

El modelo dual expuesto no contradice, en principio, a ninguna propuesta existente para la definición de aspectos. En primer lugar, la mayoría de ellas se definen a nivel de implementación, por lo que ésta podría ser una más que aceptable notación de diseño para la mayoría, si bien es cierto que no está diseñada para aprovechar las características más avanzadas de algunos modelos (como las *pointcut expressions* de AspectJ). Las que se definen también a nivel de diseño son generalmente compatibles; de hecho, estudiar sus correspondencias podría, probablemente, ayudar aclarar su semántica. Finalmente, no se ha explorado en detalle su relación con otras propuestas complejas, y en principio compatibles; resulta interesante, en particular, plantear el papel que podría darse a los patrones de composición de Theme/UML [4].

Como única objeción, se podría argumentar que, si bien se describe la correspondencia y combinación de aspectos como roles, el modelo no proporciona un mecanismo para su combinación como clases parciales (o *mixins*), que tiene que ser indicada *a mano* por el propio diseñador. No obstante, ésta no es una objeción real: no todos los modelos hacen un *weaving* automático, y la mayoría se resuelve en tiempo de compilación, igualmente previo al despliegue final; en todo caso, éste es un modelo de *diseño*. En realidad, la esencia de los esquemas de aspectos no es un mecanismo de *weaving*, sino la función de correspondencia que éste implementa, y que se define como un uso-de-colaboración en esta propuesta. Dicho esto, no obstante, se ha de señalar que en todo caso podría usarse el mecanismo descrito por Ammour *et al* [1], basado en un *merge* de paquetes, para dar una solución en este sentido sin tener que modificar ningún detalle.

Agradecimientos

Los autores han sido parcialmente financiados por la Dirección General de Universidades e Investigación de la Comunidad de Madrid y la Universidad Rey Juan Carlos dentro del Proyecto IASOMM (URJC-CM-2007-CET-1555). El trabajo ha sido también financiado por el Ministerio de Educación y Ciencia, dentro de los Proyectos META/MoMent (TIN2006-15175-C05-01) del Plan Nacional de I+D+i y AT (CONSOLIDER CSD2007-00022), dentro del Programa INGENIO 2010.

Referencias

[1] Ammour, S, Desfray, P., "A Concern-Based Technique for Architecture Modelling Using the UML Package Merge". *Electronic Notes in Theoretical Computer Science*, vol. 163, pp. 7-18, Elsevier Science, 2006.

[2] Aßmann, U. *Invasive Software Composition*. Springer, 2003.

[3] Chavez, C., Lucena, C. A., "Metamodel for Aspect-Oriented Modeling" en *AOSD'02 Workshop on Aspect-Oriented Modeling with the UML*, 2002.

[4] Clarke, S.; Walker, R.J., "Generic Aspect-Oriented Design with Theme/UML" en Filman, R., Elrad, T., Clarke, S., Aksit, M., *Aspect-Oriented Software Development*. Addison-Wesley, 2005, pp. 425-458.

[5] Conejero, J.M.; Hernández, J.; Jurado, E.; van den Berg, K. *Crosscutting, What Is and What Is Not? A Formal Definition Based on a Crosscutting Pattern.* Technical Report TR28/07, Ed. Universidad de Extremadura, 2007.

[6] Cuesta, C.E.; Romay, M.P.; de la Fuente, P.; Barrio-Solórzano, M., "Architectural Aspects of Architectural Aspects" en *Lecture Notes in Computer Science, vol. 3527*, págs. 247-262, Springer, 2005.

[7] Cuesta, C.E.; Romay, M.P.; de la Fuente, P.; Barrio-Solórzano, M.; Younessi, H. "Coordination in Architectural Connection: Reflective and Aspectual Introduction" en *L'Objet*, vol. 12, nº 1, pp. 127-151, Hérmes/Lavoisier, 2006.

[8] Filman, R., Elrad, T., Clarke, S., Aksit, M., *Aspect-Oriented Software Development*. Addison-Wesley, 2005

[9] Hanenberg, S., Unland, R., "Roles and Aspects: Similarities, Differences, and Synergetic Potential" en *Object-Oriented Information Systems (OOIS'02)*, pp. 87-94, Springer, 2002.

[10] Hannemann, J., Murphy, G.C., Kiczales, G., "Role-based Refactoring of Crosscutting Concerns" en *Aspect-Oriented Software Development (AOSD'05)*, pp. 135-146, ACM Press, 2005.

[11] Harrison, W.H., Ossher, H.L., Tarr, P.L. *Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition*. IBM Research Report RC22685 (W0212-147), Thomas J. Watson Research Center, IBM, 2002.

[12] ITU-T. *Specification and Description Language (SDL)*. ITU-T Recommendation Z-100, Agosto 2002.

[13] Katz, S., "A Superimposition Control Construct for Distributed Systems", *ACM Trans. on Programming Languages and Systems,* vol. 15, nº 2, pp. 337-356, 1993.

[14] Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W.G., "An Overview of AspectJ", en *Lecture Notes in Computer Science, vol. 2072*, pp. 327-353, Springer, 2001.

[15] Kristensen, B.B., "Object-Oriented Modeling with Roles" en *Object-Oriented Information Systems (OOIS'95)*, pp.. 57-71, Springer, 1996.

[16] Lieberherr, K., Lorenz, D.H., Ovlinger, J., "Aspectual Collaborations: Combining Modules and Aspects", *The Computer Journal*, vol. 46, nº 5, pp. 542-565, 2003.

[17] Masuhara, H., Kiczales, G, "Modular Crosscutting in Aspect Oriented Mechanisms" en *Lecture Notes in Computer Science, vol. 2743*, pp. 2-28, Springer, 2003.

[18] Mezini, M.; Ostermann, K., "Conquering Aspects with Caesar" en *Second Intl. Conf. on Aspect-Oriented Software Development (AOSD'03)*, pp. 70-79, 2003.

[19] OMG. *Unified Modeling Language (OMG UML) Superstructure, V2.1.2*, OMG Available Specification, Document formal/2007-11-02, 2007.

[20] Pawlak, R., Retaillé, J.-P., Seinturier, L., *Programmation Orientée Aspect pour Java/J2EE*. Ed. Eyrolles, 2004.

[21] Pérez, J., Ali, N., Carsí, J., Ramos, I. Dynamic Evolution in Aspect-Oriented Architectural Models en *Lecture Notes in Computer Science, vol. 3527*, 2005.

[22] Reenskaug, T., Wold, P., Lehne, O.A., *Working with Objects: The OOram Software Engineering Method*. Manning Publications, 1995.

[23] Romay, M.P., Cuesta, C.E., de la Fuente, P., Barrio Solórzano, M., "Descripción de Aspectos Mediante Conectores UML 2.0", en *Avances en Desarrollo de Software Orientado a Aspectos*, pp. 57-64, Ed. Universidad de Extremadura, 2004.

[24] Suzuki, J.; Yamamoto, Y. Extending UML with Aspects: Aspect Support in the Design Phase. En *Aspect-Oriented Programming (AOP-ECOOP'99),* 1999.

REICIS

ACTUALIDAD INVITADA

Proceso de selección de productos software en el Ministerio de Defensa

José Gonzalo Delgado de Luque
Comandante, Director de Proyectos Informáticos
Centro de Desarrollo de Software, Área de Tratamiento de la Información
Secretaría General Técnica, Ministerio de Defensa
P. de la Castellana, 109
28071 Madrid
Godelu@gmail.com

Introducción

La consecución de una de las acciones (P2-O6-AC1) que se marca en el Objetivo (P2-O6) "Definición y Diseño de la Arquitectura Técnica y Unificada" del Plan Director de Sistemas de Información y Telecomunicaciones de Ministerio de Defensa, da lugar a la instrucción técnica 01/07. Esta instrucción tiene como objetivo el establecimiento de un Modelo Técnico de Referencia, la identificación de los estándares, y productos que se establecen como de obligado cumplimiento en los desarrollos, actualizaciones e implantaciones de sistemas de información del Ministerio de Defensa.

En esta instrucción se defiende el uso de productos comerciales frente a desarrollos a medida, por razones de coste y de calidad, ya que los productos de mercado evolucionan a un ritmo que ninguna Administración Pública podría mantener mediante desarrollos propios. Para ello se establece el concepto de Arquitectura Técnica Unificada.

Arquitectura Técnica Unificada

La Arquitectura Técnica Unificada se marca como objetivo primordial el obtener una lista de productos comerciales (COTS[1]) que compondrán el núcleo común de los sistemas

[1] Commercial Off-The-Shelf software: software comercial

de información y que llevarán a los objetivos perseguidos de interoperabilidad y ahorro de costes.

Esta Arquitectura Técnica Unificada está fuertemente relacionada con la Arquitectura Técnica de Mando y Control OTAN (NATO C3 Technical Architecture NC3TA), desarrollada por el grupo de trabajo OTAN de sistemas abiertos (NOSWG), perteneciente al quinto subcomité de la estructura del consejo C3 OTAN (NATO C3 Board). Anualmente el NOSWG emite una nueva versión de la NC3TA.

Proceso de selección

Para la elección de los productos software definidos en esta arquitectura, se elabora una lista de estos en cada categoría considerada (Ofimática, Contenidos Web, Integración de Aplicaciones), que bien por su posicionamiento en el mercado o por su relevancia en el entorno de Defensa, se consideran candidatos.

Una vez definida la lista de productos a evaluar en cada categoría, se elabora un exhaustivo estudio de mercado apoyado en el estado actual de las tecnologías de la información y en el dictamen de analistas reconocidos internacionalmente (Meta Group, Forrester, Gartner, etc.), siempre enfocado a las necesidades concretas presentes y futuras del Ministerio.

Con la intención de efectuar los estudios de los productos de la forma más imparcial posible, se define un proceso de selección de productos, basado en razones de índole económica, técnica, y de mercado, con criterios de evaluación predefinidos, uniformes, y de aplicación para todas las categorías de productos. Estos estudios concluyen con la selección de uno o varios productos, elegidos en función de los resultados obtenidos en el proceso de evaluación y de las singularidades del Ministerio de Defensa (aspectos tecnológicos y económicos, compatibilidad con otros sistemas, etc.).

Estos productos se someten a una serie de pruebas que certifiquen su aptitud técnica, la satisfacción de los estándares correspondientes y su capacidad de integración e interoperabilidad con otros productos. Cuando se seleccionan productos que ya se encuentran extensamente implantados en el entorno de Defensa, y en los que el Ministerio cuenta con una amplia experiencia (Lotus Notes, Microsoft Office, etc), no se considera preciso realizar pruebas de viabilidad técnica.

El uso de algún producto diferente al seleccionado, para alguna de las categorías de productos precisada en la Arquitectura Técnica Unificada, deberá ser debidamente razonado, siendo necesario que la IGECIS (Inspección General CIS) autorice previamente su utilización. Pero no por ello se pretende que esta arquitectura sea un sistema cerrado, todo lo contrario, se considera un documento vivo, de tal forma que cualquier organismo del Ministerio de Defensa a través de su célula CIS correspondiente, puede generar una solicitud de cambio.

Acciones complementarias

Además del proceso anterior, aquellos productos para el entorno de Mando y Control, la normativa de seguridad podrá demandar, para productos que implementen funciones de Identificación y Autenticación, así como de Protección de Datos de Usuario y Auditoria de Seguridad, un nivel mínimo de certificación según los estándares obligatorios de evaluación de seguridad en los sistemas (Common Criteria e ITSEC), o bien que estén certificados por el Centro Nacional de Inteligencia (CNI). El nivel mínimo de certificación necesario en cada caso será determinado por la normativa de seguridad correspondiente.

En general, esta lista de productos elegidos se irá actualizando en sucesivas versiones atendiendo a su natural evolución tecnológica en el tiempo. También se prevé que se incremente de acuerdo a estudios o pruebas piloto que se vayan realizando, o a decisiones que se vayan adoptando por los organismos competentes del Ministerio de Defensa.

Referencias

- Orden Ministerial DEF/315/2002, de 14 de febrero, por la que se aprueba el Plan Director de Sistemas de Información y Telecomunicaciones del Ministerio de Defensa.
- Resolución ARQ01/2004 del Inspector General CIS, por la que se establece la Arquitectura Técnica de Propósito General del Ministerio de Defensa.
- Resolución ARQ02/2005 del Inspector General CIS, por la que se establece la Arquitectura Técnica de Mando y Control del Ministerio de Defensa.
- Arquitectura Técnica C3 OTAN (NC3TA v7.0).

Perfil profesional

El Comandante D. José Gonzalo Delgado de Luque (Cuerpo General de las Armas, Escala Superior de Oficiales) es Diplomado en Informática Militar. Actualmente es el Director de Proyectos Informáticos actualmente destinado en el CDS (Centro de Desarrollo de Sistemas) del ATI (Área del Tratamiento de la Información) de la SEGENTE (Secretaría General Técnica) del Ministerio de Defensa.