Universidad Carlos III de Madrid
Escuela politécnica Superior

TESIS DOCTORAL

Metodología para la Organización del Conocimiento Temprano asistido por Aspectos (OCTA)

Autor:
Eduardo Barra Zavaleta

Director:
Dr. Jorge Luis Morato Lara

Departamento de Informática
Doctorado en Ciencia y Tecnología Informática

Leganés, Madrid, España
Octubre de 2015

Universidad Carlos III de Madrid
www.uc3m.es

TESIS DOCTORAL

Metodología para la Organización del Conocimiento Temprano asistido por Aspectos (OCTA)

Autor: Eduardo Barra Zavaleta

Director: Dr. Jorge Luis Morato Lara

Firma del Tribunal Calificador:

Firma

Presidente: (Nombre y apellidos)

Vocal: (Nombre y apellidos)

Secretario: (Nombre y apellidos)

Calificación:

Leganés, Madrid, ____ de _____ de 2015

Para mi Madre

Por enseñarme que la vida es una lucha que se puede ganar y disfrutar

Para mi Padre

Por enseñarme a perseguir mis sueños y no perder la calma en el intento

Para mi Abuela

Gracias por ayudarme a no tener miedo

Para Caro

Con todo mi cariño, por su apoyo incondicional

Para Alessandra y Valeria

Gracias por aguantar a un papá desesperado

"No hay fecha que no se llegue, ni tiempo que no se cumpla."

Refrán popular mexicano

"Todos tenemos la idea perfecta cuando solo la planificamos, sin embargo, solo proporciona satisfacción aquella que se desarrolla aun siendo imperfecta."

Eduardo Barra

Agradecimientos

Ulises fue uno de los héroes legendarios de la mitología griega que luchó en la guerra de Troya durante diez años y otros diez intentando regresar a Ítaca con una serie de problemas y obstáculos que tuvo que afrontar para conseguir su objetivo. Así mismo para conseguir sus metas tuvo la necesidad de pasar por las facetas de juicioso, prudente, cobarde, vengativo, astuto, valiente y esforzado, pero sin olvidar nunca lo que quería conseguir.

Así como Ulises, también tuve mi particular toma de decisiones para resistir y adaptarme ante los obstáculos para finalmente llegar a la culminación de esta tesis doctoral. Sin embargo, es imposible hacer frente solo a los obstáculos de cualquier recorrido.

Por lo tanto, me gustaría que estas líneas sirvieran para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que han colaborado en mayor o menor medida en la realización del presente trabajo, comenzando por mi director de tesis el Dr. Jorge Morato, por apoyarme a continuar mi trabajo a pesar de todas las dificultades.

Especial reconocimiento merece el interés mostrado por mi trabajo y las sugerencias recibidas del Dr. Luis Puente, con el que me encuentro en deuda por el ánimo infundido y la confianza en mí depositada.

La publicación de este trabajo en una revista internacional ha sido posible por el apoyo de diferentes personas, sin embargo, quiero plasmar un especial agradecimiento a la participación del Dr. Christos Dimou.

Quisiera hacer extensiva mi gratitud a mis compañeros investigadores presentes y pasados del grupo de trabajo KR por su amistad y colaboración.

También quiero dar las gracias a los consultores de PYMES, por su colaboración en el suministro de los datos para la realización de la parte empírica de esta investigación.

Así mismo, me gustaría agradecer la ayuda recibida del Dr. Juan Llorens en este recorrido.

Un agradecimiento muy especial merece la comprensión, paciencia y el ánimo recibido de los que además de ser compañeros han sido amigos.

A todos ellos, muchas gracias.

Contenidos

Índice de figuras

Índice de Tablas

Índice de esquemas lingüísticos

Resumen

En la actualidad las grandes organizaciones disponen de entornos heterogéneos con procesos de negocio cambiantes. En consecuencia, las organizaciones actuales solicitan productos software que puedan adaptarse a los cambiantes procesos de negocio sin afrontar el riesgo de invertir muchos recursos en el intento.

Por otro lado, la ingeniería de software es un área de las ciencias de la computación que ha aportado métodos y técnicas con el objetivo de producir software de calidad. Entre los factores que influyen en la calidad, uno de los más importantes es gestionar los cambios a lo largo del ciclo de vida de un producto software. Sin embargo, actualmente las factorías del software carecen de metodologías efectivas para gestionar los recurrentes cambios en el sistema, provocando constantes problemas a lo largo del proceso de desarrollo software.

La propuesta de **orientación a aspectos** es un paradigma de desarrollo de software que se fundamenta en los principios clásicos de la **separación de incumbencias.** Una **incumbencia** es una consideración específica que debe ser tomada en cuenta para satisfacer los propósitos de un producto software. El paradigma orientado a aspectos surgió como una propuesta de programación que permitiera evolucionar la funcionalidad a nivel de código de un producto software. Posteriormente fue rápidamente extendida a las otras etapas del proceso de desarrollo de software hasta llegar a la ingeniería de requisitos.

Con este contexto, en esta tesis doctoral **"Metodología para la Organización del Conocimiento Temprano asistido por Aspectos"** se planteó una hipótesis exponiendo que el paradigma orientado a aspectos podría proporcionar las bases para conseguir la **evolución** eficiente de un producto software en cualquier momento a lo largo de su desarrollo. Al implicar este planteamiento una investigación muy amplia, se limitó el alcance del presente trabajo a las **etapas tempranas** de un proceso de desarrollo software.

La investigación se inició con el planteamiento del problema, encontrando que la separación de incumbencias en etapas tempranas ya es tratada por las propuestas de ingeniería de requisitos orientada a aspectos y arquitectura software orientada a aspectos. Se encontró que dichas propuestas aportan conceptos importantes pero con enfoques inmaduros para gestionar incumbencias.

La investigación continuó con el planteamiento de la solución, proponiendo la realización de un objetivo general con el cumplimiento de cinco objetivos específicos. Concretar estos objetivos proporcionó los **medios que permiten el modelado del conocimiento en las etapas tempranas de productos software** a través de los conceptos del paradigma orientado a aspectos. Esto facilitó la creación de modelos con una organización del conocimiento que permite una evolución más efectiva, evita conflictos y mejora la gestión de la trazabilidad entre incumbencias. Es decir, se observó un producto software con mejor calidad.

El camino recorrido en el desarrollo de los objetivos de esta tesis doctoral tuvo como resultado diferentes aportaciones relevantes que son comentadas a continuación:

La primera aportación relevante de esta tesis doctoral fue el desarrollo de un **"marco de referencia para la conceptualización de aspectos"** con el objetivo principal de identificar claramente los límites de los **aspectos tempranos**. El marco de referencia propone las conceptualizaciones que pueden existir a lo largo de un proceso de

desarrollo software. Por lo tanto, en el mismo marco se identifican las conceptualizaciones de **aspectos medios** y **aspectos finales**. Estos conocimientos están diseñados con la intención de continuar con el planteamiento de la hipótesis inicial en trabajos futuros.

La segunda aportación importante fue el desarrollo de la propuesta de un "**modelo conceptual de aspectos tempranos"** que tiene el objetivo de proporcionar las directrices para estructurar la especificación de las etapas tempranas de un producto software dentro del paradigma orientado a aspectos.

La propuesta del modelado conceptual de aspectos tempranos contiene varios bloques de construcción. El definido como **arquitectura de puntos de vista** es muy importante para la organización semántica pero es el más complejo de obtener. Por este motivo el planteamiento de la siguiente aportación relevante fue desarrollar **"un método para la creación de una arquitectura de puntos de vista"**. El desarrollo de este método se basa en las investigaciones de análisis facetado y garantía literaria.

En el desarrollo de esta tesis doctoral se observó la carencia de los medios para verificar la calidad de una especificación de requisitos, es decir, no era posible evaluar objetivamente la propuesta de ingeniería de requisitos orientada a aspectos que forma parte de la propuesta de este trabajo. Dada la situación descrita, la siguiente aportación fue una aproximación de "**un marco de referencia base para unificar criterios de calidad en el desarrollo de una especificación de requisitos"**.

Este punto fue el momento para el desarrollo del objetivo específico más importante de esta investigación. En él se promete crear una metodología que permita el modelado y organización del conocimiento de las etapas tempranas de un producto software que facilite la gestión de los aspectos tempranos. El resultado fue la propuesta de una **metodología para una organización del conocimiento temprano asistido por aspectos** (OCTA). El desarrollo de esta aportación se basó en los formalismos de la ingeniería ontológica presentada por la ingeniería de conocimiento, así como, en el marco para una conceptualización aspectual, en el modelo conceptual para aspectos tempranos y en el método para la creación de una arquitectura de puntos de vista, propuestos en esta tesis doctoral.

La **metodología OCTA** es la aportación más relevante de esta tesis doctoral. Esta aportación reduce la complejidad de los productos software en las etapas tempranas de un proceso de desarrollo software, facilita la creación de modelos con una alta cohesión semántica y permite una organización del conocimiento efectiva para facilitar su gestión. Estas características mejoran la evolución, reducen los conflictos y mejoran la gestión de la trazabilidad entre incumbencias.

Con el objetivo de evaluar las aportaciones de esta investigación se planteó un conjunto de procedimientos. Estos procedimientos se focalizaron en el análisis de una especificación temprana de un caso de estudio desarrollado a través de la metodología OCTA. En el primer procedimiento se verificó la estandarización y estructura de la especificación. En el segundo procedimiento se validó la inferencia del modelado midiendo la exactitud del modelado semántico a través de consultas de su conocimiento. En el tercer procedimiento se evaluó la flexibilidad al cambio de un modelado OCTA comparándola contra dos diferentes especificaciones creadas con un método clásico basadas en el mismo caso de estudio. El resultado de estos procedimientos, demostró que la solución innovadora que se propone en este trabajo reduce el impacto al cambio en las etapas tempranas sin el riesgo de invertir muchos recursos en el intento.

Abstract

Nowadays large organizations have heterogeneous environments with changing business processes, and require software products that can be adapted to such processes without facing the risk of investing too many resources in the attempt.

On the other hand, the software engineering is a computer science field that has contributed with methods and techniques aiming to produce quality software. One of the most important factors that impact the quality of software is the management of changes along the software product lifecycle. However, at the present time the software factories lack of effective methodologies to manage recurrent changes in the software products, generating constant problems along the software development process lifecycle.

The **aspect-oriented** proposal is a software development paradigm based on the classic principles of **separation of concerns**. A **concern** is a specific consideration that should be taken into account in order to satisfy the purpose of a software product. The aspect-oriented paradigm emerged as a programming proposal that allowed evolving the functionality of a software product at a code level. Later, it was quickly extended to the other software development process stages up to the requirements engineering.

In this context, in this doctoral thesis **"Methodology for the early knowledge organization assisted by Aspects"**, it was propose a hypothesis sustaining that the aspect-oriented paradigm could provide the basis to efficiently evolve a software product at any point in the software development process lifecycle. Due to this idea involves a wide research; the scope of the present work was limited to the **early stages** of the software development process.

The research work began with the problem statement; it was found that the separation of concerns in early stages was already taken into account in the aspect-oriented requirements engineering and software architecture proposals. It was found that those proposals contribute with relevant concepts but with immature approaches to mange concerns.

The research continued with the solution statement; it was proposed the accomplishment of a generic goal comprised of five specific goals. The achievement of these goals provided the **means to model the knowledge in early stages of software products** through the concepts of the aspect-oriented paradigm. Likewise, it facilitated the models creation with a knowledge organization that allowed a more effective evolution; to avoid conflicts; and to improve traceability management between concerns, i.e., to get a software product with better quality.

As a result of meeting the doctoral thesis goals, several relevant contributions were made. Those contributions are presented below:

The first contribution was the development of a **"concerns conceptualization reference framework"** aiming to clearly identify early aspects limits. The reference framework proposes the conceptualizations that can exist through the software process development. Therefore, in the framework the **medium aspects** and **final aspects** conceptualizations are identified. This knowledge has been designed with the intention of continuing with the same approach in future research.

The second contribution was the proposed of an **"early aspects conceptual model"** with the aim of provide the guidelines to structure the early stages specification of software product under the aspect-oriented paradigm.

The early aspects conceptual model proposal contains several construction blocks. The one defined as **points of view architecture** is very important for the semantic organization but the most complex to obtain. Therefore, the next contribution was a **"point of view architecture development method"**. The development of this method was based on faceted analysis and literary warrant researches.

Along the development of this doctoral thesis it was observed the lack of means to verify the requirements specification quality, i.e., it was not possible to objectively evaluate the aspect-oriented requirements engineering proposal develop in this work. The next contribution was an approximation to a **"quality criteria unification reference model for the requirements specification development"**.

At this point the development of the most important specific goal began. It was the development of a methodology for modelling and organizing the software product early stages knowledge that facilitates the early aspects management. The result was the proposal of an **aspect aid methodology for early knowledge organization** (OCTA). The development of this contribution was based on the ontological engineering formalisms included in knowledge engineering, as well as the concerns conceptualization reference framework, early aspects conceptual model, and the point of view architecture development method proposed in the doctoral thesis.

The **OCTA methodology** is the most relevant contribution in this doctoral thesis. By using it, it is possible to reduce the software products complexity in the software process development early stages, to ease the models creation with a high semantic cohesion, and an effective knowledge organization to facilitate its management. These characteristics improve the evolution, reduce the conflicts and improve the traceability management between concerns.

In order to evaluate the contributions of this research, a set of procedures were proposed. These procedures were focused in the analysis of an early specification, from a case study, developed with the OCTA methodology. In the first procedure it was verified the specification standardization and structure. In the second procedure it was validated the modelling inference by measuring the semantic modelling correctness through queries to its own knowledge. In the third procedure it was evaluated the change flexibility to an OCTA modelling, comparing it with two different specifications, based on the same case study, that were created with a classic model. The result of these procedures proved that the innovative solution proposed in this work reduces the impact due to changes in the early stages without the risk of investing too many resources in the attempt.

CAPÍTULO I
INTRODUCCIÓN

Capítulo II. INTRODUCCIÓN

1 Motivación de la investigación

En la actualidad las grandes organizaciones disponen de entornos heterogéneos con diferentes sistemas para cumplir con los objetivos de sus procesos de negocio. Los productos software son las herramientas fundamentales para simplificar y optimizar estos procesos de negocio para mejorar su productividad. Sin embargo, la sociedad actual al estar caracterizada por los continuos avances tecnológicos, así como influenciada por la tendencia a la globalización económica y cultural, hace que el entorno de las organizaciones sea cambiante (Galofre, 2013). Por lo tanto, se ha convertido en una de las mayores prioridades de las organizaciones solicitar productos software que puedan adaptarse a los cambiantes procesos de negocio.

Con este escenario, durante los últimos años una de las mayores preocupaciones de las factorías de productos software está relacionada con atender de forma eficiente las solicitudes relacionadas con la necesidad de evolucionar estos productos. Esto crea la necesidad de desarrollar software que soporte modificaciones en el código, el diseño, el análisis y la documentación asociada. En esta línea, el principal problema al que se enfrentan las actuales metodologías de desarrollo es el alto porcentaje de posibilidades de cambios en el tiempo a lo largo del ciclo de vida de estos productos software. Esto se debe a la dependencia del contexto, de las diferentes actividades del desarrollo y de los diferentes roles de los interesados en el sistema, donde la variabilidad de dichos factores a menudo implica cambios dramáticamente diferentes que pueden producir a su vez nuevos problemas, provocando que el comportamiento final no sea el esperado y aumentando considerablemente las posibilidades de fracaso.

Por otro lado, en el cada vez más diversificado mundo de la Ingeniería de Software, los avances más importantes en el desarrollo de software complejo se han obtenido con las diferentes propuestas basadas en el principio de **separación de incumbencias**. En su forma más general, dicho principio se refiere a especificar por separado las diversas **incumbencias**[1] (del inglés *concerns*) de un sistema (Elrad, Filman, & Bader, 2001). Sin embargo, la actividad compleja de identificar, dividir en módulos, encapsular, y manipular **incumbencias** de un producto software, que son relevantes a un concepto particular, a una meta o un propósito ha sido denominada **separación avanzada de incumbencias** (Brichau, Glandrup, Clarke, & Bergmans, 2002). A este respecto, se ha dicho que una incumbencia es una consideración específica que debe ser tomada en cuenta para satisfacer el propósito de un producto software (Filman, Elrad, Clarke, & Aksit, 2004). Por lo tanto, podemos decir que un producto software es la implementación o realización de un conjunto de incumbencias interrelacionadas. Sin embargo, la separación avanzada de incumbencias (del inglés *advanced separation of concerns*) depende mucho de la etapa del ciclo de vida en donde se conceptualiza y del punto de vista bajo el cual es creada una incumbencia.

Una de las aplicaciones más aceptadas del principio de separación avanzada de incumbencias se encuentra en el paradigma Orientado a Objetos (del inglés *Object Oriented*). Este establece un encapsulamiento de incumbencias funcionales a través de una entidad con estructura y comportamiento (*clase*), que captura tanto datos como

[1] Obligación y cargo de hacer algo (definición de la Real Academia Española)

operaciones con un mismo interés. Sin embargo, entre dichas entidades se dispersan incumbencias de interés, cuyo aislamiento resulta imposible con las técnicas convencionales de modelado del paradigma Orientado a Objetos (OO).

En la búsqueda de la gestión de estas **incumbencias transversales** (del inglés *crosscutting concerns*), una propuesta que dio un impulso importante a las investigaciones en la separación avanzada de incumbencias fue la **Programación Orientado a Aspectos** (del inglés *Aspect Oriented Programming*) (Gregor Kiczales et al., 1997). Dicha propuesta surgió como una posible extensión que podría ser aplicable a cualquiera de los diferentes paradigmas de programación. Sin embargo, la Programación Orientada a Aspectos (POA) solo ha trascendido a través de la simbiosis con el paradigma OO. Prometiendo que esta simbiosis permite a los desarrolladores de productos software gestionar incumbencias transversales desde un mismo punto de vista a través de una sola entidad de primera clase conocida como **Aspecto**. Asegurando que dichas entidades quedan separadas de las incumbencias que pueden ser abstraídas y compuestas claramente en las entidades conocidas como *"clases"* de la OO. Así como, proporcionar un medio para entretejer el conjunto de incumbencias (transversales y generales) en un proceso de composición que genera el producto software final.

En esta línea, es conocido por todos que el paradigma OO es aplicado con mucho éxito a lo largo de todo un ciclo de vida de un producto software con el apoyo de herramientas en cada una de las diferentes etapas de un proceso de desarrollo software. Con esta base, la POA después de dar sus primeros pasos en la etapa de codificación de forma natural ha ido extendiéndose a las otras etapas del proceso de desarrollo software. Surgiendo diferentes enfoques que pretenden proporcionar los medios para gestionar las incumbencias transversales que aparecen en los diferentes modelos creados a lo largo de un ciclo de vida de un producto software. En la actualidad se habla de **Desarrollos de Software Orientados a Aspectos** (DSOA) a todas aquellas propuestas de procesos de desarrollo software OO que aplican técnicas y métodos del paradigma Orientado a Aspectos (OA).

1.1 Hipótesis

Con el objetivo de evolucionar el software respetando su integridad en cualquier momento a lo largo de su ciclo de vida, se razonó sobre la separación avanzada de incumbencias. Este es un paradigma que proporciona los medios para soportar un modelado multidimensional con la intención de reducir la complejidad. Sin embargo, la evolución de este tipo de modelados requiere una gestión efectiva de la trazabilidad para asegurar la integridad de su semántica.

El DSOA agrupa diferentes propuestas basadas en la separación avanzada de incumbencias utilizando el concepto de "aspecto". Algunas de estas reflejan los beneficios en la utilización de los conceptos de la OA pero con carencias al modelar la semántica de forma efectiva para evolucionar el software en cualquier momento a lo largo de su ciclo de vida.

En este contexto, como base de una investigación se planteó la siguiente **hipótesis** con la meta de confirmar su validez:

Un producto software OO que sea modelado a través de un DSOA eficiente proporciona los medios para **enfrentar los continuos cambios** de información y funcionalidad de los modelos de negocio del entorno del cliente y de los desarrolladores **en cualquier momento del ciclo de vida** sin afrontan el riesgo de invertir muchos recursos en el intento.

1.2 Alcance de tesis doctoral

En base a la hipótesis de esta tesis doctoral se planteó una investigación con el objetivo inicial de proporcionar los medios para la evolución de un producto software OO en cualquier momento a lo largo de su ciclo de vida a través del apoyo del paradigma OA. Pretender investigar e innovar aplicando esta idea en todas las etapas de un Proceso de Desarrollo Software (PDS) fue un objetivo demasiado ambicioso para intentar cubrirlo en esta investigación. Por lo tanto, se requirió acotar la investigación a un alcance con una carga de trabajo aceptable para una tesis doctoral.

El alcance de esta tesis doctoral fue acotado a **las etapas tempranas de los desarrollos software**. La selección de estas etapas está basada en el objetivo de obtener los resultados más significativos en la aplicación de esta propuesta.

En la decisión del alcance de esta tesis doctoral se tuvo en cuenta cómo los modelos de mejora de calidad dirigidos a los PDS, tales como "Capability Maturity Model Integration" CMMI (CMMI_Product_Team, 2002), establecen la importancia de las **etapas tempranas** como objetivo general que sirve de guía a las factorías de software para evaluar el nivel de madurez de sus procesos.

A este respecto, desde un nivel de abstracción alto se suele hablar de etapas tempranas y etapas finales en un desarrollo software, limitando las etapas finales a partir de la etapa de bajo nivel codificación. Sin embargo, nunca está claro qué etapa de alto nivel es antes de las etapas finales y tampoco en qué etapa de bajo nivel limita el final de las etapas tempranas. El primer paso en cualquier proceso de desarrollo software es capturar la naturaleza del producto software a construir para su posterior análisis. Esta etapa se encuentra en la disciplina conocida como Ingeniería de Requisitos (IR) y por lo tanto se convierte en la primera pieza de las etapas tempranas. Ahora bien, los modelados resultantes de la **Ingeniería de Requisitos** (IR) guían de forma directa a la selección o descomposición en subsistemas del producto software y como se relacionan entre ellos, para llevar a cabo la funcionalidad especificada e introducir los factores de calidad que requiere el sistema software (reutilización, robustez, adaptabilidad, etc.). Esta segunda pieza complementaria de las etapas tempranas está contemplada dentro de la disciplina conocida como **Arquitectura Software** (AS).

2 Planteamiento del problema

En esta sección se describe el desarrollo del planteamiento del problema de esta tesis doctoral en base a la hipótesis acotada originada por la motivación. El planteamiento del problema implica la revisión de la problemática asociada a las actuales investigaciones que de alguna u otra manera plantean proporcionar los medios para resolver el problema de evolucionar un producto software en las etapas tempranas.

Los DSOA carecen de un metamodelo OA subyacente formal que cubra todo el ciclo de vida de un producto software. Las propuestas de metamodelos actuales están enfocadas a las etapas finales, descartando las etapas tempranas.

El alcance de esta tesis doctoral se acotó a las etapas tempranas de un proceso de desarrollo software, por lo tanto, se procedió a realizar un estudio de las diferentes propuestas basadas en la OA enfocadas a las etapas tempranas. El resultado del estudio arrojo que la mayor carencia de las propuestas basadas en el paradigma OA es proporcionar los medios para una efectiva gestión del conocimiento.

La observación de esta carencia implicó realizar un estudio de las propuestas de ingeniería del conocimiento en las etapas tempranas de un desarrollo software. El

estudio realizado permitió identificar la problemática de organización semántica de los modelos creados en estas etapas.

La identificación de estos problemas dio pie al planteamiento de los objetivos de la presente tesis doctoral.

2.1 Estudio de propuestas de metamodelos para la OA

Es conocido que los procesos de desarrollo que apoyan la producción de software de gran tamaño, requieren que sean incrementales e iterativos para potenciar el desarrollo en paralelo de los distintos flujos de trabajo, y por tanto el trabajo en paralelo de distintos equipos de personas. Este es un problema al que se enfrentan las actuales metodologías propuestas para un DSOA, debido a la falta de un marco de trabajo que proporcione las directrices para la creación de modelos aspectuales interdependientes sin una precedencia temporal estricta a lo largo de un ciclo de vida.

Por otro lado, en todo proceso de desarrollo software se requiere la utilización de diferentes lenguajes en cada una de las etapas para la creación de modelos. Esta necesidad es la misma en la aplicación de un DSOA con el desafío de documentar el modelado de incumbencias en diferentes artefactos correspondientes a cada una de las etapas de un proceso de desarrollo software OA. Este proceso implica la necesidad de la gestión de la trazabilidad entre aspectos e incumbencias para modelar una etapa, así como trazabilidad entre modelos de diferentes etapas. Sin embargo, diferentes propuestas de DSOA están centradas en diferentes etapas del ciclo de vida de forma independiente, proponiendo diferentes lenguajes para el modelado de aspectos en cada etapa, así como variaciones en la forma de plasmar los principios de la OA.

En este contexto, el primer paso en esta tesis doctoral fue el análisis de investigaciones basadas en los conceptos base de la OA en donde se planteara un metamodelo enfocado a cubrir todo el ciclo de vida de un DSOA. Sin embargo, era necesario seleccionar propuestas que promuevan una comunicación efectiva para permitir la integración de las actividades dentro de la red de excelencia en las diferentes propuestas de DSOA. Por lo tanto, se seleccionaron propuestas basadas en la terminología común de la AOSD-Europe (Van den Berg, Conejero, & Chitchyan, 2005) para asegurar que apuntan a una visión más integradora.

Las diferentes propuestas de metamodelos que fueron tomadas en cuenta tienen dos enfoques distintivos. Las propuestas de "Towards Visual AspectJ by a Meta Model and Modeling Notation"(Han, Günter, & Cremers, 2005) y "Towards a Common Reference Architecture for Aspect-Oriented Modeling" (Schauerhuber, Schwinger, Wieland Kapsammer, Elisabeth Retschitzegger, & Wimmer, 2006) se basan en abstraer la esencia del lenguaje de programación más aceptado para la etapa de codificación AspectJ y plasmarla en un metamodelo para posteriormente intentar adaptarlo a las etapas de análisis y diseño. Por otro lado, "A Generic MOF metamodel for aspect-oriented modelling" (Fuentes & Sánchez, 2006) es una propuesta que presenta un metamodelo para crear metamodelos de los métodos planteado en diferentes etapas de un proceso de desarrollo software.

En el análisis de las diferentes propuestas de metamodelos para un DSOA se encontró que el problema general de estos planteamientos es prescindir de las etapas tempranas de un proceso de desarrollo software.

2.2 Estudio de propuestas de IROA

Los grupos de investigación de la Ingeniería de Software (IS) por lo general manifiestan que buena parte de los problemas que surgen a lo largo de un Proceso de Desarrollo de Software (PDS) se deben a la carencia de métodos y técnicas eficientes para la representación y entendimiento del problema, además de la descripción y/o especificación poco clara de las necesidades del cliente. La Ingeniería de Requisitos (IR) es la rama de la IS que se ocupa de la captura de requisitos para su análisis permitiendo representar el entendimiento del problema. Es decir, la primera parte de las etapas tempranas de un desarrollo software.

Por lo tanto, la **Ingeniería de Requisitos Orientada a Aspectos** (IROA) (del inglés *Aspect-Oriented Engineering Requirement* -AORE) (Grundy, 1999) (Rashid, Sawyer, Moreira, & Araújo, 2002) surge como una práctica natural en los proyectos de DSOA.

Con este escenario, el siguiente paso en esta investigación fue estudiar los enfoques más representativos en la IROA. La primera propuesta identificada fue "Modularisation and composition of aspectual requirements" (Rashid, Moreira, & Araújo, 2003). En esta propuesta se puede destacar el planteamiento de utilizar el lenguaje XML para indexar la descripción de los requisitos para la gestión de las incumbencias contenidas. En la siguiente propuesta "Multi-Dimensional Separation of Concerns in Requirements Engineering" (Moreira, Rashid, & Araujo, 2005) se encontró la introducción de la propuesta de realizar una separación multidimensional en la IR, consiguiendo el modelado de diferentes puntos de vista. La propuesta de "From Goals to Aspects: Discovering Aspects from Requirements Goal Models" (Yu, do Prado Leite, & Mylopoulos, 2004) plantea utilizar técnicas de orientación a objetivo para evitar conflictos. Las propuestas "Theme: an approach for aspect-oriented analysis and design" (E. Baniassad & Clarke, 2004) y "Aspect-Oriented Software Development with Use Cases" (Jacobson & Ng, 2005) fueron encontradas en libros dedicados de forma específica a estas investigaciones. Por lo tanto, proporcionan una explicación exhaustiva de sus métodos y técnicas. Ahora bien, de la propuesta de Baniassad y Clarke se puede destacar los medios propuestos para descomponer los requisitos en otros más elementales. En el caso de la propuesta de Jacobson al relacionar los conceptos del paradigma OA con "casos de uso" se puede observar la influencia positiva para la identificación de incumbencias.

En el análisis de estas propuestas de IROA se encontró que en todos los planteamientos se consideran los requisitos como incumbencias. Por lo tanto, ayudan a la separación avanzada de incumbencias. Sin embargo, al considerar como incumbencias cualquier requisito con cualquier estructura semántica genera el problema de no tener los medios de razonar los efectos del cambio de una forma eficiente.

2.3 Estudio de propuestas ASOA

En esta investigación se expone que las etapas tempranas de un proceso de desarrollo software se componen en una primera parte por la IR, siendo complementada con el modelado de la Arquitectura Software (AS). La AS es una de las áreas fundamentales de los modernos sistemas de software, dado que es la responsable de concretar la propuesta del modelado de la solución funcionalidad y de los factores de calidad como la robustez, la reutilización y la evolución.

Es sabido que se debe justificar las decisiones tomadas en el modelado de la arquitectura software en base a una ER relacionada. Por lo tanto, al igual que la IR, en el modelado de la arquitectura software es necesario gestionar las incumbencias. Así, los arquitectos tienen la posibilidad de razonar los efectos de los cambios de las incumbencias

transversales en toda la etapa de aspectos tempranos. La separación de este tipo de incumbencias tempranas ya es tratada por las propuestas de la **Arquitectura Software Orientada a Aspectos** (del inglés *Aspect-Oriented Software Architecture, AOSA*).

En este contexto, la continuidad de la investigación corresponde al estudio de propuestas de Arquitectura Software Orientada a Aspectos (ASOA). En esta línea, se pueden encontrar en el contenido de la norma IEEE 1471 (IEEE-Computer-Society, 2000) conceptos relacionados con la separación de incumbencias que podrían ser considerados dentro del paradigma OA. Por lo tanto, se estudiaron las propuestas que han surgido de Arquitectura Software Orientada a Aspectos (ASOA) complementando las etapas tempranas de un DSOA.

En la primera propuesta "Theme/UML" (Elisa Baniassad & Clarke, 2005) se encontró el planteamiento de trazabilidad con la IROA pero con métodos complejos para su realización. En la segunda propuesta "Aspect-Oriented Software Development with Use Cases" (Jacobson & Ng, 2005) de alguna manera buscan la trazabilidad al proponer un refinamiento desde la organización en la correspondiente propuesta de IROA. Sin embargo, los modelados de ASOA resultantes son clases detalladas en lugar de componentes o capas. La tercer propuesta "Aspectual Software Architecture Analysis Method" (ASAAM) (Tekinerdogan, 2004) se centra en el modelado de la parte dinámica de los aspectos. La última propuesta "Driving and Managing Architectural Decisions with Aspects" (Garcia, Batista, Rashid, & Sant'Anna, 2006) propone modelar las incumbencias transversales separada de la AS para tener una guía.

En el análisis de las propuestas de ASOA cada una proporciona conceptos importantes a tomar en cuenta para el desarrollo de una ASOA. Sin embargo, ninguna de las propuestas proporciona los medios concretos para la gestión semántica de las incumbencias y la traza con la IROA.

2.4 Estudio de propuestas de IC en IR

Las diferentes propuestas estudiadas para las etapas tempranas de un DSOA se centran en la separación y composición de incumbencias transversales pero no están orientadas a la evolución de un producto software. En esta línea, un producto software OO proporciona los medios para evolucionar cuando es posible gestionar el conocimiento de forma efectiva de su modelado. Sin embargo, en los diferentes planteamientos de la IROA y ASOA se encontraron problemas para gestionar la semántica de las incumbencias.

En su forma más general la **Ingeniería del Conocimiento (IC)** se puede describir como la sistematización de la información basada en un lenguaje común con el objetivo de compartir conocimiento, uniendo incumbencias que parecen distintas para finalmente crear nuevo conocimiento para un determinado propósito. Como parte de la IC surgió la Ingeniería Ontológica (IO) basada en la propuesta del concepto de ontología. Las ontologías son el formalismo más utilizado para la representación del conocimiento. Estas constituyen una especificación explícita y formal de abstracciones mentales relacionadas. Así mismo, se conforman mediante el acuerdo de una comunidad experta en un dominio y un diseño para un propósito específico.

En este sentido, se planteó la necesidad de realizar un estudio de las propuestas de IC en las etapas tempranas de un desarrollo software. De esta forma, aparece la representación de conocimiento a través de ontologías como un factor importante para lograr una eficiente gestión de requisitos basada en el razonamiento.

La primera propuesta "Using domain ontology as domain knowledge for requirements elicitation" (Kaiya & Saeki, 2006) se plantea un método para el análisis de requisitos basado en la técnica de la ontología de dominio. La mayor aportación es la intención de modelar parte de la semántica de la especificación de una lista de requisitos apoyado de técnicas de descomposición léxica. Sin embargo, no están resueltos los problemas del procesado de lenguaje natural. En la segunda propuesta "Revisiting the Core Ontology and Problem in Requirements Engineering" (Jureta, Mylopoulos, & Faulkner, 2008) plantea una ontología para la clasificación rígida de requisitos basada en el tipo de comunicación de los interesados en el sistema y el ingeniero. En la siguiente propuesta "Modelling Reusable Security Requirements based on an Ontology Framework" (Velasco, Valencia-Garcia, Fernandez-Breis, & Toval, 2009) se propone un marco de trabajo para la representación de requisitos de seguridad. Este marco se basa en la combinación de una ontología de análisis de riesgo y una ontología de dominio de requisitos. La última propuesta "Using Security and Domain ontologies for Security Requirements Analysis" (Souag, Salinesi, Wattiau, & Mouratidis, 2013) es parecida a la anterior pero las ontologías utilizadas para guiar el análisis de requisitos existen previamente.

En el análisis de estas propuestas de IC se encontró que no plantean técnicas claras de descomposición léxicas para extraer la semántica de las descripciones de requisitos. Por otro lado, la idea de tener conocimiento del dominio a través de una ontología de dominio es un factor recurrente para conseguir una organización semántica de requisitos. Sin embargo, no exponen cómo crear estructuras de organización para la creación de una ER para cualquier línea de producto.

2.5 Problemas de las propuestas actuales

Se ha expuesto que esta tesis doctoral inició una investigación con el objetivo de proporcionar los medios para conseguir la evolución de un producto software OO con un alcance acotado a las etapas tempranas de un desarrollo software a través del apoyo del paradigma OA.

La búsqueda de este objetivo inició con el estudio de diferentes propuestas de metamodelos para DSOA, encontrando que el problema general de estos planteamientos es prescindir de las etapas tempranas de un proceso de desarrollo software. Por lo tanto, no se tiene un **marco de referencia** que afecte a las etapas tempranas del ciclo de vida de un DSOA.

En el análisis de estas propuestas de IROA se ha encontrado que las propuestas consideran como incumbencias cualquier descripción de requisito. Es sabido que un requisito típico está desarrollado en lenguaje natural. Esto genera el problema de no tener los medios de razonar los efectos del cambio semántico interno de un requisito de una forma eficiente. En esta línea, en el análisis de las propuestas de ASOA ninguna de las propuestas proporciona los medios concretos para la gestión semántica de las incumbencias y la traza con la IROA. Por lo tanto, entre sus mayores problemas en el estudio realizado a las propuestas de IROA y ASOA se encuentra el no proporcionar técnicas o métodos para **gestionar los cambios** efectivos de las incumbencias en cualquier momento de su desarrollo.

En un DSOA es frecuente encontrar situaciones en las que un modelo está afectado por más de un aspecto en un mismo punto. Por lo tanto, un determinado punto de unión está asociado a diferentes aspectos, donde cada uno de los aspectos puede adicionar diferentes comportamientos, provocando una interacción entre estos, cuyo resultado no es consistente ni predecible. De esta manera, un problema aparejado fue identificar que

las propuestas estudiadas de IROA y ASOA no proponen métodos para **evitar conflictos** entre las incumbencias de diferentes aspectos.

Ahora bien, en un DSOA los modelos creados con métodos de la IROA deberían tener una trazabilidad con los modelos creados con métodos de ASOA estrechamente ligada a la representación semántica. En las propuestas analizadas existen un par de intentos de trazabilidad (E. Baniassad & Clarke, 2004) (Jacobson & Ng, 2005). Sin embargo, estas propuestas carecen de medios semánticos para gestionar la **trazabilidad** en las etapas tempranas.

En general el análisis de las propuestas de IC se encontró que no plantean técnicas de descomposición léxica para extraer el conocimiento ni los medios para **modelar la semántica** de las descripciones de requisitos.

Por último, las diferentes propuestas de IC en las etapas tempranas proponen una ontología de dominio para conseguir una organización semántica de requisitos. Sin embargo, no exponen métodos parar crear **estructuras de organización del conocimiento** para la creación de una ER específica.

3 Planteamiento de la solución

El planteamiento del problema se complementa con una propuesta de solución, enmarcando así el desarrollo de la investigación propuesta en esta tesis doctoral. Así, el planteamiento de la solución inicia con el objetivo general basado en la hipótesis acotada y el desarrollo del problema de la solución.

Objetivo general. Proporcionar los medios necesarios que permitan el modelado del conocimiento en las etapas tempranas de productos software OO a través de los conceptos del paradigma OA para mejorar la gestión de la trazabilidad entre incumbencias, evitar conflictos y facilitar una evolución más efectiva.

Ahora bien, este objetivo general es el contenedor de un grupo de sub-objetivos, es decir, el cumplimento del objetivo general es la realización de un grupo de objetivos específicos encontrados en el desarrollo del planteamiento de la solución.

Inicialmente se observaron diferentes niveles de conceptualización de aspectos a lo largo de un DSOA, por lo tanto, surgió la necesidad de un marco de trabajo que permitiera crear modelos con líneas claras de separación entre los diferentes niveles de conceptualización. Una meta importante a cumplir en el marco de referencia fue que, sin importar las diferentes formas de ver las etapas de un ciclo de vida en los distintos Procesos de Desarrollo Software (PDS), siempre se puedan conceptualizar los aspectos en los mismos niveles. Así, analizando distintos paradigmas, se plantearon tres diferentes niveles de conceptualización de aspectos a lo largo de un DSOA: **Aspectos Tempranos**, los cuales se especifican desde la ingeniería de requisitos hasta las primeras líneas de solución de arquitectura; **Aspectos Medios**, los cuales se especifican en estructuras aspectuales al nivel del diseño; **Aspectos Finales**, los cuales se especifican en lenguajes de programación especializados para la implementación.

En este contexto el **primer objetivo** específico de esta tesis doctoral fue delimitar las diferentes conceptualizaciones de aspectos en las distintas etapas a lo largo de un PDS.

Con la premisa de la conceptualización de aspectos se puede decir que esta tesis doctoral es una **investigación e innovación** efectiva en el nivel de conceptualización de **aspectos tempranos**, para **solucionar los problemas** actuales en las etapas tempranas de un desarrollo software**.**

El planteamiento de los niveles de conceptualización de aspectos para cualquier PDS prepara el terreno para que los modelos de un mismo nivel conceptual se modelen con una misma semántica. Esto permite disminuir la influencia de los cambios en el tiempo de un producto software a lo largo de su ciclo de vida. Sin embargo, es necesario unificar la conceptualización del paradigma OA a lo largo de las diferentes etapas de un DSOA para proporcionar los medios que permitan la trazabilidad entre las diferentes etapas. Para conseguir este objetivo es necesaria la aceptación de un metamodelo que respete los conceptos y relaciones básicas del paradigma OA en cada una de las etapas de un DSOA.

En el estado de la cuestión se estudiaron diferentes propuestas de metamodelos para el DSOA, encontrando en ellas el descarte del modelado de las etapas tempranas bajo el paradigma OA. Sin embargo, se encontró que las propuestas realizaron un esfuerzo en el análisis de las partes importantes del paradigma OA en la etapa de codificación. Este trabajo aporta las bases para trabajar en los elementos que siempre deberían estar presentes en las otras etapas del ciclo de vida del producto software.

Teniendo en consideración lo anterior, se percibió la necesidad de desarrollar un modelo de referencia de Aspectos Tempranos (AT) en donde además de los elementos básicos seleccionados en base a las propuestas de metamodelos estudiados, es necesario añadir los conceptos básicos relacionados con la arquitectura software lógica, complemento de los aspectos tempranos.

Por lo tanto, el **segundo objetivo** específico de esta tesis doctoral fue proporcionar una forma sencilla y elegante de identificar los conceptos OA en las etapas tempranas de un desarrollo software.

En esta propuesta los aspectos tempranos son conjuntos de incumbencias tempranas que están orientadas hacia la descomposición de los dominios del problema, con propuestas de las primeras líneas de solución, buscando eliminar fallas de interpretación, mejorar la relación entre usuarios y desarrolladores, y estructurar sistemas complejos a través de subsistemas, módulos o elementos simples de una forma más natural.

Por lo tanto, la primera parte de los aspectos tempranos está concretada en la especificación de requisitos; así es como en esta tesis se consideran los requisitos como incumbencias tempranas. En una especificación de requisitos es habitual separarlos de acuerdo a diferentes criterios. Por lo general, los analistas parten de los modelos de procesos del negocio y discriminan de acuerdo a las funciones del dominio del problema, para así identificar y agrupar los requisitos en los clásicos temas funcionales, no funcionales y restricciones. Otras veces, se logra una separación que permite determinar con mayor exactitud la arquitectura del sistema, agrupando las incumbencias en los temas de reglas de negocio, requisitos de información y servicios.

En esta tesis doctoral se propone la posibilidad de lograr otro tipo de separación más efectiva, iniciando con la agrupación de los requisitos identificados como dominantes que son relativamente fáciles de modelar y los requisitos como **transversales** que comúnmente son difíciles de extraer y soportar en módulos independientes, dado que se encuentran dispersos o enmarañados en diferentes requisitos **dominantes** del sistema. Las incumbencias que tienen una naturaleza transversal, la mayoría de las veces no son abstraídas en esta etapa temprana o son descritas de forma incompleta y solapadas con las incumbencias dominantes. En consecuencia estas incumbencias tempranas no logran ser transformadas fácilmente a las siguientes etapas del ciclo de vida de un producto software.

Ahora bien, la otra parte complementaría de los aspectos tempranos es el modelado de la arquitectura software. Este área de investigación es una de las fundamentales de los modernos sistemas de software, dado que es la responsable de concretar la propuesta del modelado de la solución funcionalidad y de los factores de calidad como la robustez, la reutilización y la evolución. Esto implica que los arquitectos de software se enfrenten cada día a decisiones que tienen un amplio alcance de impacto. Por lo tanto, es importante documentar las decisiones de arquitectura de manera sistemática, relacionando los motivos de cambios y sus consecuencias para exponer la mejor propuesta.

En el planteamiento de esta tesis doctoral, se debe justificar las decisiones tomadas en el modelado de la arquitectura software en base a una especificación de requisitos relacionada. Por lo tanto, al igual que la especificación de requisitos, en el modelado de la arquitectura software es necesario identificar las incumbencias dominantes y transversales. Así, los arquitectos tienen la posibilidad de comunicarse y razonar los efectos de los cambios de las incumbencias transversales en toda la etapa de aspectos tempranos.

Por otro lado, uno de los mayores problemas en la creación de productos software con DSOA es cuando nos dedicamos a añadir aspectos de forma descontrolada. El resultado suele ser un producto software demasiado complejo, poco claro y difícil de evolucionar. Los mejores productos software son los que tienen identificados los aspectos que resuelven mejor las necesidades del cambio.

Por otro lado, las propuestas basadas en el enfoque de Puntos de Vista (PV) han sido concebidas para estructurar y organizar. El punto clave del análisis orientado al Punto de Vista es que toma en cuenta la existencia de varias perspectivas para especificar con el objetivo de reducir la complejidad y mejorar el entendimiento. En investigaciones anteriores (Kotonya & Sommerville, 1992) (Ross & Schoman, 1977) (Nuseibeh, Kramer, & Finkelstein, 1994) se ha demostrado cómo estructurar y organizar requisitos con el enfoque de PV apoya la integración y análisis de los requisitos. Sin embargo, es necesaria una arquitectura de PV que facilite la integración evitando conflictos con una articulación relativa de las incumbencias de los aspectos tempranos. Es decir, una Arquitectura de Puntos de Vista (APV) que pueda agrupar uno o más incumbencias y que puedan estar en uno o más puntos de vista. Esto hace posible identificar claramente los aspectos necesarios a observar en cualquier DSOA. Así mismo, una APV proporciona los medios para organizar las incumbencias por su semántica. En consecuencia, una organización del conocimiento de los aspectos son las bases para la evolución de un producto software. Ahora bien, la creación de una única APV no podría solventar cualquier proceso de desarrollo software. En esta línea es necesario un método para la creación de una APV específica.

En este contexto, el **tercer objetivo** específico de esta tesis doctoral fue proponer un método para la creación de una APV eficaz que permita estructurar y organizar los aspectos para apoyar la evolución de un producto software.

Por otro lado, se ha expuesto en el planteamiento del problema que son varias las técnicas y métodos OA que se han desarrollado en las etapas tempranas. Sin embargo, dichas propuestas están separadas en IROA y ASOA, quedándose el concepto de aspecto temprano en dos partes correlacionadas.

La IROA es un nueva área de la IR que aporta un conjunto de enfoques para gestionar requisitos transversales con un enfoque multidimensional que justifica el modelado de la ASOA. Por lo tanto, para que la IROA sea considerada como una disciplina que

resuelve problemas es importante dotar a esta etapa temprana del proceso de desarrollo software con propuestas ingenieriles.

La disciplina de la Ingeniería de Requisitos (IR) es responsable del proceso de descubrir y mantener la especificación de un producto software con el objetivo de comunicar, validar y analizar el conocimiento. En esta línea, el artefacto más importante para trasmitir el conocimiento entre las diferentes actividades de cualquier proceso de IR es el documento de Especificación de Requisitos (ER).

Muchas innovaciones teóricas y tecnológicas han resultado de la atención a la IR. Sin embargo, dentro de la comunidad científica se carece de un entorno de referencia homogéneo dentro del cual comunicar los conceptos motivadores en sus métodos y técnicas en la aplicación de la IR. El mayor factor que influye en esta carencia se encuentra la ambigüedad en las diferentes definiciones existentes en el tema (Christel & Kang, 1992).

En este contexto, se carece de los medios para verificar la calidad en la creación de una ER. Es decir, es necesario establecer conceptos y términos de referencia de una ER para cualquier propuesta dentro de la disciplina de la IR

Con estas ideas, el **cuarto objetivo** específico de esta tesis doctoral fue proponer un marco de referencia base para unificar criterios de calidad en el desarrollo de una ER.

Crear un producto software OO **a través de un DSOA proporciona los medios** para **enfrentar los continuos cambios** de información y funcionalidad de los modelos de negocio del entorno del cliente y de los desarrolladores **en cualquier momento del ciclo de vida** sin afrontan el riesgo de invertir muchos **recursos** en el intento. La anterior sentencia es la hipótesis base de esta tesis doctoral para el planteamiento de una investigación que pueda confirmarla. Desde los inicios del planteamiento de la investigación se acotó el alcance de esta tesis doctoral a las **etapas tempranas** de un proceso de desarrollo software. Posteriormente, al inicio del planteamiento de la solución se expuso un objetivo general que se enfoca en proporcionar los medios que permitan el modelado del conocimiento en las etapas tempranas de productos software OO a través de los conceptos del paradigma OA que permita una gestión efectiva de las incumbencias.

Ahora bien, en el planteamiento del problema se observó que las actuales propuestas OA para la creación de modelos en etapas tempranas carecen de propuestas formales basadas en las técnicas y métodos de la Ingeniería del Conocimiento (IC) para la organización semántica de los modelos. En este contexto, la utilización de ontologías como contenedores estructurados de la semántica de los elementos tempranos permite el modelado semántico que facilita la evolución y control de conflictos, acoplamientos y redundancias entre incumbencias.

Teniendo en cuenta lo ya expuesto, se han desarrollado cuatro objetivos en el planteamiento de la solución. Todos estos objetivos específicos están planteados para proporcionar los medios necesarios para el planteamiento del último objetivo específico.

El modelado de AT implica el proceso recurrente de crear y registrar las incumbencias tempranas agrupadas en aspectos para modelar su conocimiento de forma eficiente. Esta forma de modelar proporciona los medios para gestionar los efectos de las inevitables incumbencias cambiantes provocadas por la variabilidad de los deseos de los interesados en un proceso de desarrollo software, así como, los medios para evitar conflictos y mejorar la gestión de la trazabilidad. En este escenario, el modelado de aspectos e incumbencias tempranas tiene su mayor apoyo a través de la Ingeniería Ontológica (IO). Por lo tanto, es necesario proponer métodos y técnicas de

representación, organización y extracción del conocimiento de los modelos en las etapas tempranas de un DSOA basados en la IO para proporcionar una gestión efectiva del conocimiento del producto software OO. En esta línea, un marco de referencia para la conceptualización de aspectos permite definir claramente los límites de AT. Un modelo conceptual de AT proporciona una orientación clara en el desarrollo del modelado. Una APV proporciona una guía para el modelado de la organización del conocimiento temprano.

Ahora bien, es sabido que los desarrollos software basados en cualquier paradigma requieren una metodología para estructurar, planificar y controlar un proceso de desarrollo con el objetivo de crear un producto software de calidad. Por lo tanto, se requiere una metodología para estructurar, planificar y controlar la organización del conocimiento de AT.

Por último, el **quinto objetivo** específico y más importante de esta tesis doctoral es crear una metodología que permita el modelado y organización del conocimiento de las tempranas de un producto software que facilite la gestión de los aspectos tempranos.

3.1 Objetivos de la tesis doctoral

El resumen del planteamiento de la solución es un objetivo general compuesto por cinco objetivos específicos.

- **Objetivo general.** Proporcionar los medios necesarios que permitan el modelado del conocimiento en las etapas tempranas de productos software OO a través de los conceptos del paradigma OA para facilitar una evolución más efectiva, evitar conflictos y mejorar la gestión de la trazabilidad entre incumbencias.

El cumplimiento de este objetivo general da validez a la hipótesis acotada de este trabajo con la realización de los siguientes objetivos específicos.

- **Objetivo 1.** Delimitar las diferentes conceptualizaciones de aspectos en las distintas etapas a lo largo de un PDS.
- **Objetivo 2.** Proporcionar una forma sencilla y elegante de identificar los conceptos OA en las etapas tempranas de un DS
- **Objetivo 3.** Proponer un método para la creación de una APV eficaz que permita estructurar y organizar los aspectos para apoyar la evolución de un producto software.
- **Objetivo 4.** Proponer directrices para verificar la calidad en una Especificación de Requisitos
- **Objetivo 5.** Crear una metodología que permita el modelado y organización del conocimiento de las etapas tempranas de un producto software que facilite la gestión de los aspectos tempranos.

4 Visión general del documento

Respecto a la organización de contenidos de esta tesis doctoral se ha dividido en 5 capítulos, como se expone a continuación.

El Capítulo I es una introducción general, describe la motivación que llevó a la hipótesis de esta tesis doctoral. Así mismo detalla el alcance de ésta, el planteamiento del problema en donde se identifican los problemas actuales y el planteamiento de la solución en donde se plantea la realización de los objetivos específicos que han proporcionando la solución.

El Capítulo II muestra el estado de la cuestión dividido en 8 secciones; en la primera se plasma el contexto del paradigma Orientado a Aspectos para continuar en la segunda sección con una disertación sobre las etapas tempranas del ciclo de vida de un producto software. En la sección 3 se realiza un estudio de diferentes metamodelos propuestos para un DSOA. Posteriormente en las secciones 4 y 5, se describen las diferentes propuestas más aceptadas en IROA y ASOA. En esta línea, en la sección 6 se realiza un análisis global de dichas propuestas para identificar sus aportaciones y carencias. En esta sección se justifica la introducción de medios para realizar un modelado semántico en las etapas tempranas de un desarrollo software. Más adelante, en la sección 7 se diserta sobre las bondades de la ingeniería del conocimiento, pasando por la definición de conocimiento, la representación del conocimiento, las ontologías y los razonadores. Al final del Capítulo del estado de la cuestión se estudiaron las diferentes propuestas ontológicas para la organización semántica de las etapas tempranas de un DSOA.

En el Capítulo III se describe el desarrollo teórico en 7 secciones que proponen y justifican los fundamentos teóricos. En la primera sección se proponen los niveles de conceptualización de aspectos en un DSOA con un marco de referencia para su desarrollo. La segunda sección está centrada en la conceptualización de AT haciendo énfasis en los lenguajes para su modelado. En la sección 3 aparece la propuesta de modelado conceptual de AT. Posteriormente en la sección 4 se proponen y justifican las directrices para el desarrollo de una APV. En la sección 5 se analizan algunos estilos arquitectónicos conocidos para justificar el modelado de la arquitectura lógica de los AT. En la sección 6 se desarrollan los tipos de requisitos que pueden existir en una ER. Finalmente en la sección 7 del capítulo del desarrollo teórico se proponen y justifican los factores con los que hay que analizar la calidad de una ER.

Los capítulos anteriores establecen el marco teórico en el que se basa la tesis doctoral para pasar al Capítulo IV que presenta el desarrollo experimental con 4 secciones. En la sección 1 se presenta una encuesta sobre el método aplicado en la organización de una ER que proporciona las bases para que en la sección 2 se desarrolle una APV para productos software de sistemas de información con el apoyo de un corpus de requisitos. En la sección 3 se presenta el desarrollo de un caso de estudio, mostrando paso por paso la metodología OCTA para la representación del conocimiento de los aspectos tempranos en una ER efectiva y eficiente. Finalmente en la sección 4 se realiza una evaluación para demostrar la fiabilidad y eficiencia de la metodología OCTA.

Para completar el recorrido de esta investigación, en el Capítulo V se describen las conclusiones y los futuros trabajos en esta línea de investigación.

Por último, se aporta un glosario de los conceptos más importantes, se desglosan las referencias utilizadas, y se adjuntan dos anexos: el anexo A presenta la encuesta realizada sobre el método aplicado en una organización de una ER y el anexo B que describe la sintaxis de Manchester OWL.

CAPÍTULO II
ESTADO DE LA CUESTIÓN

Capítulo III. ESTADO DE LA CUESTIÓN

La Ingeniería de Software (IS) es un área de las ciencias de la computación que ofrece métodos y técnicas aplicando un enfoque sistemático, disciplinado y cuantificable para desarrollar, utilizar, mantener y evolucionar software de calidad. El paradigma OA permite una interesante aplicación de la IS.

1 Contexto de la Orientación a Aspectos

Si se mira hacia atrás en la historia de la Ingeniería de Software, se puede observar que los progresos más significativos en la búsqueda de productos software de calidad se han obtenido con la descomposición de sistemas complejos en partes más simples, relevantes a un concepto particular, a la meta o al propósito. Por ejemplo, la descomposición en módulos de un diseño de arquitectura software (Parnas, 1972) o la clasificación por temas de los requisitos no funcionales (The-European-Cooperation-for-Space-Standardization, 2009). Sin embargo, la separación por sí sola no es suficiente para obtener una solución. Es necesario que las incumbencias que se han separado se coordinen para cumplir con sus objetivos comunes, como dice la Ley de Jackson (Jackson, 1990): "considerando que se dividió para conquistar, debemos reunirnos para gobernar". Esta ley se cumple en aquellas metodologías que aportan técnicas para abstraer y componer esas partes del software que apuntan a un mismo propósito. El principio que se abstrae es llamado, de forma pragmática, **separación de incumbencias**.

1.1 Separación avanzada de incumbencias

Una **incumbencia** (del inglés *concern*) es una consideración específica que debe ser tomada en cuenta para satisfacer los propósitos de un producto software (Filman et al., 2004). Por lo tanto, podemos decir que un producto software es la implementación o realización de un conjunto de incumbencias. Sin embargo, la concepción de las incumbencias de un producto software varía dependiendo de la etapa del ciclo de vida en la que se encuentra. Por ejemplo, en las etapas tempranas son requisitos y en las etapas finales son segmentos de código. Así mismo, pueden variar dependiendo del punto de vista bajo el cual es creado. Por ejemplo, pueden ser incumbencias funcionales (como las reglas de negocio) o de otro tipo, entre las cuales se identifican incumbencias desde las de interés de alto nivel (como la seguridad y rendimiento) hasta las de intereses de bajo nivel (como el mantenimiento de la caché y sincronización).

En este contexto, se puede observar que una gestión efectiva de incumbencias de un producto software reduciría su complejidad y aumentaría su comprensibilidad aumentando su calidad. Una gestión efectiva implica tener la capacidad de identificar, modular[2], encapsular[3] y manipular los grupos de incumbencias que son relevantes a un mismo punto de vista para cumplir con el propósito de un producto software.

---

[2] Un elemento es modular cuando es más fácil, regular y económica su repetición.

[3] Ocultamiento del estado de manera que sólo se puede cambiar mediante operaciones propias.

Estas metas aunque loables e importantes, todavía no se han logrado totalmente en la práctica a causa de dos factores principales: el primer factor es la **variabilidad** del conjunto de incumbencias relevantes a lo largo del tiempo, dependiendo del contexto, tales como las diferentes actividades del desarrollo, las etapas del ciclo de vida o los interesados en el producto software en roles distintos que a menudo implican incumbencias de tipo dramáticamente diferentes. El segundo factor son los problemas con los **conflictos**, es decir, cuando existen incumbencias requeridas que al ser realizadas alguna de ellas promueven metas que bloquea a otras en el mismo proceso.

Ahora bien, basado en el principio de separación de incumbencias, el paradigma Orientado a Objetos (del inglés *Object Oriented*) propuso un encapsulamiento de incumbencias a través de una entidad con estructura y comportamiento que captura tanto datos como operaciones con un mismo interés. Sin embargo, en el proceso de encapsulación se da preferencia a incumbencias con intereses funcionales específicos descuidando todas las demás. Es decir, entre dichas entidades se dispersan incumbencias de funcionalidad o de restricción que no han sido consideradas en la descomposición previa del producto software pero que son necesarias para que su implementación cumpla con los requisitos de calidad. En consecuencia, el aislamiento de estas incumbencias resulta imposible con las técnicas convencionales de modelado del paradigma OO. Algunos investigadores han calificado a esta separación de incumbencias como el problema de "la tiranía de la descomposición dominante" (Tarr, Ossher, Harrison, & Sutton, 1999). Es decir, los lenguajes OO se agrupan entre los que tienen la influencia de la tiranía de la descomposición dominante. Por lo tanto, al implementar incumbencias transversales, que se entrecruzan en un producto software OO, es el programador quien debe realizar la tarea extra de identificarlas manualmente. Esto se debe a que dichos lenguajes proveen solamente una entidad de composición, lo cual lleva a un oscurecimiento y a un aumento de la complejidad del código.

Por lo tanto, para lograr los beneficios esperados de la separación de incumbencias son necesarias metodologías que proporcionen una buena gestión de las incumbencias en diferentes dimensiones de descomposición. La Separación Avanzada de Incumbencias (del inglés *Advanced Separation of Concerns, ASoC*) (Brichau et al., 2002) emergió como la propuesta que implica que los desarrolladores deben ser capaces de identificar, dividir en módulos, encapsular y manipular múltiples dimensiones de incumbencias simultáneamente, además de introducir nuevas incumbencias y dimensiones en cualquier etapa del ciclo de vida del software, sin el sufrimiento de los efectos agresivos de la modificación.

La separación avanzada de incumbencias ha sido concretada en la etapa de implementación y mantenimiento por diferentes grupos de investigación, proponiendo distintas y variadas técnicas de abstracción y composición, con una terminología y propiedades de lenguajes propios. Entre los trabajos más conocidos encontramos el trabajo de "Filtros de Composición" (del inglés *Composition Filters, CF*) (AkŞit, Bergmans, & Vural, 1992), el proyecto de IBM "Programación Orientada a Sujetos" (del inglés *Subject-Oriented Programming, SOP*) (Harrison & Ossher, 1993), el proyecto del grupo Demeter "Adaptación de Software Orientado a Objetos" (del inglés *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*) (Lieberherr, 1996), la "Programación Orientada a Aspectos" (del inglés *Aspect-Oriented Programming, AOP*) (Kiczales et al., 1997) y la "Separación Multi-Dimensional de Incumbencias" (del inglés *Multi-Dimensional Separation of Concerns, MDSoC*) (Tarr et al., 1999).

Todas las instancias de separación avanzada de incumbencias, mencionadas en el párrafo anterior, han propuesto metodologías y técnicas que hacen posible abstraer y componer incumbencias de forma más comprensible en el código de aplicaciones

complejas, aportando cierto soporte al mantenimiento y evolución de los productos software en la etapa de implementación. Sin embargo, el enfoque de la Programación Orientada a Aspectos (POA) es el que ha logrado un mayor grado de aceptación entre los desarrolladores, arquitectos y analistas de software especialistas en la separación de incumbencias. Es posible que la aceptación se deba a que la POA se basa en extensiones de lenguajes existentes y apunta principalmente a los basados en el paradigma OO. Esto es de interés, dado que es conocido que el paradigma OO se aplica a lo largo de todo un ciclo de vida de un producto software con mucho éxito y está apoyado por diferentes herramientas en cada una de las diferentes etapas.

1.2 Programación Orientada a Aspectos

> "An aspect, if it can not be cleanly encapsulated in a generalized procedure. Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways."[4]

La definición anterior fue dada por Gregor Kickzales y su grupo en su artículo **"Aspect-Oriented Programming"** de 1997 (Kiczales et al., 1997). En dicho documento se propone una separación conceptual en el código de los productos software en dos tipos de incumbencias: Incumbencias que pueden ser encapsuladas claramente (según el tipo de lenguaje, clase en OO) con los lenguajes de procedimiento generalizado (Java, C++,etc.), que son llamadas **incumbencias de funcionalidad básica** (del inglés *basic functionality concern*), y las **incumbencias transversales** (del inglés *crosscutting concern*) que no pueden encapsularse claramente con dichos lenguajes, dado que cortan transversalmente los modelos dominantes, produciendo un código difícil de entender. Para capturar dichas incumbencias transversales diseminadas a lo largo del código, propusieron encapsularlas sistemáticamente en la nueva entidad de primera clase llamada **"Aspecto"** (del inglés *aspect*). Es decir, modelar y gestionar la abstracción de un conjunto de incumbencias transversales con un mismo propósito en un aspecto.

La POA ha brindado un importante marco de referencia para la gestión de aspectos (incumbencias transversales), basado en los siguientes conceptos abstractos:

1. Punto de Unión (del inglés *Join point*).-Está claro que hay una relación entre los aspectos y los modelos que componen un producto software, así como entre los diferentes aspectos participantes. Por lo tanto tienen que tener algunos puntos comunes que serán los puntos de unión para unificarlos en un mismo producto.
2. Incumbencia transversal (del inglés *Crosscutting concern*).- Los puntos de unión son un tipo especial de interfaz entre los aspectos y modelos, sin embargo los elementos atómicos que se tejen realmente son los elementos transversales. Estos representan una propiedad de una o varias incumbencias diseminadas en un producto software. Los elementos transversales son representados de una forma abstracta en los aspectos y de una forma real en los modelos.

[4] "Es aspecto, si no puede encapsularse limpiamente en un procedimiento generalizado. Los aspectos no suelen ser unidades de la descomposición funcional del sistema, sino más bien a ser propiedades que afectan al rendimiento o la semántica de los componentes de una forma sistemática"

3. Tejedor (del inglés *Weaving*).-El concepto tejedor establece los mecanismos para la composición. Estos mecanismos pueden ser tanto estáticos como dinámicos entre aspectos y modelos de un producto software.

1.2.1 Los Lenguajes en la Programación Orientados a Aspectos

La POA plantea que los lenguajes funcionales y OO pueden ser agrupados como Lenguajes de Procedimiento Generalizado (LPG), basado en que sus mecanismos claves de abstracción y composición pueden verse agrupados bajo una misma raíz, encapsulando claramente el sistema en entidades de funcionalidad básica dentro de un procedimiento generalizado. Por lo tanto, para la implementación de los aspectos que se diseminan por todo el sistema sin poder encapsularse claramente en un procedimiento generalizado, propusieron una clase especial de lenguajes llamados Lenguajes Orientados a Aspectos (LOA), los cuales deben brindar mecanismos y constructores para capturar aquellas incumbencias transversales, permitiendo separarlas limpiamente de la funcionalidad básica. De forma global se distinguen dos enfoques diferentes en el diseño de los LOA: los LOA de dominio específico y los LOA de propósito general.

Los LOA de propósito específico fueron los primeros en aparecer (Lopes & Kiczales, 1997) soportando únicamente aspectos para los que fueron diseñados. Los LOA de dominio específico imponen restricciones en la utilización del lenguaje base para garantizar que los aspectos eviten conflictos de control y dependencia.

Los LOA de propósito general se diseñaron para ser utilizados en cualquier aspecto, por lo tanto, tienen el mismo conjunto de instrucciones para expresar cualquier aspecto en código.

A principios de 1998, apoyando los LOA de propósito general, fue liberada la primera versión de AspectJ (Lopes & Kiczales, 1998) como la primera plataforma que aplica los conceptos de la POA. Como es una extensión de los conceptos de Java, hoy se cuenta con una versión estable con el apoyo de la comunidad activa de Eclipse. Además de AspectJ, existen otras propuestas basadas en la extensión del lenguaje Java con cierto nivel de aceptación, tales como el proyecto de IBM HyperJ (Ossher & Tarr, 2000) que soporta aspectos con una separación e integración multidimensional; así como la propuesta de la comunidad Springsource con una aproximación a la POA, proporcionando ayuda a resolver problemas comunes en aplicaciones empresariales. Así mismo, existen otras propuestas de POA basadas en otros lenguajes: AspeCt[5], AspectC++[6], Eos[7] (basado en C#), AspectR[8] (basado en Ruby), etc.

1.3 Desarrollo Software Orientado a Aspectos

La ingeniería de software promueve el desarrollo de productos software de calidad apoyando el estudio sistemático de las características, límites y posibles soluciones de un problema informático (análisis), en base al cual se obtiene una concepción original de un modelo (diseño) para su codificación e implementación. Es decir, cumplir con todas las etapas necesarias en un desarrollo software.

---

[5] http://www.aspectc.net

[6] http://www.aspectc.org

[7] http://www.cs.virginia.edu/~eos/doc

[8] http://www.ruby-doc.org/gems/docs/a/aspectr-0.3.7/AspectR.html

En esta línea, la mayoría de los paradigmas propuestos en la ingeniería de software ha alcanzado esta madurez, aplicando inicialmente sus ideas en la etapa de implementación. Posteriormente, estos paradigmas trasladan sus conceptos a las etapas medias y tempranas de un Desarrollo Software (DS), buscando proporcionar un camino completo a través del ciclo de vida de un producto software.

Este es el caso de la POA que después de más de una década de investigación ha trasladado la manera de percibir el mundo a través de aspectos desde la etapa de implementación a las demás etapas de un DS, evolucionando la noción de POA hasta la de paradigma **Orientado a Aspectos (OA).** Esta tendencia derivada del hecho de comprender que soportar la separación avanzada de incumbencias en todas las etapas de un desarrollo software mejora la relación entre analistas, diseñadores y desarrolladores, por lo tanto, elimina fallas de interpretación. Este entendimiento plantea las bases para estructurar sistemas complejos a través de subsistemas, módulos o elementos simples de una forma más natural (Dijkstra, 1976).

Lo anterior se puede constatar con el creciente número de investigaciones que han logrado en cierta medida el modelado de los conceptos abstractos del paradigma OA en diferentes etapas de un DS. Emerge así el concepto de **Desarrollo de Software Orientado a Aspectos (DSOA)**[9] como una práctica con el objetivo de proponer nuevas técnicas y metodologías apoyadas por aplicaciones y herramientas para alcanzar la separación avanzada de incumbencias en todas las etapas de un proceso de DS basado en el paradigma OA.

Por otro lado, los mayores avances en el desarrollo de un producto software apoyado en el paradigma OA se han obtenido con la simbiosis con el paradigma OO. Por lo tanto, las propuestas de esta tesis están enfocadas a los **DSOA con base en el paradigma OO**. Sin embargo, no hay que olvidar que un desarrollo de un producto software apoyado en el paradigma OA puede estar basado en cualquier otro paradigma dentro de la ingeniería de software.

Ahora bien, una de las ventajas de producir un producto software con un DSOA es la posibilidad de gestionar el rastro entre los diferentes modelos creados durante el proceso de desarrollo, es decir, la trazabilidad. Un DSOA debe apoyar que todos los modelos de aspectos creados a lo largo de las diferentes etapas tengan una trazabilidad entre análisis, diseño y código. Esta práctica requiere el conocimiento de los diferentes lenguajes necesarios en un DSOA para su interrelación.

1.3.1 UML un lenguaje en el DSOA

En los párrafos anteriores de la sección 1.2.1 de este capítulo se ha comentado el tema relativo a los lenguajes de la programación OA, es decir, lenguajes para el modelado de aspectos en las etapas finales o de codificación; sin embargo, las otras etapas de un DSOA también requieren de lenguajes específicos.

En todo proceso de desarrollo software, antes de la etapa de codificación se encuentra la etapa de diseño. Esta tiene la necesidad permanente de un lenguaje para expresar los elementos de diseño con estructura y comportamiento. En la actividad fundamental de diseño, la abstracción desarrollada normalmente se plasma en una notación gráfica; esto se conoce como modelados visuales, permitiendo manejar la complejidad de los

[9] http://www.aosd.net/

sistemas software con el requisito deseable de que el lenguaje de modelado sea independiente de la plataforma de implementación del código.

Lenguaje Unificado de Modelado (UML) (del inglés *Unified Modeling Language*) es el lenguaje de modelado de sistemas software más conocido y utilizado en la actualidad. Esto se debe a que permite a los diseñadores de sistemas generar modelos que capturen sus ideas en una forma convencional de comprender para comunicarlas a otras personas. UML es un lenguaje estándar cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema software. En este lenguaje detrás de cada símbolo en la anotación hay una semántica claramente definida.

Por otro lado, los modelos de UML pueden conectarse de forma directa a una gran variedad de lenguajes de programación. Esto significa que es posible establecer correspondencia desde un modelo UML a un lenguaje de programación como java, C++, C#, etc. Esta correspondencia permite ingeniería directa: la generación de código a partir de un modelo UML en un lenguaje de programación. Así mismo ayuda la ingeniería inversa: se puede crear un modelo UML a partir de código implementado. La combinación de estas dos vías produce una ingeniería de ida y vuelta mientras se mantenga la consistencia. Utilizando la expresividad poco ambigua de UML, existen investigaciones donde es posible la ejecución directa de modelos, la simulación de sistemas software y la coordinación de sistemas en ejecución.

1.3.1.1 UML

Las raíces del UML provienen de las diferentes propuestas de Grady Booch, James Rumbaugh y Ivar Jacobson, fusionadas en un primer borrador en 1994. El consorcio creado para la gestión y publicación de estándares que soporte interoperabilidad en sistemas orientados a objetos, la OMG (de inglés Object Management Group) (OMG, 1989) aceptó en 1997 la primera versión v1.1 de UML.

UML está basada en una arquitectura de cuatro niveles de abstracción que van a permitir distinguir entre los distintos niveles conceptuales que intervienen en el modelado de un sistema:

El nivel M0 – Las instancias. El nivel M0 modela el sistema real, y sus elementos son las instancias que componen dicho sistema.

El nivel M1 – El modelo del sistema. Los elementos del nivel M1 son los modelos de los sistemas concretos. Existe una relación muy estrecha entre los niveles M0 y M1: los conceptos del nivel M1 definen las clasificaciones de los elementos del nivel M0, mientras que los elementos del nivel M0 son las instancias de los elementos del nivel M1.

El nivel M2 – El modelo del modelo (metamodelo). Los elementos del nivel M2 son los lenguajes de modelado. El nivel M2 define los elementos que intervienen a la hora de definir un modelo del nivel M1. En el caso de un modelo UML de un sistema, los conceptos propios del nivel M2 son "Clase", "Atributo", o "Asociación".

El nivel M3 – El modelo de M2 (meta-metamodelo). Finalmente, el nivel M3 define los elementos que constituyen los distintos lenguajes de modelado. De esta forma, el concepto de "clase" definido en UML, (que pertenece al nivel M2) puede verse como una instancia del correspondiente elemento del nivel M3, en donde se define de forma precisa ese concepto, así como sus características y las relaciones con otros elementos. Por ejemplo, una clase es un clasificador, y por tanto puede tener asociado un comportamiento, y además dispone de un conjunto de atributos y de operaciones.

La OMG ha definido un lenguaje para describir los elementos del nivel M3, que se denomina MOF (Meta-Object Facility) (Object_Management_Group-(OMG), 2006). MOF puede considerarse como un lenguaje para describir lenguajes de modelado, como pueden ser UML, CWM (Common Warehouse Metamodel), o incluso el propio MOF. Tales lenguajes pueden ser considerados como instancias del MOF, puesto que lo que proporciona el MOF es un lenguaje con los constructores y mecanismos mínimos necesarios para describir metamodelos de lenguajes de modelado, es decir, los elementos que van a constituir un lenguaje dado y las relaciones entre tales elementos.

UML 1.1 fue el lenguaje de modelado estándar suscrito por la OMG en 1997, evolucionando hasta la versión UML 1.5 en 2003. Sin embargo en 2005 fue aprobada la nueva versión UML 2.0, prometiendo mejores características, evolucionando hasta la versión actual UML 2.3 (Object_Management_Group-OMG, 2010a) (Object_Management_Group-OMG, 2010b).

Mientras que UML 1.x tenía como objetivo principal la respuesta a las necesidades clásicas de la industria del Software, la versión de UML 2.x fue una evolución mayor para el modelado visual. Las mejoras permiten describir muchos de los nuevos elementos encontrados en la tecnología de software de hoy en día, buscando la evolución de la ingeniería de software hacia la ejecución y verificación automática de modelos de diseño, además de hacer el lenguaje mucho más extensible de lo que era en las primeras versiones. La versión 2.x de UML viene acompañada de nuevas versiones de los estándares MOF (Meta Object Facility), XMI (XML Metadata Interchage) y OCL (Object Constraint Language). Esta coordinación general de varios estándares es un intento de dar un soporte más formal y completo a la filosofía general de MDA.

En las versiones de UML 1.x se hacía un fuerte hincapié en que UML no era un lenguaje de programación. Un modelo creado mediante UML no podía ejecutarse. En el UML 2.x esta asunción cambió de manera drástica y se modificó el lenguaje de manera tal que permitiera capturar mucho más comportamiento, permitiendo de esta manera la creación de herramientas que soporten la automatización y generación de código ejecutable a partir de modelos UML. Varios aspectos del lenguaje fueron reestructurados y/o modificados y la especificación se separó en cuatro especificaciones bien definidas: OCL, XMI, Infraestructura y Superestructura.

OCL (Object Constraint Language) Lenguaje de Restricciones de Objetos. El OCL fue originalmente especificado por IBM, y define un lenguaje formal para escribir restricciones y expresiones sobre elementos de un modelo. El OCL suele ser útil cuando se está especificando un dominio particular mediante el UML y es necesario restringir los valores permitidos para los objetos del dominio. El OCL brinda la posibilidad de definir en los elementos de un diagrama, entre otros: invariantes, precondiciones, poscondiciones y restricciones. El OCL fue incorporado al UML en la versión 1.1.

XMI (XML Metadata Interchange) XML de Intercambio de Metadata es una especificación para el intercambio de diagramas escrita para facilitar la manera de compartir modelos realizados mediante UML entre diferentes herramientas de modelado. En versiones anteriores del UML se utilizaba un Schema XML para capturar los elementos utilizados en el diagrama; pero este Schema no decía nada acerca de cómo el modelo debía graficarse. Para solucionar este problema la especificación XMI fue desarrollada utilizando un nuevo Schema XML que permite construir una representación SVG (Scalable Vector Graphics).

En la **Infraestructura** del UML se definen los conceptos centrales y de más bajo nivel. La Infraestructura es un metamodelo (un modelo de modelos) mediante la cual se modela el resto de UML. Típicamente la infraestructura no es utilizada por usuarios

finales del UML, sin embargo, provee la piedra fundamental sobre la cual la **Superestructura** es definida formalmente presentando los elementos de construcción de UML. La superestructura es la que sí es utilizada por el común de los usuarios. La Infraestructura brinda también varios mecanismos de extensión que hacen de UML un lenguaje configurable. Para los usuarios normales de UML basta con saber que la infraestructura existe y cuáles son sus objetivos.

La estructura de los diagramas UML está reflejada por el diagrama de la Figura 1, de acuerdo con la especificación de UML 2.5 de la OMG. Los detalles sobre estos diagramas específicos se organizan de acuerdo a esta estructura taxonómica, que da la perspectiva a los diagramas y a sus interrelaciones. Los diagramas de interacción comparten propiedades y atributos similares, como lo hacen los diagramas estructurales y de comportamiento.

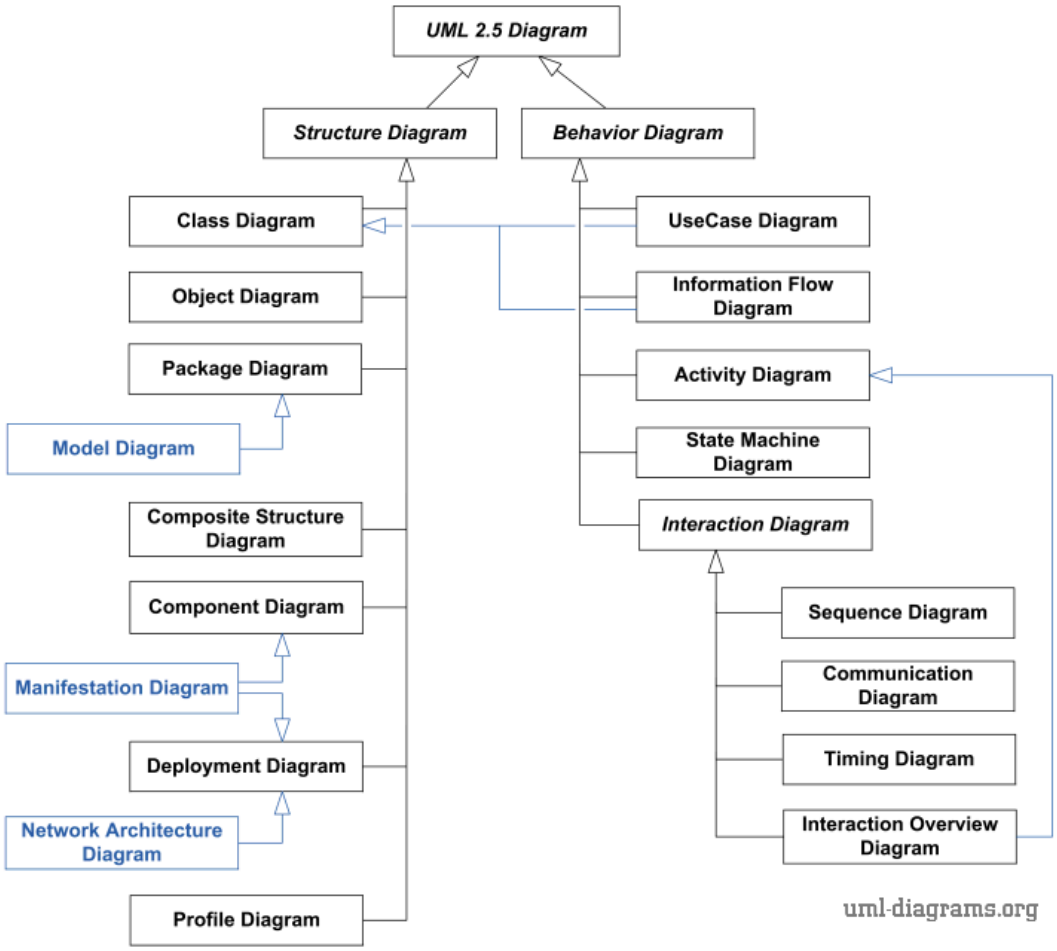


Figura 1 Estructura taxonómica del UML 2.5

Los **diagramas estructurales** representan elementos componiendo un sistema o una función. Estos diagramas pueden reflejar las relaciones estáticas de una estructura, como lo hacen los diagramas de clases o de paquetes, o arquitecturas en tiempo de ejecución, tales como diagramas de objetos o de estructura de composición.

Los **diagramas de comportamiento** representan las características del comportamiento dinámico de un sistema o procesos de negocio. Los diagramas de comportamiento incluyen a los diagramas de casos de uso, actividades, máquinas de estados, tiempos, secuencias, repaso de interacciones y comunicaciones.

Por otro lado, es sabido que el lenguaje de modelado UML es el estándar más utilizado para especificar y documentar cualquier sistema OO de forma precisa. Esto aunado a las nuevas aportaciones en los estándares UML 2.x ha propiciado la extensión de UML al DSOA. En esta línea, algunos enfoques han propuesto modificar el metamodelo UML tratando el aspecto como un clasificador. Sin embargo, hasta el momento, la comunidad OMG no ha reconocido a la entidad Aspecto como un elemento que pueda ser representado y especificado en UML. Otro enfoque que promete proporcionar soporte al modelado de aspectos en la etapa de diseño de un DSOA es analizar las características existentes de las versiones 2.x de UML, que cumpliendo con el requisito de respetar la sintaxis y semántica de UML, proporcionen soporte al modelado de la principal terminología del paradigma OA.

2 Etapas Tempranas

Hasta el momento, en el estado de la cuestión se ha expuesto cómo los conceptos de la POA están trascendiendo a las otras etapas del ciclo de vida de un producto software. Junto a la idea anterior, se comentaron los lenguajes OA relacionados con las etapas con mayor éxito en los DSOA en la actualidad: codificación y diseño.

Así, se observa la importancia de la relación que existe entre los lenguajes OA y las etapas en un DSOA. En esta línea, es necesario dejar claro a qué se hace referencia cuando se habla de cada una de las diferentes etapas en un proceso de desarrollo software.

Por lo general, desde un nivel de abstracción alto se suele hablar de etapas tempranas y **etapas finales** en un desarrollo software. Las etapas finales son limitadas a partir de la etapa de bajo nivel **codificación**. Sin embargo, nunca está claro qué etapa de alto nivel es antes de las etapas finales y tampoco en qué etapa de bajo nivel limita el final de las etapas tempranas. En esta línea, es de suponer que la etapa de diseño no está dentro de ninguna de las dos etapas de alto nivel mencionadas. Por lo tanto, podríamos decir que en su forma más general las etapas básicas de un ciclo de vida de un producto software son: etapas tempranas, etapas de diseño o medias y etapas finales.

En la actualidad, en torno al DSOA las propuestas con mayor aceptación por su gran efectividad se encuentran en las enfocadas a las etapas finales. Las propuestas enfocadas a las etapas de diseño han adquirido una efectividad aceptable pero con menor aceptación en las factorías software. Pero la efectividad de las propuestas en las etapas tempranas no es muy clara, generando un cierto escepticismo en su utilización.

Por otro lado, el DSOA está dentro de la disciplina de ingeniería del software, por lo tanto, se esperan propuestas de técnicas y métodos ingenieriles en todas las etapas de un DSOA para obtener productos software de calidad. En este sentido, deben existir métodos eficientes y aceptados por la comunidad para la creación de la especificación de los aspectos derivados de las primeras etapas de un DSOA, de lo contrario, en las etapas finales los desarrolladores tienen que obtener de un modo particular la solución sin poder comprobar el cumplimiento de los objetivos comprometidos, haciendo casi imposible futuras evoluciones en el producto software.

Ahora bien, en el capítulo de introducción se ha dicho que el alcance de esta tesis doctoral está acotado a la investigación e innovación de las **etapas tempranas** de un DSOA. En este punto, es pertinente dejar clara la importancia de estas etapas, así como sus límites, en cualquier desarrollo software.

El primer paso en cualquier proceso de desarrollo software es capturar la naturaleza del producto software a construir para su posterior análisis. Esta etapa se ha concretado en

la disciplina conocida como **Ingeniería de Requisitos (IR)** y por lo tanto se convierte en **la primera pieza de las etapas tempranas**.

La IR es el ámbito donde residen las propuestas enfocadas a evitar errores tempranos. Se puede pensar que los errores tempranos son pequeños y fáciles de solucionar. Sin embargo, un error en las etapas tempranas crea errores de código que pueden provocar aumento de costes y la causa del incumplimiento de fecha de entrega.

En este sentido se han realizado encuestas que indican la magnitud de las consecuencias por descuidar las etapas tempranas. Una de las primeras encuestas de este tipo es la realizada por la oficina de cuentas del gobierno norteamericano (del inglés *Government Account Office, GAO*), donde se realizó un estudio seleccionando 9 proyectos de desarrollo de software para el gobierno cuyos contratos sumaban una cantidad total de 6.800.000 dólares (GAO, 1979). De esta cantidad, solo 119.000 dólares correspondían a un proyecto que se había utilizado tal como se había entregado. El proyecto trataba de un problema relativamente simple cuyos requisitos no cambiaron durante el desarrollo. Un enorme porcentaje del resto del dinero fue invertido en proyectos cancelados o no satisfactorios.

El famoso Standish Group ha realizado estudios mucho más amplios, señalando entre todos sus reportes CHAOS (Standish Group, 2003) (Standish Group, 2010) una tasa promedio del 30% de proyectos en los que se alcanzan los objetivos; una tasa promedio del 50 % de proyectos en los que se incumplieron los compromisos de plazo, sobrepasaron el presupuesto o entregaron menos de las características y funciones requeridas; y con una tasa promedio del 20% se encuentran los cancelados antes de la entrega o entregado y nunca usado.

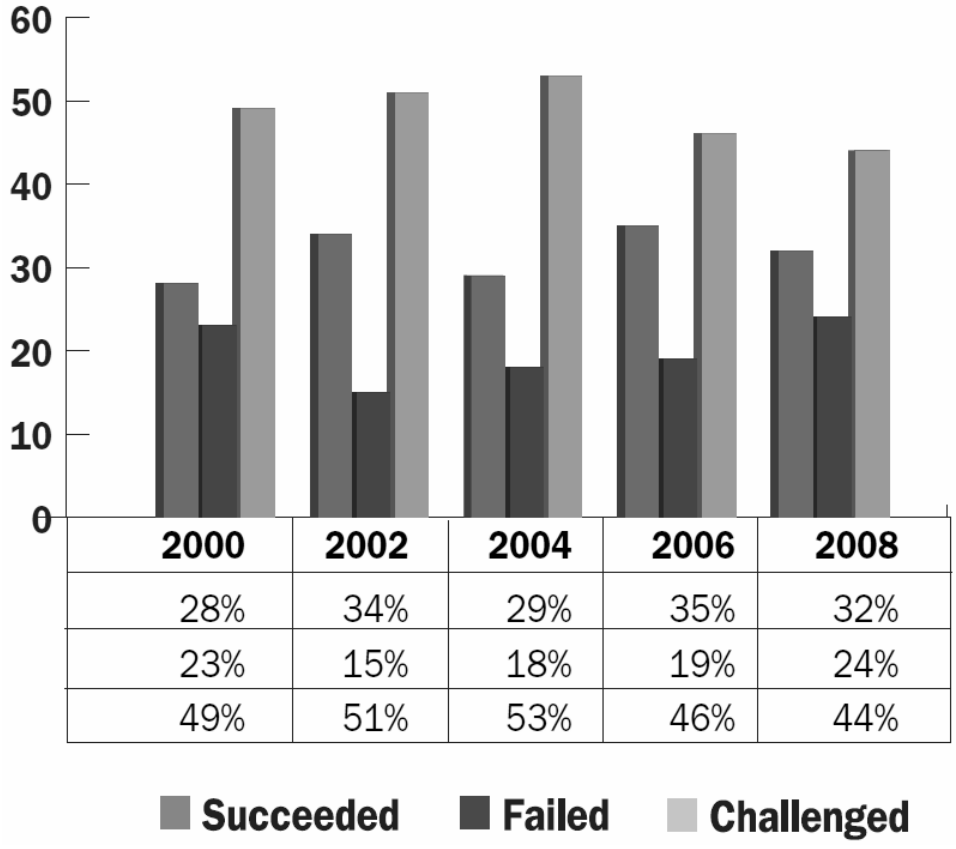


Figura 2 Resolución de proyectos CHAOS 2009

En la Figura 2 se muestra la comparación de resultados más reciente realizada por el Standish Group. Los resultados de los últimos años muestran un aumento considerable de la tasa de los proyectos entregados cumpliendo todos los requisitos, comparada con el 2% de la primera encuesta realizada por la GAO en 1979. Sin embargo, los avances en estos más de 30 años requieren mejorar para llegar a reflejar un aprobado aceptable.

Ahora bien, las encuestas realizadas a los directores de los proyectos que participaron en las encuestas indicaron que en su opinión los tres principales factores de fracaso eran:

a. Falta de información por parte de los interesados en el sistema
b. Especificación de requisitos incompleta
c. Altas posibilidades de cambios

A pesar de que las herramientas tecnológicas para construir software han evolucionado enormemente, estos reportes ponen de manifiesto la poca importancia puesta en las etapas tempranas para conseguir el éxito en la producción de software.

Por otro lado, los modelados resultantes de la Ingeniería de Requisitos (IR) guían de forma directa a la selección o descomposición en subsistemas del producto software y cómo se relacionan entre ellos, para llevar a cabo la funcionalidad especificada e introducir los factores de calidad que requiere el sistema software (reutilización, robustez, adaptabilidad, etc.). Esta **segunda pieza complementaria de las etapas tempranas** está contemplada dentro de la disciplina conocida como **Arquitectura Software (AS)**.

El concepto de AS ha generado mucha investigación en la comunidad de ingeniería de software. Han surgido dos apreciaciones importantes respecto a lo que concierne a una Arquitectura Software. Algunos favorecen un tratamiento estructural que representa a la arquitectura en su nivel de abstracción más elevado expresando la arquitectura en el sentido más formal y teórico (Perry & Wolf, 1992) (Garlan & Shaw, 1993), mientras otros apoyan cuestiones que están más cercanas al diseño. Este criterio está frecuentemente inspirado en los patrones de diseño que típicamente están en estado vivo, con constantes cambios en dichos patrones y aumento de ellos con nuevas creaciones (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996). Las dos apreciaciones son importantes para la comunidad, sin embargo, normalmente en la literatura estas dos apreciaciones se señalan con frecuencia, yuxtaponiéndolas o solapándolas, pero sin articulación formal y sistemática en su relación. Ambas apreciaciones ocurren en diferentes momentos del ciclo de vida del producto software, y por lo tanto corresponden en parte o totalmente a distintos niveles de abstracción. Con este escenario es necesario analizar el vínculo entre estas dos apreciaciones para sistematizar su relación.

Las dos piezas principales de las etapas tempranas están orientadas al entendimiento de los dominios del problema, pero con el objetivo de obtener las primeras líneas de solución, es decir, entrelazan la especificación de la IR con la descripción de soluciones de alto nivel de abstracción en la descripción arquitectónica. Cumplir con este objetivo proporciona alta calidad del producto software con una reducción de tiempo y costes de desarrollo, así como de mantenimiento y evolución.

En este contexto, en los últimos años se ha percibido la potencialidad del paradigma OA enfocado a las etapas tempranas del desarrollo software, surgiendo diferentes aproximaciones que apoyan por un lado la Ingeniería de Requisitos y por otro la Arquitectura Software.

3 Metamodelos para un DSOA

La aplicación de un DSOA implica el desafío de documentar el modelado de incumbencias en diferentes artefactos correspondientes a cada una de las etapas de un proceso de desarrollo software que requieren diferentes lenguajes OA. Así mismo, es necesario tener la posibilidad de encontrar y seguir el rastro de las incumbencias, a través de todas las etapas de un proceso de desarrollo software, es decir, la gestión de la trazabilidad para aspectos e incumbencias. Sin embargo, el DSOA presenta diferentes propuestas centradas en diferentes etapas del ciclo de vida con variaciones en la forma de plasmar los principios de la OA. Esto hace la trazabilidad entre las propuestas más complejas, evitando la amplia utilización de técnicas y métodos OA en proyectos de gran escala. Así mismo, los diferentes enfoques para un DSOA presentan diferentes lenguajes para el modelado de aspectos.

Por lo tanto, es necesario un metamodelo que pueda apoyar a cada propuesta a cumplir con los conceptos base de la OA. Una notación de modelado de aspectos que derive de un metamodelo formal subyacente, será susceptible de una automatización con herramientas de apoyo. El motivo es que para asegurar la correcta modelización de aspectos con herramientas de apoyo requiere un metamodelo de aspectos como base de especificación. Sin embargo, la especificación de un metamodelo de OA que cubra todo el DSOA puede ser complicada y contraproducente con la actual abundancia de conceptos, lenguajes y herramientas de evolución rápida.

Por lo tanto, en la línea de esta tesis doctoral, después de comprender las bases del paradigma OA es necesario proceder a la investigación de las diferentes propuestas de metamodelos enfocados a cubrir todo el ciclo de vida de un DSOA.

Ahora bien, una base común es necesaria para permitir una comunicación efectiva y para permitir la integración de las actividades dentro de la red de excelencia. La organización AOSD-Europe ha desarrollado un informe que presenta una base común para un DSOA (Van den Berg et al., 2005). Esta base común se realizó mediante el significado compartido de términos y conceptos en el ámbito de DSOA con el fin de apoyar el proceso de establecer una terminología común. Así mismo se hace hincapié en que el desarrollo consensuado sobre esta la terminología debe ser un proceso interactivo.

En esta línea, las propuestas seleccionadas de metamodelos desarrollados bajo el paradigma OA para su análisis debían estar basadas en la terminología común de la AOSD-Europe. Tomando esta consideración se resumen abajo las diferentes propuestas de metamodelos que fueron tomadas en cuenta.

3.1 Hacia un AspectJ visual con un Metamodelo y notación de modelado

El desarrollo del planteamiento **"Towards Visual AspectJ by a Meta Model and Modeling Notation"**[10] (Han et al., 2005) inicia con una propuesta de un metamodelo de Java para posteriormente extenderlo a un metamodelo de AspectJ que se muestra en la Figura 3.

[10] Hacia un AspectJ visual con un metamodelo y notación de modelado

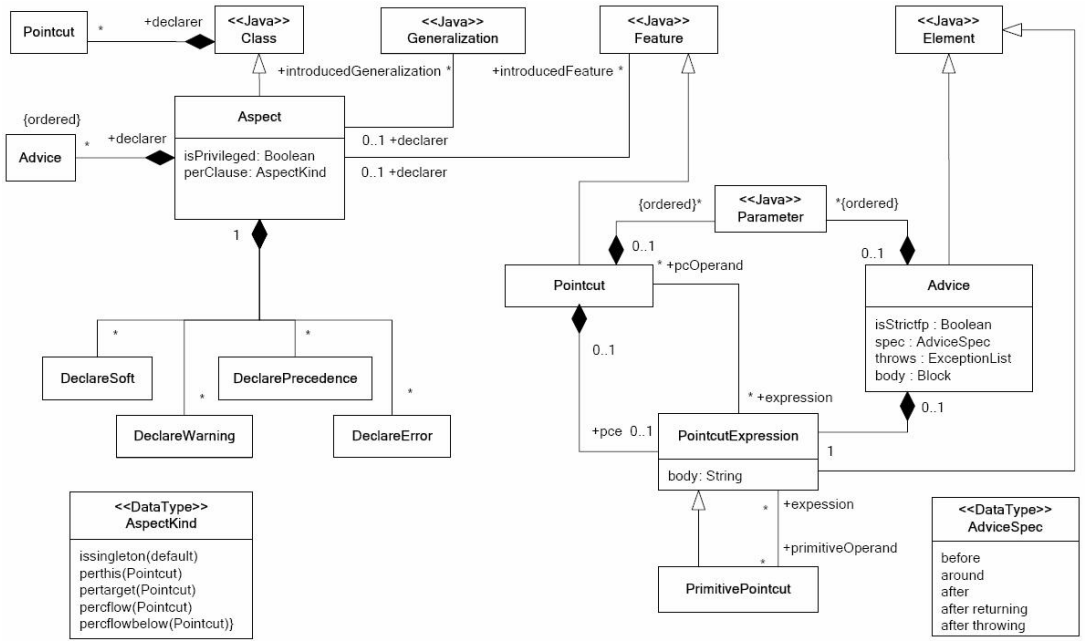


Figura 3 Metamodelo de AspectJ

La elección de Aspectj está motivada por su gran popularidad, lenguaje maduro, fortaleza industrial en herramientas de soporte y por su naturaleza como una extensión de Java.

La idea es que el metamodelo pueda utilizarse como ingeniería directa e inversa entre modelos creados con los conceptos de Aspectj y código en Aspectj. Con este objetivo plantean que un metamodelo debe cumplir los siguientes requisitos:

1. Cumplimiento de estándares: debe basarse en las normas existentes en los meta modelos.
2. Extensibilidad: debe ser fácil de incrementarse progresivamente, para que las herramientas apoyen a muchas otras extensiones de Java.
3. Minimalista: debe ser lo más pequeña y sencilla posible, a fin de aliviar la gestión de elementos relacionados.
4. Exactitud e integridad: debe ser un modelo estructural fiel a todos los conceptos de Java y Aspectj.
5. Naturaleza Visual: la notación gráfica debería formar parte del metamodelo para poder modelar el aspecto visualmente.
6. Susceptible de razonamiento formal: debe permitir razonamiento sobre semántica, sintaxis, notación, estructura de aspectos, arquitectura transversal y efectos transversales. El razonamiento también requiere que cada parte sea coherente con las otras.

Al principio da importancia al concepto de objetivo transversal, donde un tipo (del inglés *type*) puede ser estructural o de comportamiento. El primero es cuando se amplía con nuevos super-tipos o características por las declaraciones inter-tipo de algún aspecto. El segundo es cuando su comportamiento es modificado por el consejo de algún aspecto.

El metamodelo de Aspectj es una extensión de Java, por lo tanto, se extienden los conceptos del metamodelo de Java. El metamodelo AspectJ expande tres clases del metamodelo de java: clase, generalización y características, con nuevas asociaciones.

Añadiendo cinco nuevas clases de metadatos: aspecto, punto-corte (del inglés *pointcut*), consejo (del inglés *advice*), PointcutExpression y PrimitivePointcut.

El metamodelo es complementado con la propuesta de una notación gráfica para el modelado visual. La notación propuesta está basada en el núcleo del metamodelo de UML. Esto se consigue proponiendo los estereotipos <<aspect>> para Aspect, <<pointcut>> para punto-corte, <<pce>> para PointcutExpression y <<advice>> para consejo en diagramas de estructura estática. Ahora bien, para describir un aspecto añaden nuevos compartimentos para las declaraciones y para indicar cómo un aspecto corta transversalmente un objetivo este se superpone a los objetivos afectados. Con la intención de mejorar la especificación proponen notaciones para aspectos con corte transversal estructural y aspectos con corte transversal de comportamiento. Esas anotaciones pretenden detallar la visualización del resultado de aplicar aspectos a sus tipos de objetivos. Estos detalles son añadidos con diagramas de bloques de construcción del metamodelo de AspecJ anidados en las clases estereotipadas como <<aspect>>, así como superponiendo gráficamente estos elementos en los objetivos, como se puede observar en la Figura 4.

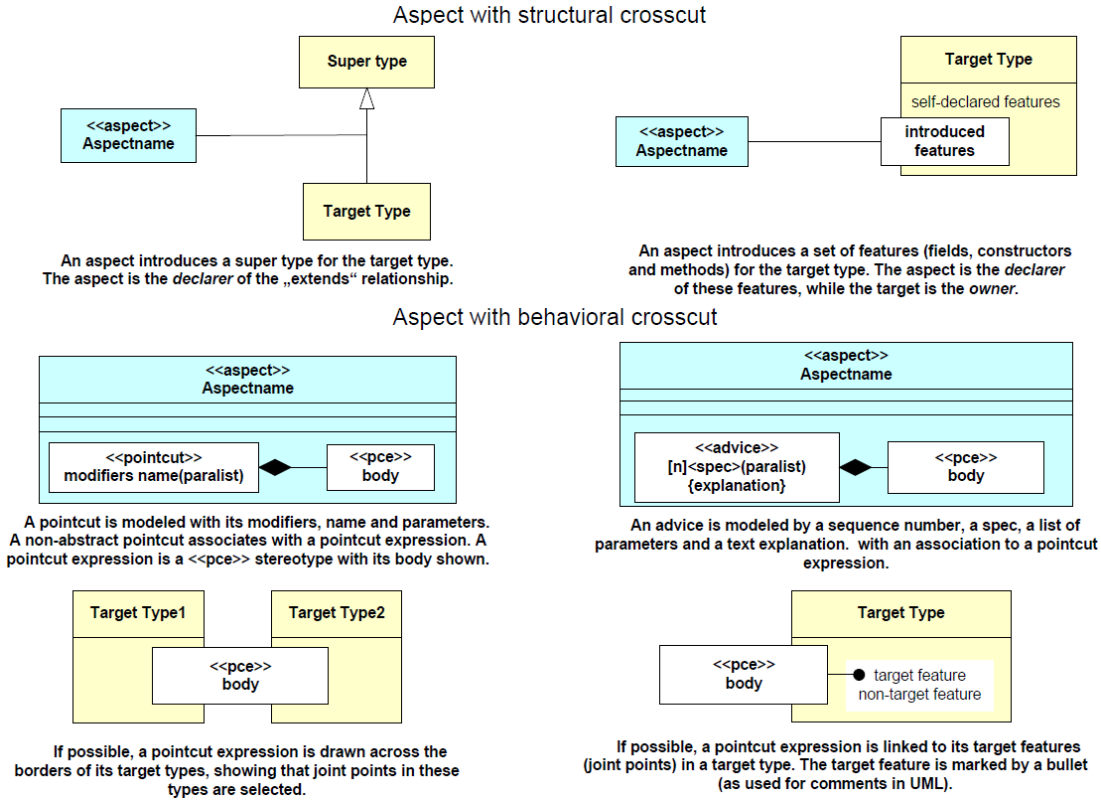


Figura 4 Notación gráfica para comportamiento y estructura transversal

3.2 Hacia una arquitectura de referencia común para un Modelado OA

Los conceptos tomados en cuenta en la propuesta **"Towards a Common Reference Architecture for Aspect-Oriented Modeling"**[11] (Schauerhuber et al., 2006) se muestran en la Figura 5.

[11] Hacia una arquitectura de referencia común para un modelado OA.

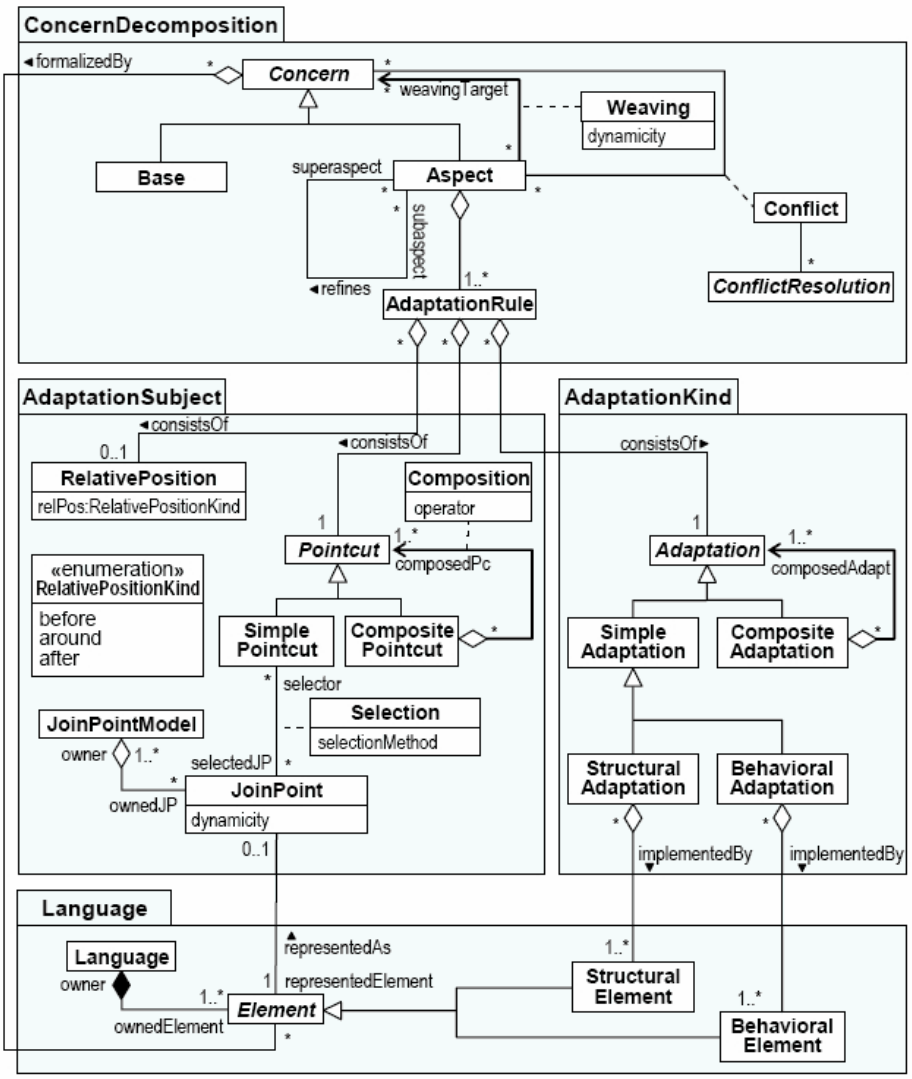


Figura 5 Arquitectura de referencia MOA

Esta propuesta pretende contribuir a la consolidación de una arquitectura de referencia donde se agrupen los conceptos básicos de la OA, abstrayendo estos de lenguajes de la POA o enfoques modelados OA.

La propuesta plantea una arquitectura de referencia con cuatro conceptos compuestos por bloques de construcción que a continuación se describen:

1. Descomposición de incumbencias (del inglés *ConcernDecomposition*). Se refiere a la descomposición general del producto software en desarrollo en las incumbencias y sus interrelaciones.

    - Incumbencia. Es un interés que se refiere al desarrollo del sistema, su funcionamiento o cualquier otro asunto que sea crítico o de otro modo importante a una o más partes interesadas.
    - Base. Es una unidad de modularización para formalizar una incumbencia no-transversal.
    - Aspecto. Es una unidad de modularización para formalizar una incumbencia transversal.

- Tejedor. Es la composición de los aspectos con otras incumbencias, que a su vez son o bien bases o aspectos. Se distinguen entre dos formas de tejer los aspectos en otras incumbencias, estática (en tiempo de diseño) y dinámica (en tiempo de ejecución).
- Regla de adaptación. Este introduce adaptaciones en determinados puntos de otras incumbencias.

2. Lenguaje (del inglés *Language*). Este concepto describe la especificación fundamental del lenguaje de la base y aspecto.

    - Lenguaje. Dependiendo del enfoque actual en el desarrollo de software de ciclo de vida de la lengua puede representar, por ejemplo, un lenguaje de modelado o un lenguaje de programación.
    - Elemento. Las incumbencias se formalizan mediante elementos de una lengua determinada.
    - Elementos estructurales. Se utilizan para especificar la estructura de un sistema.
    - Elementos de comportamiento. Se utilizan para especificar el comportamiento de un sistema.

3. Sujeto a adaptación (del inglés *AdaptationSubject*). Describe los conceptos necesarios para identificar donde se introduce la adaptación de un aspecto.

    - Puntos-unión. Específica dónde un aspecto puede insertar adaptaciones. Por lo tanto, un punto -unión es una representación de un elemento de identificación estructural o de comportamiento del lenguaje que se utiliza para capturar una incumbencia. Al mismo tiempo, los puntos de unión pueden ser estáticos o dinámicos.
    - Modelo de punto-unión. Comprende todos los elementos de un lenguaje determinado en aspectos autorizados a introducir adaptaciones.
    - Punto-corte. Representa un subconjunto de modelo punto-unión, es decir, la combinación puntos utilizados para especificar algunas adaptaciones.
    - Posición-relativa. Puede proporcionar más información en cuanto al lugar en donde las adaptaciones tienen que ser introducidas.

4. Tipo de adaptación (del inglés *AdaptationKind*). Abarca los conceptos necesarios para describir la adaptación de un aspecto.

    - Adaptación. Especifica de qué manera la incumbencia de estructura o comportamiento se adapta (afinar, sustituir o eliminar).
    - Adaptación-estructural. Está compuesto por elementos estructurales de una lengua para la adaptación de las incumbencias.
    - Adaptación-comportamiento. Comprende elementos de comportamiento de una lengua para la adaptación de las incumbencias.

- Adaptación-composición. A efectos de la reutilización, la adaptación puede estar compuesta por un conjunto coherente de adaptaciones estructurales y comportamiento.

3.3 Un metamodelo genérico MOF para el modelado OA

La propuesta de **"A Generic MOF metamodel for aspect-oriented modelling"**[12] (Fuentes & Sánchez, 2006) parte de la idea del uso de MDA como elemento de integración para las diferentes propuestas definidas en diferentes niveles del ciclo de vida de un desarrollo OA. Comentan que para poner en práctica este enfoque es necesario construir metamodelos para cada método propuesto para la mejora de los procesos OA. Con este objetivo proponen un metamodelo genérico para impulsar la construcción de metamodelos de propuestas específicas.

Este metamodelo genérico contiene un paquete núcleo de elementos de metamodelado que podrían ser especializados para cada propuesta, añadiendo anotaciones que pueden ayudar a cada propuesta para la construcción de su metamodelo. La idea principal se basa en construir el metamodelo para cada propuesta con el mismo núcleo, para que los metamodelos sean más fácilmente comprendidos y aceptados. Así, el metamodelo genérico presentado promete servir para impulsar la elaboración de metamodelos de propuestas específicas para lenguajes de diseño OA.

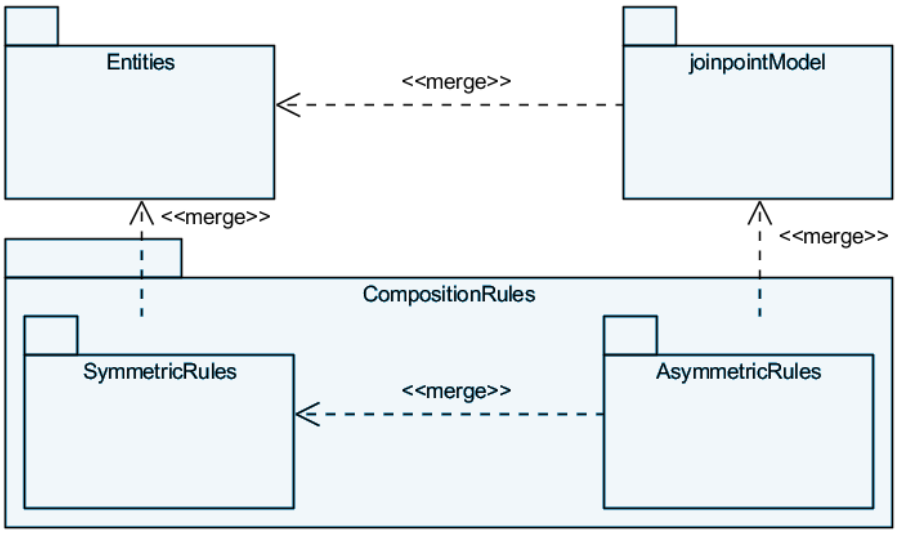


Figura 6 Diagrama de paquetes del Metamodelo genérico

Es interesante puntualizar que el metamodelo propuesto ha sido construido con MOF, que es el lenguaje estándar propuesto por el OMG para la elaboración de metamodelos.

Para definir el metamodelo genérico, utilizaron los términos y definiciones compartidos en algunas propuestas recomendadas (Van den Berg et al., 2005) (Brichau & D'Hondt, 2005) (Filman et al., 2004). Una vez que estos términos fueron identificados se agruparon en paquetes de acuerdo a su finalidad para facilitar la comprensión. Se identificaron tres paquetes principales de acción, un diagrama que muestra las relaciones de los paquetes se muestra en la Figura 6.

[12] Un meta-modelo genérico MOF para el modelado OA

1. Las entidades (del inglés *Entities*). Describen las unidades de descomposición y sus relaciones.
2. Punto-unión-modelo (del inglés *JoinpointModel*). Contiene los puntos de ejecución de la aplicación donde la interceptación de comportamiento incluido es aprobado.
3. Composición-reglas (del inglés *CompositionRules*). Tiene dos sub-paquetes, uno describe Reglas-Simétricas y otro Reglas-Asimétricas.

Una explicación más clara de cada paquete se describe a continuación:

- Entidades. Se consideran como especializaciones de BehaviouredClassifiers de UML 2.0. Las entidades cumplen todos los requisitos para ser clasificadores UML. Las entidades pueden ser especializadas en Base-Entidad o en aspecto. Para las aproximaciones simétricas no existen diferencias estructurales entre las Base-Entidad y aspectos. Ellas difieren en las reglas de composición en la que pueden participar. Para las aproximaciones asimétricas los aspectos tienen una estructura diferente de las Base-Entidad.
- Punto-unión-modelo. Cada entidad se compone de un conjunto de Ganchos, donde puede ser adjuntado un comportamiento de aspecto. Estos ganchos son los comportamientos de las Base-Entidad, porque serán los puntos de una traza de ejecución. Los ganchos son propiedad de todas las entidades, no sólo de las Base-Entidad, permitiendo deliberadamente la posibilidad de aplicar aspectos a los aspectos.
- Composición-Reglas-Simétricas (del inglés *Symmetric Composition Rules*). Desempeña el papel de una expresión, donde los operandos son las entidades, y el resultado será una entidad más grande que combina los operandos.
- Composición-Reglas-Asimétricas (del inglés *Asymmetric Composition Rules*). Especifican cómo los aspectos se aplican sobre ganchos(del inglés *hook*). Un Inconsciente-Obligatorio (del inglés *ObliviousBinding*) asocia el comportamiento de un aspecto con un gancho, lo que indica que el comportamiento tiene que ser ejecutado cuando el gancho se cumpla. Cuando el aspecto se inyecta se especifica en la unión por medio del atributo interceptionMode, lo que indica que el aspecto se ejecutará antes o después de la ejecución del gancho.

3.4 Conclusiones de los metamodelos propuestos para la OA

En la presente sección se ha hecho una revisión de algunas propuestas relevantes de metamodelos para el DSOA. Como se puede observar, los planteamientos están basados en dos enfoques principales. En el primer enfoque se encuentra "Towards Visual AspectJ by a Meta Model and Modeling Notation" y "Towards a Common Reference Architecture for Aspect-Oriented Modeling". Estos hacen referencia a abstraer la esencia del lenguaje de programación Aspectj en un metamodelo para intentar adaptarlo a las etapas de análisis y diseño. En un análisis general de sus carencias se observa que estos metamodelos prometen contribuir a la consolidación de los conceptos OA a lo largo de todo el ciclo de vida de un producto software OA, pero solo han estudiado las propuestas en Programación OA y en Modelado OA, es decir, las **etapas tempranas no están consideradas**.

Ahora bien, la mejor aportación de "Towards Visual AspectJ by a Meta Model and Modeling Notation" es proporcionar una notación de modelado con UML para el modelado previo a la codificación. Sin embargo, no está claro cómo es posible un mapeo entre las incumbencias transversales del modelado conceptual y su codificación

generada en AspectJ. En "Towards a Common Reference Architecture for Aspect-Oriented Modeling" el artículo comenta que la arquitectura de referencia es beneficiosa en tres maneras: Primero, en proporcionar la base para la construcción de un marco de criterios de evaluación. En segundo lugar, los conceptos de diferentes enfoques orientados a aspectos pueden ser mapeados cada uno a través de la arquitectura de referencia común. En tercer lugar, podría actuar como un modelo en términos de un metamodelo para el diseño de un nuevo lenguaje de modelado orientado a aspectos unificado. Todos estos deseos son precisamente los beneficios de la adopción de un metamodelo único. Sin embargo, es necesaria la aceptación de toda la comunidad científica relacionada con los DSOA para obtener estos beneficios.

La propuesta "A Generic MOF metamodel for aspect-oriented modeling" es un metamodelo para crear metamodelos que en principio implica un aumento en la reutilización de transformación y una simplificación de la especificación de transformaciones por medio de la comparación de metamodelos. Sin embargo, no aclara cómo realizar estos objetivos tan complejos. Así mismo, al igual que las otras propuestas al principio tienen la intención de cubrir todo el ciclo de vida de un producto software. Sin embargo, enuncian que la contribución de este trabajo es un metamodelo OA genérico para la creación de metamodelos de propuestas específicas en la etapa de diseño.

En resumen, en general las propuestas de metamodelos para el DSOA no consideran el modelado de las etapas tempranas. Así mismo, otra carencia observada en las diferentes propuestas es que la propuesta de modelado gráfico se enfoca únicamente a la especificación final previa a la codificación, es decir, al diseño detallado. Sin embargo, el trabajo realizado en el análisis de las partes importantes de una POA en la codificación es una aportación importante para trabajar en los elementos que siempre deberían estar presentes en las otras etapas del ciclo de vida del producto software.

En este punto, conociendo las diferentes propuestas de metamodelos enfocados a cubrir todo el ciclo de vida de un DSOA. Se procede al análisis de las diferentes aproximaciones que apoyan la Ingeniería de Requisitos y la Arquitectura Software que son las partes identificadas como etapas tempranas del desarrollo software.

4 Ingeniería de requisitos OA

La Ingeniería de Requisitos (IR) es la rama de la ingeniería de software que se ocupa de la captura de requisitos y su análisis en el Proceso de Desarrollo de Software (PDS), comprendiendo las necesidades del cliente en un determinado contexto para la especificación de estas en forma de conjuntos estructurados de requisitos que deben satisfacer un proyecto de desarrollo software. El objetivo es lograr definir qué se desea producir para disminuir significativamente la probabilidad de fracaso, dado que el conocer específicamente que se tiene qué desarrollar de un proyecto permite la proyección efectiva de las actividades, recursos, costes y tiempos, así como, permitir evaluar la calidad y medir la productividad del equipo de desarrollo para trabajos futuros. Por lo tanto, la ingeniería de requisitos es uno de los factores más importantes de éxito en un proyecto y esto se puede comprobar en los enfoques de mejora del proceso como Integración de Modelos de Madurez de Capacidades (CMMI) (del inglés *Capability Maturity Model Integrated*) (CMMI Product Team, 2010), que define la gestión y análisis de requisitos como unas de las prácticas básicas a la hora de evaluar el nivel de madurez de un proceso de desarrollo implantado en una organización.

El DSOA se fundamenta en los principios clásicos de la composición en partes y la separación de incumbencias, centrándose en el tratamiento de las incumbencias

transversales en las diferentes etapas de un desarrollo software. Por lo tanto, la Ingeniería de Requisitos Orientada a Aspectos (IROA) (del inglés *Aspect-Oriented Requirement Engineering* -AORE) (Grundy, 1999) (Rashid et al., 2002) surge como una práctica natural en los proyectos de DSOA, buscando identificar los requisitos que se entrecruzan con otros, para su organización, que de forma común son difíciles de extraer y soportar en módulos separados ya que se encuentran dispersos (del inglés *scattering*) o enmarañados (del inglés *tangling*) en diferentes requisitos del proyecto software. Los requisitos más comunes que tienen esta naturaleza transversal son los requisitos nombrados como no funcionales y los que representan atributos de calidad del producto software. Surge así un nuevo hito en la evolución de las buenas prácticas en la gestión de requisitos de software para lograr productos software de calidad.

Los enfoques que se han propuesto en esta línea tienen el propósito de fortalecer el análisis con mecanismos que facilitan la identificación, separación, clasificación y composición de incumbencias transversales solucionando los conflictos, y permitiendo el establecimiento de controles en la especificación del producto software para su posible evolución.

Entre los enfoques más representativos actualmente en la IROA están: la modulación y composición de requisitos aspectuales; la separación multidimensional de intereses; los aspectos en modelos de objetivos de requisitos; y la identificación de aspectos en los requisitos Theme/ Doc y DSOA con casos de uso. Todas estas aproximaciones se describen a continuación con mayor detalle:

4.1 Modulación y composición de requisitos aspectuales

La propuesta de IROA **"Modularisation and composition of aspectual requirements"**[13] (Rashid et al., 2003), inicia proponiendo que los requisitos interesados en el sistema pueden ser obtenidos utilizando mecanismos típicos de la IR, con una separación de incumbencias tales como puntos de vista, casos de uso (Jacobson, 1992), orientado a objetivos (van Lamsweerde, 2001) o marcos de problemas (Jackson, 2000).

En esta primera aproximación concretan su modelo únicamente con una técnica de composición de requisitos basada en un método de IR orientada al Punto de Vista (del inglés *Viewpoints*) llamado PREView (Sommerville & Sawyer, 1997). El enfoque se basa en la separación de la especificación en requisitos aspectuales, requisitos no aspectuales y reglas de composición en módulos que representan abstracciones. Así mismo se presenta XML como el lenguaje seleccionado para especificar los requisitos, los aspectos candidatos identificados y las reglas de composición para relacionar puntos de vista con aspectos. La herramienta de soporte utilizada fue ARCaDe (del inglés *Aspectual Requirements Composition and Decision*).

El modelo que en principio apoya esta propuesta se puede observar en la Figura 7. La propuesta comienza identificando y especificando incumbencias y los requisitos de los interesados en el sistema (del inglés *stakeholders*). Se infiere que una incumbencia es definida como un conjunto de requisitos, identificando las incumbencias transversales con matrices. En estas se establece de una forma heurística cuándo algunos de sus requisitos influyen de manera positiva (colabora) o afectan a otras incumbencias de manera negativa (restringe).

[13] Modulación y composición de requisitos aspectuales.

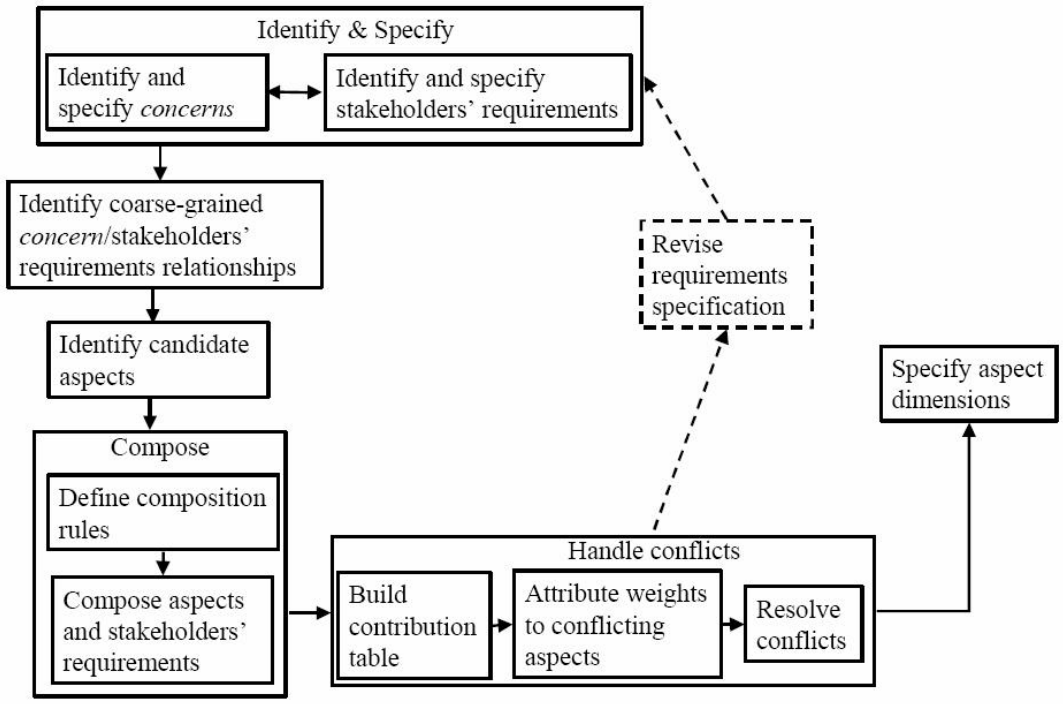


Figura 7 Modelo AORE 2002

La aproximación promete identificar, separar y componer incumbencias transversales negociando las decisiones entre grupos de interesados en el sistema y de esta forma soportar el establecimiento de cambios tempranos entre requisitos transversales y requisitos funcionales.

A través de un caso de estudio ilustran de alguna forma una instancia de su modelo genérico propuesto con los siguientes ocho pasos:

1. **Identificar y especificar requisitos de los interesados en el sistema.** En este primer paso identifican los puntos de vista, a veces con un anidamiento de puntos de vista.
2. **Identificar y especificar incumbencias.** Las incumbencias en las que se centran son las más cercanas a los conocidos como requisitos no funcionales.
3. **Identificar Relaciones Incumbencias de granularidad tosca/Punto de vista.** Para lograr este objetivo proponen realizar una matriz para relacionar incumbencias con puntos de vista.
4. **Identificar aspectos candidatos.** Basados en la matriz del punto anterior se seleccionan aspectos candidatos.
5. **Definir reglas de composición.** Las reglas de composición intentan definir las relaciones entre requisitos aspectuales y requisitos de puntos de vista en una granularidad fina.
6. **Componer aspectos y puntos de vista.** Los aspectos y los puntos de vista son compuestos utilizando las reglas de composición identificando conflictos entre aspectos.
7. **Manejando conflictos.** A través de una tabla de contribución se muestra cómo un aspecto contribuye con los otros de forma negativa o positiva. Para ayudar a resolver conflictos aspectuales se atribuyen pesos a las celdas de la matriz aspecto/punto de vista.

8. **Especificar dimensiones del aspecto.** La especificación de dimensiones de un candidato de aspecto intenta hacer posible el determinar su influencia sobre etapas de desarrollo posteriores e identificar su mapeado en una función.

4.2 Separación multidimensional de incumbencias en IR

La propuesta en "**Multi-Dimensional Separation of Concerns in Requirements Engineering"**[14] (Moreira, Araújo, & Rashid, 2005) tiene por objetivo realizar una descomposición de los requisitos de una manera uniforme, sin importar su naturaleza funcional o no funcional. Para lograr este objetivo propone un modelo para ingeniería de requisitos orientado a incumbencias. En principio este modelo hace que sea posible proyectar cualquier conjunto de requisitos en una serie de otros requisitos, por lo tanto, el apoyo a una separación multidimensional. Una proyección precisa la influencia de un problema dado en otras incumbencias. Dicha influencia se logra a través del empleo de reglas de composición. Las diferentes proyecciones hacen posible componer una serie de proyecciones que contribuyen a reflejar un interés individual. El enfoque apoya el establecimiento de los primeros intercambios entre los requisitos transversales y los solapados. Así mismo, facilita la negociación y toma de decisiones entre las partes interesadas.

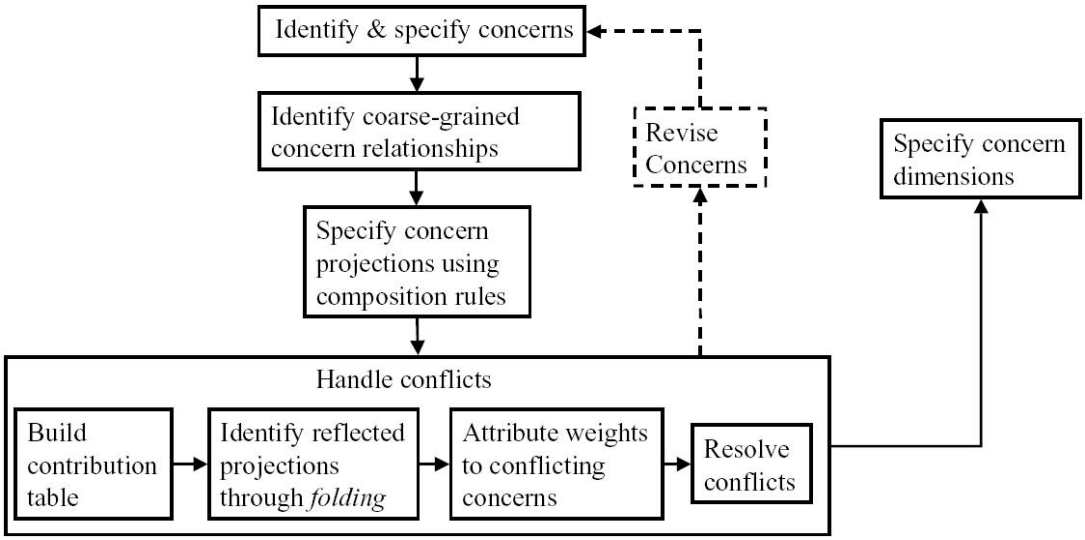


Figura 8 Modelo IR con tratamiento uniforme de incumbencias

El modelo de IR orientado a una separación multidimensional de incumbencias se muestra en la Figura 8. Se inicia con la identificación y especificación de incumbencias. La actividad de identificar y especificar incumbencias se consigue a través de alguna de las propuestas: puntos de vista, orientación a objetivos y casos de usos. El siguiente paso es especificar las posibles proyecciones de cada una de las incumbencias a otras incumbencias. Esto se logra a través de reglas de composición que funcionan en la granularidad de las necesidades individuales y no sólo con las incumbencias a encapsular. Finalmente, se identifican y solucionan conflictos entre las incumbencias.

A través de un caso de estudio ilustran de alguna forma una instancia de su modelo multidimensional de incumbencias propuesto con los siguientes cinco pasos:

[14] Separación multidimensional de incumbencias en IR.

1. **Identificar y especificar las incumbencias.** Las incumbencias se especifican en una estructura XML. La estructura inicia con una etiqueta de incumbencia para denotar el comienzo de una incumbencia, mientras que una etiqueta obligatoria indica el comienzo de un requisito. Refinamientos tales como sub-requisitos están representados a través de la anidación de las etiquetas. Cada requisito tiene un identificador que es único dentro de su ámbito de aplicación, es decir, la definición de la incumbencia.
2. **Identificar las relaciones de incumbencias de grano grueso.** A medida que se identifican y especifican las incumbencias se relacionan mediante la construcción de una matriz con una relación unidireccional entre dos incumbencias.
3. **Especificar Proyecciones de incumbencias usando las normas de composición.** La idea fundamental es la posibilidad de proyectar cada incumbencia en todas las otras con las que tiene una relación. Cada vez que un problema afecta a otros problemas que tienen un impacto amplio en el sistema se puede clasificar como una incumbencia transversal. Estas incumbencias no sólo se refieren a no funcionales, sino también a las incumbencias funcionales. La materialización de estas proyecciones se lleva a cabo aquí mediante la definición de un conjunto de reglas de composición. Definir las reglas de composición de las relaciones entre los requisitos se refiere a una granularidad fina. La especificación de las reglas de composición se hace a través de un esquema XML. La estructura de las reglas de composición muestra un conjunto coherente de reglas de composición encapsulada en una etiqueta de composición. La semántica de las etiquetas de requisito difiere de las etiquetas en la definición de incumbencias. Si un requisito de incumbencia tiene sub-requisitos deberán ser expresamente excluidos o incluidos en la restricción impuesta por el requisito de incumbencia. La etiqueta de restricción define una acción y la definición del operador, como los requisitos de incumbencia que han de ser limitados por otro requisito de incumbencia. La etiqueta final define el resultado de restringir los requisitos de incumbencia con otro requisito de incumbencia. El valor de la acción se describe si otro requisito de incumbencia o un conjunto de requisitos de incumbencia deben cumplirse o simplemente la especificación de la restricción debe ser cumplida. El punto interesante a destacar es que no todos los operadores son una especificación de incumbencia.
4. **Manejar Conflictos.** Las reglas de composición llevan a la identificación de conflictos entre incumbencias de las mismas restricciones de requisitos o grupos solapados de otros requisitos de incumbencia. La resolución de conflictos se lleva a cabo en los cuatro pasos que se describen a continuación:

    a. Tabla construcción de contribución. En una tabla se muestra de qué manera (negativa o positiva) una incumbencia contribuye a las demás.
    b. Identificar proyecciones reflejo a través de plegados. Después de haber estudiado la contribución entre las incumbencias ahora puede plegarse con el fin de reducir el alcance de las proyecciones que tenemos que tratar.
    c. Atributo de pesos a las incumbencias conflictivas. La ponderación nos permite describir el grado en que un problema puede limitar a otro. El uso de valores difusos (muy importante, importante, no tan importantes, etc.) facilita la tarea de los responsables de la atribución de prioridades a las incumbencias conflictivas. Los pesos se darán a las

incumbencias con respecto a las incumbencias que especifican una regla de composición.

d. Resolver conflictos. Los conflictos no deben ser demasiado difíciles de resolver con las prioridades que expresan los atributos de pesos. Una vez que todos los conflictos se han resuelto, el pliego de condiciones se revisa y recompone llevando a cabo la identificación de los nuevos conflictos.

5. **Especificar las dimensiones de incumbencia.** La especificación de las dimensiones de una incumbencia hace que sea posible determinar su influencia en etapas posteriores de desarrollo y determinar su asignación a una función, una decisión o un aspecto.

4.3 Aspectos desde modelos de objetivos de requisitos

En **"From Goals to Aspects: Discovering Aspects from Requirements Goal Models"**[15] (Yu et al., 2004) se propone descubrir **aspectos a través de modelos** basados en las técnicas de análisis desarrolladas en la **IR orientadas al objetivo** (van Lamsweerde, 2001).

El modelo orientado a objetivo propuesto se muestra en la Figura 9 con el nombre de V-Grafo (del inglés *V-Graph*). Utiliza un grafo en forma de V que contiene en sus vértices superiores los objetivos funcionales y los objetivos no funcionales respectivamente. Los objetivos no funcionales se describen en términos de metas (del inglés *softgoals*), basado en la idea de que el objetivo nunca se satisface claramente, además pueden entrecruzarse con otros objetivos funcionales.

El modelo es un árbol AND/OR con enlaces correlaciones laterales. En su vértice inferior se ubican las tareas u operaciones que contribuyen a la satisfacción de los objetivos y metas.

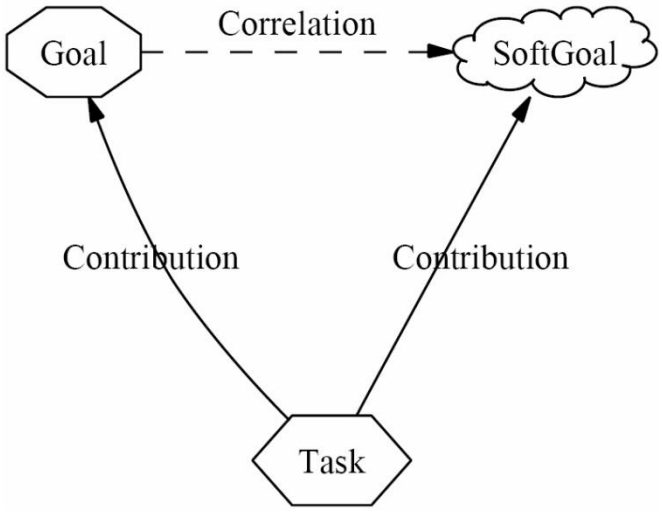


Figura 9 Modelo V-Grafo

[15] De los objetivos a los aspectos: Descubriendo Aspectos de los modelos de requisitos del objetivo.

La propuesta define un proceso sistemático e iterativo que va descomponiendo los objetivos hasta llegar al nivel de las tareas. En principio lo consiguen siguiendo el marco de referencia NFR (Chung, Nixon, & Yu, 2000), nombrando cada objetivo y tarea con dos descriptores: tipo (del inglés *type*) y tema (del inglés *topic*). Un tema captura la información contextual del objetivo / tarea / objetivo suave, que determinará las denominaciones utilizadas para identificar los objetos del mundo real relacionados con los objetivos. El proceso de construcción de V-gráficos propone una forma iterativa de perfeccionamiento de los grafos. A lo largo del proceso de descomposición se verifica el nivel de contribución de las tareas en el cumplimiento de los objetivos suaves. El proceso termina cuando todos los objetivos de la raíz y todas las metas se cumplen lo suficiente. En ese momento es posible identificar los aspectos candidatos mediante la identificación de tareas que tienen una gran cobertura. El grafo resultante se puede afinar aún más si los aspectos candidatos se agrupan en lo que llaman los aspectos objetivo. Así mismo, se comenta que durante cada paso del análisis de objetivo se permite detectar los conflictos que puedan presentarse y tomar acciones al respecto.

4.4 Identificación de Aspectos en los requisitos con Theme/Doc

En **"Theme: an approach for aspect-oriented analysis and design"**[16] se propone un enfoque de desarrollo orientado a temas que proporciona un método conocido como Theme/Doc (Elisa Baniassad & Clarke, 2004). Theme/Doc es un conjunto de heurísticas para el análisis de la documentación de requisitos software. Theme/Doc parte de una descripción textual de los requisitos en la cual se realiza un análisis gramatical para detectar las relaciones entre los requisitos y las acciones menores (del inglés *minor actions*); a partir de estas acciones se derivan las acciones mayores (del inglés *major action*) que representan la solución de software. Theme/Doc proporciona vistas gráficas de las relaciones entre los requisitos y Themes.

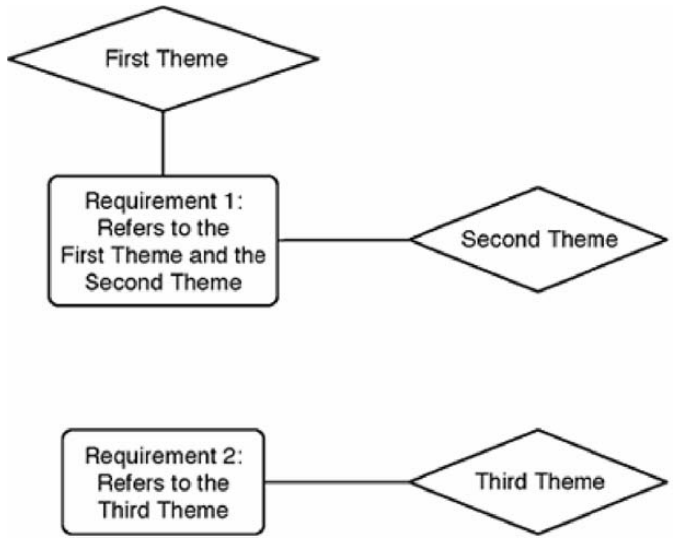


Figura 10 Ejemplo abstracto de una vista de relaciones Theme/Doc

La Figura 10 muestra un ejemplo abstracto una vista de relaciones Theme/Doc. Se puede ver en la figura que los diamantes representan Themes y las cajas redondeadas muestra el texto de un requisito. Si el requisito de un texto menciona el nombre de un

[16] Theme: un enfoque para la orientación a aspectos de análisis y diseño.

Theme o cualquier otro término que se entienda como sinónimo que esté vinculado, entonces dicho tema es el responsable de ese requisito, a menos que en algún momento más adelante en el proceso se rompa el vínculo.

Hay dos tipos de Themes: los Themes base, que pueden compartir un poco de estructura y el comportamiento con otros Themes base, dando el ejemplo de estos desde su propia perspectiva; y Themes transversales que tienen un comportamiento que se solapa con la funcionalidad de los Themes base. Los Themes trasversales podrían ser considerados Aspectos, sin embargo, esta propuesta no considera el concepto Theme como un sinónimo de aspecto. Sin embargo, el enfoque Theme/Doc proporciona una heurística para determinar cuál de los Themes son Aspectos.

El proceso comienza con la determinación de un conjunto inicial de Themes. Estos pueden ser obtenidos a través de conjuntos de incumbencias que parezcan importantes a primera vista. Así mismo, es posible contar con un conjunto de características o inquietudes coherente de comportamiento cuando se aplica con anterioridad un enfoque de análisis de requisitos.

Ahora bien, si dos o más incumbencias se describen juntas en un requisito, sus responsabilidades pueden enmarañar dos o más temas. Así, para localizar dichos temas es necesario hacer varias preguntas de competencia para identificar los relacionados con el requisito compartido:

1. ¿Puede el requisito ser dividido para aislar los temas? Si se puede, se puede reescribir la obligación y repartir las responsabilidades entre los mejores temas.
2. ¿Es un tema dominante en el requisito? Si es así, entonces el tema dominante probablemente debería ser responsable de esa obligación en lugar del requisito de ser compartida entre los temas.
3. ¿Es el comportamiento del tema dominante disparado por los otros temas mencionados en el requisito? Si es así, entonces se habrá identificado una relación de disparo entre dos temas.
4. ¿Es el tema dominante disparado en múltiples situaciones? Si, a través de los requisitos, el tema dominante es descrito como activo en múltiples situaciones, entonces es transversal. El tema dominante se convierte en el aspecto, y los temas de disparo se convierten en base.

4.5 IROA con casos de uso

Este enfoque se deriva de la propuesta de "**Aspect-Oriented Software Development with Use Cases"**[17] (Jacobson & Ng, 2005). La parte de la propuesta que abarca la IROA es la compresión y captura de las incumbencias de los interesados en el producto software.

La técnica de casos de uso proporciona diagramas de casos de uso y especificaciones textuales de cada uno de los casos de uso. La técnica de casos de uso es una técnica ampliamente adoptada para estructurar de forma natural los requisitos funcionales de un sistema. Un caso de uso es una unidad de funcionalidad útil que el producto software proporciona a sus actores. Una especificación de casos de uso contiene una serie de flujos de actividades que describen la funcionalidad básica y las posibles variaciones que

[17] Un desarrollo software Orientado a Aspectos con casos de uso

se manejan. En esta propuesta se promueve que la técnica de caso de uso proporciona los medios para el modelado de las relaciones entre las incumbencias. La base es utilizar las diferentes relaciones entre casos de uso reflejadas en sus diagramas y especificaciones: el caso de uso <<extend>>, el caso de uso <<include>> y la generalización entre casos de uso.

La propuesta destaca dos tipos de incumbencias transversales:

1. La primera es llamada **peer**. Se trata de incumbencias que son distintas una de la otra y entre peers ninguna es más importante que otra.
2. El segundo tipo de incumbencias transversales son las llamadas **extensiones**. Las extensiones son componentes que se definen en la parte superior de una base. Representan un servicio adicional o características.

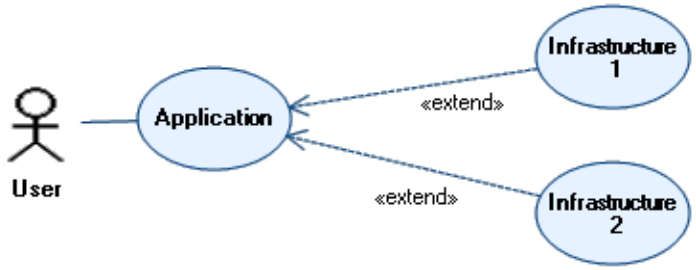


Figura 11 Casos de uso de aplicación e infraestructura

Por otro lado, en el entendimiento y la captura de requisitos se identifican dos categorías principales de casos de uso: de **aplicación** y de **infraestructura**. Los casos de uso de aplicación describen las funcionalidades básicas del producto software. Los casos de uso de infraestructura describen cómo el producto software agrega cualidades como facilidad de uso, confiabilidad, de rendimiento y de soporte para cada paso de un caso de uso de la aplicación. Una representación gráfica de esta idea se puede ver en la Figura 11.

Las actividades a realizar en esta etapa se describen a continuación:

1. La primera actividad consiste en **entender los requisitos de los interesados en el producto software**. El resultado es obtener una lista de características del producto software el cual incluye requisitos funcionales y no funcionales.
2. A continuación, se **capturan los casos de uso de la aplicación**. Esta actividad consiste en identificar actores y casos de usos a partir de los requisitos funcionales de la lista de características, y describir los casos de uso en las especificaciones de casos de uso contemplando posibles extensiones. La descripción de los casos de uso ayudará a identificar incumbencias de corte transversal.
3. En la siguiente actividad se tratan los requisitos **no funcionales, tales como la autorización**, que se pueden **capturar como casos de uso de infraestructura.** Estas pueden modelarse como extensiones modulares a los casos de uso de aplicación, modelando los puntos de unión a través de la especificación de los puntos de extensión dentro del flujo de ejecución de los casos de uso de aplicación.
4. En la última actividad se describen las incumbencias de los **requisitos no funcionales que tienen que ver con las cualidades de todo el producto**

software, tales como el rendimiento y la fiabilidad. Para conseguir este objetivo se revisa nuevamente la lista de características del producto software para identificar a los requisitos no funcionales que afectan a algún paso de los diferentes casos de uso de aplicación. Cada uno de estos pasos serán tratados como un caso de uso de transacción (del inglés *use case transaction*). Se modela la transacción base a través de un caso de uso estereotipado como <<perform transaction>>. Este caso de uso es un patrón, una plantilla limitada a cada paso de un caso de uso de la aplicación.

4.6 Conclusiones de los distintos trabajos en IROA

En la presente sección se ha mostrado la propuesta de la IROA "Modularisation and composition of aspectual requirements". Este planteamiento establece una organización libre de requisitos que se apoya en cualquier propuesta de modelado basada en puntos de vista. Por lo tanto, la propuesta no especifica de forma clara la manera en que los requisitos deben ser organizados.

Un punto destacable es la propuesta de utilizar XML como un medio para indexar la información requerida en la aplicación de una IROA, pero no existe una propuesta para modelar la semántica de los requisitos. En esta línea, otra aportación destacable de la propuesta es el desarrollo de herramienta que en principio hace posible definir el modelo propuesto, sin embargo, no fue posible obtener la aplicación para su verificación.

El siguiente planteamiento mostrado fue "Multi-Dimensional Separation of Concerns in Requirements Engineering". Esta propuesta se basa en un modelo que intenta apoyar la separación multidimensional de las incumbencias a nivel de requisitos. Esta multi-dimensionalidad se logra a través de un tratamiento uniforme de las propiedades funcionales y no funcionales. En principio, el tratamiento uniforme de las incumbencias hace posible definir las proyecciones de cada una de las incumbencias en cualquier conjunto de incumbencias a la que se referencie. Las proyecciones se utilizan para analizar la contribución de las múltiples incumbencias hacia un motivo especial de incumbencia.

Se continuó con la propuesta "From Goals to Aspects: Discovering Aspects from Requirements Goal Models". La idea más destacable en esta propuesta es el mecanismo para detectar aspectos objetivos evitando posibles conflictos a través de las influencia de las tareas. Sin embargo, el mecanismo de construcción es complejo sin apoyo de alguna herramienta.

Las siguientes propuestas analizadas han sido difundidas a través de libros, propiciando una explicación más exhaustiva de los mecanismos propuestos. La propuesta más conocida es "Theme: an approach for aspect-oriented analysis and design". En esta propuesta se puede destacar la idea de descomponer requisitos a otros más elementales, así como la intención de utilizar procesado de lenguaje natural para relacionar requisitos con temas, es decir, futuros aspectos. Sin embargo, no se utiliza ninguna herramienta para dicho procesado natural.

La última propuesta analizada fue "Aspect-Oriented Software Development with Use Cases". En esta propuesta Jacobson ha intentado relacionar los conceptos del paradigma OA con su propuesta de casos de uso. Así proporciona un sentido más claro a los bloques de construcción de la descripción de casos de uso. Sin embargo, se puede visualizar la influencia que pueden tener los casos de uso en las propuestas de IROA.

5 Arquitectura software OA

La comunidad de ingeniería de software está adoptando las bases propuestas por la norma IEEE 1471 (IEEE-Computer-Society, 2000) de cómo debe realizarse la descripción arquitectónica de un proyecto software. Sin embargo, esta norma no deja claro qué contenidos y estructura ha de tener.

La norma se centra en el concepto de **descripción arquitectónica**, definiéndolo como una colección de productos para documentar una arquitectura. Así mismo, el concepto arquitectura se refiere a la **arquitectura software,** definiéndola como la organización fundamental de un sistema encarnado en sus componentes, sus relaciones con otros y con el entorno, y los principios que guían su diseño y su plausible evolución. Estos principios guía se obtienen del **conocimiento que aportan los requisitos**, por lo tanto, la forma de organizar los requisitos debería dirigir la selección o la creación de los modelos lógicos de alto nivel de una arquitectura software.

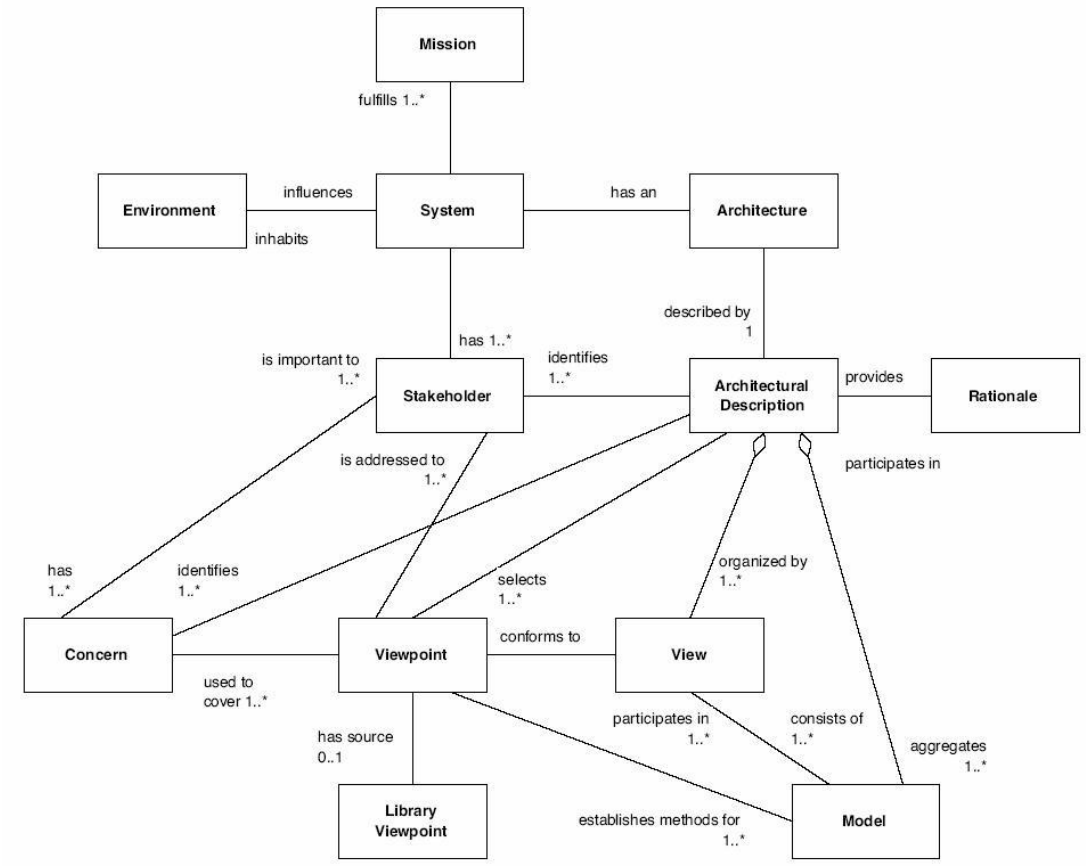


Figura 12 Modelo conceptual de IEEE Std 1471

La norma IEEE 1471 proporciona un resumen informativo de los conceptos clave de su modelo conceptual a través del diagrama de la Figura 12. Dicho metamodelo refleja que cada sistema debe poseer una arquitectura documentada o registrada por una descripción arquitectónica documentada por una colección de modelos y organizada en Vistas. Las **Vistas** deben estar conformadas por colecciones de Modelos bajo diferentes perspectivas denominadas **Puntos de Vista**. Un Punto de Vista (PV) define las reglas para extraer una o más incumbencias con una misma importancia para un conjunto de interesados en el sistema (del inglés *stakeholder*). Es decir, propone definir los PV desde las perspectivas subjetivas de un conjunto de interesados en el sistema.

En este contexto, se puede observar como la norma IEEE 1471 contiene conceptos del paradigma OA. Por lo tanto, de forma natural han surgido algunas propuestas de Arquitectura Software Orientada a Aspectos (ASOA) complementando las etapas tempranas de un DSOA.

Es interesante comentar que algunas de las propuestas de IROA descritas anteriormente continúan con las mismas bases teóricas hasta la ASOA, pero refiriéndose a esta etapa como diseño. A continuación se presentan estas propuestas de ASOA y algunos enfoques representativos concebidos en la AS:

5.1 Identificación de Aspectos en la Arquitectura Software con Theme/UML

El enfoque Theme se compone de dos partes: Theme/Doc, que es un conjunto de heurísticas para el análisis de la documentación de requisitos y Theme/UML , que es una manera de escribir los Theme con UML (Elisa Baniassad & Clarke, 2005).

"Theme/UML" permite diseñar modelos separados para cada uno de los Themes identificados en los requisitos. Los Themes que se identifican utilizando Theme/Doc, desde la perspectiva de la modularización se pueden diseñar de forma independiente, sin considerar si un Theme es transversal a otro, o si existen otros tipos de solapamiento en los Themes. Los Themes de aspectos utilizan bloques de construcción fuera del estándar de UML para capturar el comportamiento que se desencadena por un Theme base. Para proporcionar la capacidad de modular las incumbencias que se relacionan con el resto del producto software han definido un nuevo tipo de relación, llamada **relación de composición**. Esta propuesta de relación permite identificar las partes de los Themes de diseño relacionados.

Los Themes pueden estar relacionados unos con otros de la misma manera en que los requisitos o aspectos se relacionan con otras partes del producto software. Este tipo de relaciones puede provocar solapamientos entre los Themes. Hay dos formas en que los Themes se pueden relacionar: concepto compartido y transversal.

1. La primera categoría de relación es **el concepto compartido.** Esto significa la identificación de elementos de diseño en diferentes Themes para representar el mismo concepto básico en el mismo dominio, lo cual se muestra como **shared concepts** en la Figura 13. Esta relación es una categoría de corte transversal de separación simétrica.

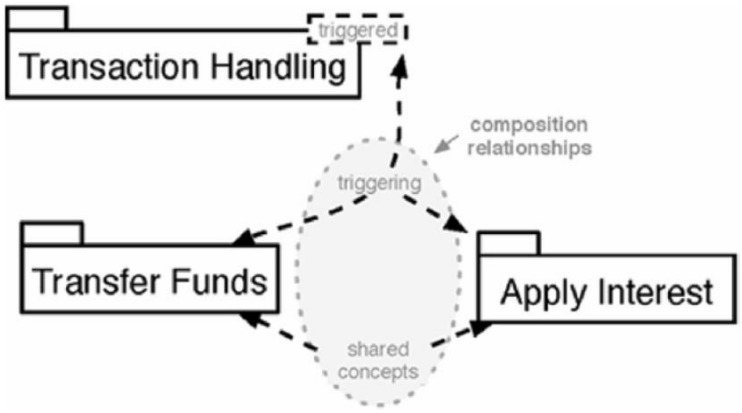


Figura 13 Composición Theme/UML

2. La segunda categoría de relación **transversal** es una categoría de separación asimétrica, donde un Theme dispara el comportamiento por el comportamiento de otros Themes; esto supone identificar cuándo y dónde en los otros Themes debe tener lugar el comportamiento dinámico adicional, lo que se refleja como **triggering** en la Figura 13.

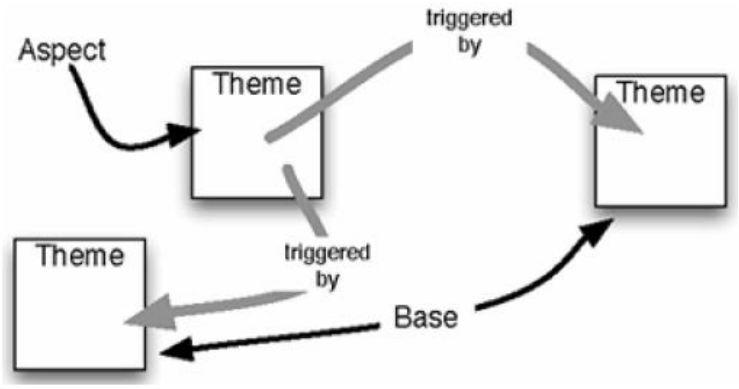


Figura 14 Themes relacionados con crosscutting: base-theme dispara el aspect-theme

A lo largo de la propuesta, se utilizan los términos base-theme, crosscutting-theme y aspect-theme. Crosscutting-theme y aspect-theme se utilizan como sinónimos y son siempre Themes que han desencadenado el comportamiento en conjunto con el comportamiento de otros Themes. Aspectos en la aproximación del Theme son los mismos aspectos en el enfoque de la separación asimétrica.

Base-Theme son los Themes que desencadenan los Aspect-Themes. Pueden ser Themes que comparten conceptos con otros Themes, y podrían ser ellos mismos aspectos y tener su propia base. También a veces hablan de una base que es el resultado de una composición de otros Themes a los que a continuación se aplica un aspecto (ver Figura 14).

5.2 ASOA con casos de uso

La propuesta de **"Aspect-Oriented Software Development with Use Cases"**[18] (Jacobson & Ng, 2005) plantea que el desarrollo de software es el proceso de construcción de modelos de casos de uso desde diferentes perspectivas. Se comienza con el modelo de casos de uso para capturar incumbencias de los interesados en el producto software. Posteriormente se refina para definir el modelo de casos de uso en un modelo de análisis, que también formula una visión de alto nivel de la aplicación, continuando con una estrategia de cómo la aplicación funcionará en la plataforma de ejecución con el modelo de diseño. Finalmente, del modelo de implementación se obtendrán los códigos y ejecutables. Al concluir el trabajo sobre un modelo de caso de uso se entregan todos los artefactos asociados en un solo paquete que llama use-case-module.

Entre las diferentes perspectivas de los modelos (análisis, diseño, etc.) se requiere mantener la separación de incumbencias transversales. Se intenta conseguir este objetivo a través de la nueva unidad modular llamada use-case-slice. Un use-case-module consta de un use-case-slice de cada modelo, ver Figura 15. La propuesta describe cómo

[18] Desarrollo software OA con casos de uso.

preservar la separación de incumbencias transversales peers y extensiones a través de use-case-slice.

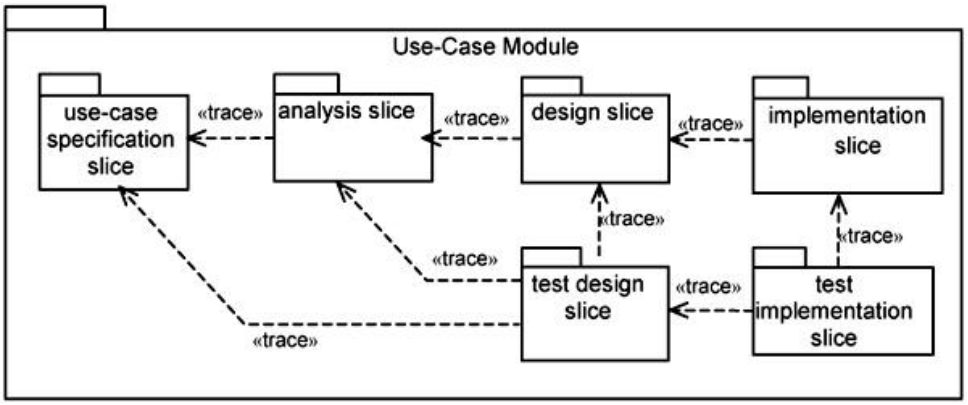


Figura 15 Use-case-slices dentro de un use-case-module

La parte de la propuesta orientada a la ASOA aparentemente abarca el modelo de análisis.

El modelo de análisis tiene el objetivo de crear una especificación precisa de las necesidades. En este modelo se plasman abstracciones en forma de clases que hace los requisitos más claros con respecto a las especificaciones de casos de uso. El modelo de análisis contiene dos estructuras:

1. **Estructura de elementos de análisis**. Abarca las clases de análisis organizadas en paquetes y capas. Estas clases de análisis están vacías.
2. **Casos de uso de la estructura de análisis**. Define la forma de la superposición de funcionalidad en la estructura de elementos .Comprende, non-use –case-specific- slices que añaden clases reales y use-case-slice que proporciona las características de las clases en la estructura de elementos.

Para abstraer detalles de bajo nivel, el modelo de análisis proporciona tres clases de análisis estereotipadas: clases estereotipadas de **límites**, de **control** y la **entidad**.

La creación de un modelo de análisis es un refinamiento de los casos de uso donde se identifican las clases de análisis que realizan los casos de uso con sus responsabilidades. Posteriormente se organizan estas abstracciones en una estructura de elementos de alto nivel del producto software. Las construcciones utilizadas en el modelo de análisis son de alto nivel e independientes de la plataforma para que no se piense en los detalles de implementación antes de tiempo.

Ortogonal a la estructura de los elementos está la estructura de casos de uso. La estructura de casos de uso contiene non-use-case-specific slices y use-case slices. Los use-case slices tienen la finalidad de preservar la separación de incumbencias transversales de los casos de uso desde el análisis al diseño y su implementación. Los non-use-case-specific slices describen comportamientos genéricos que representan el dominio de algún tipo y no contienen aspectos.

Por lo tanto, el propósito del modelo de análisis es doble. En primer lugar, es donde se comienza a describir la estructura interna del producto software. En segundo lugar, es un refinamiento del modelo de casos de uso.

En esencia el refinamiento de casos de uso se inicia con el modelado de la realización de un caso de uso con un conjunto de clases. Posteriormente se determina qué partes de

estas clases son específicas para el caso de uso y qué partes son comunes entre otros casos de uso. Se ordenan estos elementos particulares como extensiones de clase dentro de un use-case-slice. Los use-case-slice emplean aspectos para componer las extensiones de clase en sus respectivas clases dentro de la estructura de elementos. Por lo tanto, se debe determinar el contexto estructural de la ampliación indicando a qué clase u operación se añadirán las extensiones y el contexto del comportamiento, indicando en qué momento durante la ejecución de la operación principal las extensiones de la operación se ejecutarán.

5.3 Método de Análisis de Arquitectura de Software Aspectual

El **"Aspectual Software Architecture Analysis Method"**[19] **(ASAAM)** (Tekinerdogan, 2004) se centra en un método de evaluación basado en escenarios (Kazman, Abowd, Bass, & Clements, 1996) e introduce un conjunto de reglas heurísticas que ayudan a obtener los aspectos arquitectónicos transversales de los correspondientes componentes de la arquitectura.

Los artefactos que son base para el desarrollo de esta aproximación son dos: los escenarios y los componentes de arquitectura.

Los escenarios están clasificados en tres diferentes tipos:

- Directos, son los escenarios que se desarrollan de forma directa.
- Indirectos, son los escenarios que necesitan de cambios en los componentes.
- Aspectuales, son los escenarios de una combinación de los dos primeros o uno de ellos pero éstos serán distribuidos a través de múltiples componentes.

Así como los escenarios están divididos, los componentes también se dividen en cuatro:

- Cohesive component: son aquellos que están bien definidos y representan escenarios semánticamente cerrados.
- Ill-defined component: componente que consta de muchos subcomponentes el cual representa un set de escenarios semánticamente cerrados.
- Tangled component: un componente que representa un escenario aspectual el cual es también representado directa o indirectamente por el componente.
- Composite component: componente que incluye escenarios semánticamente distintos pero que no pueden ser descompuestos semánticamente o no incluyen un escenario aspectual.

El Método de Análisis de Arquitectura de Software Aspectual consta de las siguientes actividades básicas (ver Figura 16):

1. Desarrollo de la arquitectura candidata. Un diseño de arquitectura candidata establece qué será analizado en relación con los factores de calidad requeridos y los aspectos posibles.
2. Desarrollar escenarios. Los escenarios de las distintas partes interesadas en el producto software se reúnen, representando los usos importantes y usos previstos de la arquitectura de software.
3. Evaluación de escenarios individual e identificación de aspecto. Los escenarios se clasifican en primer lugar en escenarios directos e indirectos. La evaluación

[19] Método de Análisis de Arquitectura de Software Aspectual.

escenario apoya la búsqueda de posibles aspectos arquitectónicos. La aplicación de las reglas heurísticas dará lugar a una clasificación más detallada de los escenarios directos, indirectos aspectuales y los aspectos arquitectónicos.

4. Evaluación de la interacción de escenarios y clasificación de componentes. El objetivo de este paso es evaluar si la arquitectura es compatible con una adecuada separación de incumbencias. Esto incluye incumbencias no transversales e incumbencias transversales, es decir, los aspectos. Cada componente de los componentes directos e indirectos se analiza y clasifica en cohesive component, tangled component, composite component o ill-defined component.
5. Refactorización de la arquitectura. Con base en la evaluación de interacción de escenarios y las clasificaciones de componentes, se propone una refactorización de la arquitectura. La refactorización se puede hacer usando las técnicas convencionales de extracción, tales como patrones de diseño, o el uso de técnicas orientadas a aspectos. Los aspectos arquitectónicos y los componentes que están enredados se describen en la arquitectura.

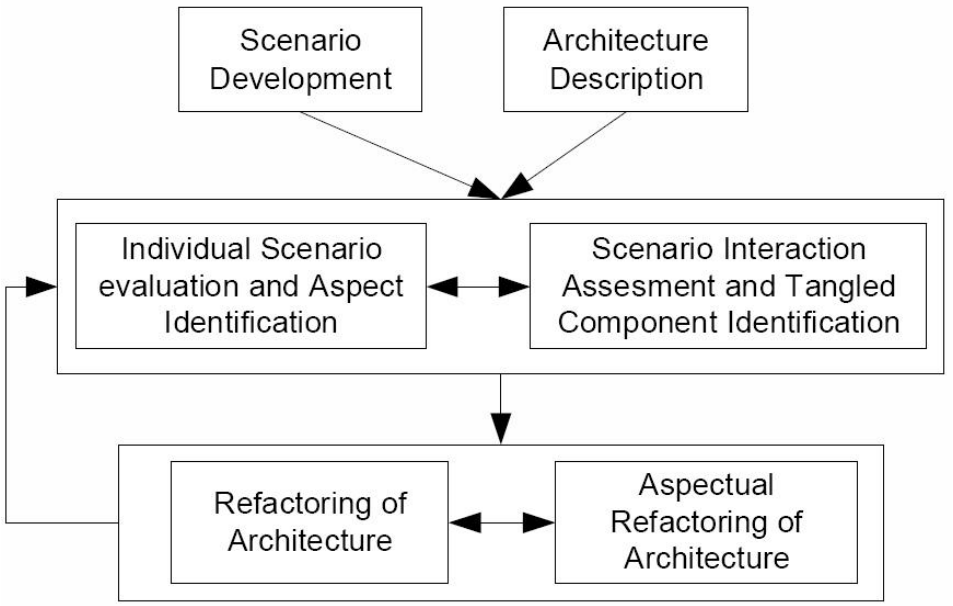


Figura 16 Actividades de ASSAM

5.4 Conducción y Gestión de las Decisiones de Arquitectura con Aspectos

La propuesta **"Driving and Managing Architectural Decisions with Aspects"**[20] (Garcia et al., 2006) se basa en que las decisiones transversales arquitectónicas deben ser manejadas como aspectos arquitectónicos separados. Estos son definidos como unidades de la modularidad para capturar las decisiones asociadas con problemas de ámbito general, permitiendo a los arquitectos que representen todas las implicaciones estructurales y de comportamiento en un solo lugar.

[20] Conducción y gestión de las decisiones de arquitectura con Aspectos

La idea es contar con abstracciones adecuadas para permitir su representación. En este sentido, intentan proporcionar los medios para facilitar su composición a través de plantillas para especificar aspectos de arquitectura con información esencial para la captura de decisiones transversales:

1. Nombre del aspecto arquitectónico.
2. Las decisiones de arquitectura estructural y de comportamiento, tales como la inclusión de componentes, interfaces, relaciones, procesos, etc. que se hicieron con el único propósito de contemplar las cuestiones relacionadas con el aspecto arquitectónico.
3. Las normas de composición, para describir cómo las decisiones transversales con respecto a este aspecto arquitectónico afectan a otros elementos arquitectónicos y otros aspectos.
4. Una sección de razonamiento que capta la razón de esas decisiones.

La Figura 17 muestra un ejemplo de cómo utilizar la noción de los aspectos arquitectónicos para apoyar la descripción modular.

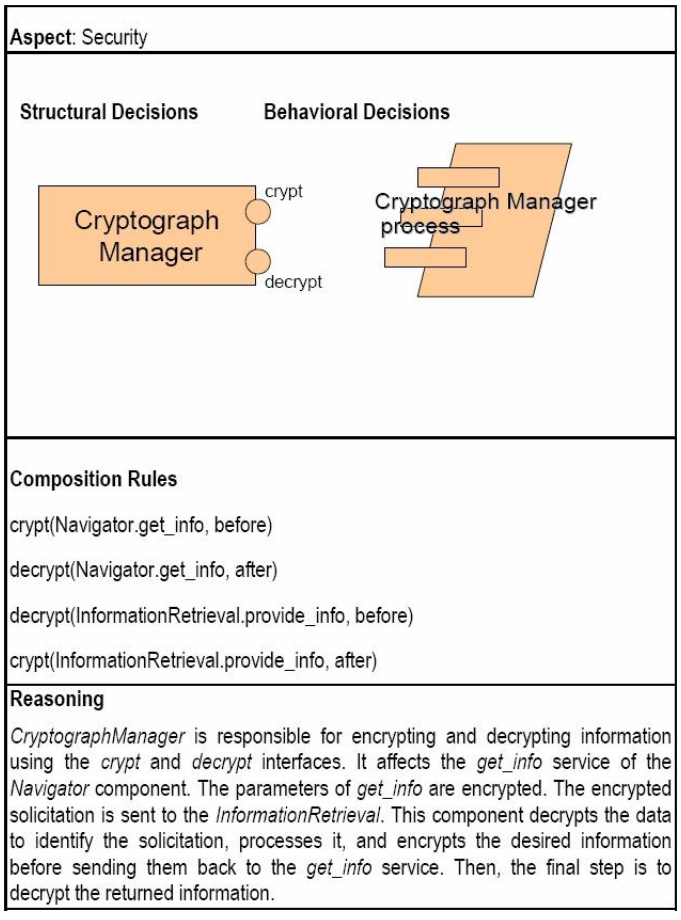


Figura 17 Plantilla para aspectos arquitectónicos

Las decisiones que afectan a varios elementos transversales de arquitectura se denominan architectural-joint-points. Un architectural-joint-point es un elemento de interés en la descripción de la arquitectura a través del cual dos o más decisiones de arquitectura se pueden componer. Las Architectural-composition-rules apoyan la especificación de composición y permiten el razonamiento de composición. Las

plantillas de OA promueven la composición de las interfaces que permiten al arquitecto hacer explícitas las relaciones e influencias mutuas de las incumbencias de ámbito general.

5.5 Conclusiones de los distintos trabajos en ASOA

La primera propuesta en el modelado de ASOA es "Theme/UML". La mejor aportación de este planteamiento es el intento de trazabilidad con la IROA. Sin embargo, propone un lenguaje independiente de UML para el modelado de los componentes de una AS.

La segunda propuesta "Aspect-Oriented Software Development with Use Cases" es la continuación de la mencionada en las propuestas de IROA. Es decir, comienza con modelos para capturar incumbencias de los interesados en el producto software para continuar refinando y llegar a definir un modelo de análisis, después uno de diseño y finalmente uno de implementación. De esta forma, la mayor aportación de esta propuesta es el intento de definir los límites del análisis y el diseño, pero con resultados muy ambiguos.

En el siguiente planteamiento "Aspectual Software Architecture Analysis Method", la propuesta se centra en la evaluación de la AS a través del modelado de escenarios. Esta es una aportación que proporciona de alguna manera el modelado de la parte dinámica de la AS.

La última propuesta "Driving and Managing Architectural Decisions with Aspects" propone modelar los aspectos arquitectónicos por separado de la AS. En esta línea, la aportación de una plantilla para especificar las decisiones transversales de los aspectos arquitectónicos es una idea interesante.

Cada una de las propuestas analizadas proporciona conceptos importantes a tomar en cuenta en la propuesta de una metodología para el desarrollo de una ASOA. Sin embargo, todas son dependientes de su aceptación por parte de la comunidad investigadora en la AS. Debido a esta premisa, se tomaron en cuenta las principales aportaciones de las diferentes propuestas de desarrollo de ASOA para la resolución propuesta en esta tesis doctoral.

6 Análisis global de IROA y ASOA

En base al alcance de esta tesis doctoral se han analizado los trabajos relacionados con la IROA y la ASOA. Se puede concluir que la OA contribuye a la solución de algunos de los problemas que se presentan en los procesos de desarrollo software dentro de las factorías de productos software:

- Las factorías de software usan técnicas de levantamiento de requisitos y el diseño de las primeras líneas de solución siguiendo modelos tradicionales, donde la experiencia en el uso de las técnicas de los analistas y diseñadores se convierte en el factor más importante de éxito. Las propuestas de IROA y ASOA intentan proporcionar los medios para normalizar la calidad del proceso y obtener productos software de calidad.
- Al aplicar IR, las factorías de productos software realizan la captura y análisis de los requisitos desde la perspectiva del cliente. Esto está orientado a lograr buenos niveles de validación de los requisitos y lograr modelos de análisis muy cercanos al producto requerido e identificar los riesgos del desarrollo en etapas posteriores. Los enfoques de IROA soportan esta labor pero desde el uso de las técnicas que involucran diferentes visiones de los clientes, proporcionando estándares de

separación y clasificación de las incumbencias según las necesidades de todos los participantes del proyecto. Acercar estos enfoques al análisis del dominio del problema ha sido importante para lograr una mejor acogida por parte de los clientes y usuarios finales, quienes son protagonistas en el proceso de IR.

Sin embargo, se identificaron algunas desventajas en los enfoques:

- Uno de los grandes problemas que actualmente enfrentan las factorías de productos software es el soporte a la trazabilidad durante el proceso de desarrollo software. Las herramientas CASE proveen soporte a la trazabilidad de los artefactos de modelado, pero la gestión de este mapeo no es suficiente, dado que es un conocimiento que crece exponencialmente y se desvirtúa con facilidad durante el proceso de desarrollo. En esta línea, las propuestas OA deberían priorizar el planteamiento de técnicas sistemáticas para la gestión de la trazabilidad y apoyarse en herramientas CASE con el objetivo de facilitar el viaje de ida y vuelta del producto software en cualquier momento a lo largo de las etapas del proceso de desarrollo software. Sin embargo, las diferentes propuestas de IROA y ASOA soportan superficialmente la trazabilidad, proponiendo con poca frecuencia la utilización de herramientas CASE.
- Por otro lado, actualmente la integración efectiva de los enfoques OA en los marcos metodológicos de la industria del software se hace únicamente a nivel de codificación. En la adopción de las propuestas de la IROA y la ASOA, no se han registrado aún trabajos en el campo industrial, solo en el campo académico. Generalmente, esto ocurre debido al escepticismo de las factorías de desarrollo software en la adopción de nuevas técnicas que prometen mejorar la calidad pero con la consecuencia de aumentar los costes.
- Para una factoría de software es muy importante contar con prácticas de aplicación flexible al integrar nuevas técnicas y tecnologías en sus marcos metodológicos de acuerdo con las condiciones de un proyecto o aplicación específica. Ahora bien, las diferentes propuestas actuales de IROA y ASOA están basadas en diferentes enfoques complejos con una reducida compatibilidad entre ellos. Así mismo, no todas las aproximaciones son adecuadas para toda la diversidad de líneas de trabajo para desarrollar software. Por lo tanto, más que trabajar en un enfoque aspectual específico, es importante contar con una propuesta de entorno de trabajo para permitir medir la calidad de las propuestas de una manera objetiva frente a los términos abstractos del paradigma OA. Así mismo, es necesario un marco de referencia, técnicas y módulos complementarios a las herramientas CASE flexibles que ayuden a seleccionar o adaptar Procesos de Desarrollo Software (PDS) con el enfoque aspectual más apropiado para todo el proceso de desarrollo o en alguna etapa específica.
- Los actuales PDS para la producción de software de gran tamaño, requieren ser incrementales e iterativos para potenciar el desarrollo en paralelo de los distintos flujos de trabajo, y por tanto el trabajo en paralelo de distintos equipos de personas. Así mismo, en el ámbito de la OA se requiere una creación de modelos aspectuales interdependientes sin una precedencia temporal estricta de todo el producto software. Esta necesidad de un método que controle la lógica temporal de los modelos de los posibles niveles de aspectos actualmente no se contempla en las propuestas de IROA y ASOA.
- Es frecuente encontrar situaciones en las que un modelo de incumbencias, claramente encapsuladas, está afectado por más de un aspecto trasversal en un mismo punto, es decir, un determinado punto de unión está asociado a diferentes aspectos y por lo tanto cada uno de los aspectos puede añadir diferente comportamiento, provocando una interacción entre estos, cuyo resultado no es

consistente y predecible. Este factor crítico de gestión de los conflictos entre aspectos en cualquiera de las etapas tempranas del DSOA no es contemplado por las propuestas anteriores con el nivel de importancia necesario.

- El desarrollo de productos software complejos, ya sean Orientado a Objetos puros o con la simbiosis OA, siempre implica un alto porcentaje de posibilidades de cambio en los requisitos a lo largo del tiempo del ciclo de vida, aumentado considerablemente las posibilidades de fracaso. Sin embargo, a pesar de la importancia de esta necesidad ninguna de las propuestas propone algún método para solventar los cambios a lo largo del tiempo.
- Finalmente, una necesidad que no se ha tomado en cuenta en las propuestas OA es una metodología que cubra la especificación eficaz y formal del conocimiento de las etapas tempranas. En su mayoría la especificación de requisitos utiliza técnicas basadas en lenguajes naturales. Así, las diferentes propuestas de IROA proponen utilizar cualquier técnica o simplemente continuidad en la libertad del lenguaje natural. Sin embargo, desarrollar software con una verdadera OA requiere representación semántica en las etapas tempranas de un desarrollo software.

Muchas de estas desventajas de las actuales metodologías de IROA y ASOA ponen de manifiesto lo poco eficaz que suelen ser los modelos tempranos. Dichos modelos contienen una gran cantidad de conocimiento que requiere ser gestionada. Sin embargo en todas estas metodologías no se contempla cómo se realizará la especificación del conjunto de aserciones y reglas que modelan el conocimiento necesario. Tampoco se explica cómo gestionarán el conocimiento de dichos modelos para obtener conclusiones y resolver un problema determinado.

En este punto, se observa que la actividad más complicada de realizar en las etapas tempranas es el modelado del conocimiento. Esto planteó estudiar los métodos propuestos por la ingeniería del conocimiento, en donde, resuelven gran parte de este problema con la utilización de ontologías. Las ontologías permiten representar el conocimiento de manera intuitiva y extensible. Así mismo, en la actualidad muchas investigaciones han invertido muchos esfuerzos para facilitar su creación y mantenimiento.

7 Ingeniería del Conocimiento

Una parte importante en la propuesta de esta tesis doctoral es la aplicación de técnicas y métodos de ingeniería del conocimiento. El principal objetivo es lograr una representación y organización del conocimiento útil para un DSOA.

Actualmente, Gestión del Conocimiento (del inglés *Knowledge Management*) (Wiig, 1997) es el concepto utilizado en las organizaciones cuando pretenden extraer datos relevantes de situaciones concretas, con el fin de organizarlos en contextos específicos buscando contener información formal y transmitirla entre sus miembros, de modo que pueda ser utilizada como un recurso disponible para su transformación en conocimiento. Usualmente este proceso implica técnicas y metodologías para capturar, clasificar y almacenar información relevante de la organización y transformarla en un activo intelectual que preste beneficios y se pueda compartir.

Estas técnicas y metodologías son proporcionadas por la disciplina de la i**ngeniería del conocimiento.** La ingeniería del conocimiento forma parte de la Inteligencia Artificial, cuyo fin es el diseño y desarrollo de **sistemas basados en el conocimiento**. Esta disciplina conduce a una nueva concepción estructural en el modelado y codificación del conocimiento, haciendo una separación entre el conocimiento de un dominio común

codificado en la máquina (base de conocimientos) y los mecanismos deductivos (motor de inferencias), obteniendo un alto grado de representación semántica.

En este escenario, el concepto conocimiento adquiere mucha importancia; sin embargo, en la comunidad científica no existe una única definición de conocimiento. Por lo tanto, parece lógico intentar aclarar en esta tesis doctoral qué se considera conocimiento.

7.1 Conocimiento

Buscando una referencia de conocimiento desde un punto de vista poco filosófico y más ligado a la tecnología, se encontró una definición en el "Free On-Line Dictionary of Computing"[21] (Howe, 1994):

> The objects, concepts and relationships that are assumed to exist in some area of interest. A collection of knowledge, represented using some knowledge representation language is known as a knowledge base and a program for extending and/or querying a knowledge base is a knowledge-based system. Knowledge differs from data or information in that new knowledge may be created from existing knowledge using logical inference. If information is data plus meaning then knowledge is information plus processing[22].

Dicho de otra manera, el conocimiento es el resultado de un proceso de análisis de la información, que ha sido representada y organizada por el emisor con datos relevantes, que conlleva un cambio de comportamiento en un receptor después de interiorizar la información buscando resolver un problema. Por lo tanto, la información requiere un proceso de comunicación entre un emisor y un receptor, sobre un dominio específico con un lenguaje de representación común.

Por otro lado, se ha propuesto conceptualizar diferentes tipos de conocimiento dentro de los sistemas basados en conocimiento. Una de las ideas más difundida de tipos de conocimiento y base de la ingeniería de conocimiento, es la formulada por Polanyi en 1958, donde expone que las personas tienen un **conocimiento tácito** (Polanyi, 1958). Una referencia que fortalece esta idea es la definición del conocimiento tácito del Dictionary of Philosophy of Mind[23] (Barbiero, 2004):

> Knowledge that enters into the production of behaviors and/or the constitution of mental states but is not ordinarily accessible to consciousness[24].

Así, la ingeniería del conocimiento distingue entre el **conocimiento explícito**, que puede capturarse y expresarse en fórmulas, documentos, plantillas, procedimientos, o

---

[21] Diccionario de la computación en línea y gratuito.

[22] "Conocimiento. Los objetos, conceptos y relaciones que se supone que existen en un área de interés. Una colección de conocimientos, representada utilizando un lenguaje de representación del conocimiento, se llama base de conocimiento, y un programa para ampliar y/o consultar una base de conocimiento es un sistema basado en conocimiento. El conocimiento difiere de los datos o la información en que a partir del conocimiento existente puede crearse nuevo conocimiento utilizando la inferencia lógica. Si la información es datos más significado, el conocimiento es información más procesamiento".

[23] Diccionario de filosofía de la mente.

[24] "El conocimiento que toma parte en la producción de comportamientos y/o la constitución de estados mentales pero que no es normalmente accesible a la consciencia".

cualquier otro medio de expresión, y el **conocimiento tácito**, que reside en las mentes de las personas y es inaccesible incluso a su consciencia, y que sólo se manifiesta por sus resultados. Por lo tanto, el conocimiento tácito no puede expresarse explícitamente en los sistemas basados en conocimiento, que es lo que aquí nos interesa.

Ahora bien, en los trabajos relacionados con la ingeniería del conocimiento se ha adoptado que el conocimiento, así como la información, son conceptos que no pueden definirse como un conjunto de condiciones necesarias y suficientes universalmente válidas (Hanson & Bauer, 1989), es decir, son conceptos **polimorfos** (del inglés *polymorphy*). Por lo tanto, el conocimiento debe definirse sobre el marco de un contexto particular, desde una perspectiva determinada y con un propósito concreto, que ha sido la base en la compresión del conocimiento en los desarrollos basados en conocimiento.

En este sentido, se ha dicho que el conocimiento que se aplica para resolver determinados tipos de problemas puede expresarse de dos formas: **declarativa**, donde se especifican los objetos, las propiedades y las relaciones generales, y se deja al cuidado del receptor que ha de resolver los problemas la aplicación de mecanismos generales de razonamiento; y **procedimental,** en el cual se especifica un procedimiento para resolver los problemas.

El modelado de este conocimiento puede ser de dos tipos: el conocimiento **normativo** que es el contexto particular de un dominio y el conocimiento **factual** que se refiere a los datos concretos de un problema a resolver, así como, a las conclusiones intermedias y finales. Este conocimiento factual incluye dos tipos que son de la misma forma, aunque de distinto origen: el que está basado en observaciones, o sea, datos que al interpretarlos el sistema cobran un significado para él, convirtiéndose en información, y en conocimiento al integrarse; y el que procede de razonamientos, que podemos llamar **inferencial**, y que puede ser o no ser verificable por observación o experimentación.

Otro tipo de conocimiento, el **estratégico**, se refiere a la manipulación del conocimiento factual y normativo. Si el normativo está en forma de reglas, se refiere a la estrategia de búsqueda que decide en cada momento qué regla se aplica. Pero, independientemente de la estrategia, al aplicar la regla se está usando un cuarto tipo de conocimiento: el que dice que si se satisface el antecedente, la regla permite afirmar que se satisface el consecuente, llamado conocimiento de tipo **táctico**.

7.2 Representación del Conocimiento

Se ha dicho que el objetivo de la ingeniería del conocimiento es el diseño y desarrollo de sistemas basados en el conocimiento. Estos sistemas son un conjunto de programas que actúan sobre una **base de conocimiento** usualmente con el objetivo de realizar **razonamiento** deductivo automático. El objetivo de una base de conocimiento es el de representar y almacenar bajo forma digital el conocimiento de una manera que facilite su gestión e inferencia. Por lo tanto, la **representación de conocimiento** es un medio de expresión y comunicación imprescindible.

Una representación del conocimiento es la creación de modelos de información bien definida, de manera que pueda ser interpretada por agentes humanos y agentes automáticos. Esta trasferencia y creación de conocimiento requiere de un lenguaje que proporcione una estructura formal para facilitar la extracción y recuperación de información para realizar una determinada incumbencia. No obstante, el principal problema en la representación del conocimiento es llegar a un consenso sobre qué representar y cómo hacerlo, implicando siempre una manifestación semántica (Fernández-López, Gómez-Pérez, & Corcho, 2004).

Diferentes disciplinas han propuesto una serie de normalizaciones para la representación semántica de su información. Las metodologías y técnicas para el modelado del conocimiento dependen de la finalidad y del contexto.

Existe una propuesta muy aceptada sobre cómo en cualquier contexto la semántica se puede representar mediante tres elementos básicos de diferente nivel de construcción (Daconta, Obrst, & Smith, 2003):

- El primer nivel equivale a la descripción de conceptos simples u objetos.
- El segundo nivel representa un modelado de conocimiento sobre las cosas u objetos, es decir, establecer relaciones entre los objetos del primer nivel.
- Por último, el tercer nivel representa una superestructura de un "modelo del mundo cerrado" y este nivel de representación semántica requiere una serie de reglas y lógica que permitan hacer inferencias de conocimiento. De este último punto se extrae que el grado de representación semántica que se quiera plasmar condiciona los lenguajes en el proceso de creación de los modelos de conocimiento.

La ingeniería del conocimiento muestra cómo se puede describir un concepto con un conjunto de características. Dicha descripción representa el conocimiento que se posee de un concepto. La suma de todas las características de un concepto es lo que se conoce como su **intensión**, mientras que el conjunto de objetos a los que un concepto hace referencia es su **extensión**. La extensión de un concepto es inversamente proporcional a su intensión; es decir, un concepto que posea muchas características tiene una intensión amplia y, en consecuencia, su extensión será limitada, ya que sólo puede hacer referencia a un número limitado de objetos. Por otro lado, un concepto que posea pocas características tendrá una intensión limitada y una extensión amplia. Los conceptos han de ser organizados de forma sistemática y caracterizados de acuerdo con las relaciones que establecen con otros conceptos en el seno de un sistema conceptual. Tradicionalmente, han sido cuatro las clases de relaciones conceptuales que se han considerado fundamentales para la estructuración de un campo conceptual:

- Relaciones genérico / específico: relación jerárquica en la que se identifica a los conceptos por su pertenencia a una categoría, en la que un concepto genérico se considera padre de otros conceptos más específicos. Los conceptos hijos comparten las características del concepto genérico pero, además, poseen algunas peculiaridades propias que los diferencian y hacen más específicos. Se establece, por tanto, una relación que va en dos sentidos diferentes: vertical, es decir, la que se establece entre un concepto específico y su genérico; y horizontal, la que sostienen varios conceptos específicos que poseen el mismo genérico y que se diferencian entre sí por poseer alguna característica distintiva.
- Relaciones parte / todo: se refiere a la que existe entre conceptos que están formados por más de una parte y dichas partes constituyentes.
- Relaciones polivalentes: en las que se da cuenta de la posibilidad de que un concepto pueda colocarse en lugares diferentes en un mismo sistema conceptual.
- Relaciones complejas: categoría en la que se engloban una serie de interrelaciones que se establecen entre conceptos en una jerarquía, pero que no pueden considerarse ni genéricas ni partitivas. Ejemplos de este tipo de

relaciones son las de causa-efecto, actividad-lugar de realización, proceso-producto, etc.

7.2.1 Representación formal

Cuando se diseña una representación del conocimiento la decisión más importante que hay que tomar es la expresividad de la representación del conocimiento. La expresividad está estrechamente relacionada con el lenguaje y el marco de trabajo. Cuanto más expresivo es un modelo, más difícil es derivar inferencias automáticamente de él, sin embargo, cuanto menos expresiva, es más fácil y compacta. Las representaciones del conocimiento poco expresivas pueden ser tanto completas como consistentes, así como las más expresivas pueden ser incompletas e inconsistentes. Por lo tanto, el principal problema es encontrar una representación del conocimiento y un sistema de razonamiento que pueda hacer las inferencias que se necesiten dentro de los límites de recursos del problema a tratar.

Para comprender la representación del conocimiento es interesante observar el desarrollo de la disciplina de **documentación**, disciplina donde siempre se han compartido los mismos objetivos de la ingeniería del conocimiento. Dicha disciplina siempre ha tenido la necesidad de representar semánticamente los objetos de información para su posterior recuperación. Para conseguir este objetivo surgieron propuestas de representación con un claro aumento de complejidad y eficiencia en el tiempo; desde listas de términos (ejemplos: listas de autoridades, glosarios, índice de libros, etc.); pasando por sistemas de clasificación y categorización (ejemplos: encabezamientos de materia, esquemas de clasificación, taxonomías, etc.); hasta agrupaciones con relaciones normalizadas (ejemplo: tesauros).

Entre todas las propuestas de modelado formal para la representación semántica de información la más representativa en documentación son los tesauros. Los **tesauros** son vocabularios controlados de términos en el lenguaje natural definido por diferentes normas (ISO 2788:1986, UNE-50-106-90:1990, ANSI/NISO Z39.19:2005, ISO 25964-1:2011, etc.); dichos términos están relacionados entre ellos en estructuras jerárquicas con relaciones de equivalencia (controla la sinonimia, homonimia, antonimia y polisemia entre los términos) y relaciones asociativas (mejoran la semántica para su posterior recuperación y ayudan a reducir la polijerarquía entre los términos). Podría decirse que los tesauros son un intermediario entre el lenguaje natural y el que emplean los especialistas de un determinado dominio.

La **lingüística** también tiene un desarrollo de representación del conocimiento, teniendo su mayor aportación formal en las Redes Semánticas (del inglés *Semantic Networks*) (Minsky, 1969)(Shapiro, 1969). Los conceptos y sus relaciones se representan mediante un grafo donde los elementos semánticos se representan por nodos unidos por arcos y un conjunto de procedimientos de inferencia que operan sobre las estructuras. Existen diferentes técnicas utilizadas como: redes IS-A, grafos conceptuales o redes de marcos.

Sin embargo, la disciplina con un alto grado de interés en esta tesis doctoral es la **ingeniería de software**. En la ingeniería del software se pueden encontrar diferentes propuestas formales de modelado para la representación de semántica de información. Entre las más importantes podemos identificar las siguientes:

- El modelo Entidad-Relación (ER) (Barker, 1994) es una propuesta de modelado conceptual mediante el cual se pretende modelar los objetos que pertenecen a la Base de Datos de un sistema informático utilizando

entidades que pueden contener atributos y se vinculan mediante relaciones, acotadas por las cardinalidades de la relación.

- UML (Unified Modeling Language) es una especificación de notación para la representación de modelos en la programación OO. Para representar la semántica de una aplicación informática UML dispone de dos tipos diferentes de diagramas: los que dan una vista estática del sistema y los que dan una visión dinámica. Así, los diagramas estáticos y dinámicos pueden representar conceptos (objetos, clases, componentes) con sus relaciones (asociaciones, enlaces, conectores) con la posibilidad de añadir restricciones por el tipo de dato, el rango de valores, la multiplicidad entre clases y las restricciones entre relaciones.
- RSHP (Juan Llorens, Morato, & Génova, 2004) es un modelo de representación que permite manejar todo tipo de objetos (textos, modelos de diseño, código, bases de datos, etc.) utilizando el mismo esquema de representación, y por lo tanto es posible una generalización de su gestión en los sistemas informáticos. La fortaleza del modelo de RSHP depende de la definición de las relaciones permitidas. Términos dinámicos (verbos) también deben ser modelados para representar acciones dentro de las relaciones. Las particularidades del modelo de RSHP permiten representar tipos de artefactos que usan la misma información que contienen. Esto significa que no hay pérdida de información en el proceso de indexación de los artefactos, y se hace posible generar el artefacto original de su representación. Este modelo permite la reutilización de la representación real de todo tipo de artefactos, especialmente en la disciplina de ingeniería de software.

Estas propuestas de modelado para la representación semántica de información están pensadas para desarrollar aplicaciones informáticas a lo largo del ciclo de vida.

7.2.2 Lenguajes de marcado

Topic Maps ™ es una propuesta de representación del conocimiento previa a las actuales con una codificación de marcado. Topic Maps ™ es una especificación para caracterizar y categorizar, documentos y secciones de documentos en la web a partir de su contenido. Un Topic Map está orientado a la indización de contenidos en conjuntos de documentos. El paradigma de Topic Maps permite indizaciones múltiples para la misma colección de documentos web. Los Topic Maps están definidos por la norma ISO 13250 y están especificados en sintaxis XML para crear Topic Maps, que se denomina XTM. En el estándar XTM se especifican los elementos y los conceptos claves: topic (es el término que expresa determinado concepto o idea), name (un topic puede tener varias denominaciones, pero debe estar representado por una forma base), association (es un enlace que establece una relación entre dos o más topics) y occurrence (son enlaces a recursos informativos).

Sin embargo, los desarrollos más importantes en la representación del conocimiento con lenguaje de marcado han sido liderados por la web semántica. La **web semántica** (Berners-Lee, 2000) es una web extendida que está suponiendo una revolución en el uso de la web. Esta web extendida promete estar dotada de mayor significado en la que cualquier usuario en internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. Al dotar a la web de más semántica, se pueden obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización de una infraestructura común, mediante la cual, es

posible compartir, procesar y transferir información de forma sencilla. La web semántica está basada en la representación del conocimiento. Para que esto ocurra, se requieren lenguajes y formalismos universales que resuelvan los problemas ocasionados por una web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante.

En este contexto, la web semántica se ha enriquecido con el desarrollo de lenguajes y estándares de representación del conocimiento basados en lenguajes de marcado para su codificación. Estas propuestas formales están pensadas para diseñar y representar bases de conocimiento, donde las expresiones típicas que proponen son la determinación de conceptos, relaciones, restricciones formales, reglas de inferencia y funciones (Antoniou & Harmelen, 2004). Para obtener un adecuado modelado de la base de conocimiento, la web semántica utiliza esencialmente RDF SKOS y OWL.

RDF (Resource Definition Framework) (W3C Recommendation, 2004c) es un modelo de representación desarrollado por el consorcio de la red mundial W3C (World Wide Web Consortium) originalmente diseñado como un modelo de datos para metadatos, usado como un método para el modelado de la información que se implementa en los recursos Web. Los conceptos de RDF son: modelo de dato gráfico, vocabulario basado en URI, tipos de datos, literales, sintaxis de serialización XML, expresión de hechos simples y vinculación. El modelo RDF se basa en la idea de hacer declaraciones sobre los recursos en forma de expresiones sujeto-predicado-objeto. Estas expresiones son conocidas como tripletas. El sujeto indica el recurso y el predicado denota rasgos o aspectos del recurso y expresa una relación entre el sujeto y el objeto. Una colección de sentencias RDF representa intrínsecamente una multigrafo dirigido y etiquetado. Como tal, un modelo de datos basado en RDF se adapta de forma más natural a cierto tipo de representación del conocimiento que el modelo relacional. Por otro lado, la especificación RDF Schema (Resource Definition Format Schema – RDF Schema) es una extensión de RDF, que enriquece el modelo básico, y permite especificar una jerarquía explícita de clases de recursos y las propiedades de las clases, así como las restricciones sobre las combinaciones admitidas entre clases, propiedades y valores.

OWL (Ontology Web Language) (W3C Recommendation, 2004a) es una normalización desarrollada por el W3C para crear y compartir ontologías en la Web. OWL tiene más facilidades para expresar significado y semántica que el contenido Web soportado solo por los mecanismos admitidos por XML, RDF y RDF Schema. OWL añade más vocabulario para describir propiedades y clases tales como relaciones entre clases, cardinalidad, igualdad, tipologías de propiedades más complejas, caracterización de propiedades o clases enumeradas; es decir, proporciona vocabulario adicional junto a una semántica formal. OWL se deriva como extensión del lenguaje de ontología Web DAML+OIL en la que inicialmente apareció OWL 1 como la primera propuesta. Posteriormente, un nuevo grupo de trabajo del W3C anunció nuevas características en la versión OWL 2 (W3C Recommendation, 2012), incluyendo mayor poder expresivo y definiendo nuevos perfiles para atender mejor ciertos requisitos de desempeño.

SKOS (W3C_Working_Group_Note, 2009) (del inglés *Simple Knowledge Organization Systems*) es un esquema RDF proporcionado por la W3C para la representación de la estructura básica y el contenido de esquemas de conceptos como tesauros, esquemas de clasificación, listas de encabezamientos de materia, taxonomías, folksonomías y otros vocabularios controlados similares. Un uso básico de SKOS es permitir identificar los recursos conceptuales mediante URI, etiquetarlos con literales de uno o varios idiomas, documentarlos con diversos tipos de notas, relacionarlos entre sí mediante estructuras jerárquicas informales o redes asociativas, y agregarlos a esquemas de conceptos. SKOS ha sido diseñado para proporcionar un modo de migrar a la Web Semántica sistemas de

organización del conocimiento ya existentes con un coste bajo. SKOS también puede contemplarse como una tecnología de transición que proporciona un nexo de unión entre el formalismo lógico riguroso de los lenguajes de ontologías como OWL y el mundo caótico, informal y débilmente estructurado de las herramientas colaborativas basadas en Web, ejemplificadas por las aplicaciones de etiquetado social. La aplicación más avanza de SKOS permite realizar mapeos entre conceptos pertenecientes a diferentes esquemas de conceptos, lo que se puede entender como una asociación semántica entre conceptos pertenecientes a distintas ontologías.

7.2.3 ODM

La W3C no es la única organización que ha buscado facilitar la representación de conocimiento con ontologías. La OMG ha buscado mejorar la interoperabilidad entre los distintos lenguajes con la publicación del Metamodelo para la Descripción de Ontologías (del inglés *Ontology Definition Metamodel –ODM-*) (Object_Management_Group-(OMG), 2009).

ODM es una especificación para el modelado de ontologías desarrollado por la OMG. ODM incluye a seis metamodelos (cuatro que son normativos, y dos que son informativos). Estos se agrupan en función de la naturaleza de los formalismos de representación: formales de primer orden y lógica descriptiva; estructurales y de subsunción o representaciones descriptivas; y conceptuales tradicionales o de modelado de software orientado a objetos.

En el núcleo están dos metamodelos que representan los lenguajes lógicos formales: Lógica Descriptiva (del inglés *Description Logics –DL-*), que se incluye como informativo y CL (del inglés *Common Logic*). Mientras que la jerarquía de estos lenguajes es distinta, juntas abarcan una amplia gama de representaciones que van desde un orden superior de forma probabilística y las representaciones intencionales hasta la expresión de taxonomía muy simple. Posteriormente, hay tres metamodelos que aportan la sintaxis abstracta para las representaciones de carácter más estructural o descriptivo de RDFS (RDF Schema), OWL y Topic Maps (TM). Finalmente, hay dos metamodelos adicionales que se consideran esenciales para los ODM. Estos representan los enfoques más tradicionales de la ingeniería de software para el modelado conceptual: UML2 y ER. Estos son probablemente los dos lenguajes de modelado más utilizados en ingeniería de software, particularmente para modelado conceptual y lógico. Sin embargo, ODM simplemente referencia el estándar de UML2 y proporciona un mecanismo adicional no normativo para el manejo de claves desde la perspectiva de ER. Tres perfiles UML se han identificado para su uso en ODM para RDF, OWL, y Topic Maps. Estos permiten el uso de la notación UML (y herramientas) para el modelado de la ontología y facilitar la generación de descripciones de la ontología correspondiente en RDF, OWL, y TM, respectivamente.

7.3 Ontologías

El concepto de Ontología data de los tiempos del filósofo Aristóteles, sin embargo, en la actualidad coexisten dos usos diferenciados del término ontología. En el primero el término ontología se origina en el campo de la filosofía y la epistemología[25], donde es una rama de la metafísica que se ocupa del estudio de la naturaleza, de la existencia de

[25] Rama de la filosofía que estudia la naturaleza y las fuentes del conocimiento.

los seres y de sus propiedades transcendentales; en filosofía, por tanto, una ontología se considera como una explicación sistemática de la existencia. Este concepto de ontología desde el punto de vista filosófico queda fuera de los alcances de esta tesis doctoral. El segundo uso del concepto de ontología contemplado en este documento tiene un entendimiento mucho más pragmático y aplicado, usado en el ámbito de la ingeniería del conocimiento. Una ontología es entendida como un cuerpo estructurado de conocimiento, que no ha de ser considerada como una entidad natural que se descubre, sino como un recurso artificial que se crea. Una ontología ha de entenderse como un entendimiento común y compartido de un dominio, que puede comunicarse entre científicos y sistemas computacionales. El hecho de que puedan compartirse y reutilizarse en aplicaciones diferentes, explica en parte el gran interés suscitado en los últimos años en la creación e integración de ontologías.

Cuando hablamos de ontologías como modelos de representación de conocimiento debemos especificar a qué tipo de conceptualización de conocimiento nos referimos. La naturaleza humana ha tendido a la clasificación de objetos en forma de taxonomía para representar el conocimiento de forma jerárquica, estableciendo entre los conceptos una relación de generalización-especialización. En los últimos años para categorizar un conjunto de conceptos con semántica más compleja surgieron otras propuestas de organización cognitiva como los tesauros, modelos conceptuales y finalmente las ontologías. En este contexto, la disciplina de la Ingeniería Ontológica ha considerado estructuras existentes de organización del conocimiento como ontologías pero con diferentes cargas semánticas, aunque tenga características y rasgos distintivos en otras disciplinas. La propuesta más aceptada para diferenciar estas conceptualizaciones de ontologías es el estudio del Espectro de las Ontologías (del inglés *Ontology Spectrum*) (Lassila & Mcguinness, 2001). El marco en el que se desarrolla la clasificación toma como elemento principal la riqueza semántica de su estructura, caracterizando entre semántica débil y semántica fuerte. El propósito de este marco es comparar la capacidad de representación semántica y las formas de representación del conocimiento.

En las ontologías de los sistemas basados en el conocimiento, lo que existe es lo que se puede representar, y lo que se representa es la conceptualización sobre la cual se quiere hablar y razonar. Por lo tanto, una definición de ontología a tomar en cuenta es la de Gruber (Gruber, 1995), donde dice que **una ontología es una especificación explícita formal de una conceptualización compartida**. En esta definición, **conceptualización** se refiere a un modelo abstracto de algún fenómeno del mundo del que se identifican los conceptos que son relevantes; **explícito** hace referencia a la necesidad de especificar de forma consciente los distintos conceptos que conforman una ontología; **formal** indica que la especificación debe representarse por medio de un lenguaje de representación formalizado; y **compartida** refleja que una ontología debe ser resultado de un consenso de conocimiento en una determinada comunidad. Así mismo, propone modelar ontologías con marcos y lógica de primer orden con los siguientes seis tipos de componentes:

1. Las Clases o conceptos representan ideas básicas que se intentan formalizar en algún dominio de interés.
2. Las Instancias representan objetos determinados de un concepto en una ontología.
3. Las Relaciones entre las cosas que representan un tipo de asociación entre conceptos de un dominio.
4. Las Propiedades y sus valores de los anteriores elementos.

5. Las Funciones o desarrollo de los procesos es un tipo especial de relación donde uno de los elementos de la relación se identifica mediante un cálculo de una función que considera otros elementos de la ontología.
6. Los Axiomas formales son restricciones y reglas declaradas para que los elementos de la ontología los cumplan.

Se puede observar que desde este punto de vista, una ontología trata de representar dos tipos de conocimiento: el primero se puede llamar "genérico", que se refiere a las clases y los modelos de los problemas del dominio (también denominado intensión); y el segundo "concreto", que se refiere a las instancias (también denominado extensión). Así, este punto de vista de la ontología se puede utilizar en los programas informáticos para una variedad de propósitos, incluyendo el razonamiento inductivo, la clasificación, y una variedad de técnicas de resolución de problemas.

Por otro lado, actualmente no existe consenso sobre los diferentes tipos de ontologías, detectándose discrepancias entre las clasificaciones de un investigador a otro. Se ha observado que la concepción y clasificación de los diferentes tipos de ontologías se hace en función de los objetivos de aplicación. En este sentido, un objetivo común a tomar en cuenta es **el grado de posible reutilización**; dado que la construcción de una ontología se traduce generalmente en una labor manual, lenta y costosa; Corcho, Fernández-López y Gómez-Pérez (Fernández-López et al., 2004) lo ponen de manifiesto al analizar dicha cuestión.

Uno de los primeros trabajos relacionados con la clasificación de tipos de ontologías que toman en cuenta la posible reutilización es el de Mizoguchi (Mizoguchi, Vanwelkenhuysen, & Ikeda, 1995). Esta propuesta clasifica cuatro tipos ontologías (para reutilizar el conocimiento, para transmitir conocimiento, para extracción y las superiores) en donde los tipos de ontologías para reutilizar el conocimiento contienen tres subcategorías: Ontologías de tarea, de dominio y general o común. Van Heijst (van Heijst, Schreiber, & Wielinga, 1997) dio un paso más allá, clasificando las ontologías en dos dimensiones ortogonales: por el tipo de estructura y por conceptualización. En la primera se distinguen tres categorías: ontologías terminológicas, de información y que representen conocimiento. En la segunda dimensión se identifican cuatro categorías: ontologías de representación del conocimiento, general, de dominio y de aplicación. En cada una de estas categorías de la segunda dimensión está contenida la intención de reutilización.

Bajo este principio se han encontrado en la literatura algunos tipos fundamentales de ontologías basados en el tema de conceptualización. Estas ontologías son definidas para utilizar un vocabulario común en los trabajos de ingeniería ontológica desarrollados en esta investigación:

Ontologías de Representación del Conocimiento (van Heijst et al., 1997) **o de nivel superior** (Mizoguchi et al., 1995) (del inglés *Top-level o upper-level*) capturan las primitivas de representación (por ejemplo: clases, subclases, atributos, valores, relaciones y axiomas) y proveen nociones generales bajo las cuales todos deben ser enlazados para formalizar el conocimiento bajo un paradigma de representación del conocimiento. Entre las ontologías de representación del conocimiento conocidas tenemos RDF, RDF Schema, OIL, DAML+OIL y OWL.

Ontologías Generales (van Heijst et al., 1997) **o comunes** (Mizoguchi et al., 1995) son usadas para representar conocimiento de sentido común reutilizable a través de dominios. Estas ontologías incluyen vocabulario relacionado con cosas, eventos, tiempo,

espacio, causalidad, comportamiento, función, mereología (parte-todo) (Borst, 1997), etc.

Ontologías de Dominio (Mizoguchi et al., 1995)(van Heijst et al., 1997) son reutilizables en un dominio específico dado (médico, farmacéutico, ingenieril, jurídico, empresarial, automotriz, etc.). Estas ontologías proveen vocabularios acerca de conceptos dentro de un dominio y sus relaciones acerca de las actividades que toman lugar en ese dominio, y acerca de las teorías y principios elementales que gobiernan ese dominio.

Ontologías de Tareas (Mizoguchi et al., 1995)(Guarino, 1998) describen el vocabulario relacionado con una tarea o actividad genérica (diagnosticar, planificar, vender, etc.) mediante la especialización de los términos en las ontologías de nivel superior. Las ontologías de Tareas proveen un vocabulario sistemático de los términos usados para resolver problemas asociados con tareas que pueden o no pertenecer al mismo dominio.

Ontologías de Aplicación (van Heijst et al., 1997) son dependientes de las aplicaciones. Contienen todas las definiciones necesarias para modelar el conocimiento requerido para una aplicación particular. Las ontologías de aplicación frecuentemente extienden y especializan el vocabulario de un dominio y de ontologías de tareas para una aplicación dada.

Ontologías lingüísticas (Fernández-López et al., 2004) son las que consideran las palabras como unidades gramaticales y están pensadas para describir construcciones semánticas, más que para proponer modelos específicos de un dominio. La finalidad de las ontologías lingüísticas (por ejemplo WordNet (Miller, Beckwith, Fellbaum, Gross, & Miller, 1990)) ha sido conformar bases de datos léxicas, aplicándolas a las máquinas de traducción y a la generación de lenguaje natural.

7.4 Tipos de lógica

Los sistemas basados en el conocimiento están compuestos por una **base de conocimiento** en un **lenguaje formal** y un **motor de inferencia** que es lo que permitirá sacar conclusiones y resolver problemas que se le planteen. La base de conocimiento se codifica en un conjunto de sentencias que le permiten al motor de inferencia obtener nuevas sentencias o conclusiones. Así es posible determinar si una sentencia es deducible a partir de las sentencias de su base de conocimiento; también permite determinar si una determinada sentencia lleva a una contradicción a partir de su base de conocimiento.

Un **lenguaje lógico** es un lenguaje formal para representar información de la que se puedan sacar conclusiones, y que dispone de una sintaxis y una semántica. Las lógicas sirven para abstraerse del mundo real y modelar un área de estudio. Si en la abstracción existen solo hechos donde se puede decir que son ciertos, falsos o desconocidos, entonces se está tratando con lógica proposicional. Si el mundo abstraído además de hechos tiene objetos y relaciones, se está utilizando lógica de primer orden. Existen otras lógicas derivadas para la representación de conocimiento que permita implementar razonadores que realicen inferencia.

La **lógica proposicional** carece de signos para variables de tipo entidad pero incluye signos para variables proposicionales y signos simples para la conectividad lógica. Por lo tanto, en este tipo de lógica puede utilizarse la inferencia lógica de proposiciones. Una lógica proposicional es un lenguaje formal cuyos elementos más simples representan proposiciones, y sus conectivos lógicos representan operaciones sobre preposiciones capaces de formar preposiciones de mayor complejidad que son tratadas como

funciones de verdad. En la Tabla 1 se despliegan las conectivas lógicas que ocupan a la lógica proposicional.

Conectiva	Expresión en el lenguaje natural	Símbolo
Verdadero		T
Falso		F
Negación	no	¬
Conjunción	y	∧
Disyunción	o	∨
Condicional material	si... entonces	→
Bicondicional	si y sólo si	↔
Negación conjunta	ni... ni	↓
Disyunción excluyente	o bien... o bien	↮

Tabla 1 Conectivas de la lógica proposicional

La **lógica de primer orden** se ocupa de estudiar el modo en que hablamos y razonamos con expresiones lingüísticas con el suficiente poder expresivo para definir prácticamente todas las matemáticas. La lógica de primer orden se puede ver como una extensión de la lógica proposicional que añade sintaxis y semántica con interpretaciones de términos cuantificados. Entre los conceptos básicos tenemos el *predicado* que es una expresión lingüística que, cuando se conecta con una *expresión* manifiesta una *propiedad* y, cuando se conecta con dos o más expresiones, expresa una relación donde los predicados son tratados como *funciones*. Una función recibe *argumentos* que procesa devolviendo *valores* como resultados. Así mismo, una *constante de individuo* es una expresión lingüística que se refiere a una entidad determinada; sin embargo, cuando no está determinada se utilizan las *variables*. En esta línea, aparentemente en una expresión con una variable sin determinar no es posible asignar un valor de verdad. Sin embargo, es posible dar un valor de verdad a la expresión si se le antepone una expresión que afirma que una condición se cumple para un cierto número de individuos; a estas se les llama **cuantificadores**. Así la lógica de primer orden combina las conectivas de la lógica proposicional con los predicados, constantes, variables y cuantificadores.

Una propuesta que pretende expresar el contenido completo de la lógica de primer orden para la transmisión de conocimiento es la Lógica Común Simple (del inglés *Simple Common Logic - SCL*) (P. J. Hayes & Menzel, 2005). La SCL es un lenguaje propuesto para el intercambio y transmisión de información a través de una red abierta. Además, permite una gran variedad de formas sintácticas, llamados dialectos, expresable todo dentro de una sintaxis basada en XML común y que comparten una misma semántica. El lenguaje tiene una semántica declarativa, lo que significa que es posible entender el significado de las expresiones escritas en SCL sin necesidad de un intérprete para manipular esas expresiones. SCL es un formalismo que extiende notaciones convencionales de primer orden de varias maneras. SCL está basado en la definición de la Lógica Común (del inglés *Common Logic* - CL) fomentando el desarrollo de una variedad de formas sintácticas diferentes.

7.4.1 Lógica Descriptiva

Una de las razones más importantes detrás del creciente interés en las lógicas descriptivas es su relación con la web semántica. Este proyecto global busca añadir metadatos semánticos a los recursos en la web con lenguajes lógicos que puedan ser utilizados para una mejor explotación automática de los mismos. La idea central de este proyecto es la creación de ontologías que contengan semántica que permita a los sistemas basados en conocimiento la capacidad de inferir conocimiento nuevo. Por lo tanto, los lenguajes utilizados en el modelado de estas ontologías son variaciones

sintácticas de algunas lógicas descriptivas, lo cual ha convertido a los DL en el fundamento formal de la web semántica.

Las **Lógicas Descriptivas** (del inglés *Description Logic*- DL) pueden ser vistas como un subconjunto de la lógica de primer orden que permite hacer resoluble sus inferencias. La DL fue diseñada como una extensión de sistemas computacionales y de las redes semánticas. Las DL son una familia de lenguajes de representación de conocimiento formales. Su sintaxis está diseñada especialmente para facilitar la representación de la información de una manera organizada, mientras que su semántica está basada en la lógica clásica de predicados. Cada una de las diferentes DL define un lenguaje para especificar bases de conocimiento. La función de una base de conocimiento es describir la estructura de los conocimientos utilizados en un área específica, utilizando un lenguaje compuesto generalmente por tres tipos de elementos fundamentales: conceptos, también llamados clases, que describen un conjunto de objetos que comparten ciertas características; roles, que describen relaciones binarias entre conceptos; e individuos, que son de instancias específicas de algún concepto. En lógica clásica, estos tres elementos equivalen, respectivamente, a predicados unarios, predicados binarios y constantes.

Los LD brindan un conjunto diferente de constructores, que permiten formar conceptos y roles complejos utilizados para escribir bases de conocimiento.

En general los DL contienen un base de conocimiento que tiene dos partes, TBox (caja de términos) y la ABox (caja de aserciones). En general el componente TBox contempla el vocabulario de un dominio de aplicación en función de conceptos (denotan clases o conjuntos de individuos), roles (denotan relaciones binarias entre individuos) y un conjunto de descripciones complejas sobre este vocabulario. El componente ABox contiene afirmaciones acerca de los conceptos y roles especificando a quién pertenecen los individuos (se le puede considerar información extensional). Una base de conocimiento TBox y ABox es equivalente a un conjunto de axiomas de la lógica de primer orden, por lo tanto se pude considerar a las DL como lógicas computacionales por excelencia. Estos lenguajes fueron específicamente diseñados para la representación y organización de datos. La separación entre la información intencional y extensional es poco común en las lógicas formales, por lo tanto, hace a las DL particularmente útiles en varias áreas de las tecnologías de información.

Al estar separada la base de conocimiento en bloques TBox y ABox, el razonamiento también es dividido.

El razonamiento en la TBox hace posible verificar la posibilidad de un concepto (Concep Satisfiability); esto consiste en verificar si existe un modelo para un concepto, es decir, encontrar al menos un individuo que lo cumpla. Así mismo es posible comprobar si una nueva aserción se puede inferir directamente de las aserciones de la TBox; esta característica es una inclusión (Subsumption). Una característica importante es la posibilidad (Satisfiability) de ver si no hay ninguna incongruencia o inconsistencia en dicha TBox.

Con el razonamiento en la ABox es posible comprobar si una instancia pertenece a una clase determinada (Instance Checking o InferredTypes). Una característica difícil de realizar es verificar la consistencia de si lo que se dice en la ABox es coherente con ella misma y con lo descrito en la TBox.

7.4.2 Familia de DL

Uno de los principales objetivos de los DL es proporcionar un equilibrio con sus constructores permitidos entre la expresividad del formalismo y la dificultad computacional de hacer inferencias. Por lo tanto, la existencia de los diferentes lenguajes de DL permite una selección adecuada que permita hacer inferencias con la mayor eficacia posible en cada contexto.

En la Tabla 2 podemos ver una comparativa de los distintos DL, indicando sus características y ordenados de menor a mayor complejidad. Como se verá más adelante, el lenguaje utilizado en esta tesis doctoral para describir las ontologías es OWL. Este lenguaje dispone de varios niveles de expresividad, donde cada nivel implementa uno u otro lenguaje de DL según su nivel de expresividad.

Nombre	Elementos	Complejidad	Soporte
ALC	⊤, ⊥, ⊔, ⊓, ¬, ∃, ∀	ExpTime	
S	*ALC + roles transitivos*	ExpTime	
SI	*S + roles inversos (I)*	ExpTime	
SH	*S + jerarquía de roles (H)*	ExpTime	OIL
SHIQ	*SHI + restricciones numéricas cualificadas (Q)*	ExpTime	OWL Lite
SHIF(D)	*SHI + roles funcionales (F)*	ExpTime	OWL Lite
SHOIN(D)	*SHIQ + restricciones numéricas no cualificadas (N) + dominios concretos (D)*	NExpTime	OWL DL
SHOIQ(D)	*SHIQ + nominales , objetos (O) + dominios concretos (D)*	NExpTime	DAML+OIL
SROIQ	*SHOIN + tipos de roles (R)*	N2ExpTime	OWL 2

Tabla 2 Familia DL

En la misma, se pueden observar órdenes de complejidad con un coste temporal exponencial respecto a la complejidad del problema. Entre los aumentos de expresividad está la inclusión de **roles transitivos**, ejemplo de expresión $R(a,b) \wedge R(b,c) \rightarrow R(a,c)$. **Roles inversos**, un rol es inverso a otro cuando al aplicar ambos elementos sobre el que se aplica da como resultado el mismo elemento, ejemplo de expresión $R-(R(x)) \equiv x$. **Jerarquía de roles**, cuando un rol es una especialización de otro, ejemplo de expresión $R \sqsubseteq P$. **Nominales**, estos permiten definir una clase enumerando uno a uno los elementos que la contienen. Restricciones numéricas, ejemplos de expresión $n \in \{1,2 \ldots\}, \leq n$. **Dominios concretos o tipos de datos**, pueden ser números, cadenas de texto, etc., que se estudian como un conjunto disjunto de Δ^I, llamado Δ^I_D, definiendo nuevos tipos de roles.

7.5 Lenguajes ontológicos

Una ontología define los términos usados para describir y representar un área de conocimiento. Normalmente las ontologías se expresan en un **lenguaje basado en lógica** (W3C Recommendation, 2004b) con el fin de detallarla, precisarla y hacerla consistente para obtener sólidas distinciones significativas entre las clases, propiedades y relaciones. Algunas herramientas gestoras de ontologías basadas en estos lenguajes realizan razonamientos automatizados, y proporcionan servicios avanzados a las aplicaciones inteligentes. En esta tesis doctoral, se investigaron los lenguajes lógicos

utilizados para describir ontologías. Estos lenguajes se pueden definir como lenguajes formales para representar información con una sintaxis que define el tipo de sentencia que puede darse en el lenguaje, y una semántica que defina el significado de dichas sentencias. Por lo tanto, estos lenguajes incluyen en su sintaxis definiciones lógicas que permiten utilizar razonadores lógicos para inferir resultados. En estos sistemas cuanto más restrictivas son las definiciones lógicas más sencillo es su manejo, pero más difícil es que el mundo real se adapte a dichos sistemas.

Una clasificación de los lenguajes ontológicos lógicos según la lógica utilizada de más restrictivo a menos inicia con los lenguajes basados en **lógicas probabilísticas** y **difusas** que suelen ser utilizados por sistemas de redes neuronales. Los siguientes son los lenguajes basados en las **lógicas no clásicas** que se caracterizan por no utilizar el concepto de verdadero o falso, sino el concepto de grado de verdad de una aserción. A continuación encontramos los lenguajes basados en las **lógicas de mayor orden** que se centran en relacionar conjuntos de elementos de discusión directamente, sin hacerlo a pares.

En el contexto de esta tesis doctoral, las siguientes agrupaciones de lenguajes ontológicos son las más importantes:

Los primeros lenguajes son los **basados en reglas**; a este respecto la regla se entiende como unidades de conocimiento contenidas que implican algún tipo de razonamiento. Entre los más importantes en este tipo está RuleML, que es un lenguaje basado en reglas orientado a su uso en la web, donde las reglas comprenden a la lógica clásica de forma que puede haber reglas en el lenguaje para definir más profundamente las ontologías. Entre los tipos de reglas están las de integridad, que sirven para definir las restricciones de integridad que se colocan en una base de datos relacional; otra es la regla de derivación que se basa en un conjunto de condiciones y consecuencias cuando se cumplen las condiciones anteriores. Las condiciones y las consecuencias se representan mediante una fórmula lógica. Una más es la regla de reacción, que es otra que tiene un disparador, condición de inicio y una acción a realizar. También tiene la regla de producción que expresa qué se ha de hacer cuando se da una condición. Finalmente, las reglas de transformación que explican cómo pasar de una instancia de un tipo a su equivalente de otro tipo. Sin embargo, RuleML aún se encuentra en estado de desarrollo, estudiando otro tipo de reglas necesarias.

A continuación tenemos los lenguajes ontológicos basados en **lógica de primer orden** de los que ya se ha hablado. El más conocido en esta agrupación es KIF/SKIF. El formato de intercambio de conocimiento (del inglés *Knowledge Interchange Format - KIF*) es un lenguaje que se desarrolló con la idea de que fuera un lenguaje de intercambio de conocimiento entre los distintos sistemas desarrollados para tratar bases de conocimiento. SKIF (P. Hayes & Menzel, 2001) se puede considerar una versión de KIF más adaptada a la web semántica, que permite el uso de URI y UNICODE con una sintaxis más sencilla, donde no distingue a priori entre instancias, clases y relaciones entre ellos, agrupando todo como términos. Ambos lenguajes se basan en lógica de primer orden, utilizando una sintaxis similar al lenguaje de programación LISP. Es un buen lenguaje para intercambio, sin embargo, hay pocos razonadores que trabajen con SKIF.

Finalmente en esta clasificación tenemos a los lenguajes ontológicos lógicos basados en la **lógica descriptiva** encarnados en los lenguajes OIL y DAML+OIL, sobre todo, en el lenguaje OWL. Este último es el lenguaje utilizado en esta tesis doctoral por lo que se describirá con más detalle a continuación.

7.5.1 OWL

OWL (del inglés *Web Ontology Language*) apareció como recomendación en febrero del 2004 como un lenguaje de marcado para publicar y compartir información usando ontologías en la web. El diseño de OWL está influenciado por más de diez años de investigación en DL. El origen de OWL se remonta a SHOE (del inglés *Simple HTML Ontology Extensions*), el cual fue el primer lenguaje para diseñar ontologías en la Web con las siguientes características apreciables:

- Una sintaxis construida sobre XML, lo que permitía embeberlo dentro de las páginas HTML.
- Los nombres son URIs, lo que le permite reutilizar la gran ventaja de los nombres únicos de los recursos, y evitar ambigüedades.
- Suposición de que las ontologías pueden reutilizarse, cambiar y ampliarse en el tiempo, por lo que ofrece posibilidades para importaciones, alias locales para los nombres largos, e información sobre versiones y compatibilidades hacia atrás.

Otro lenguaje que influyó de forma destacable en OWL fue DAML-ONT. La iniciativa americana (DARPA Agent Markup Language – DAML) DAML fue propuesta en 1999 con la idea de ser la base de la web semántica, la cual ya se basaba en RDF, pero se decidió enriquecer su sintaxis para aumentar su expresividad, surgiendo el lenguaje DAML-ONT. La alternativa europea fue la Capa de Inferencia de Ontología (del inglés *Ontology Inference Layer - OIL*), que buscaba objetos similares a DAML. OIL fue el primero en basarse en DL. Sin embargo, su compatibilidad con RDF no era completa. Los esfuerzos de DAML y OIL se unieron en una iniciativa entre Europa y Estados Unidos, llamada DAML+OIL. Este lenguaje ya se basó en DL y al mismo tiempo fue más compatible con RDFS. Así surgió OWL como una extensión de DAML+OIL, siendo la apuesta para la web semántica, dado que permite varios grados de complejidad, así como una sintaxis robusta y probada. Existen dos versiones: OWL 1 y OWL 2

OWL 1 (W3C Recommendation, 2004a) se presentó con tres niveles del lenguaje en función de su expresividad. Esto permite a sistemas más sencillos limitar la expresividad buscando una mayor efectividad de razonamiento.

- OWL Lite está diseñado para aquellos usuarios que necesitan representar las necesidades básicas de una ontología, principalmente una clasificación jerárquica y restricciones simples. Por ejemplo, a la vez que admite restricciones de cardinalidad, sólo permite establecer valores cardinales de 0 ó 1.
- OWL DL está diseñado para aquellos usuarios que quieren la máxima expresividad conservando completitud computacional (se garantiza que todas las conclusiones sean computables), y resolubles (todos los cálculos se resolverán en un tiempo finito). OWL DL incluye todas las construcciones del lenguaje de OWL, pero sólo pueden ser usados bajo restricciones controladas. Por ejemplo, mientras una clase puede ser una subclase de otras muchas clases, una clase no puede ser una instancia de otra.
- OWL Full está dirigido a usuarios que quieren máxima expresividad y libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser considerada simultáneamente como una colección de clases individuales y como una clase individual propiamente dicha. OWL Full permite una ontología para aumentar el significado del vocabulario preestablecido (RDF u OWL).

La sintaxis de OWL se divide en constructores, también denominados como descripciones de clases, y axiomas. Los constructores nos permitirán definir clases, tanto anónimas como no anónimas, y los axiomas son los elementos por los que podremos

realizar aserciones sobre las clases y sobre las instancias de las mismas, así como sobre las propiedades que relacionan a las instancias. En la definición formal de OWL se consideran los constructores de clase como un subconjunto de los axiomas de clase pero es claro que el significado semántico de constructores y axiomas es diferente.

OWL tiene tres grupos de propiedades (P), es decir, constructores de propiedades:

- Propiedades de instancia (P_C), donde el objeto de la propiedad es un individuo de Δ^I (owl:ObjectProperty). Es la única válida en OWL Lite.
- Propiedades de tipos de datos (P_D), donde el objeto de la propiedad es un individuo de $\Delta^I{}_D$ (owl:DatatypeProperty).
- Propiedades de anotación y de ontología, externas a la parte de lógica (owl:AnnotationProperty, owl:OntologyProperty) donde se almacenan breves explicaciones, temas de importación de ontologías, versiones, compatibilidad. Los principales tipos predefinidos son: owl:versionInfo, owl:imports (para definir la ontología, junto con otros sobre la compatibilidad), rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs: isDefinedBy

Las instancias tienen dos tipos de constructores:

a) De pertenencia a una clase (aserciones sobre conceptos). Por ejemplo:

```
<Profesor rdf:ID="Juan"/>
```

En este caso definimos la instancia *Juan* como un elemento de la clase Profesor.

b) Cuando le damos el valor a una propiedad (aserciones sobre roles). En el ejemplo podemos ver cómo definimos la instancia *Juan* indicando que la instancia *Juan* se relaciona con la instancia *ingeniería* por medio de la propiedad *imparte*.

```
<rdf: Description rdf:about="#Juan">
<imparte rdf:resource="#Ingenieria">
</rdf Description>
```

En la Tabla 3 y la Tabla 4 se ve la notación OWL para los constructores de clases y los axiomas.

En la Tabla 3 se presentan las distintas formas de definir una clase, indicando el nombre de la clase o como una función lógica entre otras clases. Las clases también se pueden definir con los cuantificadores de existencia $\exists$ y universal $\forall$. Así mismo, se introducen cuantificadores de cantidad ($\leq$nP, $\geq$nP). En las tablas C y D representan una clase, x_i una instancia, P una propiedad cualquiera, P_D una propiedad de tipo de datos y P_C un propiedad de objetos.

Constructor	Sintaxis DL OWL	OWL Lite	OWL DL	OWL Full
owl:Class	C, $[C^I \subseteq \Delta^I]$	X	X	X
owl:intersectionOf (3)	$C \sqcap D$	X	X	X
owl:unionOf	$C \sqcup D$		X	X
owl:complementOf	$\neg C$		X	X
owl:allValuesFrom	$\forall P.C$	P_C	P	P
owl:someValuesFrom	$\exists P.C$	P_C	P	P
owl:maxCardinality (1)	$\leq nP$	P_C , n={0,1}	P	P
owl:minCardinality	$\geq nP$	P_C , n={0,1}	P	P
owl:Thing	T, $[\Delta^I]$	X	X	X
owl:Nothig	$\perp$	X	X	X
owl:oneOf (2)	$\{x_1,\ldots,x_n\}$		X	X
owl:hasValue	P=x ó P ∋ x, x Datatype		X	X

Tabla 3 Constructores de clases en OWL

Axioma	Sintaxis DL	OWL Lite	OWL DL	OWL Full
rdfs:subClassOf (1)(3)	$C_1 \sqsubseteq C_2$	X	X	X
owl:equivalentClass (3)	$C_1 \equiv C_2$	X	X	X
owl:disjointWith	$C_1 \sqsubseteq \neg C_2$		X	X
owl:sameAs	$x_1 = x_2$	X	X	X
owl:differentFrom (2)	$x_1 \neq x_2$	X	X	X
rdf:subPropertyOf	$P_1 \sqsubseteq P_2$	P_C	P	P
owl:equivalentProperty	$P_1 \equiv P_2$	P_C	P	P
owl:inverseOf (4)	$P_1 \equiv [P_2]^-$	P_C	P	P
owl:transitiveProperty	$P[P] \sqsubseteq P$	P_C	P_C	P_C
owl:functionalProperty (4)(5)	$T \sqsubseteq \leq 1P$ (5)	P_C	P	P
owl:inverseFunctionalProperty (4)	$T \sqsubseteq \leq 1P^-$	P_C	P_C	P_C
owl:SymmetricProperty	$P \equiv [P]^-$	P_C	P_C	P_C

Tabla 4 Axiomas en OWL

Se pueden ver en la Tabla 4 los distintos axiomas del lenguaje ontológico junto con las marcas XML utilizadas para cada uno. Así mismo, están indicadas las diferencias existentes entre los niveles de OWL.

En 2009 el grupo de trabajo de W3C publicó **OWL 2** (W3C Recommendation, 2012) como una ampliación y revisión de OWL 1. OWL 2 tiene una estructura general muy similar a OWL 1, casi todos los bloques de construcción de OWL 2 estaban presentes en OWL 1, aunque posiblemente con nombres diferentes.

- En OWL 1, la sintaxis abstracta desempeñó el papel de la estructura y la sintaxis funcional en OWL2. La sintaxis funcional OWL 2 difiere en la forma de la sintaxis abstracta en OWL 1, pero su papel dentro de la estructura general de OWL es idéntico. La sintaxis funcional de OWL 2 es mucho más cercana a la representación gráfica RDF y puede capturar más gráficos RDF, que también tiene una correspondencia directa con la especificación estructural en UML.
- Como OWL 1, OWL 2 especifica un mapa detallado de las estructuras ontológicas (representada mediante la sintaxis abstracta / funcionales) para grafos RDF. Sin embargo, OWL 2 también se beneficia de un mapeo específico explícito de grafos RDF a las estructuras ontológicas.
- Las dos semánticas (*Direct* y *RDF-Based*) de OWL 2 tienen sus contrapartes directas en OWL 1, bajo los nombres *Direct Model – Theoretic Semantics* y *RDF-Compatible model Theoretic Semantics* respectivamente.
- Una presentación de sintaxis XML también estaba disponible para OWL 1 (aunque no como una recomendación). Por otra parte, el *Manchester syntax* no existía para OWL 1.
- OWL 1 define un sub-lenguaje (OWL Lite), donde OWL 2 define tres perfiles (EL, QL y RL). OWL Lite no ha sido re-especificado para OWL 2, pero debido a la compatibilidad con versiones anteriores, OWL Lite termina como un sub-lenguaje de OWL 2.

El papel central de RDF / XML como sintaxis de intercambio sólo se requiere para herramientas OWL 2 y las relaciones entre las semánticas Direct y RDF-Based no han cambiado. Más importante aún, la compatibilidad hacia atrás con OWL 1 es completa, tanto sintáctica y semánticamente.

- Al igual que en OWL 1, OWL 2 puede manejar todos los grafos RDF. El vocabulario al que se le da un significado especial en OWL 2 incluye el vocabulario especial de OWL 1. Sin embargo, el uso de *owl: RangoDatos* , mientras que todavía es posible, ahora es obsoleto, *rdfs : tipo* de datos se debe utilizar en su lugar.

- La semántica directa para OWL 2 es casi totalmente compatible con la semántica directa para OWL 1. La única diferencia es que las anotaciones semánticas son libres en la semántica directa para OWL 2. Es muy poco probable, sin embargo, que los usuarios notaran una diferencia: en primer lugar, la semántica dada a las anotaciones en semántica directa de OWL 1 era muy débil y es poco probable que conduzca a ninguna vinculación significativa, y en segundo lugar, las herramientas OWL 1 utilizando la semántica directa, normalmente tratan anotaciones como si fueran semántica libre .
- La semántica RDF-based de OWL2 es completamente compatible con la semántica de RDF-based de OWL 1. Algunos de los detalles de esta semántica han cambiado, pero el conjunto de inferencias es el mismo.
- El tratamiento de la importación de documentos RDF ha cambiado ligeramente en OWL 2 si los grafos RDF deben ser conforme a documentos ontológicos OWL 2 DL. En OWL 1, la importación ocurrió primero, por lo que todo el grafo resultante de la fusión se considera como una unidad. En OWL 2, los documentos individuales se consideran por separado en la mayoría de los casos. Esto significa que los documentos OWL 1 DL RDF que no tienen un encabezado ontológico bien determinado pueden necesitar ser ligeramente modificados para ser conforme a los documentos de ontología OWL 2 DL.

7.6 Razonadores

Un razonador es también conocido como motor de razonamiento o motor de reglas. Es una pieza de software capaz de funcionar como un motor de inferencia, deduciendo conclusiones lógicas de un conjunto de axiomas. Los razonadores proporcionan un conjunto de mecanismos para trabajar con reglas de inferencia que son definidas con lenguajes ontológicos lógicos. Entre los razonadores más utilizados para OWL están los siguientes.

7.6.1 RacerPro

En el área de los razonadores de software comercial el más conocido es RACER (del inglés *Renamed ABox and Concept Expression Reasoner*). RacerPro (Haarslev & Möller, 2001) fue uno de los primeros razonadores para ABoxes que apareció en el mercado desde el año 2002. Inicialmente se desarrolló por las universidades alemanas de Hamburgo. Este razonador se puede utilizar para la gestión de ontologías de la web semántica basadas en OWL. Sin embargo, también puede ser usado como un repositorio de información de web semántica con un motor de recuperación optimizado.

El razonador RacerPro es un razonador optimizado para *SHIQ(D)*, por lo tanto, soporta OWL DL excepto para los nominales y para tipos de datos no estándar (fuera de xsd:X). Se basa en el algoritmo *tableaux* con optimizaciones y cacheos de inferencias obtenidas. Actualmente es capaz de manejar varios TBoxes y varios ABoxes, además de las tareas de razonamiento básicos ofrece ABox consultas sobre la base de las optimizaciones nRQL. Se implementa en el lenguaje de programación Common Lisp.

Racer es un razonador con más posibilidades que un simple razonador OWL, ya que incluye un cliente (RacerPorter), para hacer consultas en OWL-QL con su propio lenguaje de consultas (nRQL, new Racerpro Query Language) e incluye una herramienta de cliente exclusiva (RICE, Racer Interactive Client Environment), y un interfaz de programación de aplicación API en Java para poder interactuar con él directamente (JRacer).

7.6.2 KAON2

Entre los razonadores conocidos de uso libre pero de código cerrado encontramos KAON2[26]. Es un razonador semántico, que permite la ejecución de inferencias con ontologías representadas en OWL DL, SWRL y F-logic; a diferencia de otros razonadores que trabajan con lógica descriptiva no implementa cálculo *tableaux* como base para su proceso de razonamiento; sino que utiliza una serie de algoritmos que reducen una base de conocimiento $\mathcal{SHIQ}(\mathcal{D})$ a un programa datalog disjunto (Hustadt, 2004) lo que aumenta de forma considerable el rendimiento del razonador a la hora de ejecutar los procesos de inferencia.

KAON2 ofrece una librería de programación para la manipulación de ontologías en OWL DL y F-logic. Así mismo es capaz de resolver consultas conjuntivas que están expresadas en el lenguaje SPARQL. Posee un API que permite el acceso a la funcionalidad de Kaon2 desde otras herramientas, tales como Protégé.

La API de ontologías facilita servicios de manipulación de una ontología, como la adición y recuperación de axiomas de la ontología; es completamente compatible con OWL y SWRL en el nivel sintáctico. Las ontologías pueden ser guardadas en archivos, utilizando sintaxis OWL, RDF, XML. Alternativamente, las afirmaciones ABox se pueden almacenar en una base de datos relacional (RDBMS) asignando entidades a las tablas en una base de datos de la ontología, kaon2 consulta la base de datos sobre la marcha durante el razonamiento.

La arquitectura flexible y escalable de KAON2 está organizada en tres estratos: los adaptadores para almacenamiento de la ontología, el software personalizado para los servicios ofrecidos por la suite de herramientas, y las aplicaciones del cliente que acceden al software personalizado de la ontología y ofrecen aplicaciones a los usuarios finales. Toda la suite de herramientas de Kaon2 es implementada en Java.

El modelo de conocimiento trabaja con la suite de la herramienta Kaon2 con un lenguaje propio basado en una extensión de RDF (S). Con Kaon2 se modelan ontologías con los conceptos, las propiedades (para expresar relaciones y atributos), las instancias de conceptos y de propiedades. Los conceptos son organizados en taxonomías de concepto (con la relación del subclassOf) y las propiedades son organizadas en jerarquías de la propiedad (con la relación PropertyOf).

7.6.3 PELLET

Los razonadores más utilizados son los del área de software libre y código abierto. Entre estos tenemos el razonador PELLET (Parsia & Sirin, 2004) que fue desarrollado por la Universidad de Maryland.

Este razonador es capaz de gestionar ontologías OWL 2. Pellet incorpora el algoritmo *tableaux* desarrollado para lógica descriptiva que incluso soporta el razonamiento sobre las clases enumeradas y resuelve consultas conjuntivas. Pellet es capaz de analizar y reparar ontologías incorporando un conjunto de heurísticas para detectar y corregir automáticamente las ontologías para asegurarse que son OWL DL.

En Pellet es posible utilizar los servicios de razonamiento desde la línea de comandos, a través de las bibliotecas de programación OWL-API (Horridge & Bechhofer, 2009) y

[26] http://kaon2.semanticweb.org/

Jena[27], usarlo integrado como plug-in en el editor de ontologías Protégé, o usarlo como servidor DIG (Bechhofer, Möller, & Crowther, 2003). Pellet contiene múltiples optimizaciones para responder consultas conjuntivas. Además puede responder consultas SPARQL de tipo SELECT, CONSTRUCT y ASK por sí mismo, y consultas de tipo DESCRIBE o que usen OPTIONAL y FILTER si se usa como razonador integrado en la arquitectura de Jena. Este razonador permite razonar con todos los datatypes predefinidos en XML Schema (numéricos, cadenas, fecha y hora, etc.) y con algunos tipos definidos por el usuario como extensiones de los anteriores. Así mismo, permite gestionar ontologías extendidas mediante reglas representadas en el lenguaje SWRL.

Pellet incorpora heurísticas que permiten detectar ontologías marcadas como OWL Full que pueden ser expresadas usando OWL-DL. En algunos casos permite corregir estas ontologías automáticamente. En la detección de conceptos que no se pueden satisfacer, un servicio de razonamiento básico Pellet ayuda al usuario en el diagnóstico de las causas de la inconsistencia así como su resolución. Es capaz de calcular los axiomas y hechos de la ontología responsable de la inconsistencia, y cómo las dependencias entre conceptos hacen que el error se propague. Este razonador gestiona eficientemente bases de conocimiento dinámicas mediante el uso de algoritmos incrementales de clasificación y chequeo de consistencia.

7.6.4 Hermit

El razonador HermiT[28] es uno de los proyectos de investigación del laboratorio de computación de la Universidad de Oxford. Hermit es código abierto y liberado bajo LGPL[29]. En general es un razonador para ontologías codificadas con OWL. Aunque inicialmente solo era empleado en ontologías médicas, puede ser usado en cualquier tipo de ontología.

La aportación más importante de Hermit es la implementación del novedoso algoritmo *hypertableaux* (Motik, Shearer, & Horrocks, 2007) a fin de proveer razonamiento más eficiente. El aspecto principal de este algoritmo es que es más rápido que los algoritmos *tableaux* existentes.

7.6.5 FACT ++

Creado por la Universidad de Manchester, el FaCT++ (Tsarkov & Horrocks, 2006) es la continuación del razonador de código abierto para TBoxes FaCT (del inglés *Fast Classification of Terminologies*) desarrollada en Lisp. La versión actual FaCT++ está desarrollada en C++ para crear una herramienta de desarrollo del software eficiente, para el uso correcto de sus algoritmos.

La nueva versión permite razonar con ABoxes y en las últimas versiones ha mejorado bastante su interfaz con DIG. FaCT++ está optimizado para la lógica de OWL DL pero sobretodo es conocido por su clasificación acelerada de terminología. Este razonador soporta $\mathcal{SHIF}$ y $\mathcal{SHIQ}$ en su proceso de razonamiento. Su lógica $\mathcal{SHIQ}$ es

---

[27] http://jena.apache.org/

[28] http://www.hermit-reasoner.com/

[29] GNU Lesser General Public License.

expresiva para ser usado como razonador de la lógica DL para utilizar esquemas de base de datos.

Está diseñado como una plataforma para experimentar con nuevos algoritmos del *tableaux,* que actualmente se ha convertido en un estándar para los sistemas DL. Fact++ incluye una lista de tareas pendientes que es más adecuada para los algoritmos *tableaux*, donde cuenta con un rango más amplio de optimizaciones heurísticas. La simplificación y normalización léxica de Fact++es una optimización estándar que reescribe principalmente el diseño para detectar la inconsistencias. La idea básica de estos conceptos es transformarlos en una forma normal simplificada.

8 Ingeniería del conocimiento en etapas tempranas

En las secciones 4 y 5 de este capítulo se han descrito diferentes trabajos de la aplicación del paradigma OA a las etapas tempranas. Ahora bien, al seleccionar la ingeniería de conocimiento como el medio para resolver los problemas semánticos de las propuesta IROA y ASOA, parece pertinente analizar la aplicación de las técnicas de **ingeniería del conocimiento en las etapas tempranas** en el desarrollo de un producto software.

Los trabajos en este área argumentan la importancia de utilizar ontologías para facilitar la elicitación, representación y extracción del conocimiento de una ER. Así mismo, enfatizan la importancia de tomar en cuenta la organización de la semántica durante las etapas tempranas.

Son bien conocidas las consecuencias relacionadas con el coste y la funcionalidad de una inapropiada gestión de requisitos en el desarrollo del sistema (Glass, 2003). En este contexto, aparece la representación de conocimiento a través de ontologías como un factor importante para lograr una eficiente gestión de requisitos basada en el razonamiento.

8.1 El uso de ontología de dominio como conocimiento del dominio para la elicitación de requisitos

La propuesta **"Using domain ontology as domain knowledge for requirements elicitation"** (Kaiya & Saeki, 2006) se enfoca en el conocimiento del dominio como un factor crucial para el éxito de la elicitación de requisitos de alta calidad. Se basa en el hecho de que típicamente el conocimiento del dominio sólo lo conocen los expertos de dicho dominio y no los analistas de requisitos.

Los autores proponen un método donde una ontología de dominio es utilizada como una representación del conocimiento del dominio. Este método establece una correlación entre la ontología de dominio que representa una base semántica y una descripción de requisitos. A este respecto, el método permite realizar un análisis de las descripciones de requisitos con el fin de detectar propiedades como la completitud e inconsistencia.

Una de las ideas en esta propuesta es que sin tener una ontología completa para el dominio del problema, es posible obtener parte a través de técnicas de ingeniería ontológica. La posibilidad está en la reutilización de ontologías del mismo dominio del problema de una amplia variedad de dominios construidas por diferentes investigadores y desarrolladores de la comunidad de ingeniería del conocimiento. Ontologías representadas con el lenguaje OWL con una estandarización para ser intercambiado por muchas personas. Aunada a esta idea está la relativa a la utilización de fuentes documentos de especificaciones de requisitos en lenguaje natural existentes del dominio del problema para extraer conceptos y sus relaciones mediante la aplicación de técnicas

de minería de texto. En el contexto de estas ideas plantean un ciclo de elicitación de requisitos y conocimiento de forma incremental para crear una ontología de dominio y descripciones de requisitos del dominio del problema de alta calidad.

Ahora bien, la propuesta de este artículo se centra en un método para elicitar y definir requisitos de forma sistemática mediante el uso de ontologías.

Este método, parte de una lista de requisitos iniciales a los cuales se les aplican técnicas de descomposición léxica para su descomposición en varios términos para su asignación a elementos de una ontología. Para este proceso de asignación propusieron un metamodelo de ontología organizado jerárquicamente con diferentes tipos de conceptos y relaciones. La estructura de conceptos y las relaciones del metamodelo de ontología propuesto pretenden guiar el modelado de la semántica entre elementos de los requisitos. Entre los conceptos propuestos algunos se utilizan para representar los elementos de requisitos funcionales y otros los no funcionales.

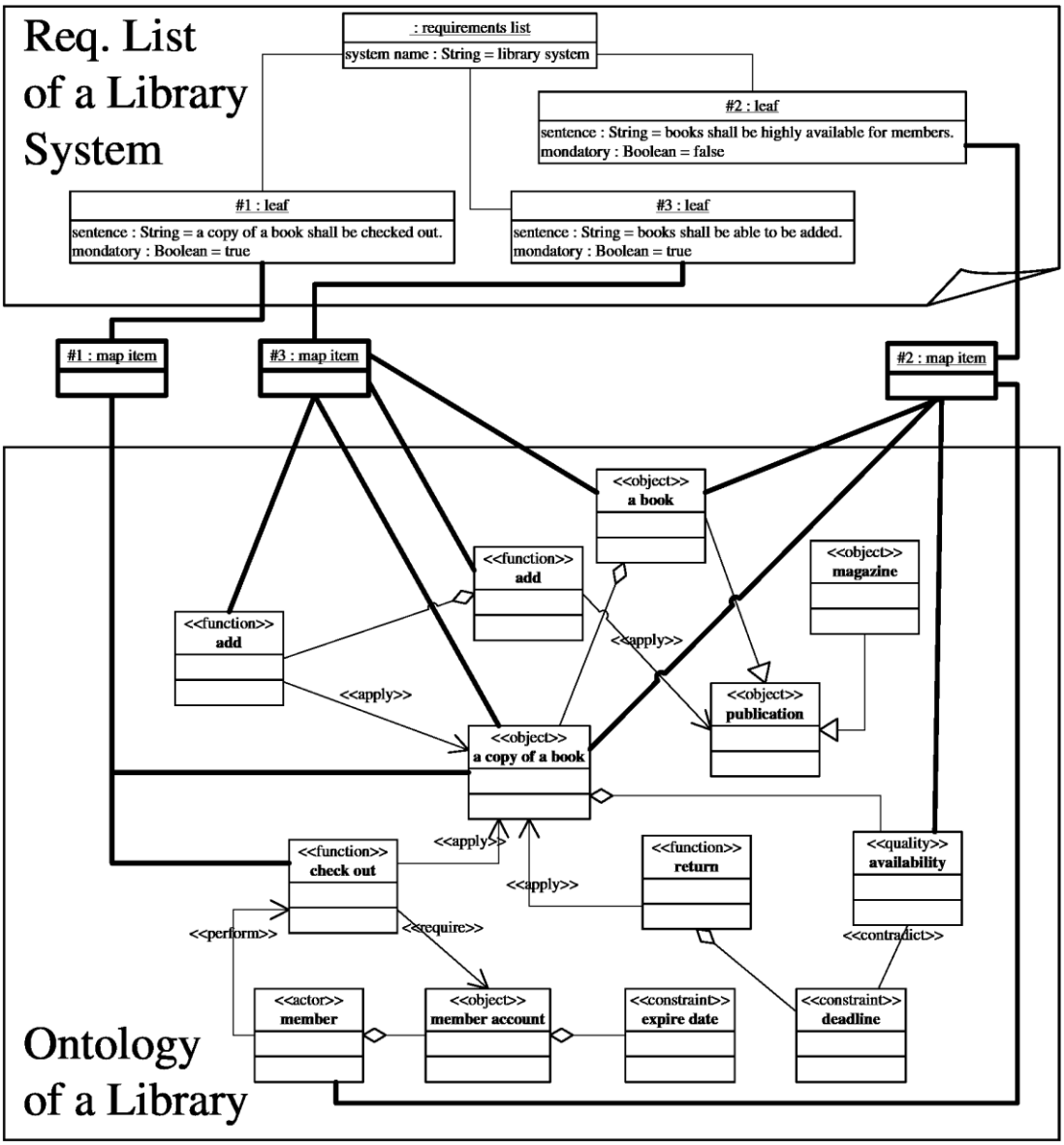


Figura 18 Ejemplo mapeo semántico

En base a este metamodelo de ontología un analista asigna los elementos de requisitos para crear una instancia de esta ontología mapeada contra la lista de requisitos. Parte de un ejemplo del resultado de este mapeado se presenta en la Figura 18.

En base a este mapeo es posible aplicar un procesamiento semántico de *peso ligero* para analizar las descripciones de requisitos escritas en lenguaje natural mediante el uso de una ontología de dominio. En esta propuesta, una ontología de dominio desempeña un papel en el dominio semántico que da significado a las declaraciones de los requisitos. Así, mediante el uso de reglas de inferencia en la ontología es posible la medición de las descripciones de los requisitos midiendo los factores de calidad: exactitud, completitud, consistencia y ambigüedad.

8.2 Revisando la ontología núcleo y el problema en ingeniería de requisitos

La segunda propuesta **"Revisiting the Core Ontology and Problem in Requirements Engineering"** (Jureta et al., 2008) está basada en un planteamiento anterior (Zave & Jackson, 1997) en donde se estableció una ontología núcleo que describe normas mínimas para la información que debe ser desarrollada en la ingeniería de requisitos. En la ontología núcleo fundamental que presentaron Zava y Jackson se sugiere que el problema de los requisitos equivale a encontrar la especificación **S** que para supuestos de un dominio dado **K** satisface los requisitos dados **R**. Si los tres están escritos en una lógica matemática, el problema se resuelve una vez que el ingeniero encuentra **S** tales que **K**, **S** $\vdash$ **R**.

Ahora bien, en esta nueva propuesta se expone que esta anterior caracterización es deficiente. Declarando en primer lugar, que no permite el cumplimiento parcial de requisitos. En segundo lugar, que la caracterización no deja lugar a que una especificación sea mejor que otra, dado un dominio de suposiciones **K** y requisitos **R**. En tercer lugar, que las nociones tan importantes como las de requerimiento no funcional y la preferencia se dejan fuera del marco, con el que no hay sustitutos para compensar esta omisión. Para solucionar estos problemas, esta nueva propuesta plantea una ontología núcleo extendida para la ingeniería de requisitos con una nueva formulación de los problemas de los requisitos. En esta ontología núcleo se añaden incumbencias básicas relacionadas con la comunicación de los interesados en el sistema y el ingeniero.

Inician recomendando que frente a un acto de hablar, el ingeniero primero debe distinguir su tipo, entonces separar su modalidad de su contenido. En esta línea, se plantean cuatro modalidades que corresponden a los estados mentales que subyacen a los actos de hablar: *creencias* (**B**), *deseo* (**D**), *intención* (**I**), y *actitud* (**A**). Con este escenario, exponen que mientras los *deseos* conducen a requisitos y *creencias* a los supuestos de dominio, *intenciones* conducen a especificaciones. Por lo tanto, se observa que Zave y Jackson en su suposición de dominio, requerimientos, y conceptos de especificaciones cubren las actitudes proposicionales (es decir, **B**, **D** y **I**) ya que suelen ser conceptualizados en la filosofía. En cuanto a la noción de *actitud* en psicología (que es lo que transmiten los actos de habla expresivos), se identifica más estrechamente con el afecto, es decir, una reacción evaluativa general. Así, la ontología fundamental de Zave y Jackson es incompleta al no considerar *actitudes*, que tienen importantes consecuencias en la formulación de los problemas de los requisitos.

En este sentido, el enfoque de la nueva propuesta consiste primero en la clasificación del contenido comunicado por los interesados mediante la asociación de modalidades a la misma, por lo que la elección de la modalidad depende del tipo de acto de habla utilizado en la comunicación. Las reglas para la asociación de las modalidades para el contenido son:

- El contenido ø de un acto de habla *asertiva* o *declarativo* o *declarativa representante* que los interesados comunican al ingeniero de software es una *creencia*, **B**ø.
- El contenido ø de un acto de habla *directiva* que los interesados comunican al ingeniero de software es un *deseo*, **D**ø.
- El contenido ø de un acto de habla de *autorización* que las partes interesadas comunican al ingeniero de software es una *intención*, **I** ø.
- El contenido ø de un acto de habla *expresiva* que los interesados comunican al ingeniero de software es una *actitud*, **A**ø.

Con estas reglas los requisitos de Zave y Jackson corresponden a **D**ø, mientras que **I**ø y **B**ø corresponden a una especificación (es decir, un fragmento de la misma) y un supuesto dominio. No hay divergencia a este respecto de las intuiciones aceptadas en IR: los requisitos abarcan lo que se desea, mientras que los supuestos de dominio refieren lo que es verdad. Las intenciones dan fragmentos de especificación para determinar lo que se hará para satisfacer los requisitos.

Con este contexto, se expone cómo en esta propuesta el ámbito de la ontología núcleo es determinado por los tipos de expresiones que los interesados en el sistema comunican al ingeniero de software, para que el ingeniero asocie una modalidad para el contenido de un acto de habla dado. Finalmente procede a determinar si el resultado obtenido es una instancia de un concepto o de una relación de esta ontología núcleo.

Ahora bien, los conceptos de la ontología núcleo son **objetivo**, **meta**, **restricción de calidad**, **plan**, y **suposición de dominio**; y las relaciones son **opcionalidad basada en la actitud y la preferencia**, **aproximación justificada**, y **consecuencia no monótona**.

El contenido creído en **B**ø comunicado a través de afirmaciones, declaraciones o declaraciones representativas es una *suposición de domino*, denominado generalmente como **k**.

El contenido deseado en **D**ø que se comunica a través de un acto de habla directiva es una *restricción de calidad* (denominado como **q**) si describe calidades y valores de restricciones de calidad; es un *objetivo* (denominado como **g**) si no describe calidades y valores de restricción de calidad; y es una *meta* (denominado como **ˆq**) si describe las calidades o restringe los valores de calidad, en donde las calidades descritas deben tener un espacio de calidad con una estructura subjetiva y/o mal definida.

El contenido intentado en **I**ø que se comunica a través de un acto de habla con la intención de autorizar es un *plan,* denominado **p**.

Existe una relación de *aproximación justificada,* denominada **jAprprox (ˆq, q)**, si y sólo si existe una justificación para una *declaración* **"q aproxima ˆq"** y hay suficiente correlación entre los valores en el espacio de calidad de **q** y el espacio de calidad de **ˆq**.

El contenido de actitud comunicado a través de un acto de habla expresiva es una relación de *actitud* (denominada **a**) si y sólo si se evalúa en términos de favorecer o desfavorecer uno o más elementos constituidos de **K** (incluye las suposiciones de dominio **k**), **P** (incluye todos los planes correspondientes **p**), **G** (incorpora los objetivos **g**), **Q** (incluye una o más restricciones de calidad **q**) o **ˆQ** (incluye las metas **ˆq**).

Se puede observar que las definiciones propuestas para los conceptos de *objetivo*, *meta* y *restricción de calidad* pretenden cubrir la taxonomía clave de los requisitos funcionales y no funcionales.

Los conceptos de esta ontología son plasmados con la intención de cubrir todas las expresiones, buscando así que la ingeniería del software pueda clasificar todo el

contenido de las comunicaciones de los interesados en el sistema como ejemplos de conceptos o relaciones de la ontología propuesta.

La definición que resume la resolución de los problemas de los requisitos basados en la ontología núcleo propuestas describe, que a partir de supuestos de dominio obligatorios y optativos (K_C y K_O), objetivos (G_C y G_O), restricciones de calidad (Q_C y Q_O), metas ($\hat{}Q_C$ y $\hat{}Q_O$), planes (P_C y P_O) y actitudes (A_C y A_O) comunicados por las partes interesadas, el problema de los requisitos equivale a la búsqueda de la especificación (es decir, planes) (P*) tal que:

1. $K^*, P^* \mid\sim G^*, Q^*, A$.

2. Preferencias en A indican que no hay otra combinación (Ki_C; Gi_C; Qi_C) que se prefiere a (K^*_C; G^*_C; Q^*_C).

3. No hay Pi diferente de P^* tal que K^*_C; $P^i \mid\sim G^*_C$, Q^*_C, A_C verifica y

> (a) K^*_C, K^i_O, $P^i \mid\sim G^*_C$, G^i_O, Q^*_C; Q^i_O, A_C, A_O verifica, donde K^i_O es un subconjunto más grande de K_O de K^*_O,
>
> y/o G^i_O es un subconjunto más grande de G_O que G^*o,
>
> y/o Q^i S es un subconjunto más grande de Q_O de Q^*_O,
>
> y/o
>
> (b) Las preferencias en A indican que K^i_O es preferible a K^*_O, y/o G^i_O es preferible a G^*_O y/o Q^i_O es preferible a Q^*_O.
>
> 4. Para cada meta ^q en ^Q, hay una restricción de calidad q que se encuentra en la relación aproximación justificada con la meta, es decir, jApprox (q; q).
>
> 5. Para cada orden de preferencia en A sobre meta, hay un orden de preferencia que mantiene ese mismo orden sobre las restricciones de calidad que se interponen en la relación de aproximación justificada para las metas dadas.

8.3 Modelando requisitos reutilizables de seguridad basados en un marco ontológico

La tercera propuesta **"Modelling Reusable Security Requirements based on an Ontology Framework"** (Velasco et al., 2009) plantea el objetivo de reutilizar requisitos de seguridad basados en la tecnología de la Web semántica.

Se inicia definiendo que los requisitos de seguridad incluyen los tipos y niveles de protección necesaria para el equipamiento, los datos, la información, las aplicaciones y las facilidades para encontrar una política de seguridad. A continuación, basándose en el estándar para la gestión de la seguridad de la información ISO 27002 (2005) especifican que los requisitos de seguridad son identificados por análisis de riesgo. Con este escenario, apoyan que los grandes beneficios de la reutilización son obtenidos cuando es considerado durante las fases tempranas del ciclo de vida de un desarrollo software. En consecuencia, para una eficiente gestión de requisitos la reutilización es un factor importante. A este contexto se añade que la comunidad de la Web semántica tiene experiencia en el diseño y desarrollo de componente reutilizables a través de utilización de tecnología ontológica.

Los autores presentan un marco de trabajo basado en ontologías para el modelado de requisitos de seguridad en el análisis de riesgo. Este marco de trabajo se basa en dos ontologías: la ontología de análisis de riesgos y la ontología de requisitos.

Iniciaron con la conceptualización del dominio de análisis de riesgos basándose en métodos y estándares oficiales de seguridad de análisis de riesgos. En esta primera ontología se identifican cinco tipos de elementos de riesgo de acuerdo con MAGERIT V.2 (López Crespo, Amutio Gómez, Candau, & Mañas, 2006):

- **Activos** (del inglés *asset*): todo lo que aporta valor a la organización. Se clasifica dentro de una jerarquía de clases de activos. MAGERIT distingue nueve tipos disjuntos de activos: *Servicio*, *Medios, Comunicación*, *Software, Hardware*, *Información de datos*, *Equipos auxiliares*, *instalaciones* y *el personal*. Estos activos se agruparon en cuatro clasificaciones disjuntas: *Funciones de la Organización*, *Sistema de Información*, *Información* y de *activos de entorno*.
- **Amenaza** (del inglés *Threat*): las posibles amenazas asociadas a los activos en un sistema de información. Se han identificado cuatro tipos principales disjuntos de amenazas: los *desastres naturales*, *Origen Industrial*, *errores y fracasos accidentales* (fallos involuntarios causados por las personas) y los *ataques deliberados* (fallos deliberados causados por personas).
- **Protección** (del inglés *Safeguard*): las salvaguardias que permiten las amenazas que deberá enfrentar. Por ejemplo, el control de acceso, registro de actuaciones o copias de respaldo.
- **La dimensión de valoración** (del inglés *valuation dimension*): las características o atributos que lo hacen un activo valioso. Esta es la medida de la pérdida causada por daños en un activo en una cierta dimensión: la disponibilidad, la integridad, la confidencialidad, la autenticidad y la rendición de cuentas.
- **Los criterios de valoración** (del inglés *valuation criteria*): Los criterios que hicieron un activo valioso para una organización, en otras palabras, lo interesante que es el activo para el sistema.

En la ontología de análisis de riesgos además de estos conceptos añadieron relaciones, restricciones, axiomas y reglas. La semántica de las relaciones genéricas afecta directamente a cada concepto participantes mediante la adición de las correspondientes restricciones de alcance y dominio en OWL.

En el desarrollo de la segunda ontología correspondiente al modelado de requisitos fue tomada en cuenta la experiencia del método para IR basado en la reutilización denominado SIREN (Toval, Olmos, & Piattini, 2002). SIREN se basa en un repositorio de requisitos reutilizables organizado por catálogos basados en normas de IR (IEEE-Computer-Society, 1998b)(IEEE-Computer-Society, 1998a).

Los principales elementos meta-información o atributos identificados para cada requisito se describen a continuación:

- Los requisitos se caracterizan por un **identificador único** y una **descripción textual**. Ambos se han definido en OWL como *propiedades de tipo de datos* (del inglés *datatype properties*).
- **Prioridad** (del inglés *priority*): este valor debe ser establecido por el analista y muestra el orden de desarrollo de la *propiedad de tipo de dato* {"alto", "medio", "bajo"}.
- **Justificación** (del inglés *rationale*): por qué se consideran los requisitos. *Propiedad de tipo de datos*.

- **Estado** (del inglés *state*): nueve estados son posibles para un requisito. *Propiedad tipo de datos* {a determinar, decidido, pendiente de revisión, ser descartado, aprobado, modelado en análisis, modelado en diseño, implementación o verificar}
- **Trazabilidad** (del inglés *traceability*): en algunos documentos de requisitos, estos requisitos pueden aparecer relacionados entre sí por medio de trazas. El modelo de trazabilidad incluye tres tipos de relaciones: las relaciones inclusivas, las relaciones entre padres e hijos y relaciones exclusivas que han sido modeladas utilizando las siguientes *propiedades de objetos*:
  - **Incluido**: las relaciones de trazabilidad *incluido* se definen entre dos requisitos A y B, lo que significa que para satisfacer A, B también tiene que ser satisfecho y, por lo tanto, la reutilización de una implicará la reutilización de B. Esto es una relación de dependencia y satisface un conjunto de propiedades (reflexividad, asimetría y transitividad).
  - **Padres e Hijos**: se trata de las relaciones a través de las cuales los requisitos de los hijos refinan el significado de los padres.
  - **Exclusivo**: las relaciones exclusivas de trazabilidad significan que los requisitos implícitos son mutuamente excluyentes. Esta relación está directamente relacionada con la reutilización ya que indica los requisitos que no se pueden seleccionar desde el repositorio para el mismo proyecto.
- **Fuente** (del inglés *source*): la fuente del requisito. Aunque las necesidades de los clientes son la fuente principal, los demás requisitos se podrían derivar de la solución técnica, la legislación vigente o las normas (es decir, las normas de seguridad). *Propiedad de tipo de datos*.
- **Método de verificación** (del inglés *verification method*): éste es el método utilizado para verificar que el requisito se cumple en el producto final. *Propiedad Tipo de datos* { "inspección", "analizar", "demostración", "test"}
- **Sección** (del inglés *section*): la sección del documento en la que el requisito se encuentra, si el requisito se almacena en algún documento. *Propiedad de tipo de datos*.
- **ValidatedBy**: los criterios de validación necesarios para probar los requisitos. *Propiedad de tipo de datos*.
- **ProposedBy**: las personas involucradas en el desarrollo del proyecto que solicitan que el requisito sea incluido. *Propiedad de tipo de datos*.
- **Responsable de**: la parte interesada que debe satisfacer o realizar el requerimiento. *Propiedad de tipo de datos*.
- **El valor parametrizado** (del inglés *parameterized value*): requisitos parametrizados que contienen algunas partes que tienen que ser adaptadas a cada aplicación o sistema y que tienen que ser una instancia en tiempo de reutilización. Este tipo de requisito fomenta la reutilización al factorizar los detalles específicos del sistema como parámetros de la exigencia, lo que permite a los ingenieros especificar los requisitos con un mayor grado de flexibilidad. *Propiedad de tipo de datos*.

Ahora bien, a través de una evaluación de riesgos, se identifican las amenazas a los activos cumpliendo con la política de seguridad. Los requisitos modelados pueden ser del software o del sistema, que apoyan a algunos problemas de seguridad en el sistema. El resultado es una **ontología de requisitos** en la que se clasifican los requisitos de seguridad. A continuación la ontología es extendida con los conceptos de la **ontología de dominio de análisis de riesgos**. Obteniendo un modelado de los requisitos de seguridad con la metainformación relacionada con el análisis de riesgos y las restricciones, axiomas y reglas, propiedades que ayudan a mantener la coherencia.

8.4 El uso de seguridad y ontologías de dominio para el análisis de requisitos de seguridad

La última propuesta **"Using Security and Domain ontologies for Security Requirements Analysis"** (Souag et al., 2013) presenta un método que explora el uso combinado de una ontología de seguridad y una de dominio existentes para desarrollar una especificación de requisitos de seguridad.

En una anterior investigación (Souag, Salinesi, & Comyn-Wattiau, 2012) revisaron y clasificaron un grupo de ontologías de seguridad existentes. Este estudio fue la base para identificar los conceptos comunes de alto nivel con los que se desarrolló una visión genérica de las ontologías de seguridad revisadas. Esta ontología de alto nivel se muestra en la Figura 19.

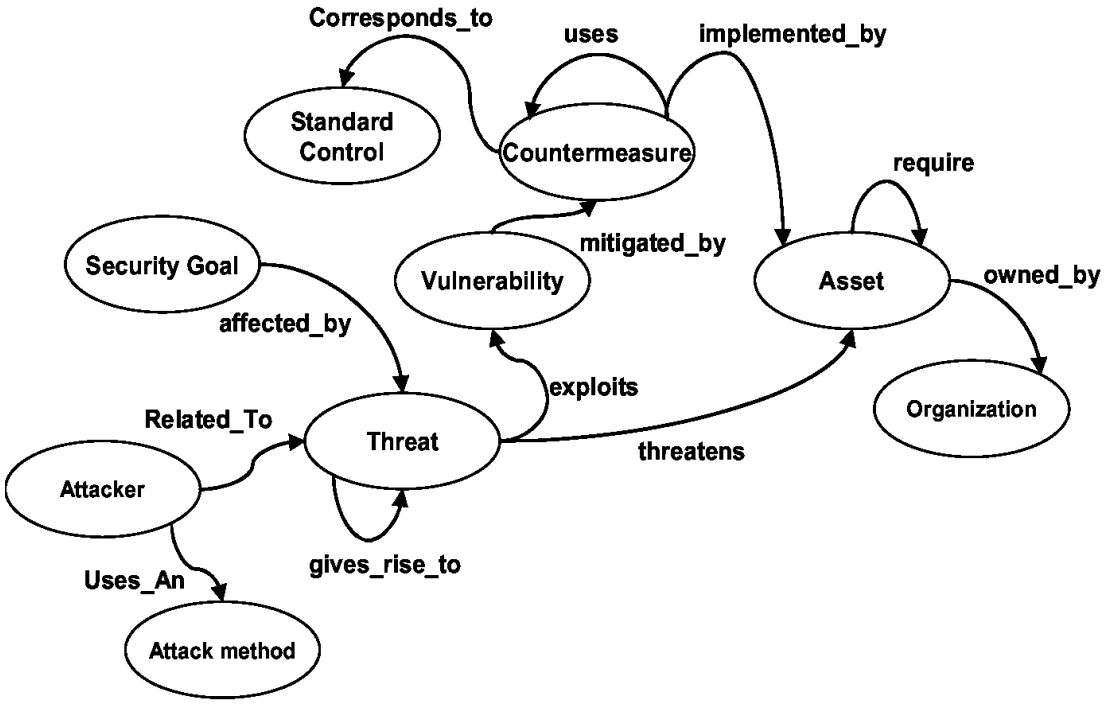


Figura 19 Ontología de alto nivel de seguridad

En lo relacionado con la ontología de dominio desarrollaron una visión genérica de ontologías de dominio. Esta ontología de dominio de alto nivel representa un dominio de acuerdo principalmente a entidades y propiedades.

Ahora bien, el método requiere la exploración combinada de un par de ontologías existentes: una de seguridad y una de dominio. Esta actividad se logra con la propuesta de dos sub-conjuntos de reglas heurísticas que especifican llamadas de alto nivel, basadas en las propuestas de ontologías de alto nivel. El primer conjunto se ocupa del análisis de la ontología de dominio y el segundo grupo del análisis de la ontología de

seguridad, es decir, las reglas heurísticas proporcionan los medios para extraer el conocimiento relevante de estas ontologías.

Por otro lado, el objetivo principal de esta propuesta es proporcionar una guía para la elicitación de requisitos de seguridad de un dominio específico. En esta línea, para obtener la especificación de requisitos de seguridad esperada a través de este método es necesario crear una especificación de requisitos de seguridad de alto nivel. Este modelo de requisitos inicial es creado por un analista a partir de las necesidades y preocupaciones expresadas por los interesados acerca de la seguridad en el inicio del proyecto.

Con este contexto, para la aplicación de las reglas heurísticas se requiere el alineamiento del modelo de requisitos de seguridad (SR) construido en la fase inicial del proceso, la ontología de seguridad (SO), y la ontología de dominio (DO). Los autores de esta propuesta proponen el siguiente alineamiento.

Un *objetivo de seguridad* en la ontología de seguridad, se alinea con una *meta* en el modelo de requisitos. Una *amenaza*, está alineada con un *evento* en la ontología de dominio, y con una *tarea maliciosa* en el modelo de requisitos. Una *vulnerabilidad* está alineada con el *punto vulnerable* en el modelo de requisitos. Un *activo* se alinea como una *entidad atómica o compuesta* en un dominio. Gracias a los *recursos* en los conceptos del modelo de requisitos, el *activo* de la ontología se caracteriza en el modelo de requisito.

Se observa que el método está basado en reglas de razonamiento para descubrir y añadir nuevos elementos para el modelado esperado de requisitos de seguridad de un dominio específico. Este proceso de razonamiento se obtiene aplicando reglas de análisis a SR, SO y DO. Las reglas de análisis mostradas en esta publicación son:

- **Identificación de objetivo de Seguridad** (regla 1): Cada meta (*softgoal*) en el modelo de requisitos de entrada es mapeado a los objetivos de seguridad de la ontología (*secgoal*), utilizando la función *EquivalentClass*. La equivalencia se basa en una correlación de léxico y sinonimia usando un tesauro como Wordnet. Si se encuentra una equivalencia, entonces la meta es considerada como un objetivo de seguridad y por lo tanto, se puede especificar como tal en la especificación de requisitos de seguridad.
- **Identificación de amenazas** (regla 2): La ontología de seguridad indica que cada objetivo puede estar relacionado con una amenaza por la relación *affectedBy* que indica que la amenaza puede ser un riesgo que causa el fracaso *SecGoal*. Esta relación se explora a lo largo del modelo de requisitos de entrada y la ontología de seguridad de la siguiente manera. Una amenaza se puede identificar cuando una tarea maliciosa se lleva a cabo por un agente que es un atacante. Si se cumple la situación, entonces se crea la tarea amenazante en el modelo de seguridad, y unidos por un vínculo de contribución a la meta.
- **Identificación de amenaza de dominio** (regla 3): Usando la ontología de dominio podemos obtener una mayor precisión en relación con dicha amenaza. Dado que las amenazas pueden diferir de un dominio a otro. Una amenaza de dominio puede ser identificada a través de eventos de dominio.
- **Identificación de vulnerabilidad** (regla 4): En esta regla, cada amenaza que se relaciona por la relación *exploits* a una vulnerabilidad se busca en toda la ontología de seguridad. Una vulnerabilidad es una debilidad que permite a un atacante reducir un sistema de seguridad. La regla analiza esta información y lo agrega al modelo de requisitos (SR) cuando la ontología de seguridad revela que hay una posible vulnerabilidad en el sistema bajo la forma de una amenaza que normalmente afecta a

una vulnerabilidad de la que una de las metas en el modelo de entrada puede ser asimilada.

- **Identificación de contramedidas** (regla 5): Esta regla busca los controles (contramedidas) que se pueden utilizar para manejar estas amenazas. Las contramedidas se buscaron en la ontología de seguridad de entrada a través de las relaciones *MitigatedBy* que están asociadas con una contramedida correspondiente al control estándar.
- **La identificación de recursos** (regla 6): De acuerdo con la ontología de seguridad un control necesita un activo para ser implementado. Este es el caso cuando una relación *implementedBy* aparece en la ontología entre el control y el activo. La regla escanea la ontología con el fin de encontrar los activos que se utilizan para la implementación de un control como sistemas de control de acceso, puertas de seguridad, o protección de virus. Cuando se encuentran, se introducen en la especificación requisitos software bajo la forma de recursos, entonces vinculados por un enlace de descomposición a la tarea agregada previamente.
- **Identificación de recursos de dominio** (regla7): Los modelos de entrada contienen mucha información acerca de los recursos, algunos de los cuales son de independiente-seguridad, otros de los que son más relevantes en la especificación de los requisitos de seguridad. Por otro lado, la ontología de dominio indica qué activo se utiliza típicamente en el dominio para manejar cuestiones de seguridad. La ontología de dominio (DO) es por lo tanto explorada en busca de entidades que puedan ser asimiladas a los recursos identificados en el modelo de entrada. Cada recurso en el modelo de requisitos de entrada es mapeado al activo correspondiente en la ontología de dominio utilizando la función *equivalentClass*.

Las reglas específicas de la ontología de seguridad y las reglas específicas de la ontología de dominio se pueden aplicar de forma iterativa hasta que uno cree que todos los posibles artefactos se han añadido a la especificación de requisitos. El proceso de identificación de requisitos se puede considerar como completo cuando no hay un nuevo objetivo de seguridad que se puede añadir en la especificación. Así mismo, la especificación de requisitos de seguridad puede considerarse completa cuando no se pueden añadir nuevas amenaza o activo a la seguridad.

8.5 Conclusiones de las propuestas ontológicas para la IR

En la presente sección se ha hecho un estudio de diferentes propuestas relevantes relacionadas con la utilización de la ingeniería ontológica para la representación y análisis del conocimiento de una ER. A continuación se describen las características y conclusiones más relevantes de cada una de ellas.

En la primera propuesta "Using Domain Ontology as Domain Knowledge for Requirements Elicitation" se plantea un método para el análisis de requisitos basado en la técnica de la ontología de dominio para el modelado del conocimiento del dominio general. El objetivo principal es mejorar la eficiencia de la elicitación de requisitos bajo un procesamiento semántico de forma automática con herramientas informáticas.

Ahora bien, en la búsqueda de este objetivo la parte más delicada de esta propuesta es la de modelar la semántica de la especificación de una lista de requisitos en una ontología. Este proceso en principio se apoya en técnicas de descomposición léxica como la que está definida en "Software Development Process From Natural Language Specification" (Saeki, Horai, & Enomoto, 1989). Sin embargo, en esta referencia no están resueltos todos los problemas del procesado de lenguaje natural que se requieren para modelar toda la semántica de un requisito. Otra carencia se encuentra en la propuesta del metamodelo de ontología para la organización del modelado de los elementos de las

descripciones de requisitos al estar compuesto por pocos conceptos y relaciones muy generales para modelar la semántica de la gran diversidad de tipos de requisitos.

Por otro lado, trataron de evaluar la facilidad de uso y la eficacia de este método a través de un experimento pero ellos mismos comentan que el experimento es demasiado pequeño para sostener la generalidad de los resultados experimentales.

La segunda propuesta "Revisiting the Core Ontology and Problem in Requirements Engineering" está basada en una ontología básica propuesta por Zave y Jackson en 1997 con el propósito de desarrollar con éxito una ER. Los autores argumentan que esta ontología no cubre todos los tipos de incumbencias básicas que comunican los interesados en el sistema comunican al ingeniero de software. Ellos proponen una nueva ontología núcleo ampliando la formulación de la primera propuesta basándose en las incumbencias clasificadas en creencias, deseos, intenciones y actitudes. Con esta base, este documento revisa los conceptos centrales posibles y propone una serie de definiciones de estos. Es verdad que esta nueva base de organización del conocimiento de una ER es mejor, pero no queda claro por qué esta es la mejor opción. En el mismo artículo se expone que no existe consenso sobre los conceptos precisos de la ontología fundamental, dado que existen diferentes formas de entender un requisito.

Los conceptos son mostrados para cubrir todas las expresiones. La intención es asegurar que se puedan clasificar todos los tipos de contenidos de las comunicaciones de los interesados en el sistema como instancias de conceptos o relaciones de la ontología propuesta. En esta ontología propuesta los conceptos son: objetivo, meta, restricción de calidad, plan y suposición de dominio; las relaciones están basadas en opciones de actitud y preferencia, de aproximación justificada, y de consecuencia no monótona (no repetitiva). El concepto objetivo pretenden cubrir los requisitos funcionales y los objetivos meta y restricción de calidad cubren la taxonomía clave de los requisitos no funcionales. Los autores exponen que esto permite la separación aclara entre estos conceptos. Sin embargo, no ha estado particularmente claro cómo realizar esta separación.

En el siguiente planteamiento "Modelling Reusable Security Requirements based on an Ontology Framework" se propone un marco de trabajo para la representación ontológica de requisitos de seguridad reutilizables. El marco de trabajo propone una combinación de una ontología de análisis de riesgo y una ontología de requisitos para la representación de los requisitos de seguridad. Sin embargo, exponen que este marco de trabajo solo es la base para elaboración de un método para obtener y especificar requisitos de seguridad.

La utilización de estándares oficiales de seguridad de análisis de riesgos en el desarrollo de la ontología de análisis de riesgo proporciona una conceptualización uniforme relacionada con los diferentes proyectos dentro de la organización y/o con otras organizaciones similares. Sin embargo, la ontología de requisitos se desarrolla bajo una experiencia propia basada en los estándares IEEE 830 y IEEE 1233 que no se especializan en la organización de requisitos de seguridad.

Ahora bien, la propuesta para el modelado de la organización de requisitos basada en ontologías proporciona medios para detectar defectos e inconsistencia en los requisitos. Sin embargo, en lo que respecta a los procesos semánticos solo hay un pequeño ejemplo de la utilización de restricciones semánticas mediante el uso de reglas (Semantic Web Rules - SWRL) para la verificación de la consistencia de la metainformación asociada a la organización y no a la semántica propia de cada uno de los requisitos.

En el último trabajo analizado "Using Security and Domain ontologies for Security Requirements Analysis", se propone un método que intenta aprovechar la semántica de ontologías existentes de seguridad y de dominio para guiar la elicitación de requisitos de seguridad para un dominio específico a través de una colección de reglas heurísticas.

El planteamiento de utilizar un conocimiento existente en ontologías desarrollas por otros grupos de trabajo para crear una especificación de requisitos de seguridad no es del todo verdad cuando para el modelado de requisitos es necesaria una especificación de requisitos previa para ir añadiendo nuevos elementos al modelo.

La propuesta presenta una evaluación cualitativa conducida a través de un análisis por cuatro expertos. Las conclusiones son que la idea es útil e interesante y que la utilización de un dominio general y otro de requisitos hacen una diferencia con respecto a la utilización de solo ontologías de seguridad. Sin embargo, manifiestan que se encuentra limitado por las técnicas de desarrollo. En esta línea, los autores manifiestan que el proceso de evaluación todavía necesita ser refinado con validación cuantitativa a través de un prototipo que está bajo construcción. Con este escenario, manifiestan que es un trabajo en desarrollo con la intención de conseguir extraer más conocimiento del dominio y adaptar el análisis basado en riesgo y en ataques.

En este punto, parece pertinente comentar que en esta línea de investigación cabría esperar propuestas con rigurosas técnicas de procesamiento de lenguaje natural a las descripciones de requisitos. Sin embargo, las propuestas relacionadas en esta línea de investigación proponen métodos de análisis de requisitos con un procesado ligero.

En este contexto, entre las propuestas analizadas en esta línea de investigación la más interesante se encontró en "uso de ontología de dominio como conocimiento del dominio para la elicitación de requisitos". Esta propuesta es la única que intenta extraer la semántica de cada descripción de requisito a través de una técnica de descomposición léxica.

Por otro lado, la idea de tener conocimiento del dominio es un factor recurrente para conseguir una elicitación de requisitos de alta calidad. En general proponen métodos donde una ontología de dominio puede ser utilizada como el conocimiento del dominio. En donde es posible establecer una correlación entre una ER y elementos ontológicos de dominio que representan semántica mediante el uso de reglas de inferencia. Es decir, proporcionan medios para guiar la organización de una ER en base a su conocimiento. Sin embargo, no plantean técnicas y métodos para crear estructuras de organización para la creación de una ER para cualquier línea de producto.

El principal aporte de estas ideas es exponer las bases para la creación de métodos que permitan a un analista navegar por una ER para tener la posibilidad de mejorar la calidad del conjunto de sus requisitos.

CAPÍTULO III
DESARROLLO TEÓRICO

Capítulo IV. DESARROLLO TEÓRICO

En el presente capítulo se describen las aportaciones desarrolladas desde el punto de vista teórico. La primera aportación definida es el **marco de referencia para la conceptualización de aspectos** de un producto software OA en cualquier momento de su desarrollo. En el contexto del marco de referencia se describen los lenguajes necesarios para cada una de las diferentes conceptualizaciones.

En el primer capítulo se manifestó que el objetivo de esta Tesis Doctoral es proporcionar una metodología para la organización y representación del conocimiento en las etapas tempranas de un producto software OO para la gestión efectiva de su evolución en un DSOA. En este escenario, las Etapas Tempranas son acotadas al ámbito de la conceptualización de Aspectos Tempranos (AT). Por lo tanto, la segunda aportación que se describe en este capítulo es la propuesta de **un modelo conceptual de AT**.

El modelo conceptual de AT propone conceptos clave para la organización del conocimiento. Entre los elementos clave se encuentra la **Arquitectura de Puntos de Vista** (APV). Por este motivo se planteó desarrollar **un método para la creación de una** APV.

Entre las directrices necesarias para la creación de una APV se encuentra la creación del ámbito de la **arquitectura lógica**. El desarrollo para justificar el modelado de este ámbito requirió el análisis de algunos estilos arquitectónicos, el cual es descrito en este capítulo.

Una parte importante del modelado de los AT es la representación semántica de la **Especificación de Requisitos** (ER) de un producto software. En una ER clásica existe un modelado semántico muy pobre que afecta a la calidad de la especificación. En este contexto surgió la tercera aportación que describe un **marco de referencia para medir la calidad de una ER** en donde se proponen y justifican los factores con los que hay que analizar la calidad.

1 Conceptualización de Aspectos en un DSOA

A lo largo de un DSOA cada etapa es afectada por la especificación del concepto de aspecto. Típicamente, en las diferentes etapas de un DSOA el concepto de aspecto es especificado respetando las restricciones del lenguaje OA que se utilice en ese momento. Sin embargo, existen muchas propuestas con varios lenguajes que proporcionan diferente sintaxis y semántica en cada etapa del DSOA. Así mismo, las personas implicadas pueden tener diferentes concepciones de lo que es un aspecto. En consecuencia, existe discrepancia en el cómo y dónde se pueden identificar los aspectos que tienen relevancia, así como en la forma de relacionar estos aspectos para que evolucionen conjuntamente. Por lo tanto, existe la necesidad de una propuesta de clasificación conceptual de aspectos para la creación consistente de aspectos en cualquier instante de tiempo de cualquier Proceso de Desarrollo de Software (PDS).

Varios grupos de investigación han trabajado para poder llevar a la práctica la meta de separación avanzada de incumbencias a través del concepto de aspecto en diferentes etapas del ciclo de vida de un producto software. Se han propuesto distintos mecanismos de abstracciones y composición diferenciando matices del concepto de Aspecto dentro de dichas etapas.

- **Aspecto** (Kiczales et al., 1997): La POA propuso el aspecto al más bajo nivel (aspecto de implementación) abstrayendo y componiendo aspecto en la etapa de codificación y mantenimiento.
- **Aspecto Temprano** (Rashid et al., 2002): La iniciativa de los Aspectos Tempranos se concentra en gestionar los aspectos creados en la primera etapa de desarrollo, abstrayendo y componiendo aspectos en el dominio de la ingeniería de requisitos.
- **Aspecto Arquitectónico** (Tekinerdogan, 2004): En el nivel arquitectura, las incumbencias transversales requieren ser especificadas en componentes arquitectónicos individuales tomando en cuenta las etapas tempranas y apuntando a las finales del ciclo de vida de un producto software.

Se observa que el concepto de aspecto ha evolucionado naturalmente a diferentes niveles de conceptualización a lo largo de un DSOA. Se intenta brindar un contexto general al desarrollador que le permita separar claramente aspectos en sus diferentes roles en las diferentes etapas del ciclo de vida del producto software OA para producir software de calidad. Por lo tanto, es necesario dejar abierta la posibilidad de desarrollar diferentes niveles de conceptualización de aspectos a lo largo de un desarrollo de un producto software. Se ha propuesto anteriormente una nueva definición del concepto de aspecto que cumple con dicha característica (Barra, Fraga, & Llorens, 2007).

> "Un Aspecto es una unidad modular que agrupa incumbencias, examinadas y seleccionadas bajo puntos de vista específicos, diseminadas a través de los diferentes modelos creados en las diferentes etapas del ciclo de vida de un desarrollo software. Los Aspectos cambiarán su nivel de abstracción en cada una las etapas del ciclo de vida de su desarrollo Software, pudiendo o no existir una traza entre cada uno de los diferentes niveles de conceptualización de un Aspecto"

Esta definición proporciona las bases para desarrollar un contexto general que le permita al desarrollador, en sus diferentes roles, separar claramente aspectos en diferentes etapas del ciclo de vida de un producto software OA. Así mismo, deja abierta la posibilidad de plantear diferentes niveles de conceptualización de aspectos que respeten la semántica a lo largo de un DSOA.

Para la creación consistente de aspectos en cualquier instante de tiempo de un Proceso de Desarrollo Software (PDS) fue necesario definir un marco de referencia con principios, guías y términos útiles que permitiera diseñar modelos con líneas claras de separación entre diferentes posibles niveles de conceptualización de aspectos a lo largo de un DSOA.

Las conceptualizaciones de los aspectos deben ser creadas a través de puntos de vista establecidos por las diferentes etapas que componen un PDS. Se tomó siempre en cuenta la necesidad de gestionar la traza entre los modelos de cada uno de los aspectos propuestos respetando claramente la semántica dentro de un mismo nivel de conceptualización. Así mismo era deseable que los DSOA conceptualizasen los aspectos siempre en los mismos niveles en cualquiera de las diferentes formas de ver las etapas de un ciclo de vida.

1.1 Evaluación de las etapas de un PDS en el contexto de la OA

En el contexto de un Proceso de Desarrollo Software (PDS), una etapa está conformada por un conjunto de tareas y actividades con un objetivo común bien definido. Se

desarrolla en un intervalo de tiempo entre dos hitos importantes de un proceso que produce artefactos software. Un PDS tiene el objetivo de ordenar parcialmente dichas etapas con la intención de obtener un producto software de calidad que cumpla con las necesidades del cliente. Sin embargo, cada proyecto de software requiere de una forma particular de abordar el problema, donde la composición y sincronía de las etapas están basadas en el tipo de proyecto.

Un factor estratégico en el éxito de un proyecto software es la creación del PDS, donde su conformación está basada en un conjunto de principios aportados por diferentes modelos genéricos de ciclos de vida obtenidos de forma heurística a lo largo de la historia de la ingeniería de software. Entre los modelos genéricos más importantes tenemos (Royce, 1987)(Boehm, 1988): el Modelo en cascada (definió las primeras etapas), el Modelo de desarrollo evolutivo (introdujo el enfoque iterativo), el Modelo de desarrollo incremental (una comprensión incremental del problema y un crecimiento incremental de una solución a través de varios ciclos) y el Modelo de desarrollo en espiral (este modelo toma explícitamente en consideración el riesgo siendo una actividad importante en la administración del proyecto). Con este contexto, las propuestas comerciales y académicas de PDS actuales han recogido las mejores aportaciones de los modelos genéricos y enfocan todo un ciclo de vida a un solo paradigma. Por lo tanto, un proceso de DSOA debe contener todos estos elementos.

Se ha comentado que esta tesis está enfocada a los DSOA con base en el paradigma OO, por lo tanto, las propuestas de desarrollo software a tomar en cuenta en este documento son las que ofrecen detallar un proceso basado en el paradigma OO y que ofrezcan las características antes mencionadas, entre las cuales podemos mencionar DSDM (del inglés *Dynamic System Development Method*) (Stapleton & Constable, 1997), OPEN (del inglés *Object-Oriented Process, Environment and Notation*) (Graham, Henderson-Sellers, & Younessi, 1999) o el Proceso Unificado de Desarrollo Software (del inglés *Unified Software Development Process*) (Jacobson, Booch, & Rumbaugh, 1999).

Los diferentes Procesos de Desarrollo Software (PDS) componen y sincronizan sus etapas con diferentes enfoques y su propia terminología. Sin embargo, analizando varios PDS conocidos se ha encontrado que algunas actividades disciplinares fundamentales se encuentran presentes en todas las propuestas (Sommerville, 2005): **Especificación**, **Desarrollo**, **Validación** y **Evolución**. Típicamente se descompone la **Especificación** en captura de requisitos y análisis, así como el **Desarrollo** en diseño e implementación. Sin embargo, otra actividad fundamental que debe ser considerada es la **Arquitectura**, actividad básica que ha adquirido relevancia como elemento de enlace entre la Especificación y el Desarrollo. Así mismo, la arquitectura tiene el objetivo general de controlar el desarrollo iterativo e incremental de un proyecto a lo largo de todo su ciclo de vida. Por otro lado, las actividades fundamentales de Validación y Evolución han trascendido a todo el ciclo de vida del producto software. En consecuencia, se han convertido en actividades fundamentales transversales a las actividades fundamentales de Especificación y Desarrollo que son dominantes.

Dentro de las propuestas de PDS que permiten descomposición por etapas, ha surgido una disciplina diferente que actualmente está acaparando el interés en la comunidad investigadora. El Desarrollo Dirigido por Modelos (Model Driven Development - MDD-) (Stahl & Voelter, 2006) es una disciplina con el objetivo primordial de restar predominio a la etapa de **implementación** en el proceso de desarrollo software para depositar más responsabilidad en los modelados de las **etapas tempranas**. Dichos modelos dejan de ser utilizados únicamente para la documentación o como guía para la implementación, convirtiéndose en el artefacto principal del proceso de desarrollo, permitiendo a los desarrolladores centrarse en la semántica del sistema a construir,

abstrayéndose de todos los detalles de la implementación relativos a la plataforma subyacente.

Basado en este paradigma surgió para hacer frente a los problemas en el desarrollo de software actual, la aproximación Arquitectura Dirigida por Modelos (del inglés *Model Driven Architecure -MDA-*) (Object_Management_Group-OMG, 2003) de la OMG (del inglés *Object Management Group*) (OMG, 1989). Dicha propuesta ha desarrollado tres perspectivas con diferentes niveles de conceptualización en el proceso de desarrollo software: Modelo independiente de la computación (del inglés *Computational Independent Model -CIM-*), Modelo independiente de la plataforma (del inglés *Platform Independent Model -PIM-*) y Modelo específico de la plataforma (del inglés *Platform Specific Model-PSM*). Estas perspectivas o actividades fundamentales disciplinares están basadas en el concepto de plataforma, el cual típicamente se relaciona con la tecnología de implementación (Kaindl, 1999) (Graham et al., 1999). Por lo tanto, podríamos decir que los PIM son modelos independientes de la tecnología y los PSM son modelos dependientes de la tecnología, dejando el modelado del negocio y del dominio al CIM.

1.2 Un marco de referencia para la conceptualización de Aspectos

Cada una de las diferentes propuestas existente de Procesos de Desarrollo Software (PDS) compone y sincroniza sus etapas con diferentes enfoques y su propia terminología pero todas están basadas en actividades fundamentales disciplinares. Una etapa contiene por lo menos una actividad fundamental que proporciona la pauta para delimitar los niveles de conceptualización de aspectos con puntos de vista coherentes en cualquiera de los contextos de las diferentes propuestas de PDS.

Con este planteamiento se ha desarrollado una propuesta de marco de referencia de tres diferentes niveles de conceptualización de aspectos a lo largo de cualquier PDS:

- **Aspectos tempranos**, los cuales se especifican en la ingeniería de requisitos hasta las primeras líneas de solución de arquitectura, representando vistas lógicas independientes de la plataforma tecnológica.
- **Aspectos medios**, los cuales se especifican en estructuras aspectuales al nivel del diseño con vistas físicas dependientes de la plataforma tecnológica.
- **Aspectos finales,** los cuales se especifican en lenguajes especializados de programación para la implementación.

El objetivo principal de esta propuesta de clasificación conceptual de aspectos es definir términos útiles, principios y guías para la creación consistente de aspectos en cualquier instante de tiempo de cualquier proceso de desarrollo de software, teniendo siempre la posibilidad de gestionar la traza entre los modelos aspectuales dentro de un mismo nivel de conceptualización.

1.2.1 Límites entre aspectos tempranos, medios y finales

Delimitar las competencias entre los niveles de conceptualización de aspectos propuestos requiere el análisis de las actividades fundamentales disciplinares en el contexto de otros procesos de desarrollo software. Esto es suficiente para aclarar los límites entre aspectos medios y aspectos finales. Sin embargo, para definir los límites entre aspectos tempranos y aspectos medios se requieren dos disertaciones sobre conceptos relacionados con la arquitectura software. Uno de los razonamientos importantes está relacionado con la simbiosis de los conceptos de Vista y Aspecto, y el otro es aclarar el vínculo entre estilos y patrones arquitectónicos.

1.2.1.1 Actividades fundamentales disciplinares en los diferentes tipos de Procesos de Desarrollo Software

En la mayoría de los PDS, de la Actividad Fundamental Disciplinar (AFD) de **Especificación** típicamente se derivan las etapas de **Captura** y **Análisis** de Requisitos. Actualmente estas etapas se han recompuesto en la popular disciplina conocida como Ingeniería de Requisitos (IR). Así mismo, de la AFD de **Desarrollo** típicamente derivan las etapas de **Diseño** e **Implementación**.

Estas etapas se caracterizan por utilizar diferentes lenguajes y perspectivas en cada uno de sus modelos resultantes pero representando siempre la misma semántica del producto software.

La etapa de **Implementación** está delimitada claramente por los lenguajes de programación y en los DSOA no es la excepción. El nivel de conceptualización de aspectos nombrada como **Aspectos Finales** representa los modelos en código de la etapa de implementación. Diferentes investigaciones de la POA han proporcionado lenguajes OA para la codificación y gestión de productos software OA (Ossher & Tarr, 2000) (Laddad, 2003) con un alto nivel de eficiencia. Dicho esto, podríamos decir que el nivel conceptual de aspectos finales ha sido resuelto. En este sentido, los aspectos finales están delimitados por la utilización de lenguajes de codificación y la dependencia de la plataforma.

Ortogonalmente, MDA delimita la perspectiva de PSM a la implementación con la promesa de una trasformación automática de los modelos en código. Ahora bien, entre los ingenieros de software existe un consenso en que la etapa de **Diseño** plantea una solución a un problema o situación que ha sido comprendida en los modelos de la AFD de **Especificación**. En esta línea, MDA delimita la perspectiva de CIM con PIM con el mismo criterio. Por lo anterior, el PIM y PSM de MDA encajan en la AFD de Diseño. Sin embargo, los modelos PIM y PSM son diferentes. Así mismo, hay diferentes tipos de PIM con matices de PSM y viceversa. Una directriz que nos permite ubicarnos mejor entre dichas perspectivas es la mencionada en párrafos anteriores: *"considerar los PIM como modelos independientes de la tecnología y los PSM como modelos dependientes de la tecnología"*.

Al evaluar las etapas de un PDS en el contexto de la OA se encontró que las actividades de **Validación** y **Evolución** son Actividades Fundamentales Disciplinares (AFD) que en la actualidad deben realizarse en todas las etapas. A este respecto, existen AFD encapsuladas claramente en una o más etapas que se han denominado dominantes y otras AFD que pueden estar diseminadas a lo largo de todo el ciclo de vida del producto software, que se han denominado transversales. Por lo tanto, las AFD de Validación y Evolución corresponden a las AFD **trasversales** y las actividades de **Especificación** y **Desarrollo** son AFD **dominantes**.

En la actualidad la AFD de **Arquitectura** no tiene un consenso en la comunidad de investigadores de la ingeniería de software de su influencia transversal o dominante. En la concepción de arquitectura existen dos grandes grupos: algunos consideran la arquitectura una organización abstracta para guiar el producto software a lo largo de un ciclo de vida y otros la consideran como una etapa donde se obtienen una colección de productos para documentar las primeras líneas de solución. En otras palabras, los primeros visualizan su influencia a lo largo de todas las etapas existentes en un PDS y los segundos consideran la Arquitectura como una etapa dominante no trasversal. La postura de esta tesis doctoral es considerar la arquitectura como una etapa dominante.

El resultado del análisis de las AFD en los diferentes tipos de Procesos de Desarrollo Software se resume en la Tabla 5.

<table>
<tr><th>Actividades Fundamentales (AF)</th><th>USDP</th><th>MDA</th><th>Etapas Clásicas Docentes</th><th>Conceptualización de Aspectos</th></tr>
<tr><td rowspan="2">Especificación</td><td>Requisitos</td><td rowspan="3">CIM</td><td>Ingeniería de Requisitos</td><td rowspan="3">Aspectos Tempranos</td></tr>
<tr><td>Análisis</td><td rowspan="2">Arquitectura Software (nueva AF)</td></tr>
<tr><td rowspan="3">Desarrollo</td><td rowspan="2">Diseño</td><td>PIM</td></tr>
<tr><td>PSM</td><td>Diseño Detallado</td><td>Aspectos Medios</td></tr>
<tr><td rowspan="2">Implementación</td><td rowspan="2">-</td><td>Codificación</td><td rowspan="2">Aspectos Finales</td></tr>
<tr><td>Compilación</td></tr>
</table>

Tabla 5 Actividades Disciplinares vs diferentes tipos de Etapas

1.2.1.2 Simbiosis de los conceptos de Vista y Aspecto

En la actualidad ninguna de las definiciones de Arquitectura de Software (AS) conocidas es respaldada por la totalidad de la comunidad. Sin embargo, existe una convergencia en la definición del estándar IEEE 1471 (IEEE-Computer-Society, 2000) que ha sido revisado en la nueva norma (ISO/IEC-IEEE, 2011).

La norma IEEE 1471 propone que un sistema posee una descripción arquitectónica especificada por una colección de modelos organizados en Vistas. La norma define **Vista** como una representación de todo un sistema desde la perspectiva de un grupo relacionado de incumbencias. La creación y utilización de una Vista está dirigida por convenciones coherentes especificadas en un **Punto de Vista**. Por lo tanto, una Vista permite concentrarse en algunas incumbencias concretas del sistema, abstrayéndose del resto. La norma considera la necesidad de contemplar la consistencia y completitud de las Vistas, manifestando que una descripción arquitectónica es consistente si ninguna de sus Vistas impone requisitos contradictorios sobre alguna de las otras Vistas y es completa si contiene todas las necesidades de los interesados en el sistema.

La norma IEEE 1471 establece la información que ha de contener la especificación de cualquier Punto de Vista:

1. El nombre del Punto de Vista.
2. Los interesados en el sistema para los que el Punto de Vista es relevante.
3. Las incumbencias del sistema tratadas por el Punto de Vista.
4. Los lenguajes, notaciones, técnicas de modelado y metodologías a usar para construir Vistas relativas a su Punto de Vista.
5. La Fuente, para un punto de vista de biblioteca.

De la misma manera la norma plasma que cada Vista incluirá:

1. Un identificador y otra información introductoria, como se definió por la organización usuaria.
2. La representación del sistema construido con los lenguajes, métodos y técnicas de modelado o analíticas del "Punto de Vista" asociado.
3. Información de configuración, como se definió por la organización usuaria.

La norma IEEE 1471 propone el concepto de Punto de Vista para agrupar conceptos, lenguajes y reglas estructurales propias de abstracción que permitan concentrase en algunas incumbencias del sistema abstrayéndose del resto y concretándose en una Vista.

En esta línea, diferentes propuestas de desarrollo de descripción arquitectónica tienden a estructurar el sistema software en torno a diferentes Puntos de Vista, incluso sin manifestarlo abiertamente. Por ejemplo en el desarrollo de las aplicaciones Web se proponen tres: la navegación, la presentación y la lógica de negocio. En el enfoque MDA se propone igualmente tres Puntos de Vista: el independiente de la computación, el independiente de la plataforma y el específico para una plataforma. Una de las propuestas precursoras del uso de Vistas bajo diferentes Puntos de Vista es el modelo de Kruchten (Kruchten, 2003) en el que se definen cinco vistas.

La primera norma que se ha acogido abiertamente al estándar IEEE 1471 es el RM-ODP (Reference Model – Open Distributed Processing) (ITU-T/ISO/IEC, 1994), la cual establece una serie de conceptos fundamentales para el desarrollo de aplicaciones abiertas y distribuidas, definiendo cinco Puntos de Vista: el de la empresa, de la información, la computación, la ingeniería y la tecnología. Este último modelo es el que de alguna manera mejor describe las Vistas a crear, por ese motivo actualmente está siendo adoptado con más frecuencia por la comunidad.

En este contexto, se observa que el modelado de Puntos de Vista es una técnica muy efectiva al especificar sistemas. En consecuencia, la especificación de Punto de Vista es la mayor aportación de esta propuesta de descripción arquitectónica.

La norma IEEE 1471 ha definido los modelos de Vista como los productos de trabajo y los Puntos de Vista como los elementos que establecen los convenios de construcción, interpretación y uso de las Vistas. Sin embargo, la especificación de Vista en la norma carece de conceptos abstractos para su modelado. Ortogonalmente, la OA ha proporcionado el modelo Aspecto con los conceptos de incumbencia transversal y punto de unión que le da fortaleza, sin embargo, en la OA no existe el concepto de Punto de Vista. Por lo tanto, se ha propuesto la simbiosis entre Vista y Aspecto (Barra et al., 2007). No se quiere decir con esto que debiera desaparecer alguno de los dos conceptos, únicamente aceptarse como uno solo y utilizar el nombre indicado en el momento oportuno.

Recientemente la norma IEEE 1471 ha evolucionado a la norma ISO/IEC 42010 (ISO/IEC-IEEE, 2011). Entre los nuevos objetivos de la nueva norma encontramos añadir coherencia con sistemas y modelos de ciclos de vida software, así como alinearse con estándares relacionados con arquitectura y arquitectura empresarial de ISO. Sin embargo, la revisión de la norma no añade conceptos nuevos que modifiquen el enfoque en esta disertación.

1.2.1.3 Estilos y patrones arquitectónicos como partes complementarias

En las primeras aproximaciones de la AS, un análisis realizado a varios productos software permitió observar ciertas regularidades de configuración. Las configuraciones aparecían una y otra vez como solución al mismo problema y estas eran modeladas en torno a dichos conceptos clave (Garlan & Shaw, 1993). En dichos estudios se acuñó el concepto de **estilos arquitectónicos** (Garlan, Allen, & Ockerbloom, 1995) por analogía del uso del término en arquitectura de edificios (Perry & Wolf, 1992). Los estilos arquitectónicos fueron definidos como una codificación particular de elementos de diseño con acuerdos formales, es decir, un estilo arquitectónico limita tanto los elementos de diseño como las relaciones entre ellos. Inicialmente los elementos identificables de los estilos arquitectónicos se identificaron empíricamente, encontrando repetidamente en la práctica copias de decisiones estructurales lógicas a un mismo tipo de producto software. Como concepto, los estilos arquitectónicos fueron formulados cuando el escenario tecnológico era distinto al que tenemos hoy en día. Por lo tanto, el

primer objetivo de la disciplina de la arquitectura de software fue apoyar una **arquitectura lógica independiente de la tecnología**. Proporcionando líneas para un desarrollo homogéneo del producto software en las etapas tempranas.

Tras los estilos arquitectónicos en una relación a veces de yuxtaposición (Shaw & Garlan, 1996) y otras de completitud (Buschmann et al., 1996) (Shaw & Clements, 1997), surge el concepto **patrón arquitectónico**. Se señala con frecuencia el aire familiar entre estilos y patrones arquitectónicos, pero no se articula formal y sistemáticamente su relación. Quienes trabajan con estilos favorecen un tratamiento que concierne a la arquitectura como una estructura de componentes lógicos, mientras que quienes favorecen a los patrones se ocupan de estructuras de componentes que están más cerca del diseño e implementación. Por lo tanto, el siguiente objetivo fue apoyar una **arquitectura física dependiente de la tecnología** proporcionando más líneas para una codificación homogénea del producto software.

En el contexto de la hipótesis de esta tesis doctoral se recomienda que en la creación de una arquitectura de software, los estilos y patrones arquitectónicos tengan una relación de completitud.

1.2.1.4 Especificación de los límites entre aspectos tempranos y medios

La norma IEEE 1471 define la descripción arquitectónica como:

> "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution"[30].

Según esta definición se extraen conceptos clave de la arquitectura:

a. La **organización**, como un concepto cualitativo y estructural.
b. Los **componentes**, que representan los elementos tempranos de la solución.
c. Las **relaciones** entre los componentes y el entorno, que representan la estructura de interacción interna y externa.
d. Los **principios** para conducir el diseño y evolución, donde las decisiones de estos principios están justificados en los requisitos del naciente sistema.

Por lo tanto, el modelado de una arquitectura software debe contener partes relacionadas interna y externamente con una determinada organización. Así mismo deben estar contenidas en la descripción arquitectónica de un producto software el conjunto de **decisiones** tomadas para obtener los principios que guíen el diseño de las primeras líneas de solución.

Las decisiones involucran compensaciones y estrategias para hacer una resolución evitando conflictos con las incumbencias dominantes y transversales. Por lo tanto, la descripción arquitectónica especifica la evaluación de alternativas, selección y toma de decisiones sobre las soluciones técnicas y estrategias.

---

[30] La organización fundamental de un sistema incorporada en sus componentes, las relaciones entre los mismos y con el entorno, y los principios que orientan su diseño y evolución.

Por otro lado, se ha probado que los cambios en una arquitectura que respete la semántica de los requisitos permite detectar conflictos en los cambios de su organización de requisitos (Weston, Chitchyan, & Rashid, 2009). En consecuencia, la descripción arquitectónica está dirigida por los requisitos pero orientada a establecer el diseño detallado que será implementado.

Por todo lo anterior, la descripción arquitectónica es un producto que contiene dos partes complementarias. La parte que compete a esta tesis es la primera, la cual es reflejada en los modelos más tempranos del proceso de desarrollo de software que representan la descomposición lógica modular arquitectónica que tiene la obligación de describir el viaje de ida y vuelta a los requisitos. Por lo tanto, es coherente decir que las decisiones en la organización de requisitos están igualmente en la arquitectura lógica independiente de la tecnología y viceversa. En consecuencia, es plausible decir que los aspectos tempranos llegan hasta la arquitectura lógica. Se puede observar que esta disertación es respaldada por la evaluación de las etapas de un PDS en el contexto de la OA.

Ahora bien, es necesario puntualizar que una descripción arquitectónica se compone de la estructura lógica de una arquitectura y de la arquitectura física disertada en párrafos anteriores; sin embargo, esta parte de la arquitectura no fue competencia de esta tesis doctoral.

La conclusión es un compromiso de interrelación explícita entre la especificación de requisitos y la representación lógica arquitectónica. Esto implica la necesidad en la creación de requisitos de clasificar y priorizar las incumbencias para disminuir los conflictos y seleccionar la más ventajosa entre las diferentes perspectivas que se pueden tener. Por lo tanto, deben existir los mismos conceptos en la especificación de requisitos y en la arquitectura lógica.

En el contexto de las secciones anteriores, una definición para delimitar las competencias entre los niveles de conceptualización de aspectos tempranos y aspectos medios se describe a continuación.

Los Aspectos Medios se especifican en estructuras aspectuales al nivel del diseño con Vistas Físicas dependientes de la plataforma tecnológica (PSM) y los Aspectos Tempranos se detallan desde la Especificación hasta las primeras líneas de solución de Arquitectura Software, representando Vistas Lógicas independientes de la plataforma tecnológica (CIM y PIM).

1.2.2 Características del proceso basado en el marco de referencia

El marco de referencia para la conceptualización de aspectos permite desarrollar un **proceso de desarrollo software genérico para la construcción de productos OA**. Este proceso tiene la característica de ser configurable para diferentes áreas de aplicación y organizaciones. Promueve una de las más importantes buenas prácticas, el ciclo de vida iterativo e incremental. Así mismo, impulsa la creación de modelos semánticos formales.

1.2.2.1 Iterativo e incremental

Los actuales modelos de PDS utilizados para la producción de software de gran tamaño, requieren ser incrementales e iterativos para potenciar el desarrollo en paralelo de los distintos flujos de trabajo. Así mismo, debe proporcionar un enfoque disciplinado acerca de cómo asignar tareas y responsabilidades para poder trabajar en paralelo con

distintos equipos de personas y adaptarse a las siempre cambiantes necesidades del proyecto.

Este marco de referencia para la conceptualización de aspectos permite la creación de modelos aspectuales que serán interdependientes entre las diferentes conceptualizaciones. Por lo tanto, no existe una precedencia lógica temporal estricta a lo largo del proceso de un DSOA. Los modelos aspectuales de una misma conceptualización son interdependientes, por lo que pueden evolucionar en paralelo. Los modelos de Aspectos Tempranos son anteriores a los modelos de Aspectos Medios sólo dentro de la misma secuencia de versión. Así mismo, las versiones de los modelos aspectuales producidos en una misma iteración no son necesariamente compatibles entre ellas. Esta flexibilidad es plasmada en la Figura 20.

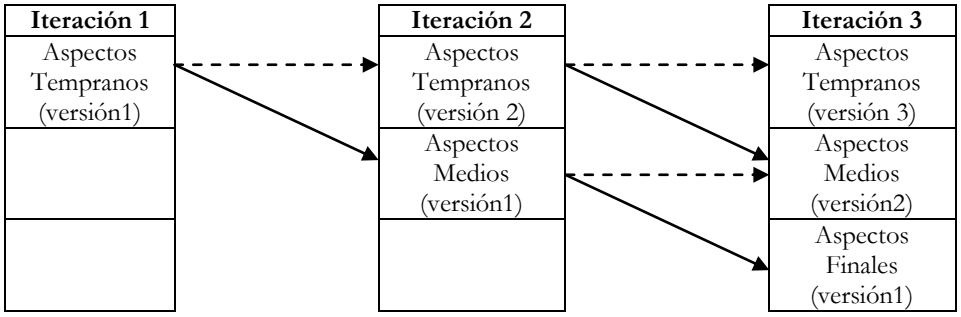


Figura 20 Relación lógico-temporal entre conceptualizaciones de Aspectos

1.2.2.2 Modelos semánticos

Este marco de referencia para la conceptualización de aspectos promueve la creación de modelos que proporcionen semántica formal del producto software. Estos modelos deberían representar el conocimiento con la finalidad de ser capturados y controlados digitalmente. La razón subyacente es minimizar los costes asociados a la gestión y evolución del producto software y maximizar la calidad de información.

Basándose en la semántica de los modelos se pueden clasificar y priorizar incumbencias para disminuir los conflictos entre los diferentes aspectos de un mismo nivel conceptual, así como guardar en un repositorio todos los artefactos generados en el desarrollo para mejorar las decisiones heurísticas de las técnicas utilizadas en la etapa correspondiente en futuros desarrollos.

1.2.2.3 Configurable

El marco de referencia para la conceptualización de aspectos es adaptable y puede configurarse a toda una familia de procesos de desarrollo OO, y además puede variarse para adaptarse a los procesos tradicionales y a los ágiles. Así mismo, es adaptable a las necesidades de la factoría de software compuesta de pequeños equipos de desarrollo de software hasta grandes organizaciones.

Este marco de referencia promueve un control de validación adaptable, así como una gestión objetiva y continua. La validación no se trata como una etapa dominante, si no como una etapa trasversal configurable.

2 Aspectos Tempranos

En la comunidad científica de la ingeniería del software es conocida la importancia de las etapas tempranas en cualquier proceso de desarrollo software. Sin embargo, tradicionalmente las etapas tempranas del ciclo de vida de un producto software no

están claramente delimitadas. En ocasiones las etapas tempranas pueden ser todas las desarrollas antes de la implementación.

Ahora bien, se ha dicho en la introducción que el alcance de esta tesis doctoral está acotado a la investigación e innovación de las etapas tempranas de un desarrollo software OA. Por lo tanto, los límites de las etapas tempranas deben estar basados en el marco de referencia para la conceptualización de aspectos, es decir, en el límite de los AT. La conceptualización de AT se centra en la gestión de incumbencias dominantes y transversales en las etapas tempranas de desarrollo. En esta línea, la comunidad científica enfocada a este campo de investigación (Rashid, 2005) acota los AT en las etapas relativas a la ingeniería de requisitos y la etapa del diseño de arquitectura (Elisa Baniassad et al., 2006). Sin embargo, la comunidad científica en arquitectura software no tiene un consenso del contenido de la etapa de arquitectura.

A este respecto, en párrafos anteriores se ha plasmado una visión justificada de un marco de referencia que limita las diferentes conceptualizaciones de aspectos. En consecuencia, estos lineamientos han dado las pautas para concretar una definición formal de AT:

> "Los Aspectos Tempranos son la gestión de incumbencias en todas las etapas de la ingeniería de requisitos, obteniendo modelos de especificación de las propiedades prometidas en el futuro producto software, así como la gestión de incumbencias en la etapa enfocada a las primeras líneas de solución de la arquitectura software, obteniendo modelos de Vistas lógicas independientes de la plataforma tecnológica".

Así, a partir de esta sección se describe cómo se enfocó el desarrollo de una metodología para la extracción, organización, representación, y validación del conocimiento de Aspectos Tempranos (AT) en un DSOA, con la meta de mejorar los costes de la creación y evolución de los modelos de un producto software.

2.1 Lenguajes en los Aspectos Tempranos

En torno a la aproximación MDA, algunos autores consideran que las correspondientes transformaciones de modelos PIM-PSM y PSM-PIM son verticales, mientras que las correspondencias PIM-PIM y PSM-PSM son trasformaciones horizontales (Sendall & Kozaczynski, 2003) (Clark, Sammut, & Willans, 2008). A este respecto, la propuesta de MDA ha logrado una mayor trascendencia en las técnicas de **transformación vertical**, proponiendo técnicas que logren una traza entre sus diferentes modelos conceptuales. Sin embargo las **transformaciones horizontales** carecen de metodologías eficientes que proporcionen la traza entre los modelos de un mismo nivel conceptual. En este sentido, han surgido diferentes propuestas que unen MDA con DSOA (Clemente, Hernández, & Sánchez, 2007). Las técnicas que ofrece DSOA complementan la carencia de la traza horizontal, pero actualmente no existe una unificación estructural clara. Por lo tanto, la conceptualización de aspectos permite la gestión de la traza horizontal independiente de la traza vertical; dicho de otra manera, la traza entre los modelos dentro de un mismo nivel de conceptualización.

Esta tesis doctoral se fundamenta en la combinación prometedora donde MDA ofrece separación vertical y DSOA ofrece separación horizontal. La coherencia de esta propuesta depende de la producción de modelos software en todo el ciclo de vida con **lenguajes compatibles entre niveles conceptuales de aspectos**. Sin embargo, el ámbito de esta tesis se centró únicamente en la problemática de la separación horizontal, por lo tanto, se ha dejado para trabajos futuros investigar la transformación vertical y llegar a respetar la semántica Aspectual entre niveles conceptuales.

Por ahora, los lenguajes propuestos deben permitir la gestión de la traza entre los modelos dentro de un mismo nivel de conceptualización. Con esta idea se han propuesto lenguajes que puedan cumplir con las características de la propuesta de marco de referencia para la conceptualización de aspectos a lo largo de cualquier PDS de esta tesis doctoral. Este marco de referencia específica tres niveles: Aspectos Tempranos, Aspectos Medios y Aspectos Finales.

En este contexto, las diferentes conceptualizaciones están delimitadas en cierta medida por el lenguaje de modelado. El nivel de conceptualización de los aspectos Finales está delimitado claramente por los **lenguajes de codificación OO y OA**, con la dependencia de la plataforma. Sin embargo, los Aspectos Medios y Aspectos Tempranos no están delimitados por un solo lenguaje específico para la creación de los modelados de software de su respectivo nivel conceptual. La conceptualización de Aspectos Medios está embebida en la actividad fundamental de Diseño, y por lo tanto, la totalidad de los modelos de este nivel serán desarrollados con un **lenguaje de diseño**. Finalmente, la conceptualización de Aspectos Tempranos es la más importante de modelar con el objetivo de ahorrar costes en el desarrollo de un producto software. Sin embargo, depende de varios factores para obtener la calidad prometida, entre ellos, los diferentes lenguajes necesarios para su cohesión: **Lenguajes de diseño OA** y **lenguajes de análisis OA**.

2.1.1 Lenguaje para modelos de Diseño OA

Entre los ingenieros de software existe consenso en que el diseño está relacionado con la tarea de especificar los modelos de un proyecto antes de construirlo. Por lo tanto, la actividad fundamental de Diseño tiene la necesidad permanente de un lenguaje para expresar los elementos de diseño con estructura y comportamiento. Así mismo, es necesario un lenguaje de modelado que soporte la conceptualización de **aspectos medios**.

Por otro lado, sabemos que un modelo es una simplificación de la realidad con el objetivo de capturar partes esenciales. En la actividad fundamental de diseño la abstracción desarrollada normalmente se plasma en una notación gráfica. Esto se conoce como modelado visual, permitiendo manejar la complejidad de los sistemas software con el requisito deseable de que el lenguaje de modelado sea independiente de la plataforma de implementación.

UML (Unified Modelling Language, UML) es el lenguaje gráfico más utilizado para especificar, visualizar, construir y documentar los artefactos de los sistemas software OO de forma precisa, proporcionando una forma estándar de escribir los planos de un producto software. UML es un método formal de modelado, permitiendo realizar una verificación y validación del modelo realizado. Además, se pueden automatizar determinados procesos y permite generar parte del código a partir de los modelos, así como realizar ingeniería inversa. Esto permite que el modelo de diseño y el código estén actualizados, con lo que siempre se puede mantener la visión de la estructura de un proyecto a más alto nivel. Por lo tanto, UML debería ser un lenguaje para modelados de diseño OA.

2.1.2 Modelado de Aspectos en diseño con UML 2.x

En una primera aproximación, se ha demostrado (Barra, Génova, & Llorens, 2004) que los nuevos bloques de construcción de UML 2.x tienen la suficiente semántica para modelar las propiedades de los conceptos abstractos de la OA.

2.1.2.1 Puerto

En investigaciones anteriores se ha determinado que un objetivo clave al modelar un producto software OA está en el detallado del concepto **Punto de Unión** (Stein, Hanenberg, & Unl, 2002). Las características del concepto abstracto punto de unión son solventadas con el sub-paquete **Puerto** incluido en UML 2.3, descrito en el capítulo de estructuras de composición (Object_Management_Group-OMG, 2010b)(p.185):

Un Puerto es una característica estructural de un clasificador que especifica un punto de interacción entre ese clasificador y su entorno o entre el clasificador y sus partes internas. Un Puerto puede especificar los servicios que un clasificador proporciona a su entorno así como los servicios que espera de su entorno. Las interfaces asociadas con un puerto especifican la naturaleza de las interacciones que pueden ocurrir sobre un puerto. Las interfaces requeridas de un puerto caracterizan las peticiones que pueden ser hechas desde el clasificador a su entorno a través de este puerto. Las interfaces proporcionadas de un puerto caracterizan peticiones para el clasificador que su entorno puede hacer a través de este puerto.

Las interfaces requeridas y proporcionadas de un puerto especifican todo lo que es necesario para las interacciones por medio de ese punto de interacción. Si todas las interacciones de un clasificador con su entorno son logradas por medio de los puertos, entonces la estructura interna del clasificador está totalmente aislada del entorno. Esto permite que un clasificador sea utilizado en cualquier contexto que satisfaga las obligaciones especificadas por sus puertos.

Los puertos permiten que las interfaces de un componente se dividan en paquetes discretos y que se utilicen de manera independiente. La independencia que proporcionan los puertos permite un grado mucho mayor de encapsulación y posibilidad de sustitución.

Un puerto de un clasificador es mostrado como un pequeño símbolo cuadrado y por defecto tiene visibilidad pública. El nombre del puerto es situado cerca del símbolo cuadrado. El nombre del clasificador junto con el nombre del puerto identifica de manera única un puerto específico en un clasificador específico para que sea utilizado por otros clasificadores.

Cuando el clasificador se desarrolla, se conectan puertos en sus partes interiores a sus puertos externos de acuerdo con las interfaces declaradas. Un puerto puede especificar cualquier petición que llegue a este puerto que sea manejada por el comportamiento de la instancia del clasificador poseedor. Las instancias de los puertos se crean y se destruyen junto con las instancias del clasificador al que pertenecen. Los puertos también pueden tener multiplicidad, indicando el posible número de instancias de un puerto particular dentro de una instancia del clasificador, es decir, un puerto en una instancia de un clasificador tiene un vector de instancias. Aunque todas las instancias de un puerto satisfacen la misma interfaz y aceptan el mismo tipo de solicitudes, pueden tener diferentes estados y valores para los datos. Por ejemplo, cada instancia en el vector puede tener un nivel de prioridad diferente.

2.1.2.2 Estructura interna

La distinción más obvia en un producto software OA es su conformación con aspectos. Es sabido que la composición de un aspecto es un conjunto de incumbencias trasversales (del inglés *crosscutting concerns*) con el mismo propósito. Por lo tanto, otro objetivo clave al modelar un producto software OA está en el detallado del concepto

incumbencia transversal, el cual es solventado con el sub-paquete EstructuraInterna en UML 2.3 (Object_Management_Group-OMG, 2010b)(p.167-197).

El sub-paquete EstructuraInterna (del inglés *InternalStructure*) proporciona mecanismos para la especificación de las estructuras de elementos interconectados que se crean dentro de una instancia de un clasificador contenedor. Una estructura de este tipo representa una descomposición de ese clasificador contenedor que se conoce como su **estructura interna**. Cuando una instancia de clasificador contenedor es creado, un conjunto de instancias correspondientes a sus propiedades se puede crear de forma inmediata o en algún momento posterior. Estas instancias son las instancias del clasificador contenedor determinando la propiedad (del inglés *property*). Una **propiedad** especifica que un grupo de instancias pueden existir; este conjunto de instancias es un subconjunto del conjunto total de las instancias especificadas por el clasificador contenedor determinando la propiedad.

La estructura interna de un clasificador contenedor contiene propiedades que pueden definirse en términos de **puertos**, **partes** y **conectores**. Cada uno de estos elementos se define dentro del contexto proporcionado por la definición de un **clasificador estructurado**. Un clasificador estructurado es una meta-clase abstracta que representa cualquier clasificador contenedor cuyo comportamiento puede ser total o parcialmente descrito por la colaboración de las instancias de su propiedad o de referencia.

Una **parte** es un pedazo estructural de un clasificador contenedor que declara que una instancia de este clasificador estructurado puede contener un conjunto de instancias por composición. Así mismo, una parte tiene un nombre, un tipo y una multiplicidad. Si la multiplicidad de la parte excede uno, una instancia del clasificador estructurado puede tener instancias múltiples de una parte pero cada instancia de la parte está separada. Si la multiplicidad esta ordenada, las instancias individuales pueden distinguirse por el índice de su posición dentro de la instancia. Cada parte puede tener un tipo, y el tipo restringe el tipo de las instancias que pueden ligarse a un rol.

En un **clasificador estructurado** cada parte es una instancia distinta en su propio contexto único. Pueden existir múltiples partes con el mismo tipo, donde cada uno tiene un conjunto de relaciones diferentes a diferentes partes. Una parte es una descripción de todas las instancias que pueden ligarse a la otra parte. Es necesario aclarar que todas estas instancias se destruyen cuando la instancia del clasificador contenedor se destruye.

En un clasificador estructurado las instancias de roles tienen relaciones contextuales que son representadas con **conectores**. Un conector sólo tiene significado dentro del contexto de un clasificador estructurado. Al contrario que una asociación, es una relación entre partes o roles, no entre clasificadores que son los tipos declarados de los roles. Un conector tiene una multiplicidad en cada extremo que indica cuántas instancias pueden estar conectadas a una sola instancia.

En algunos casos, un conector puede ser considerado como usos de una asociación general entre los clasificadores contenedores participantes; en ese caso, la asociación general se utiliza para un propósito particular. En otros casos, las conexiones entre las partes o roles no tienen validez fuera del contexto del clasificador estructurado. Es decir, si un conector no tiene una asociación explícita, entonces define una asociación implícita válida sólo dentro del clasificador estructurado.

Los conectores se pueden representar de dos formas. Si dos clasificadores contenedores se enlazan de manera explícita, ya sea directamente o a través de sus puertos, simplemente se dibuja una línea entre ellos o los puertos. Por otro lado, si dos clasificadores contenedores se conectan porque tienen interfaces compatibles, se puede

usar la notación de una junta circular para mostrar que no hay una relación inherente entre los clasificadores, aunque estén conectados dentro de este clasificador. Podemos colocar en este sitio a cualquier otro clasificador contenedor que satisfaga la interfaz.

Se puede conectar puertos internos a puertos externos del componente global. Esto se denomina **conector de delegación**, ya que los mensajes sobre el puerto externo son delegados al puerto interno. Esto se representa con una flecha desde el puerto interno al externo. Los conectores de delegación deben emparejar elementos de la misma polaridad, es decir, elementos requeridos con elementos requeridos o elementos proporcionados con elementos proporcionados. La interfaz de los elementos que reciben un mensaje delegado debe incluir los tipos que puede producir el elemento productor del mensaje delegado. Los conectores de delegación permiten la implementación de servicios de alto nivel mediante clasificadores de bajo nivel.

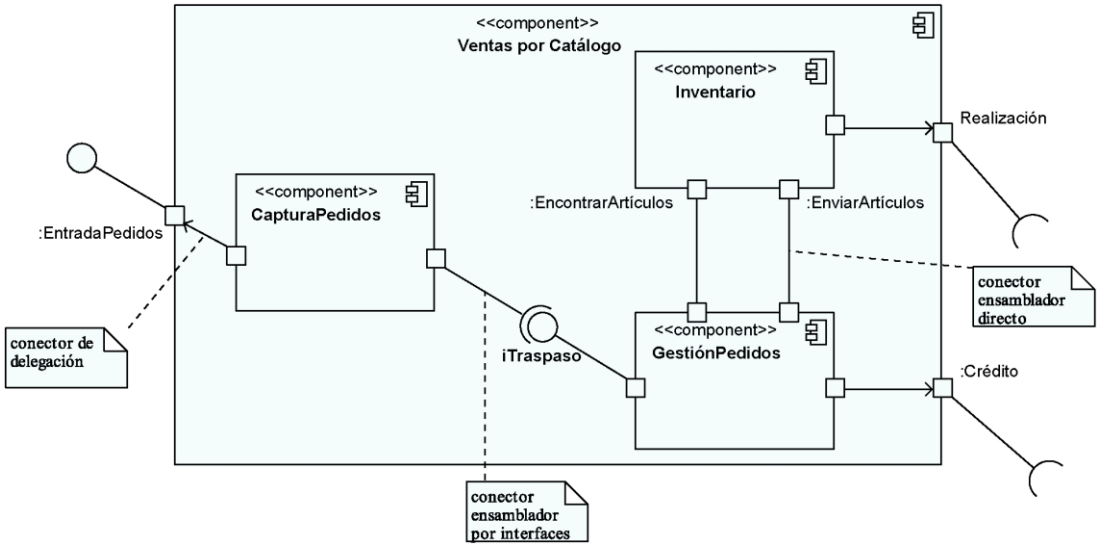


Figura 21 Ejemplo de clasificador estructurado de UML 2.x

Existe otro tipo de conector denominado **conector ensamblador**, el cual se refiere a un conector entre dos elementos en la especificación de la estructura interna del clasificador global estructurado.

La Figura 21 muestra un ejemplo de cómo un clasificador estructurado define la aplicación de un clasificador. Las interfaces describen lo que un clasificador debe hacer y su estructura interna describe cómo hace el trabajo a través de puertos, partes y conectores.

2.1.2.3 Organización de diseño

Los productos software de gran tamaño necesitan varios niveles de estructura hasta llegar a las clases que encapsulan el estado y comportamiento de una parte de la implementación del software. Durante las primeras etapas estas clases de implementación se encuentran definidas por sus propiedades externas. En este sentido, un sistema software se descompone en una colección de subsistemas organizados para lograr un propósito general. Recursivamente un subsistema es un elemento de alto nivel que puede expandirse en partes constituyentes. Un clasificador estructurado es un clasificador con partes internas conectadas dentro del contexto del clasificador contenedor. Un clasificador estructurado puede tener una frontera débil o fuerte, donde todas las comunicaciones tienen lugar sobre puntos de interacción bien definidos.

Una convención útil y recurrente es utilizar el clasificador contenedor de UML **componente** para modelar el diseño lógico. Los componentes pueden especificar o

describir desde el sistema software con sus elementos resultantes del segundo nivel de descomposición hasta el penúltimo nivel. En el último nivel de descomposición suele usarse el clasificador **clase** para designar los elementos de implementación, es decir, la representación simbólica de un fragmento de código en un lenguaje de programación OO. Es importante aclarar que el estándar de UML no implica mucha distinción semántica entre una clase y un componente.

En el estándar de UML 2.x los componentes representan una parte modular de un sistema software que encapsula sus contenidos lógicos y cuya manifestación es reemplazable dentro de su entorno. Un componente puede ser expresado por uno o más artefactos de despliegue, y a su vez, esos artefactos pueden ser desplegados para su entorno de ejecución.

Los componentes y artefactos son clasificadores, sin embargo, hay algunas diferencias significativas entre ellos. Los componentes representan abstracciones lógicas y los artefactos representan elementos físicos de los bits. Los artefactos representan el empaquetamiento físico sobre la plataforma de implementación. Los componentes y las clases que pueden componerlos pueden tener atributos y operaciones, sin embargo, los artefactos solo pueden implementarlos.

Ahora bien, un componente representa un elemento modular lógico y un artefacto representa un elemento modular físico. Sin embargo, la relación entre un artefacto y un componente o clase es una manifestación de implementación que convencionalmente es un componente por un artefacto de despliegue, en cuyo caso, muchas veces un componente representa un futuro artefacto de despliegue y por lo tanto es una representación física.

2.1.2.4 Comenzando el modelado de Aspectos con UML

Finalmente en esta sección se manifiesta una aproximación al modelado con UML 2.x a través de algunas directrices. Antes de continuar es necesario recordar que esta tesis doctoral está enfocada a las etapas tempranas, por lo tanto, de ahora en adelante únicamente se hablará de modelado del nivel conceptual de Aspecto Tempranos.

Dentro del modelado OA es observable la importancia de definir abstracciones precisas que nos permitan trabajar pensando en la vista lógica de un sistema software. El modelado lógico se hace para visualizar, especificar y documentar las decisiones acerca de las capas del dominio y sobre cómo colaboran estos elementos tanto estructuralmente como desde el punto de vista del comportamiento. Sin embargo, todo esto ha sido pensado sólo para las incumbencias que pueden ser encapsuladas claramente, por lo tanto, es recomendable complementar el modelo lógico con diagramas que representen las incumbencias trasversales. Para este objetivo se propone la directriz de utilizar el diagrama de **componentes estructurados** de UML 2.x especificando una clara separación entre componentes de modelado clásico y componentes estereotipados como <<aspecto>>, de manera que las relaciones de los componentes que encapsulan claramente incumbencias con los aspectos, se describan principalmente con las estructuras internas de cada componente.

Las clases y componentes son los bloques de construcción más importantes de cualquier sistema orientado a objetos. Estos clasificadores forman el vocabulario a diferentes niveles de abstracción del sistema software. Al realizar estas abstracciones, se observa que las entidades no se quedan aisladas, la mayoría se relacionan entre sí. El diseño de estas relaciones es una distribución equilibrada de responsabilidades entre las clases o componentes para modelar la estructura estática. Por lo tanto, se requiere modelar el comportamiento dinámico de las colaboraciones entre componentes clásicos o contra

componentes aspectuales, así como entre componentes aspectuales. Es necesario representar sociedades de instancias que desempeñan roles específicos que colaboren entre sí para desarrollar un comportamiento mayor que la suma de los comportamientos de sus elementos.

El comportamiento se especifica como flujo de control que puede incluir simples hilos de secuencia a través de un producto software, así como flujos más complejos que impliquen bifurcaciones, iteraciones, recursión y concurrencia. Estos flujos de control se modelan con iteraciones que documentan el comportamiento dinámico. Una iteración establece un escenario presentando todas las instancias que colaboran, incluyendo los mensajes y señales enviados, para realizar algún servicio. Para modelar estas interacciones UML propone dos tipos de diagramas: diagramas de comunicación y de secuencia. Un diagrama de comunicación es un diagrama de interacción que destaca la organización estructural de las instancias que envían y reciben mensajes; y un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes y señales enviados.

En el contexto de modelado de aspectos en diseño los diagramas de interacción aspectuales (secuencia y colaboración) son un arma importante de diseño de aspectos. Estos diagramas son el complemento necesario para describir las relaciones específicas entre componentes de aspecto y componentes clásicos o entre los propios componentes de aspectos.

2.1.3 Lenguaje para modelos de Análisis OA

En esta tesis doctoral, el modelado con UML 2.x está enfocado al concepto de diseño que está relacionado con la tarea de especificar los modelos de un proyecto antes de construirlo. Por lo tanto, los modelos de diseño pueden ser desde las primeras líneas de solución de arquitectura software que representan vistas lógicas independientes de la plataforma tecnológica, hasta las soluciones en los modelos de diseños detallados en vistas físicas dependientes de la plataforma tecnológica. Con este escenario, los Aspectos Medios contienen modelos de diseño en todo el nivel conceptual con las vistas físicas dependientes de la plataforma tecnológica. Así mismo, los **Aspectos Tempranos** (AT) contienen modelos de diseño a través de las primeras líneas de solución de Arquitectura software que representan vistas lógicas independientes de la plataforma tecnológica. Sin embargo, para el modelado de los AT también se utilizan los lenguajes para modelos de análisis OA. Por lo tanto, se requieren dos lenguajes con un mecanismo formal de mapeo entre ellos, proporcionando una alta cohesión de la semántica a lo largo del nivel conceptual de AT de un producto software.

Entre los ingenieros de software existe un consenso por el cual el análisis está relacionado con la tarea de especificar la solución a las necesidades para las que un producto software se desarrollará. La actividad fundamental de análisis tiene la necesidad de un lenguaje para expresar su especificación de elementos de análisis.

Los AT en la arquitectura tienen matices de diseño al representar vistas lógicas independientes de la plataforma tecnológica, motivo por el que se recomienda que los modelos visuales sean representados con UML. Ahora bien, los AT en la Ingeniería de Requisitos (IR) están basados en la actividad fundamental de análisis. El desarrollo de este proceso tiene como salida modelos que normalmente se plasman utilizando el lenguaje de los interesados en el producto software, que a menudo es el lenguaje natural. Sin embargo, entre los ingenieros analistas de software no existe un consenso en cómo especificar los requisitos con lenguaje natural (Weston et al., 2009). A pesar de todo, la actividad fundamental de análisis tiene la necesidad de representar conocimiento con

lenguaje natural. El principal problema en la representación del conocimiento es llegar a un consenso sobre qué representar y cómo hacerlo, implicando siempre una manifestación semántica (Fernández-López et al., 2004). Una representación del conocimiento es la creación de modelos de conocimiento en un lenguaje con una determinada organización, para facilitar así la extracción de conocimiento necesaria a determinadas incumbencias. En consecuencia, el lenguaje natural utilizado debe ser trasformado a un lenguaje formal de modelado a través de técnicas de ingeniería del conocimiento enfocadas a la especificación de requisitos aspectuales. Una propuesta aceptada para la gestión de semántica en el dominio de la OO es el uso de **ontologías** como modelado de conocimiento en la IR (Evermann & Wand, 2005).

2.1.3.1 Objetivo de las ontologías en la IR

En la etapa de análisis de un producto software, los analistas tienen dificultades para controlar la información que en un momento dado puede ser de utilidad en la especificación temprana del producto software, máxime cuando los cambios en la misma se suceden de forma frecuente. En tales circunstancias, se han ideado técnicas que nos ayudan a decidir qué información puede aportar conocimiento ante un problema. Sin embargo esta información se especifica en requisitos con lenguaje natural, y por lo tanto, no suele estar organizada formalmente. Ello provoca, sin duda, falta de precisión y exhaustividad en el modelado para su refinamiento posterior.

La especificación de requisitos con ontologías trata de solucionar estos problemas, donde las aplicaciones serán capaces de efectuar un procesamiento de la información mucho más profundo. Estas aplicaciones serán capaces de comprender el contenido semántico de los requisitos, y por tanto, de relacionar la información contenida en requisitos aislados y de deducir o inferir conocimiento no detectado previamente, tomando decisiones con un cierto grado de autonomía. Para que estas aplicaciones inteligentes sean posibles es necesario que la información de los requisitos esté organizada, es decir, perfectamente especificada y clasificada de manera que su significado exacto esté al alcance de las máquinas.

En consecuencia, las ontologías son el medio idóneo para lograr este objetivo, al facilitar la especificación formal de las entidades y conceptos presentes en los diferentes dominios, la jerarquía que les sustenta y las diferentes relaciones que los unen entre sí. Así garantizamos una representación formal legible por las máquinas, basada en un lenguaje común que puede ser compartido y utilizado por cualquier sistema de manera automática. Las ontologías no son formalismos cerrados y están sujetos a procesos evolutivos. Es por eso que metodologías y herramientas que soporten estos procesos se hacen esenciales. El proceso de desarrollo de ontologías debe tener el apoyo necesario tanto desde el punto de vista metodológico como de herramientas que lo faciliten.

3 Propuesta de un modelo conceptual de AT

Ha surgido en los últimos años una amplia gama de propuestas basadas en el paradigma OA con diferentes lenguajes y formas de modelado. Cada una de estas propuestas suelen centrarse en una etapa específica, en conjunto cubren desde las etapas finales hasta las etapas tempranas de un desarrollo software. Sin embargo, existen variaciones de los conceptos de la OA a lo largo de un DSOA completo. En consecuencia esto hace más compleja la trazabilidad entre las diferentes propuestas y en su transformación entre etapas.

En este escenario, se observa la necesidad de un proceso de DSAO cohesivo que impulse la transformación de modelos elaborados en una cierta etapa de desarrollo a los

modelos de la siguiente etapa, hasta alcanzar un producto software final. Para conseguir este objetivo es necesaria la aceptación de un metamodelo que respete los conceptos y relaciones básicas del paradigma OA en cada una de las etapas de un DSOA. En busca de un metamodelo para el DSOA, en el estado de la cuestión se estudiaron diferentes propuestas que respetaban la terminología común de la AOSD-Europe (Van den Berg et al., 2005), encontrando en este estudio que el principal problema observado en el estudio de todas las propuestas de metamodelos para el DSOA es el descartarte del modelado de las etapas tempranas bajo el paradigma OA. A este respecto, anteriormente se mencionó que debido a la gran cantidad de investigación y desarrollo necesario para aplicar la hipótesis inicial a lo largo de todo ciclo de vida de un producto software en un DSOA el ámbito de esta tesis doctoral se centró en el nivel de conceptualización de los Aspectos Tempranos (AT). Esta conceptualización es una de las tres contenidas en la propuesta de conceptualización de aspectos a lo largo de cualquier PDS desarrollada en párrafos anteriores. Estas conceptualizaciones permiten centrarse en las llamadas trasformaciones horizontales en los PDS; es decir, las transformaciones de los modelos en el mismo nivel de conceptualización. Así mismo, en este marco de referencia se ha expuesto que los aspectos tempranos están presentes desde la especificación de requisitos hasta la descripción de las primeras líneas de solución de la arquitectura software representadas por vistas lógicas independientes de la plataforma tecnológica. En el contexto de MDA los AT se encuentran en las etapas abstractas de CIM y PIM.

Teniendo en consideración lo anterior, se encontró que las diferentes propuestas estudiadas de metamodelos para un DSOA no proporcionaban los medios para el modelado efectivo de AT. Es decir, no procuran la transformación de sus modelos a la siguiente etapa ni permiten la gestión de su trazabilidad. Por lo tanto, era necesario el desarrollo de un marco de referencia para el modelado de AT.

Ahora bien, un marco de referencia para AT requiere el desarrollo de un modelo conceptual. El desarrollo del **modelo conceptual de Aspectos Tempranos** debe considerar una selección de los elementos básicos del paradigma OA. En este sentido, se encontró que en las propuestas de metamodelos estudiadas realizaron un esfuerzo en el análisis de los elementos importantes del paradigma OA en las etapas de codificación y diseño. A este respecto, este trabajo aporta las bases para trabajar en los elementos que siempre deberían estar presentes en las etapas tempranas del ciclo de vida del producto software con el objetivo de establecer criterios unificados pertenecientes al contenido y uso. Aunado a estos elementos básicos es necesario añadir los conceptos básicos relacionados con la arquitectura software lógica complemento de los AT.

Anteriormente (Barra et al., 2007) se ha demostrado que la simbiosis de los conceptos de *Vista* expuesto por el estándar IEEE 1471 (IEEE-Computer-Society, 2000) y AT es provechosa para los DSOA. En esta simbiosis aparecen los elementos a considerar en el modelado de AT soportados en una recomendación consensuada para el desarrollo de una descripción arquitectónica de software.

Unido a los elementos antes mencionados otra idea conceptual que recomienda añadir esta tesis doctoral son los elementos que eviten la tendencia de modelar con una única dimensión. Es decir, proporcionar los medios para tener de manera simultánea diferentes dimensiones de AT.

El resultado fue la propuesta de un **modelo conceptual de aspectos tempranos** que proporciona un marco de trabajo para el modelado de incumbencias en las etapas tempranas que facilita la transformación del modelado temprano a la siguiente etapa de un DSOA.

En la Figura 22 se plasma el modelo recomendado para la especificación del nivel de abstracción de Aspectos Tempranos.

3.1 Aspectos Tempranos en contexto

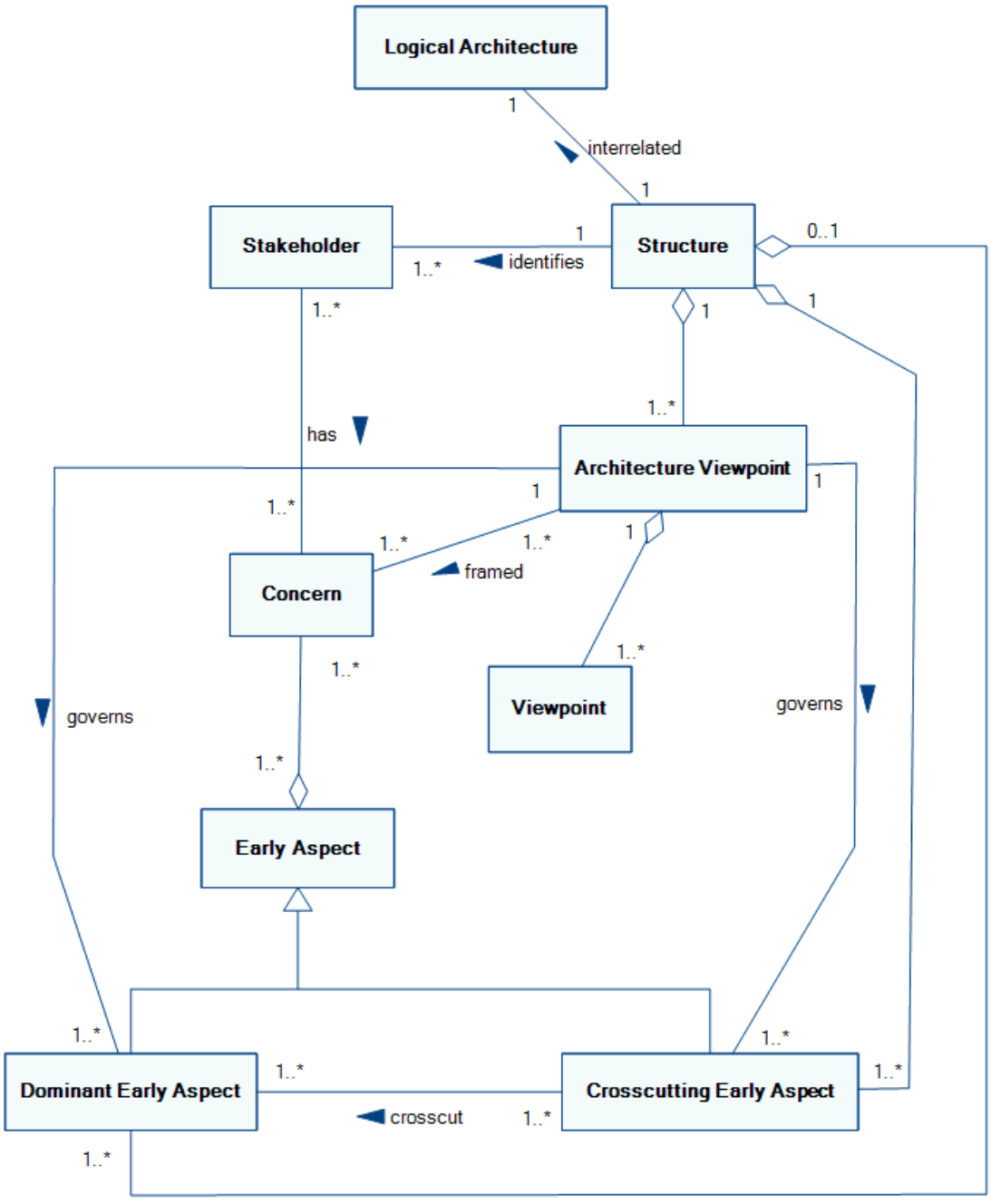


Figura 22 Modelo de referencia de Aspectos Tempranos

En el modelo que se propone, el término **Arquitectura Lógica** representa las vistas lógicas independientes de la plataforma tecnológica. La conformación de las vistas o aspectos de la arquitectura lógica está basada en una serie de abstracciones clave para la completitud estructural. El conjunto de principios para la selección de dichas abstracciones es aportado por diferentes estilos arquitectónicos obtenidos de forma heurística a lo largo de la historia de la ingeniería de software. Su principal representación es una colección de paquetes, capas o componentes organizados con posibilidades articulatorias en sus interrelaciones. Así mismo, busca agrupar entidades

del mismo tipo con el objetivo de hacer homogéneos los trabajos de desarrollo del producto software.

Una Arquitectura Lógica está interrelacionada con una Estructura. La **Estructura** es una especificación de requisitos compuesta de **Aspectos Tempranos Dominantes** y **Transversales**. En los términos de este modelo recomendado, un aspecto temprano es una agregación de incumbencias. Al mismo tiempo los Aspectos Tempranos Dominantes y Transversales son tipos del término más general denominado **Aspecto Temprano**. Las incumbencias son consideraciones específicas o requisitos que deben ser tomados en cuenta para que los propósitos de un producto software se cumplan.

Por otro lado, las **partes interesadas en un sistema** (del inglés *stakeholders*) tienen incumbencias. En este modelo, una incumbencia podría ser de interés para uno o más interesados en el sistema y un interesado en el sistema puede interesarse por una o más incumbencias. Existe una relación indirecta entre **estructura** e **incumbencia**, dado que la estructura identifica a los interesados en el sistema, la Arquitectura de Puntos de Vista enmarca las incumbencias y la estructura tiene agregada la arquitectura de puntos de vista.

La **Arquitectura de Puntos de Vista** son Puntos de Vista organizados que establecen los convenios específicos para la creación, organización y análisis de la estructura. Es decir, la arquitectura de puntos de vista debe proporcionar una serie de conceptos y reglas estructurales propias para conseguir la abstracción de Aspectos Tempranos. Las entidades de Aspecto Temprano Dominantes y Trasversales son gobernados por la arquitectura de puntos de vista. Por lo tanto, los Puntos de Vista nos permiten realizar los modelados de Aspectos Tempranos, concentrándonos en las incumbencias relevantes del sistema y abstrayéndose del resto.

4 Arquitectura de Puntos de Vista

En un DSOA, la estructura de aspectos tiene diferentes intereses para diferentes interesados en el sistema, así como restricciones del entorno. Satisfacer a todos puede conllevar una gran cantidad de Puntos de Vista para la creación de Aspectos. Es decir, uno de los mayores problemas en la creación de productos software con DSOA es cuando nos dedicamos a añadir aspectos de forma descontrolada. El resultado suele ser un producto software demasiado complejo, poco claro y difícil de evolucionar. La gran mayoría de ellos no son más completos porque posean un mayor número de aspectos. Los mejores productos software son los que tienen identificados los aspectos que resuelven mejor las necesidades de variabilidad. Una Arquitectura de Puntos de Vista eficaz permitiría identificar los aspectos que apoyarían una evolución del producto software.

En la ingeniería de requisitos, las propuestas basadas en el enfoque de Puntos de Vista han sido concebidas para estructurar y organizar los requisitos. El punto clave del análisis orientado al Punto de Vista es que toma en cuenta la existencia de varias perspectivas para especificar los requisitos, con el objetivo de reducir la complejidad y mejorar el entendimiento grupal de los requisitos.

En esta línea, la propuesta de modelo conceptual de AT expuesta en la sección anterior 3.1, incluye el concepto Arquitectura de Puntos de Vista (APV). Esta es la conceptualización más significativa en la especificación de requisitos y arquitectura temprana.

En investigaciones anteriores (Kotonya & Sommerville, 1992) se ha demostrado cómo una APV apoya la integración y análisis de los requisitos. Algunos (Ross & Schoman,

1977) se han centrado en proponer tres puntos de vista comunes en todos los requisitos: el contexto de análisis, las especificaciones funcionales y limitaciones del diseño, obteniendo homogeneidad. Otros (Nuseibeh et al., 1994) han propuesto una APV con grandes resultados en la identificación de conflictos pero específica para un sistema en concreto. Por lo tanto, se requiere un APV que facilite la integración evitando conflictos con una articulación relativa de los requisitos. Es decir, una arquitectura de diferentes puntos de vista que puedan agrupar uno o más requisitos y que los requisitos puedan estar en uno o más puntos de vista. Así es posible identificar claramente los aspectos necesarios a observar en cualquier DSOA.

En esta tesis doctoral, se ha propuesto una APV como una guía clara para desarrollar con la mayor completitud el nivel de conceptualización de aspectos tempranos. Para obtener esta u otra APV específica se desarrolló un método descrito en el capítulo de experimentación sección 2.2. El desarrollo del método se realizó a través de experimentaciones con un corpus de de 6337 requisitos. El corpus es una recopilación de 1658 en PYMES[31], 754 en proyectos de fin de carrera y 3925 en proyectos académicos. Entre los proyectos académicos se distinguen los realizados por analistas junior (estudiantes de ingeniería informática en último curso) y analistas sénior (profesores docentes investigadores).

En el proceso de creación del método para la obtención de una APV específica se encontró la necesidad de realizar un proceso de grano grueso a grano fino para proporcionar eficacia en la APV creada. En este sentido, el análisis del corpus de requisitos con grano grueso demostró que los Puntos de Vista creados usualmente pertenecen a conjuntos delimitados que por su naturaleza deben considerarse aisladamente; estas agrupaciones se han llamado **Ámbitos**.

Como se ha comentado, se ha encontrado a través de un análisis heurístico de grano grueso que en un DSOA los Aspectos pertenecen a un pequeño grupo de ámbitos generales. En su forma básica las incumbencias de la especificación de un producto software siempre se pueden agrupar en tres grandes temas: dominio, entorno operacional y producto software. Sin embargo, estos ámbitos son ambiguos en la definición de sus límites y de las características de las incumbencias que agrupan.

En este sentido, se observó que la agrupación de los aspectos del dominio siempre es limitado por los analistas responsables; en este sentido es un ámbito de domino específico que debe ser justificado. Al mismo tiempo muchos de los aspectos creados por los ingenieros del software no se refieren al producto software y tampoco al entorno de los roles del presente o futuro producto. En este sentido, los aspectos se refieren al dominio específico de aplicación, es decir, esa parte de la realidad que posiblemente afectará al producto software y a su vez será afectada por dicho producto software. Así mismo, típicamente el entorno se adecúa a un posible producto software de forma ambigua, en donde convive la descripción del problema con la abstracción de la solución sin especificar. Aunado a todo esto se observa que los aspectos pueden modelar el producto software, de forma parcial o completa. Es importante contemplar que pueden tratarse de aspectos de un producto existente o de uno en desarrollo.

No obstante, estas agrupaciones fueron la base para la definición de los tres ámbitos generales propuestos: **Ámbito del dominio específico**, **Ámbito del entorno operacional** y **Ámbito del producto software.**

---

[31] Pequeña y mediana empresa

A este respecto, cada uno de estos Ámbitos permite identificar un subconjunto de **dimensiones básicas** que sólo pueden pertenecer a un ámbito específico pero con un comportamiento complementario entre ellas con el cual se pueden clasificar los aspectos (ver Figura 23).

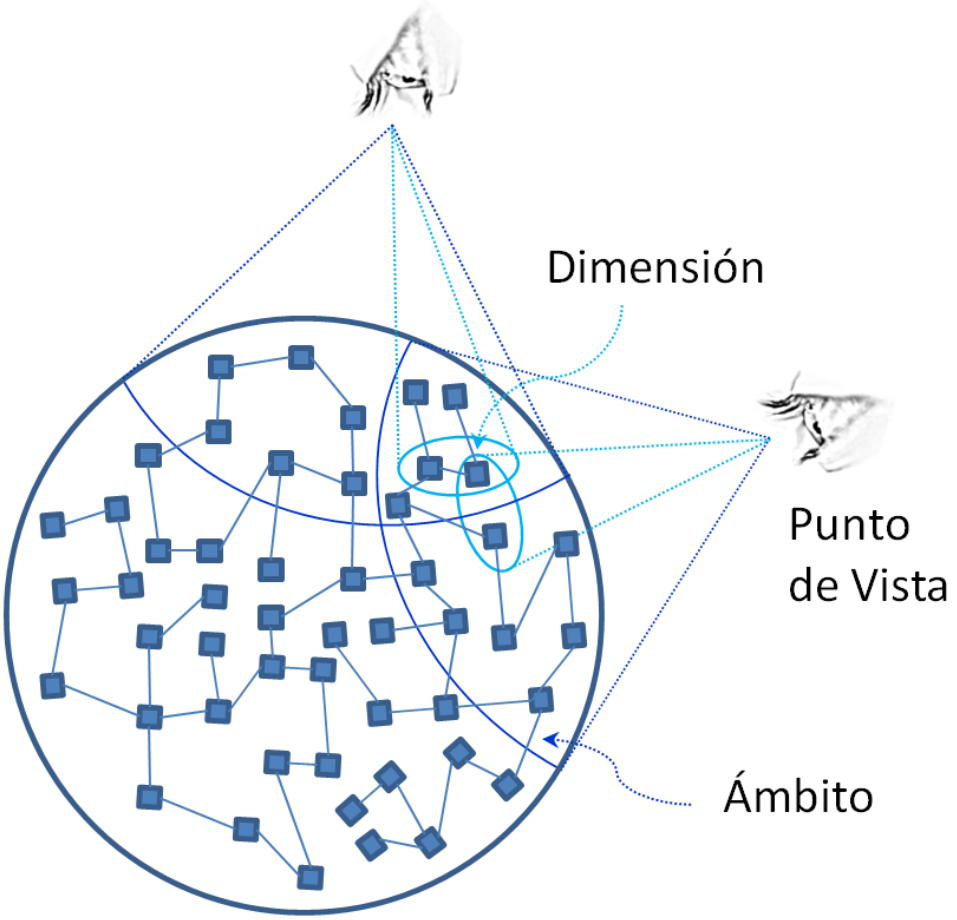


Figura 23 Ámbito y dimensión del punto de vista

4.1 Ámbito del dominio específico

La especificación inicial de un producto software obtiene fuerza semántica con el uso de conceptos estructurados que pertenecen a un domino específico. Es decir, un modelado de dominio específico podría ser la base para la representación de un dominio de aplicación de una familia de productos software relacionados (Parnas, 1976).

En este contexto, algunos aspectos proveen un vocabulario de conceptos y sus relaciones acerca de las actividades que toman lugar en ese dominio, así como reglas y principios elementales que gobiernan ese dominio. Estos Aspectos pertenecen al Ámbito del dominio específico en la **dimensión de la información de dominio**.

En el mundo de las organizaciones todo producto software forma o formará parte de un proceso de negocio. Algunos investigadores apuntan (Kent, 2002) que desarrollar con éxito un producto software resulta difícil si no se define previamente el proceso de negocio del que formará parte el software que se pretende construir. Por lo tanto, resulta necesario realizar un modelo del proceso de negocio actual o futuro. En este punto es importante aclarar que el modelo del proceso del negocio es una herramienta para definir qué servicio se puede informatizar o qué nuevo servicio se puede ofrecer al

automatizar la información. Estos aspectos son del Ámbito del dominio específico en la **dimensión del proceso de negocio**.

4.2 Ámbito del entorno operacional

Es sabido que un producto software debe interactuar con un conjunto de elementos con propiedades que no forman parte del producto. Sin embargo, un cambio en uno de estos elementos puede inducir o producir cambios en el sistema. Por consiguiente todos los elementos que pueden condicionar el futuro producto software conforman el entorno operacional.

El producto software que se desea desarrollar es el que promete informatizar servicios plasmados en el proceso de negocio. Este es el motivo por el cual típicamente el modelado del producto software es identificado como la solución a un problema de gestión de información en una determinada organización. En este punto podemos decir que se ejemplifica en análisis un modelo **problema-solución**. El modelo de proceso de negocio implica de forma plausible los elementos del entorno de aplicación a un futuro producto software. En el contexto del problema-solución es deseable especificar un entorno operacional de aplicación a medida para el producto software. El modelado debe visualizar el producto software como una o varias cajas negras que interactúan con el entorno. Por lo tanto, los aspectos que surgen de este punto de vista son del ámbito del entorno operacional en la **dimensión del entorno de aplicación**.

El entorno de aplicación modela los elementos informáticos existentes externos al producto software dejando clara la existencia de un punto de vista **información de entorno**. Sin embargo, existen requisitos que describen cómo se comunican los elementos informáticos del entorno. Por lo tanto, se propuso el punto de vista de **comunicación externa** para agrupar los requisitos que especifican las redes y protocolos de comunicación utilizados para relacionar la información del entorno de aplicación.

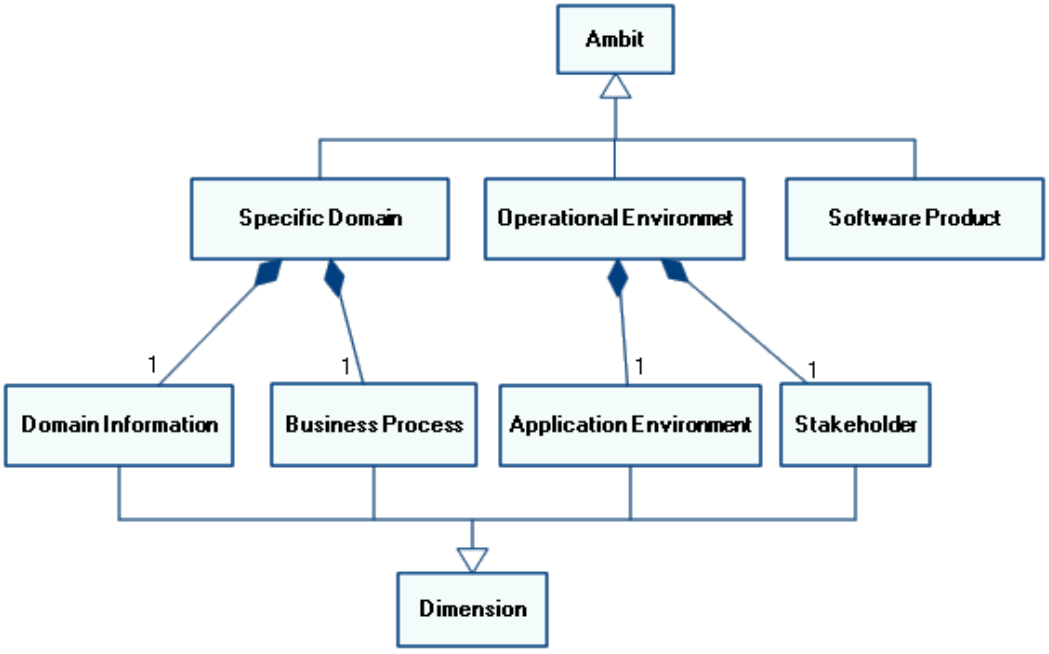


Figura 24 Ámbitos y dimensiones

El entorno también está compuesto por interesados que pueden afectar o son afectados por un sistema. Desde el punto de vista del entorno específico un interesado (del inglés

stakeholder) es aquella persona que está involucrada en la realización de un producto software, haciendo posible el mismo ya sea por tener poder de decisión en su creación y desarrollo o bien por su capacidad de convertirse en usuario del mismo. En este sentido, una buena planificación de sistemas debe involucrar la identificación y clasificación de los interesados. Posiblemente estos aspectos son los más simples y fáciles de identificar pero necesarios en el modelado del conocimiento de un producto software. Así, estos aspectos surgen de puntos de vista del ámbito del entorno específico en la **dimensión de stakeholder.**

La situación de los ámbitos y dimensiones hasta ahora mencionadas puede observarse en la Figura 24. Debe tenerse en cuenta que en la siguiente sección 4.3 se explicará el ámbito del producto software.

4.3 Ámbito del producto software

Las dimensiones del ámbito del dominio específico guían la creación de los modelos de las dimensiones del ámbito del entorno operacional. Al mismo tiempo surge una interrelación entre estos ámbitos que en conjunto guían el diseño de los modelos del ámbito del producto software. Por lo tanto, en los ámbitos de dominio específico y entorno operacional aparecen puntos de vista bajo los cuales se crearán aspectos que contendrán las incumbencias que justifican las decisiones del diseño del producto software.

Por otro lado, en el campo académico las etapas tempranas se identifican como la definición del problema. Sin embargo, como se ha visto en anteriores párrafos, también contienen las primeras líneas de la solución. Estas especificaciones se reflejan con mayor naturalidad en la especificación o descripción de requisitos del ámbito del producto software.

Al realizar el análisis de un producto software normalmente se inicia con la descripción y especificaciones de sesiones de interacciones entre el producto software y los agentes externos. Es decir, describir lo que el producto software debe ser capaz de llevar a cabo en su entorno operacional. Esta actividad, arroja un enorme número de ejecuciones potenciales. Sin embargo, la capacidad de un producto software se puede concebir en términos de un número relativamente pequeño de escenarios típicos. La descripción de dichas sesiones de interacciones o escenarios está compuesta por grupos de incumbencias con una **alta cohesión semántica**. Por lo tanto, de las descripciones de estas sesiones se extraen dimensiones dominantes para visualizar los aspectos dominantes del ámbito del producto software.

La primera dimensión dominante identificada es la **dimensión de servicios.** Esta dimensión tiene el objetivo de identificar puntos de vista para modelar los servicios que proporcionan un resultado valioso para el usuario. Así mismo, es característica la modificación de datos permanentes a lo largo de la sesión de interacciones de los servicios. Es decir, los aspectos de la dimensión de servicios modelan las incumbencias dominantes que al ser realizadas modifican el estado del sistema.

Otros servicios que puede proporcionar un producto software son los que no modifican su estado pero al igual que los aspectos dominantes de los puntos de vista de la dimensión de servicios proporcionan un resultado valioso para el usuario. Los aspectos dominantes resultantes de estas incumbencias dominantes se encuentran en la dimensión identificada como **dimensión de consulta**.

Hay un tipo de servicios que son modelados bajo puntos de vista específicos en aspectos dominantes que agrupan incumbencias dominantes básicas. Dichas

incumbencias están enfocadas a modificar datos permanentes del sistema y por tanto a cambiar su estado. Los aspectos que surgen de estos puntos de vista son de la **dimensión de gestión básica**.

Los aspectos de la dimensión de servicio, dimensión de consulta y dimensión de gestión básica modelan las incumbencias que satisfacen los servicios esperados de un producto software. Los aspectos dominantes son el objetivo principal para un DSOA.

Sin embargo, para que una factoría de software se comprometa con la producción de un producto software es necesario que se establezcan las necesidades sobre cómo cumplir dicho objetivo. Es decir, qué aspectos trasversales son imprescindibles en el producto software. Por lo tanto, es necesario definir las dimensiones de este ámbito. Las dimensiones a tomar en cuenta en el ámbito del producto software para la creación de aspectos transversales se mencionan en los siguientes párrafos.

El inmediato objetivo relevante es la identificación de las entidades conceptuales con información que las incumbencias dominantes necesitan para satisfacer las necesidades de información de los servicios prometidos. Por lo tanto, los aspectos creados en la búsqueda de este objetivo están compuestos por incumbencias transversales. Estos aspectos se modelan con los puntos de vista encontrados en la **dimensión de información**.

La estructura de información temprana de un producto software la definen los atributos. Éstos en las etapas finales contendrán datos con un determinado formato que suelen ser predefinidos en las etapas tempranas. Los datos tienen la necesidad de respetar siempre el mismo formato, tanto en las diferentes entidades en tiempo de ejecución, como en las bases de datos almacenados en memoria auxiliar. En este contexto, parece clara la necesidad de la **dimensión de formato de datos**.

Actualmente los productos software con Interfaces Gráficas de Usuario (del inglés *Graphical User Interface-GUI*) han evolucionado con grandes consecuencias para la industria del software. Su principal misión consiste en proporcionar un entorno intuitivo para permitir la interacción de los usuarios con el producto software y el sistema operativo. Dicha interacción se basa principalmente en formas visuales que representan funciones, acciones e información. Sin embargo, cumplir con dicha misión llega a monopolizar un gran porcentaje del código y la utilización de muchos recursos. Dada la situación descrita, en el desarrollo de un producto software es necesario especificar aspectos transversales en la **dimensión de la interfaz de usuario**.

Existen incumbencias trasversales que especifican los valores numéricos de las variables medibles de rendimiento, tales como frecuencia de acceso, capacidad y velocidad de proceso. Los aspectos que surgen en este ámbito del producto software pertenecen a la **dimensión de rendimiento**.

En un producto software siempre es necesario proteger el sistema contra las amenazas a la confidencialidad, integridad y disponibilidad. Estas suelen ser incumbencias transversales que especifican el nivel y la frecuencia de acceso permitido a los usuarios autorizados del software, el nivel y medios para la prevención contra el uso no autorizado del software, el nivel de protección física de los datos del ordenador a través programas relacionados con copias de seguridad. Por lo tanto, es necesario especificar los aspectos transversales desde el punto de vista de la seguridad en la **dimensión de seguridad**.

Normalmente se espera que un producto software implementado tenga fallos de ejecución debido a carencias en su construcción o errores internos. Sin embargo, éstos no deben existir, por lo tanto no se deben cubrir los errores con un código interminable

de manejo de fallos. Así mismo, también se esperan errores del entorno que pueden generar fallos en el nuevo producto software. Sin embargo, en principio estos errores no son controlables pero es factible intentar gestionar los fallos críticos. Este tipo de errores suelen corregirse en el nivel conceptual de aspectos finales, sin embargo, es importante especificar cómo debe responder la aplicación a los errores del sistema desde el nivel conceptual de aspectos tempranos. Los responsables en esta etapa son los actores que interaccionan con el producto software. Por lo tanto, el objetivo es ayudar a asegurar la calidad de la aplicación, atrapando e identificado los errores lógicos potenciales. En consecuencia, es necesario especificar los aspectos desde el punto de vista del manejo de fallos en la **dimensión de gestión de fallos**.

Una dimensión que es necesario tomar en cuenta es la **dimensión de restricciones de diseño.** En esta dimensión los aspectos contienen incumbencias transversales que especifican criterios que pueden restringir el diseño o la implementación. En esta dimensión ha sido posible identificar puntos de vista de bajo nivel. Entre los más importantes guiados por las propuestas de los expertos en el dominio tenemos los puntos de vista donde se agrupan los requisitos que especifican características del software a utilizar. Este punto de vista es llamado **software necesario**, donde debe existir compatibilidad entre sistema operativo, sistemas de gestión de bases de datos, lenguaje de programación, etc. Otro punto de vista de bajo nivel consensuado es el de **portabilidad** del futuro producto software, donde se especifican las necesidades para mover la aplicación de un ambiente a otro.

La dimensión de restricciones de diseño es un contenedor de puntos de puntos de vista de mayor subjetividad pero necesarios. Los puntos de vista propuestos en esta dimensión requieren verificar su aportación objetiva a la creación del diseño para su propuesta en una APV.

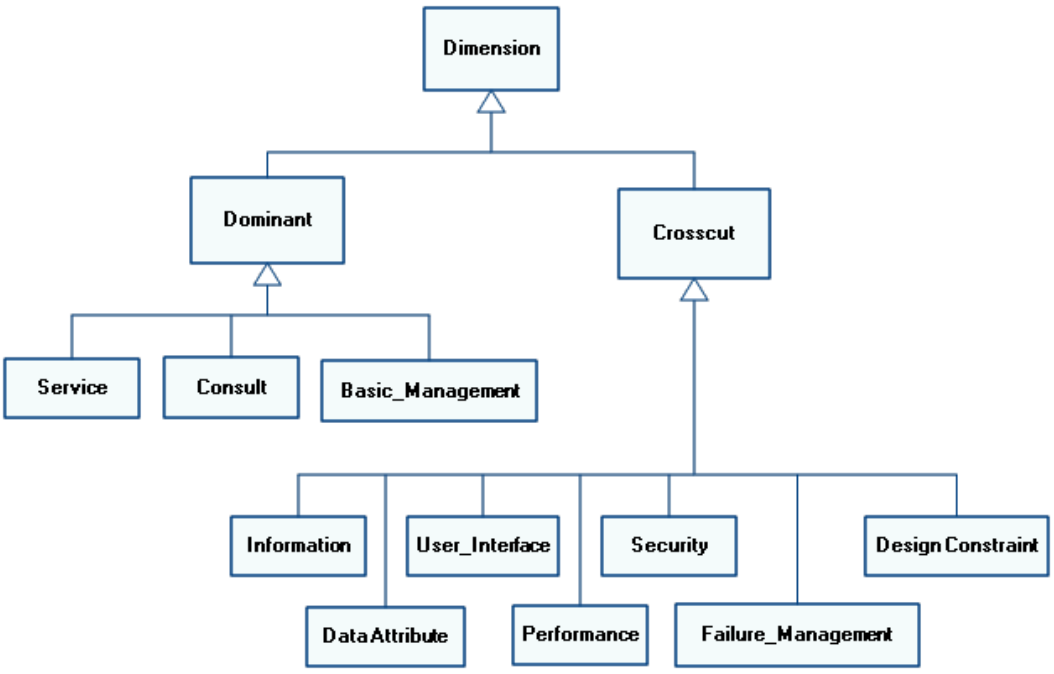


Figura 25 Dimensiones del ámbito del producto software

En resumen, en el ámbito del producto software se ha identificado un esquema de diez dimensiones básicas en las que se pueden clasificar los aspectos (ver Figura 25). Estas diez dimensiones podrán contener más de un punto de vista para la creación de aspectos transversales claramente justificados.

4.4 Dimensiones de Aspectos clásicos

Las incumbencias fueron desgranadas en el análisis del corpus de requisitos de esta tesis doctoral. A partir de las incumbencias extraídas se crearon los aspectos con los que se identificaron las dimensiones en las secciones anteriores. Sin embargo, existen dimensiones clásicas a considerar dada la recurrencia en la creación de aspectos.

Inicialmente los avances del paradigma OA se lograron a través de lenguajes OA que soportan únicamente aspectos para los que fueron diseñados. En la tesis de C. Lopes (Lopes & Kiczales, 1997) se propusieron los dos primeros lenguajes OA de dominio específico, COOL y RIDL, para especificar los **aspectos de sincronización** y **distribución** respectivamente, motivo por el cual dichos aspectos son los más señalados en la literatura genérica relacionada con la OA.

En un DSOA los aspectos finales son el resultado de la trasformación de aspectos medios y estos a su vez de aspectos tempranos. Sin embargo, no todas las incumbencias de los aspectos finales son descendientes de incumbencias de aspectos medios o las incumbencias de los aspectos medios descienden de las incumbencias de aspectos tempranos. Existen incumbencias que se van añadiendo como nuevas en cada una de las conceptualizaciones aspectuales. En el caso de los dimensiones de sincronización y distribución la mayoría de sus incumbencias aparecen en los aspectos medios y finales. Es decir, las incumbencias a cumplir en estos aspectos son por criterios implícitos, de diseño o derivados de la plataforma elegida.

Anteriormente se ha dicho que los aspectos medios se especifican en estructuras aspectuales al nivel del diseño con vistas físicas dependientes de la plataforma tecnológica y es claro que los aspectos finales sólo se modelan con una tecnología determinada. Dichos aspectos están relacionados con las propiedades que proporciona su especificación para llegar a concretar su implementación con tecnología en un producto software final. Podríamos decir que son aspectos de bajo nivel los que contienen características más cercanas a la capacidad de interpretación del ordenador. En oposición llamaremos aspectos de alto nivel a los que expresan las incumbencias de una manera más cercana a la capacidad cognitiva humana. En este sentido, todos los aspectos de alto nivel llegan a generar aspectos de bajo nivel, pero existen aspectos que se crean sólo en bajo nivel sin tener traza directa con ningún artefacto de las etapas tempranas. Así, los aspectos clásicos llevan tiempo en estudio, buscando diferentes técnicas y metodologías para resolver sus necesidades en las etapas finales, obteniendo soluciones aceptables. Sin embargo, esto no prohíbe proponer nuevas soluciones a dichos aspectos con independencia de la tecnología.

4.5 Ámbito de la arquitectura lógica

Los ámbitos de dominio específico, entorno operacional y producto software se centran en ofrecer facilidades para especificar el concepto **estructura** del modelo de referencia de AT (ver Figura 22). En este modelo, el concepto **arquitectura lógica** está vinculado estrechamente con la **estructura** con un objetivo complementario. La arquitectura lógica es la parte de la arquitectura contenida en los AT que en principio modela componentes lógicos del futuro código software. Sin embargo, el desarrollo de un sistema es mucho más que un problema software, hay que conocer el mundo físico, en el cual existen elementos móviles que pueden fallar, en los que las señales tienen ruido y el comportamiento no es lineal. En esta línea, a veces existen componentes de alto nivel que son lógicos, pero que toman en cuenta la localización física en la topología hardware; por ejemplo un dispositivo móvil contiene software para la utilización del sistema. Por lo tanto, es necesario facilitar la comunicación entre los ingenieros del

hardware del proyecto y de los desarrolladores de software. Así mismo, se requiere crear modelos comprensibles para ambos grupos de personas para la toma de decisiones de ingeniería del sistema. Los modelos resultantes son útiles para razonar acerca de los compromisos entre el hardware y el software. En resumen, un sistema requiere modelar aspectos hardware a un nivel suficiente para que un ingeniero de software pueda especificar la topología de procesadores y dispositivos sobre los que se ejecutará el software del sistema.

En este contexto se han propuesto para el modelado de las vistas lógicas de la arquitectura el **ámbito arquitectura lógica** con las dimensiones **hardware** y **software**.

En el ámbito del producto software dentro de la dimensión de restricción de diseño, es posible agrupar en el tema *recursos hardware* los requisitos que especifiquen los equipos informáticos disponibles y necesarios donde se ejecutará el software a desarrollar. Sin embargo, este posible punto de vista de recursos hardware identifica los conceptos de la topología. En esta línea, se considera que dicho punto de vista aporta mayor semántica al relacionarlo con el modelado de la arquitectura lógica. Por lo tanto, se ha considerado plasmarlo en la dimensión hardware del ámbito de la arquitectura lógica.

El punto de vista de **recursos hardware** modela los elementos físicos de procesamiento considerados del sistema; sin embargo, en el proceso de especificar o describir una topología hardware se debe considerar cómo interactúan o se comunican. Estas relaciones entre elementos considerados del sistema están descritas en los requisitos que especifican las redes y los protocolos de red utilizados en la topología hardware. Por lo tanto, es necesario agrupar estos requisitos dentro del punto de vista de bajo nivel de **comunicación interna** en la dimensión arquitectura lógica.

Se han identificado las dimensiones de hardware con sus respectivos puntos de vista; sin embargo, para la identificación de los puntos de vista de la dimensión software es necesario saber otra serie de factores relacionados con el futuro producto software. Por lo tanto, el modelado de los puntos de vista de la dimensión de software será guiado a través de estilos arquitectónicos.

5 Modelado de la Arquitectura Lógica

En la sección 1.2.1.3 de este capítulo se explicaron los límites entre aspectos tempranos y aspectos medios con el apoyo del análisis de la dicotomía de la arquitectura software en estilos arquitectónicos y patrones arquitectónicos, enmarcando las propuestas de estilos arquitectónicos en la creación de la arquitectura lógica independiente de la tecnología. Ahora bien, a continuación se realiza un análisis de los estilos arquitectónicos para apoyar esta idea.

Usualmente un estilo arquitectónico se selecciona de un catálogo definido por la organización que los aplicará. Las restricciones del estilo arquitectónico seleccionado guían la forma de modelar las agrupaciones lógicas y sus relaciones con el fin de cumplir con los servicios ofrecidos.

En el área académica la selección de un estilo arquitectónico no tiene solo un valor distintivo sino que define una situación pragmática. Así mismo, se dice que los estilos a utilizar tienen que tener una dimensión semánticamente precisa y diferencial, asociando experiencia y antecedentes específicos. Sin embargo, en las factorías de productos software diferentes estilos arquitectónicos podrían servir para modelar una misma arquitectura lógica. Del mismo modo es permitido mezclar los estilos arquitectónicos al momento de aplicarlos al desarrollo de un producto software.

Por otro lado, la idea de estilo arquitectónico ha sido uno de los conceptos mejor aceptada entre los arquitectos de software. Sin embargo, se manifiestan discrepancias cuando se trata de enumerar y clasificar los estilos conocidos. Por ejemplo, el estilo cliente-servidor ha sido clasificado como estilo de flujo de datos (Shaw & Clements, 1997), como arquitectura distribuida (Garlan & Shaw, 1993), como estilo de llamada y retorno (Bass, Clements, & Kazman, 2003) o como estilo independiente (Taylor, Medvidovic, Anderson, Whitehead, & Robbins, 1995).

Se puede observar que la agrupación de estilos arquitectónicos es susceptible de realizarse desde múltiples puntos de vista. Esta decisión depende de los intereses del responsable del desarrollo de arquitecturas en una determinada organización. Sin embargo, estas agrupaciones definen las restricciones generales de los estilos arquitectónicos.

Con base en la arquitectura lógica y en los paradigmas actuales se proponen tres puntos de vista de alto nivel para agrupar estilos arquitectónicos, los cuales son enumerados a continuación:

- Un punto de vista es el que agrupa los estilos arquitectónicos **orientados a procesos**. Esta clasificación se adecúa a modelos que se entienden mejor en términos de datos que fluyen entre unidades de procesamiento. Es decir, en estos estilos los datos y los procedimientos están separados. Los estilos arquitectónicos de este punto de vista animan a pensar en términos de componentes solo con comportamiento definido por las funciones, dejando en segundo lugar las estructuras de datos. Se enmarcan en esta agrupación los estilos de flujo de datos como "secuencial por lotes" y "tuberías y filtros" (donde los filtros no realizan forzosamente tareas de filtrado sino que pueden ejecutar diferentes transformaciones).
- El segundo punto de punto de vista importante es el que agrupa estilos arquitectónicos **orientados a capas**. Esta clasificación se adecúa a los modelos que se entienden mejor en términos de llamada y retorno. Es decir, sus elementos principales son entidades que pueden tener un estado, comportamiento e identidad. Por lo tanto, en estos estilos la estructura de datos y el comportamiento están estrechamente relacionados. Los estilos arquitectónicos de este punto de vista animan a pensar en términos de componentes conformados de comportamiento funcional y estructura de datos, dándoles prioridad a ambos conceptos indistintamente. Esta familia de estilos es la más generalizada en sistemas en gran escala. En esta familia podemos agrupar los estilos arquitectónicos en "capas", "orientados a objetos", etc.
- El tercer punto de vista significativo es el que agrupa los estilos arquitectónicos **orientados a la responsabilidad**. Esta clasificación se basa en las responsabilidades de las capas o componentes según se percibe desde el resto de las partes del sistema que orientan su diseño. Los estilos arquitectónicos en este punto de vista constituyen un apoyo para entender la creación de componentes esenciales. Se enmarcan en esta agrupación los estilos "centrado en datos", "máquinas virtuales", etc.

Por otro lado, cuando un analista piensa en un estilo arquitectónico normalmente piensa en Modelo-Vista-Controlador (MVC). Ahora bien, MVC es una agrupación lógica que ha sido propia de desarrollos con el lenguaje Smaltalk, antes de acuñarse el término de estilos arquitectónicos. En ese momento se le conoció como buenas prácticas; posteriormente a la definición de sus relaciones se le consideró como un estilo arquitectónico. Sin embargo, han ido variando sus relaciones pero conservando el nombre de MVC.

Esta tendencia se puede observar en la guía para la aplicación de las normas de la ESA de ingeniería de software para proyectos basados en métodos OO (ESA-Board-for-Software-Standardisation-and-Control, 1998), donde el modelo de análisis representado por un diagrama de clases debe tener clasificadas las clases en tres tipos:

- Clases de interfaz. Encargado en gestionar las transacciones entre el sistema y el mundo exterior.
- Clases entidad. Mantener la información del sistema.
- Clases control. Encargadas de coordinar el comportamiento de las otras clases.

Así mismo, la propuesta DSOA con casos de uso (Jacobson & Ng, 2005) sugiere estereotipar clases para acotar el lenguaje de UML en la creación de un modelo de análisis. Esta propuesta es deliberadamente elegida para que el modelo sea independiente de la plataforma. Las clases de análisis estereotipadas para detallar la abstracción del modelo de análisis son tres:

- Clases Límite (del inglés *Boundary*). Se utilizan para modelar la interacción entre los sistemas y sus actores. La interacción consiste en recibir y presentar la información de las solicitudes de los usuarios y sistemas externos.
- Clases control. Son las responsables de la coordinación, la secuencia, la transacción y el control de otros objetos y se utiliza para encapsular el control relacionado con un caso de uso.
- Clases entidad. Se utiliza para modelar la información en el dominio del problema.

Se puede observar cómo estas propuestas están inspiradas en los estilos arquitectónicos de **Modelo-Vista-Controlador** (MVC) (Krasner & Pope, 1988) con el objetivo principal de localizar agrupaciones lógicas. Sin embargo, en estos casos no se especifica de qué forma deben crearse las restricciones de las relaciones entre las agrupaciones o componentes.

El siguiente estilo arquitectónico en el que se piensa es el construido en **capas**. Este estilo aparece con mayor frecuencia mencionado en los catálogos. Algunos han identificado (Rising, 2000) cerca de cien patrones relacionados con las capas. Observando los diferentes estilos arquitectónicos en capas se puede deducir su cronología.

El más minimalista de los estilos del plausible punto de vista de arquitectura en capas sería el estilo **cliente-servidor** o dos capas. Es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. De esta separación en dos partes es fácil deducir que existirán elementos lógicos de interés mezclados internamente en cada una de las capas. La necesidad de simplificar otros elementos lógicos identificados surge el estilo tres capas. Este estilo tiene el objetivo de separar la lógica de negocio de la lógica de interfaz de usuario o presentación y de la capa de datos. Si MVC cumpliera la restricción donde cada componente sólo puede solicitar servicios secuencialmente desde la capa superior a la capa inferior, observaríamos que MVC es un estilo en capas. En este punto converge la arquitectura MVC como una relación muy directa en las agrupaciones lógicas. Cronológicamente se observaron aspectos de infraestructura que era necesario separar. El estilo cuatro capas intenta solventar esta necesidad pero es

prácticamente imposible cumplir con la restricción de las relaciones, dado que la capa de infraestructura se relaciona directamente con las otras capas.

La clave de los estilos arquitectónicos en capas está en la gestión de dependencia. En un estilo arquitectónico en **capas tradicional**, los componentes de una capa pueden interactuar sólo con componentes de la misma capa o bien con otros componentes de capas inmediatamente interiores. Sin embargo, en la actualidad hay otra aproximación de diseño en **capas laxo** que permite que los componentes de una capa interactúen con cualquier otra capa inferior. El uso del estilo estricto ayuda a reducir las dependencias entre componentes de diferente nivel pero el estilo laxo mejora el rendimiento al eliminar las redundancias de llamada y retorno.

Anteriormente, se ha dicho que se pueden detectar tres puntos de vista generales en la clasificación de estilos arquitectónicos: orientado a procesos, orientado a capas y orientado a la responsabilidad. En este escenario, los estilos arquitectónicos clasificados como orientados a capas se refieren a la aproximación de estilos arquitectónicos en capas laxo.

Los estilos arquitectónicos **orientados a procesos** separan los datos de las funciones, por lo tanto, no son recomendables para un desarrollo OO. En esta línea, se puede decir que la factorización propuesta por los estilos arquitectónicos orientada a procesos está fuera del ámbito de esta tesis doctoral, dado que esta memoria se enfoca a desarrollos OO. Por otro lado, los estilos arquitectónicos **orientados a capas** están enfocados al correcto modelado de la estructura y comportamiento de los componentes para una simplificación en las relaciones para una fácil gestión de llamadas y retornos entre ellos. Por lo tanto, los estilos arquitectónicos orientados a capas proporcionan los principios para facilitar una separación lógica de componentes en desarrollos OO. Sin embargo, estos principios se complementan con los principios de los estilos arquitectónicos **orientados a la responsabilidad**. Dado que dichos estilos proporcionan los principios para modelar componentes lógicos con una conducta sobresaliente percibida desde el resto del diseño. Esta conducta se centra en la creación de componentes pero se modela con una estructura de relaciones abierta que proporcione interacciones articulatorias que permitirá ejecutar los servicios prometidos.

Por lo tanto, las pautas de creación de componentes de una arquitectura lógica están basadas en estos principios de los estilos arquitectónicos orientados a capas y orientados a la responsabilidad. En esta línea, las restricciones de las relaciones de los estilos arquitectónicos sólo sugieren posibilidades que serán muy dependientes del producto software y la opinión subjetiva del responsable de la arquitectura lógica.

En el nivel conceptual de aspectos tempranos, una **arquitectura lógica** de un producto software debe especificar qué **bloques de construcción lógicos** de alto nivel formarán la aplicación. Sin embargo, entre estos bloques de construcción, capas o componentes lógicos se pueden distinguir capas que toman en cuenta la localización tangible. Por ejemplo, cuando una capa es vista como un componente de alto nivel del producto software en un dispositivo hardware diferente. Las capas de alto nivel de abstracción influenciadas por la localización tangible sólo anidan capas que cumplirán la distinción de descomposición lógica no tangible. Es decir, la descomposición lógica no debe plasmar una propuesta organización de artefactos de implementación. La única influencia en la descomposición lógica son dispositivos hardware obligados en una topología física inicial. Esta observación de la **influencia de la localización tangible** es un principio a tomar en cuenta para el diseño de bloques de construcción lógicos que no es explicitado en los estilos arquitectónicos.

6 Tipos de requisitos respecto a la calidad

Por más de dos décadas existe un unánime consenso en que la Especificación de Requisitos (ER) de un sistema identifica una distinción entre requisitos relativos a la **funcionalidad** y requisitos con características **no funcionales**.

Por otro lado, en la norma ISO 9000:2000 (Senlle, Ayuso, & Manchado, 2001) la calidad es definida como "el grado en que un conjunto de características inherentes cumple con los requisitos", donde el requisito es "una necesidad o expectativa establecida, generalmente obligatoria o implícita". De esta definición, se deduce que todas las agrupaciones de tipos de requisitos están enfocadas a las calidades, por lo tanto, la funcionalidad no es eficaz sin las características no funcionales. Sin embargo, debido a la ambigua y subjetiva naturaleza de las necesidades no funcionales, la mayoría de la atención en la ingeniería de software en el pasado se ha centrado en notaciones y técnicas para definir o especificar las funciones de un sistema de software.

Hoy en día el software controla innumerables sistemas que afectan a prácticamente todos los aspectos de nuestra actividad diaria, algunas de ellas críticas. El incorrecto funcionamiento de él puede producir inconformismo, riesgos económicos o incluso poner en peligro la salud o la vida de las personas. Ofrecer productos de calidad es una responsabilidad que comienza por un compromiso personal de las personas responsables del producto software, es decir, poner todos los medios para conseguir que el proceso de desarrollo produzca bienes activos que cumplan las especificaciones requeridas. Por lo tanto, **la calidad de un producto debe referirse al grado de cómo el producto cumple lo que se esperaba de él**.

En la ingeniería del software se hace hincapié en la calidad del proceso con la premisa de que un buen proceso asegura un buen producto. La calidad del software (Pressman, 2010) se define como el cumplimiento de los requisitos de funcionalidad y de rendimiento explícitamente especificados, de los estándares de desarrollo explícitamente documentados y de las características implícitas que se espera de todo software desarrollado profesionalmente. En este punto podemos de decir en el sentido más amplio que inicialmente existen dos tipos de requisitos: **explícitos** e **implícitos**. Los explícitos se dividen en funcionales y no funcionales. Por otro lado, cuando un requisito implícito se percibe como una necesidad no evidente, se documenta, pasando a ser explícito no funcional. Este proceso no es trivial, dado que depende del conocimiento del dominio de los interesados en el sistema.

En este contexto, en el corpus de requisitos de esta investigación se han identificado requisitos con objetivos de alto nivel que no definen directrices claras en el modelado de la ER. Son requisitos que son especificados para el entendimiento del dominio o del negocio con una imprecisión de su influencia sin la posibilidad de ser verificados. Estos requisitos son explícitos pero a veces podrían ser implícitos, incluso en ocasiones se observa que se dan por implícitas incumbencias que podrían ser explícitas. Dado que este tipo de requisitos no funcionales son ambiguos y dependen de la experiencia del analista para especificar su influencia o su descarte, han sido llamados **requisitos apócrifos**.

Al realizar modelados de ER con diferentes técnicas y métodos pertenecientes a la IR se identifican dos formas de estructurar los requisitos. El modelado estructural sugerido implícitamente por los estándares es el **secuencial**, referido a requisitos en lista. El otro modelado estructural identificado como en **árbol** es sugerido por algunas herramientas de gestión de requisitos al proporcionar los medios para modelar jerarquía de requisitos. En el segundo tipo de estructura de requisitos se pueden identificar requisitos padre o requisitos contenedores. Estos requisitos de alto nivel describen el conjunto de

requisitos hijos contenidos. Es sabido que los requisitos de una ER tienen que ser desarrollados hasta la implementación, esto quiere decir que los requisitos contenedores no se realizan hasta que cada uno de los requisitos contenidos se desarrolla. En esta línea, estos requisitos deben ser considerados apócrifos dado que son explícitos pero podrían ser implícitos.

Cuando a los requisitos de una ER se les realiza un análisis lingüístico para entenderlos mejor, aparecen requisitos que contienen más de una incumbencia que es necesaria separar. Sin embargo, hay algunos requisitos que al ser desgranados se identifican como definiciones de términos del dominio o del negocio, por lo tanto, también son considerados requisitos apócrifos.

7 Calidad en la Especificación de Requisitos

La disciplina de la IR en su forma más general es el proceso de descubrir y mantener los requisitos que describen o especifican un producto software de calidad. Para obtener beneficios de estas actividades es necesario comunicar, validar y analizar el conocimiento. El éxito de la aplicación de la IR depende de la trazabilidad con los otros artefactos creados durante el desarrollo del producto software. Sin embargo, otro factor importante a tomar en cuenta es la eficiencia para conservar la integridad semántica de los requisitos al transformarlos a otros artefactos. Ahora bien, el artefacto más importante para trasmitir el conocimiento entre las diferentes actividades de cualquier proceso de IR es el documento de Especificación de Requisitos (ER).

La característica más importante de una ER es que sea capaz de transmitir el conocimiento. Sin embargo, una clásica ER es la realizada por sentencias en lenguaje natural las cuales contienen potencialmente diferentes interpretaciones. En esta línea, si una ER debe trasmitir conocimiento es imprescindible que sea comprendida por todos los participantes en el proceso de IR. En este sentido, algunas normas establecen una distinción entre una ER que describe el conjunto de servicios requeridos al nivel de entendimiento de un cliente y la ER que describe el conjunto de requisitos enriquecidos con tecnicismos comprendidos por el desarrollador (IEEE-Computer-Society, 1998b) (ESA-Board-for-Software-Standardisation-and-Control, 1994). El ámbito de esta tesis doctoral está dirigido a los expertos ingenieros en el proceso de desarrollo de un producto software.

Por otro lado, muchas innovaciones teóricas y tecnológicas han resultado de la atención a la Ingeniería de Requisitos (IR). Dichas innovaciones están ocurriendo y madurando rápidamente, dentro de muchas comunidades de aplicación. Sin embargo, dentro de la comunidad científica se carece de un entorno de referencia homogéneo dentro del cual comunicar los conceptos motivadores en sus metodologías y técnicas en la aplicación de la IR. Típicamente, la base para formalizar un marco de referencia son las definiciones pero ninguna de las definiciones conocidas de la disciplina en la IR es respaldada por la totalidad de expertos. Entre otras cosas el mayor factor que influye en esta discrepancia es la ambigüedad en las diferentes definiciones existentes en el tema (Christel & Kang, 1992). Por lo tanto es recomendable comenzar con la adopción de un marco de referencia con un nivel de abstracción que satisfaga la mayoría de las propuestas. Para obtener dicho marco de referencia preferentemente se requiere una participación de forma iterativa e interactiva de la comunidad relacionada en la IR.

A hora bien, para que la IROA sea considerada como una disciplina que resuelve problemas es importante dotar a esta etapa temprana del proceso de desarrollo software con propuestas ingenieriles.

En este contexto, esta sección propone una aproximación de un **marco de referencia base para unificar criterios de calidad en el desarrollo de una ER**, estableciendo conceptos y términos de referencia de una ER para cualquier propuesta dentro la disciplina de la IR. Para su creación se han contemplado diferentes recomendaciones y buenas prácticas extendidas en el desarrollo de una ER.

7.1 Atributos de la Especificación de Requisitos

En el desarrollo de una Especificación de Requisitos (ER), dependiendo del contexto en que se desarrollan los atributos de requisitos, pueden distinguirse dos grandes grupos de atributos de distinta naturaleza: extrínsecos e intrínsecos.

Los atributos extrínsecos son los que disponen de un enlace persistente entre el registro del atributo y el requisito que describen, enlace que existe separadamente de estos requisitos. Entre los principales beneficios de los atributos extrínsecos está la flexibilidad a la hora de alterarlos, dado que modificaciones y agregaciones se efectúan sobre el atributo extrínseco sin modificar el requisito.

Los atributos intrínsecos son los que están sincronizados con el o los requisitos que describen, dado que son sensibles al contexto en que se utilizan, siendo muy difícil aportar semántica sin el requisito que describen. El problema principal de las propiedades intrínsecas es que su administración está estrechamente interrelacionada con el requisito o requisitos que describen, generando problemas técnicos a la hora de alterar alguna de las dos partes.

7.1.1 Atributos extrínsecos

El número y tipo de atributos extrínsecos dependen del propósito del proyecto concreto y de la organización de la factoría de software. Sin embargo existen atributos extrínsecos que se pueden considerar **obligatorios**. Para comprender los atributos extrínsecos a continuación se nombran algunos de los más utilizados.

Aplicar ingeniería a la producción de un producto implica catalogar cuidadosamente los elementos que lo componen para que puedan ser rastreados para su posterior composición. Por lo tanto, si un requisito es un elemento temprano en la producción de un producto software, cada requisito debe contener un **identificador**. Así mismo, es obligatorio determinar a qué, a quién o quiénes se deben consultar en caso de duda, por lo tanto, es obligatorio el atributo extrínseco de **fuente** que nos dice qué persona, documento u otro medio concibió el requisito. Otro atributo extrínseco obligatorio que proporciona la información para gestionar y administrar las líneas base de los requisitos es la **fecha de creación**.

Por otro lado, se ha dicho que los atributos extrínsecos dependen del proyecto u organización, es decir, existen atributos extrínsecos **opcionales** con la posibilidad de permitir añadir conocimiento a los requisitos. Un ejemplo sería la propiedad **asignado** que se refiere a la persona o grupo de trabajo interno o externo de la factoría de software que sea responsable de cumplir con la realización del requisito. Esta propiedad está vinculada fuertemente con la estructura de la vista lógica independiente de la tecnología, por lo tanto, supone una mayor semántica. Vinculado a este atributo encontramos el de **estado** de la implementación de la incumbencia.

En el contexto de la hipótesis de esta tesis doctoral un atributo importante es el que describe la posibilidad de **variabilidad** de una incumbencia a lo largo del tiempo de un desarrollo software. Por lo tanto, en el contexto de esta memoria este atributo es

obligatorio dado que su semántica es la principal a tomar en cuenta en la selección detallada de los aspectos en un DSOA.

Podríamos decir que un atributo extrínseco de un requisito se encarga de mantener un registro sobre el contexto y propósito de un requisito, de tal forma que se pueda identificar, extraer y gestionar dicho requisito para describir el estado de colecciones de requisitos, los procesos en los que están involucrados y las restricciones que se les aplican.

7.1.2 Atributos intrínsecos

Los atributos intrínsecos se refieren a la inferencia de atributos a partir de otros atributos o del propio requisito, es decir, podemos inferir atributos aplicando el uso del contexto que envuelve el requisito.

Un ejemplo clásico de atributo intrínseco en la especificación de requisitos de un producto software es la búsqueda de asegurar que cada uno de los requisitos debe expresar los participantes en la interacción. Por ejemplo, tenemos el requisito: *"La aplicación proporcionará al usuario la opción para crear incidencias";* al no ser lo único que tendrá que proporcionar la aplicación, típicamente en posteriores requisitos el atributo intrínseco que indica el participante *"aplicación"* es despreciado; por ejemplo *"Existirá una opción dónde los usuarios de distintos servicios podrán borrar una incidencia"*. Por lo tanto existe el atributo intrínseco de **roles participantes**. Las soluciones comunes a estos errores son intentar ordenar los requisitos por usuarios participantes u ordenar jerárquicamente los requisitos. Sin embargo, en los dos casos existen muchas posibilidades de que los requisitos tengan que ser repetidos varias veces. En el segundo caso los requisitos padre no son realizados si los requisitos hijos no son realizados, convirtiéndose éstos en requisitos contenedores que finalmente pueden considerar otros atributos intrínsecos.

Podríamos decir que un atributo intrínseco de un requisito se encarga de proporcionar semántica con una alta cohesión. Por lo tanto, requieren un trato especial a través de una correcta organización de requisitos.

7.2 Propiedades

Existen diferentes libros (Sommerville & Sawyer, 1999) (Braude, 2000) (Robertson & Robertson, 2006), así como estándares (IEEE-Computer-Society, 1998b) (The-European-Cooperation-for-Space-Standardization, 2009) de organismos de prestigio, con recomendaciones para el desarrollo de una Especificación de Requisitos (ER) de calidad. Sin embargo, cada propuesta recomienda cumplir con un conjunto de propiedades de calidad de alto nivel demasiado imprecisas. Un ejemplo típico de estas propiedades deseadas es: *"la especificación de requisitos global debe ser completa"*. Esta y otras propiedades de calidad deseadas, no orientan demasiado acerca de cómo especificar requisitos de calidad.

En este contexto, existe la necesidad de comprender el error con relación a las propiedades idóneas. Ante todo, es imprescindible no confundir un ideal con la realidad. Este error es el que se comete cuando se asumen como dogma algunas propiedades, presuponiendo o postulando como axioma aquello que es un factor de calidad y que sólo casualmente algún producto software satisface. Por lo tanto, una base fundamental es aceptar que algunas propiedades son ambiguas, pero se asumen como necesarias en toda ER y que sólo normalizando un modo de entenderla puede constituir un ideal racional. Ahora bien, para normalizar es necesario medir que se cumple en menor o mayor grado con las propiedades deseables, por lo tanto, sólo cuando a una propiedad se le asigna una o varias métricas para validarla y deducir sus directrices debe ser tomada

en cuenta. Es decir, aumentando la calidad del proceso de desarrollo aseguramos la calidad del producto. Por lo tanto, la calidad en el desarrollo de una ER está ligada de forma estrecha a la sistematización de las propiedades a cumplir. Una base para iniciar este objetivo es el grupo de diferentes propiedades mencionadas repetidamente en los libros y estándares (ver Tabla 6), a veces con diferentes matices pero claramente con el mismo objetivo. Sin embargo, dichas propiedades pueden ser tomadas como **propiedades típicas** a cumplir en cualquier desarrollo de producto software como factores de calidad.

IEEE 830	ESA	Pressman	Sommerville	Braude
Completo	Completitud	Completo	Completitud	Completitud
Ambigüedad	Claridad	Ambigüedad	Comprensibilidad	Ambigüedad
Consistencia	Consistencia	Consistencia	Consistencia	Consistencia
Modificable	Modificable	-	Adaptabilidad	-
Trazable	Trazabilidad	Trazabilidad	Trazabilidad	Trazabilidad
Verificable	Verificable	Comprobable	Verificabilidad	Comprobable

Tabla 6 Listas de propiedades típicas

Ahora bien, las diferentes propiedades a que los libros y estándares hacen referencia se pueden observar desde dos puntos de vista generales: **propiedades globales** y **propiedades individuales**. Esta diferencia facilita la creación de métricas que permiten identificar directrices específicas para cada una de las propiedades seleccionadas.

Por otro lado, los lenguajes naturales utilizados en la ER tienen una gramática con reglas y principios que regulan su uso. Por lo tanto, es recomendable tomar en cuenta la gramática general en la conformación de las propiedades típicas. Así mismo, las propiedades de calidad necesitan cubrir las funciones de la ER desde una perspectiva cognitiva.

En esta línea, también se puede observar que las propiedades típicas identificadas están relacionadas. Es decir, algunas se cumplen parcial o totalmente como consecuencia de cumplir otras propiedades. En esta tesis doctoral se han identificado las **propiedades base: completitud explícita**, **consistencia gramatical** y **coherencia semántica.**

Esta tercia de propiedades de calidad para el desarrollo de una Especificación de Requisitos (ER) cubre de forma explícita las dos propiedades típicas de **completitud** y **consistencia**, a través de directrices apoyadas en métricas específicas. Así mismo de forma implícita al cumplir con las directrices base propuestas se logra la **no ambigüedad**[32] y la **verificabilidad** dado que para cumplir dichas propiedades éstas requieren completitud explícita, consistencia gramatical y coherencia semántica. Observando la lista de propiedades de calidad típicas encontramos dos especialmente relacionadas con la propuesta de esta tesis doctoral: **modificable** y **trazable.** Es de todos sabido que si no es fácil modificar los requisitos será más complicado modificar el producto software en cualquier otra de sus etapas. Modificar la ER depende de la existencia de relaciones explícitas entre los requisitos, es decir, la trazabilidad horizontal. Así mismo, para modificar modelos en otras etapas del desarrollo de un producto software es necesario plasmar las relaciones entre ellos, es decir, la trazabilidad vertical. Con este escenario, cumplir con la coherencia semántica depende de que se gestione de forma efectiva la trazabilidad. Así mismo, el conocimiento de la trazabilidad proporciona los medios para modificar o evolucionar el producto software.

[32] Que puede entenderse de varios modos.

Es prudente comentar que a pesar de no aparecer en las propiedades típicas, también la **reutilización** es apoyada por las diferentes propiedades base propuestas en esta tesis doctoral. Sin embargo, es una propiedad que requiere una estrategia extra de modelado que no siempre es ventajosa.

En la siguiente sección se especifican las propiedades base propuestas con sus directrices y métricas para el desarrollo de la ER.

7.2.1 Completitud explícita

Un producto software es de calidad cuando se cumplen todas las partes de la ER, por lo tanto una ER es de calidad cuando es **completa**. Los requisitos funcionales son la parte de la ER que siempre se especifican explícitamente. Sin embargo, todos los demás requisitos (requisitos de rendimiento, de restricción, etc.) suelen ser en parte explícitos y en parte implícitos. Por lo tanto, una ER de calidad debería contener toda la funcionalidad convenida previamente con los interesados en el futuro producto software y los requisitos que cumplan con las propiedades no funcionales mínimas que la factoría de software propone para obtener un producto software de calidad.

Cuando la propiedad de completitud explícita se toma de forma **individual,** la directriz básica es **evitar términos que generen ideas incompletas** en la especificación de requisitos. La métrica mínima a aplicar es asegurar cero identificaciones de los términos reconocidos con dicha intención, tales como "más adelante", "en un futuro", "por determinar", etc.

Cuando la propiedad de completitud explícita se toma de forma **global** la directriz es verificar que la ER describe todos los **requisitos explícitos a cumplir por el futuro producto software**. Las métricas a utilizar para su validación deberían ser determinadas por la factoría de software. Sin embargo, una recomendación es utilizar la técnica de caso de uso o la de escenarios para verificar que la especificación de requisitos contenga la funcionalidad convenida. Así mismo, se recomienda utilizar algún estándar para completar la especificación mínima de los requisitos conocidos como no funcionales.

En el contexto de esta tesis doctoral, sería necesario verificar que se hayan tomado en cuenta todas las dimensiones típicas de aspectos de los proyectos realizados en el mismo dominio en dicha factoría de software y así complementar la información explícita.

7.2.2 Consistencia gramatical

El lenguaje natural es típicamente el medio con el que se modela la Especificación de Requisitos (ER). Es conocido que el lenguaje natural permite describir con diferentes formas gramaticales un mismo mensaje. Por lo tanto para su análisis asistido por ordenador, es deseable que la ER sea creada a través de un discurso[33] textual dirigido por directrices gramaticales específicas de lingüística textual. En pocas ocasiones dichas directrices son propuestas en las factorías de productos software. Así mismo, sólo algunos investigadores proponen indicadores a la hora de modelar la ER (J Llorens, Velasco, Moreiro, & Morato, 1998) (Alexander & Stevens, 2002) (Génova, Fuentes, Llorens, Hurtado, & Moreno, 2011).

La gramática es una disciplina combinatoria, centrada en la constitución interna de los mensajes y en el sistema que permite crearlos e interpretarlos. Por lo tanto, la gramática

[33] La palabra "discurso" es utilizada como unidad de comunicación.

estudia la estructura de las palabras, las formas en que estas se enlazan y los significados a los que tales combinaciones dan lugar (R.A.E, 2010). En este sentido, la gramática comprende la morfología, que se ocupa de la estructura de las palabras, su constitución interna y sus variaciones; y la sintaxis, a la que corresponde el análisis de la manera en que se combinan y se disponen linealmente, así como el de los grupos que forman. A partir de las unidades léxicas simples, la sintaxis puede articular unidades llamadas frases o sintagmas. Se considera hoy que son estos grupos los que realmente desempeñan las funciones sintácticas. Los grupos sintácticos son estructuras articuladas en torno a su núcleo, que admite diversos modificadores y complementos, que pueden formar parte de otros distintos de los que les dan nombre. Ahora bien, las clases de palabras y los grupos sintácticos establecen relaciones que permiten interpretar su aportación semántica al contenido de una oración o de otro grupo sintáctico. Estas funciones dependen muy a menudo de la posición que las palabras ocupan pero también de otras marcas o exponentes sintácticos. Se denomina función, en el sentido de relaciones de dependencia que permiten interpretar la manera en que se vinculan gramaticalmente ciertos segmentos con alguna categoría de la que dependen. Suelen distinguirse tres clases de funciones: **sintácticas, semánticas e informativas**. Las funciones sintácticas se establecen a partir de marcas o índices formales además de la posición sintáctica. Las funciones semánticas especifican la interpretación semántica que debe darse a determinados segmentos en función del predicado del que dependen. Las funciones del tercer tipo hacen referencia a la partición informativa de la oración, es decir, a la separación entre lo que se da por conocido y lo que se presenta como nuevo. La contribución de cada fragmento del mensaje depende de buena medida del discurso previo y de su papel en la articulación del texto, pero, a diferencia de los otros dos tipos de funciones, no está determinado por el significado de las piezas léxicas.

Las funciones sintácticas representan las formas mediante las que se manifiestan las relaciones que expresan los argumentos. Cada función sintáctica se caracteriza por la presencia de diversas marcas o exponentes gramaticales, como la concordancia, la posición y la presencia de preposiciones. Las marcas de función son los índices formales que permiten reconocerlas. Ahora bien, una misma función semántica puede corresponder a funciones sintácticas distintas y también desempeñar la misma función pero manifestarse mediante categorías diferentes.

En este contexto, el **marco de referencia base para unificar criterios de calidad en el desarrollo de una ER** propone la propiedad de **consistencia gramatical**. Esta propiedad se centra en un grupo de directrices que permiten formalizar en cierta medida las funciones sintácticas, semánticas e información en la descripción de cada requisito. Cumplir con las diferentes directrices proporciona elegancia y simplicidad a la ER. Estas directrices se enfocan a la reducción de posibles sintagmas que impliquen cambios de contenido de naturaleza gramatical. Por lo tanto, son las que mayor número de métricas requieren, donde siempre una nueva es bien recibida.

La primera directriz se basa en la idea de regular las **funciones sintácticas** y **semánticas** en la estructura de los requisitos. Las métricas propuestas para esta directriz son las siguientes:

1. La primera métrica tiene la intención de reducir al mínimo los **adverbios ambiguos** para no dar pie a la especulación y dificultar las pruebas de los requisitos. Entre dichos términos podemos mencionar: “bastante”, “suficiente”, “bonito”, “usable”, “generalmente”, “típicamente”, etc. Así mismo, es necesario evitar las palabras polisémicas y homógrafas.

2. La segunda métrica intenta minimizar la incorrecta interpretación de las palabras causada por la utilización de múltiples términos equivalentes: **sinónimos, siglas, acrónimos o abreviaturas**. Por lo tanto, para un entendimiento único es necesario un **vocabulario controlado**.
3. La tercera métrica apoya la idea de que los requisitos son más simples cuando el sujeto realiza la acción del verbo. Por lo tanto es necesario comprobar que todos los requisitos están escritos en **oración activa** y nunca en pasiva.
4. La cuarta métrica busca mejorar la comprensión de los requisitos en el tiempo. Las formas verbales en tiempo condicional expresan la acción como hipotética, tales como "debería", "podría", "habría", etc. Por lo tanto se debe asegurar que los requisitos siempre usen **frases en forma imperativa**.
5. La quinta métrica monitoriza la **subjetividad humana,** evitado la incorporación de puntos de vista personales, con frases tales como "en mi opinión", "pienso que", etc.

La segunda directriz de la propiedad de consistencia gramatical se basa en la idea de regular las **funciones informativas** con las siguientes métricas propuestas:

1. La primera métrica tiene el objetivo de **evitar expresiones deícticas**[34], dado que para su correcta interpretación dependen del contexto. Es decir, se busca evitar hacer referencia a elementos del discurso equivocados o presentes sólo en la memoria humana. Por lo tanto es necesario evitar las palabras anafóricas que sirven para indicar otros elementos, tales como "tú", "hoy", "aquí", "esto", etc.
2. La segunda métrica busca evitar la **mezcla de dos o más necesidades**. Para reducir esta característica es necesario, dentro de lo posible, evitar conectores en los requisitos que puedan estar formados por adverbios, preposiciones, conjunciones o por segmentos complejos.
3. La tercera métrica tiene la responsabilidad de evitar **requisitos incompletos**, buscando obtener cero ocurrencias de términos, ya sean simples o compuestos, de los que se infieren necesidades no completamente expresadas, tales como "más adelante", "en un futuro", "posteriormente", etc.
4. La cuarta métrica busca evitar confusión a través del uso de **énfasis** en los requisitos. La idea es evitar adverbios que enfaticen los requisitos. Por ejemplo: cuando "no" está en presencia de otros elementos que tienen también sentido negativo, tales como "nunca", "jamás", "tampoco", "nada", "ninguno", etc.

7.2.3 Coherencia semántica

Uno de los mayores problemas que se presentan en una Especificación de Requisitos (ER) es evitar contradicciones entre requisitos. En esta línea, podemos decir que la vista global de los requisitos de una ER debe tener **coherencia semántica**.

La propiedad de coherencia semántica depende de la consistencia gramatical pero sobre todo de la metodología propuesta por una factoría de productos software para la **organización** de requisitos. En este sentido, se puede observar que cuando se aplica una metodología de simple agrupación en un requisito donde describe una funcionalidad completa se obtiene una disminución de la dependencia entre requisitos. Sin embargo, aumenta la longitud de los requisitos y esto aumenta exponencialmente las posibilidades

[34] Elemento gramatical que realiza una deixis.

de contradicciones. Por lo tanto, los requisitos independientes disminuyen la coherencia semántica. Por otro lado, en una ER al aumentar la dependencia entre requisitos disminuye el tamaño y por ende las contradicciones. En este caso aumenta sustancialmente la coherencia semántica pero con un aumento sustancial del acoplamiento entre requisitos. Por lo tanto, es necesario dejar el control de las dependencias entre requisitos a herramientas tecnológicas para expresar los pensamientos de forma coherente.

Una ER de calidad debe facilitar la utilización de la semántica de los requisitos. Las siguientes directrices apoyan este objetivo:

En la propiedad de coherencia semántica, al contrario de la propiedad de consistencia gramatical, todas sus directrices se realizan de forma **global.**

La primera directriz a seguir es hacer **desaparecer los conflictos** entre requisitos. Los conflictos existen cuando dos o más requisitos se contradicen en su necesidad plasmada y no se pueden satisfacer simultáneamente.

La segunda directriz que hay que cumplir es **evitar las redundancias** entre requisitos. Esto se refiere básicamente a requisitos que quieren decir lo mismo y uno de los dos tiene que desaparecer.

La tercera directriz es la **gestión de un correcto acoplamiento** entre requisitos dependientes. Los requisitos dependientes tienen acoplamientos con otros requisitos, es decir, si cambia uno de ellos puede afectar al otro. En este sentido, se requiere matizar la navegabilidad de sus relaciones: la relación simétrica se da cuando existe navegabilidad bidireccional y las relaciones asimétricas se da cuando la navegabilidad es unidireccional.

Lo ideal en la construcción de una ER es tener cero ocurrencias en conflictos, redundancias y reducir al mínimo el número de acoplamientos. El mayor problema de identificar los posibles conflictos y redundancias, así como acoplamientos indeseables, es el hecho de que en la actualidad la ER más aceptada es la representada con un conjunto de requisitos descritos con lenguaje natural con una gran libertad de formas de expresión que generan una mayor cantidad de interpretaciones. Un problema añadido, es la variabilidad para modelar la semántica de una ER, dado que depende de la experiencia y habilidades del analista creador. Con este escenario, resulta complejo especificar métricas para las tres directrices de la propiedad de la coherencia semántica. Por lo tanto, actualmente se proporcionan pocas métricas para cumplir con las directrices necesarias en la propiedad de coherencia semántica en la creación de una ER de calidad.

7.3 Resumen del marco para medir la calidad de una ER

Los requisitos de una ER tienen dos tipos de atributos: Extrínsecos e Intrínsecos. En la actualidad la gestión de los atributos extrínsecos es resuelta con herramientas de gestión de requisitos configurables. Sin embargo, la gestión de los requisitos intrínsecos no es posible con las mismas herramientas.

Se ha dicho que los atributos intrínsecos son los relacionados con la semántica del atributo, por lo tanto no es posible su separación del propio requisito. Sin embargo, tienen una estrecha relación con las propiedades de calidad que se espera cumpla una ER. En este sentido una de las aportaciones del marco de referencia propuesto en esta sección es facilitar el desarrollo de una ER con modelos de una alta coherencia semántica, y así proporcionar los medios para la gestión de los atributos intrínsecos.

El entendimiento previo de este marco de referencia facilita el proceso de modelado de calidad para una buena transmisión del conocimiento de una ER, así mismo, su aplicación permite medir la calidad de una ER. La medición de la calidad se consigue al verificar el cumplimento de las directrices propuestas con sus métricas establecidas para las propiedades base identificadas.

Las propiedades base propuestas son:

- Completitud explícita
- Consistencia gramatical
- Coherencia semántica

Las directrices para cumplir con las propiedades pueden ser aplicadas desde los puntos de vista: global e individual.

En la propiedad base de completitud explícita existen los dos puntos de vista, es decir, se aplica en requisitos individuales y en grupo de requisitos. En el primer caso para cumplir con la directriz es necesario evitar un grupo de palabras preestablecidas que generan ideas incompletas. En el segundo caso actualmente no hay actividad clara que asegure la calidad; en este sentido este fue uno de los problemas que se buscó resolver con la metodología propuesta en esta tesis doctoral.

En la propiedad base de consistencia gramatical las directrices son aplicadas solo desde el punto de vista individual. En este caso para cumplir con las directrices se busca la reducción de posibles sintagmas que impliquen cambios de contenido de naturaleza gramatical; esto se logra formalizando la gramática de la descripción de los requisitos en el desarrollo de una ER.

Finalmente en la propiedad de coherencia semántica las directrices son aplicadas solo desde el punto de vista global. En este caso las directrices a cumplir son evitar conflictos, redundancias y acoplamientos entre requisitos. La propiedad de coherencia semántica implica gestionar la semántica de toda la ER. En estos casos no hay metodologías que permitan gestionar claramente la organización del conocimiento, por lo tanto, se torna complicado medir la calidad de una ER.

Por otro lado, esta tesis doctoral propone una metodología que proporciona los medios para gestionar las propiedades de completitud explícita desde el puto de vista global y la propiedad de coherencia semántica.

Las directrices de las propiedades de completitud explícita desde el punto de vista individual y la propiedad de consistencia gramatical se pueden aplicar en el desarrollo de una ER a través de metodologías clásicas, así como de la metodología OCTA propuesta en esta memoria.

CAPÍTULO IV
DESARROLLO EXPERIMENTAL

Capítulo V. DESARROLLO EXPERIMENTAL

El capítulo anterior establece el marco teórico de esta tesis doctoral, base para pasar a describir en el presente capítulo el desarrollo experimental.

En 1.2.1.4 del capítulo III del desarrollo teórico se justificó por qué los AT comprenden las etapas de la ingeniería de requisitos y las primeras líneas de solución de la arquitectura software. Así mismo, se definió en el modelo conceptual de AT que existen incumbencias dominantes y transversales. En este escenario, el desarrollo experimental descrito en este capítulo inició con una encuesta sobre el método aplicado en la organización de una ER para definir claramente las directrices al clasificar las diferentes incumbencias. Estas directrices proporcionaron las bases de la primera aportación definida como un **método para la creación de una APV** para productos software de sistemas de información con el apoyo de un corpus de requisitos que es descrito en este capítulo.

A través de un caso práctico, tomando en cuenta todas las bases teóricas propuestas anteriormente, se describe la principal aportación de esta tesis doctoral definida como una **Metodología para la Organización del Conocimiento Temprano asistido por Aspectos** (OCTA). Así mismo, se describe la evaluación realizada para demostrar la fiabilidad y eficiencia de la metodología OCTA.

1 Encuesta del método aplicado para organizar la ER

Se ha realizado una encuesta para analizar la información sobre el método utilizado por los ingenieros informáticos al organizar una Especificación de Requisitos (ER). La encuesta se ha llevado a cabo correspondiendo a la necesidad de identificar los puntos de vista que agrupan los aspectos dominantes. Es oportuno indicar que hay poca información escrita sobre la **justificación** de los criterios utilizados para la organización del modelado de una ER.

La encuesta se ha conducido en una fase, llevando a cabo una consulta masiva por vía Web[35] a través de un enlace comunicado por correo electrónico a diferentes universidades de España, universidades de Latinoamérica y factorías de productos software. La lista de correspondencia está compuesta de nombres de docentes universitarios en el último año de ingeniería informática, profesores docentes investigadores de universidad del área de ingeniería de software y analistas sénior de empresas.

La información que se solicita en la encuesta de cada participante se comenta a continuación[36]. Inicialmente se solicita que valoren su nivel de conocimiento en ingeniería de requisitos en una escala específica. En la segunda pregunta tipo test se pide definir su perfil principal en su organización de trabajo. En la lista del test se establecen dos perfiles universitarios y dos perfiles de factorías software. La tercera pregunta depende del perfil seleccionado, preguntando la universidad de procedencia en caso de los perfiles docentes o preguntando nombre de la empresa de procedencia en caso de

---

35 https://docs.google.com/spreadsheet/viewform?formkey=dG40MGd6TnI2U3N1Q0VNY3JtVEM0S0E6MQ

36 Anexo A

los perfiles de factorías de software. En el caso de ser de la universidad, se pregunta el área general de trabajo y en el caso de pertenecer a una empresa se pide clasificar el tamaño de la empresa según su número de empleados[37]. En la siguiente pregunta común a través de un par de opciones se solicita indicar qué tipo de estructura de requisitos prefiere. En la penúltima pregunta se pide indicar de una serie de casillas de verificación qué tipo de agrupación usa en el modelado de la ER. Finalmente de una lista se pide elegir qué herramienta de gestión de requisitos es utilizada.

En los resultados obtenidos en la encuesta, de un total de 168 personas entrevistadas, el 60% pertenecen a la categoría de estudiantes universitarios del último curso de ingeniería informática, el 25% a profesores y/o investigadores de universidad en Ingeniería, el 2% a analistas junior en empresas y el 14% a analistas séniores en empresas (ver Figura 26).

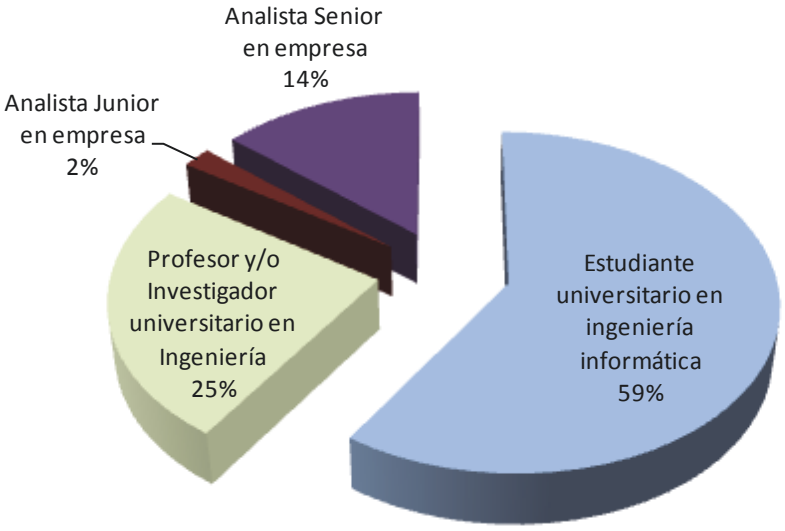


Figura 26 Visión general encuestados

En esta tesis se menciona la estructura de requisitos como la primera visualización de una organización, es decir, la distribución y orden básico de todos los requisitos de una ER. La lista secuencial y la jerárquica en forma de árbol son las principales formas de estructurarlos. En este contexto, el 48% de todos los participantes de los diferentes grupos prefieren la jerárquica y el 52% prefieren la secuencial. En una primera instancia es equilibrado el método aplicado en la estructura.

El grupo de personas evaluadas puede ser dividido en las categorías de analistas en universidades y analistas en empresas. Enfocando el análisis a los perfiles universitarios, se obtiene que de los estudiantes de último curso de ingeniería informática, el 37% prefiere la estructura jerárquica y el 63% la secuencial. Por otro lado, entre los profesores docentes investigadores el 65% prefiere estructurar los requisitos en jerárquica y un 35% en secuencial.

En el área universitaria, se puede observar que los estudiantes prefieren una estructura secuencial de requisitos y los profesores investigadores una jerárquica. Es factible

[37] Las empresas en España se clasifican en microempresas, pequeña, mediana, grande y muy grande; dependiendo del número de empleados asalariados y sus ingresos.

imaginar que los expertos del mundo académico tienen claras las ventajas de una organización de requisitos con cierta semántica.

Centrando el análisis únicamente en los perfiles de empresas privadas se obtiene que los perfiles de microempresas, pequeñas y medianas empresas prefieren el 100% la estructura jerárquica, y en los perfiles de empresas grandes y muy grandes el 33 % prefieren jerárquica y 67% secuencial. Los métodos aplicados para estructurar los requisitos en la empresa privada tienden a ser secuenciales cuanto más grande es la empresa. En esta línea se identificó que el 100% de los perfiles de las microempresas, pequeñas y medianas usan como herramienta de gestión de requisitos Excel o Word. Así mismo el 75% de los perfiles de empresas grandes y muy grandes frente a un 25% que prefieren usar herramientas de gestión de requisitos tipo CASE.

Una posible interpretación a este comportamiento es que derivado de la facilidad que existe para estructurar los requisitos en productos software de pequeño tamaño con herramientas simples es posible plasmarlos en jerarquía sin perder el control de la organización semántica. En contraparte cuando la aplicación a diseñar es grande y compleja se tiende a modelar los requisitos secuencialmente para intentar asegurar el control de la semántica.

Por otro lado, el análisis más importante sobre esta encuesta recae sobre la solicitud a los participantes de comunicar qué tipo de agrupación usa en el modelado de la ER. La pregunta permitía seleccionar una respuesta de una lista de seis posibilidades, más la opción de "ninguna de la lista" con la intención de cubrir todas las posibilidades. El resultado global de esta pregunta se puede ver en la Figura 27.

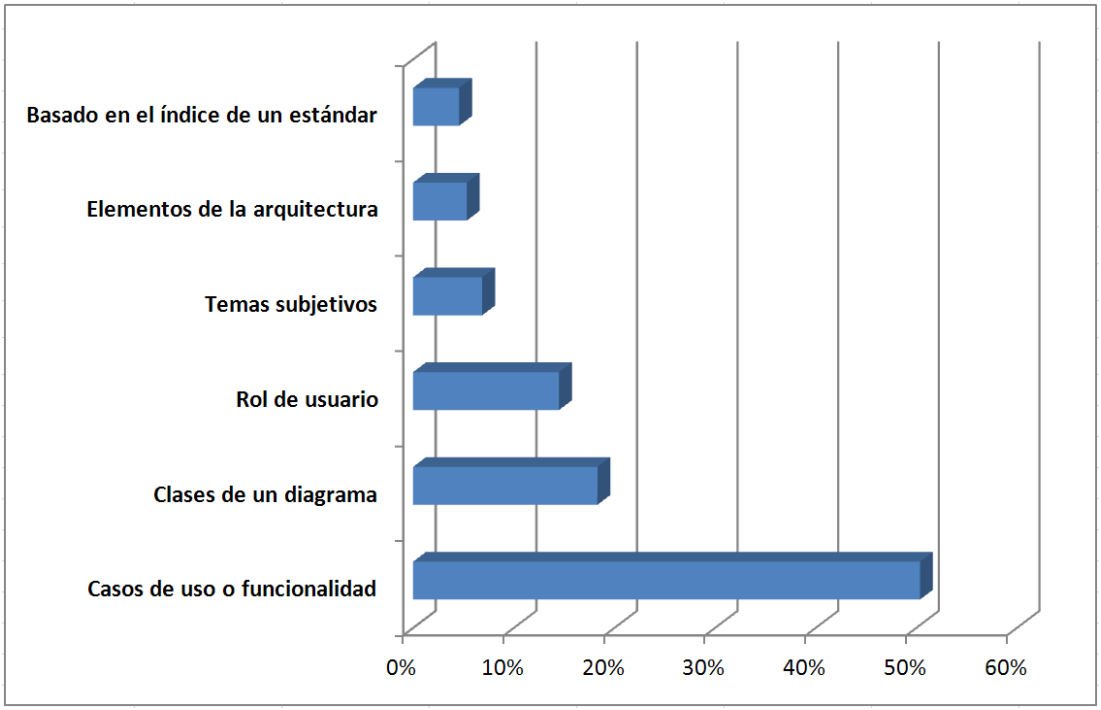


Figura 27 Agrupación usada en el modelado de la ER

El análisis revela que el 50% de los perfiles encuestados tienen una preferencia por agrupar los requisitos en casos de uso o funcionalidad. Esto revela que un punto de vista claramente dominante a la hora de agrupar requisitos o incumbencias es el enfocado a la funcionalidad o servicios ofrecidos por la aplicación.

Si bien la agrupación de requisitos siempre ha estado presente en la Ingeniería de Requisitos (IR), muchos analistas todavía tienen algunas dudas respecto a los beneficios

de una organización de requisitos que se pueda gestionar. Esto se puede observar en cómo el 39% de los encuestados indicó la utilización de más de un tipo de agrupación y el 85% de estos utiliza Excel o Word cuando para realizar una multiagrupación es necesaria como mínimo una herramienta de gestión de requisitos. Sin embargo, aquellos que no saben exactamente para qué sirve la organización intuyen que tiene beneficios y que es un catalizador para la calidad de la gestión de requisitos.

La segunda preferencia de agrupación es la de clases de un diagrama con un 18% de los encuestados. Si recordamos que los diagramas de clases creados en las etapas tempranas de ingeniería de requisitos son los diagramas conceptuales, en el contexto de esta tesis doctoral, es un diagrama de las primeras líneas de solución y por lo tanto es una vista lógica independiente de la tecnología parcial de la arquitectura. En esta línea, sería plausible sumar a esta preferencia los encuestados que prefieren una agrupación por elementos de la arquitectura.

En tercer lugar un 15% se encontró la preferencia de agrupación por rol de usuario, lo cual no es extraño cuando ha sido la primera aplicación de punto de vista en la ingeniería de requisitos.

Las preferencias con menor aceptación han sido las agrupaciones basadas en el índice de estándares y agrupadas por elementos de la **arquitectura con un 5%**. Así mismo, la agrupación por temas subjetivos se ha quedado con un 7% de aceptación. Es importante recordar que en la encuesta se tenía la opción de decantarse por "ninguno de la lista" y mencionar la que se utiliza, sin embargo, un único encuestado propuso una agrupación diferente, "prioridad de negocio".

2 Creación de una Arquitectura de Puntos de Vista

En este punto de la investigación aparece la necesidad de una metodología para la creación de una arquitectura de puntos de vista a través de formalismos de clasificación de contenidos. Así, surgen las vinculaciones con las disciplinas de documentación y biblioteconomía.

2.1 Análisis facetado y garantía literaria

Una APV es un esquema de clasificación capaz de expresar relaciones jerárquicas, así como las relaciones entre conceptos que pertenecen a diferentes jerarquías. En biblioteconomía existe un tipo de clasificación con contenido semántico conocido como **análisis facetado** (Ranganathan & Gopinath, 2006). La idea de partida dice que cualquier entidad compleja puede ser analizada desde múltiples facetas diferentes. Así mismo, se basa en la generalización de partes específicas de documentos que proporcionan afirmaciones sobre el tema a crear. Los nuevos temas generales de las partes elementales de documentos se llaman facetas. Ahora sabemos que la noción de faceta se basa en la creencia de que hay más de una manera de ver el mundo. Típicamente estas clasificaciones son vistas como estables, sin embargo, deben ser consideradas provisionales y dinámicas (Kwaśnik, 1992) . El desafío es construir clasificaciones que sean flexibles y puedan adecuarse a nuevos fenómenos.

La aplicación del formalismo de análisis facetado en la construcción de una APV para una determinada factoría de software, crea la necesidad de la utilización de un software para la gestión de una gran cantidad archivos digitales que proporcione un soporte integrado para la vida de los documentos, controlando información de adquisición, evaluación, catalogación, almacenamiento, preservación, búsqueda y acceso. Dicha actividad posibilita la identificación de los archivos más adecuados para proporcionar la información que permita proponer las mejores facetas o temas. En consecuencia, se

realizó una evaluación de las aplicaciones de software libre más conocidas para la gestión de archivos y determinar el más idóneo a utilizar en esta metodología (Barra, Palacios, Sánchez-Cuadrado, & Moreiro, 2011). En la comparativa se tuvieron en cuenta doce requisitos concernientes a la gestión de metadato, normas de descripción, análisis y ciclo de vida del documento. Las aplicaciones de software libre proporcionan las ventajas de mantenimiento, actualización e integración a nuevos contextos, convirtiéndolas en una alternativa prometedora en la gestión de archivos. Los resultados indican que todas las aplicaciones analizadas son factibles de utilización y acoplamiento para la gestión de archivos digitales.

Por otro lado, una de las directrices orientadas a la selección y organización de las partes más adecuadas en los sistemas de clasificación e incluso para sistemas de organización del conocimiento, es la formulación del principio de **garantía literaria** (del inglés *literary warrant*) (Hulme, 1950). El principio de garantía literaria en su forma genérica consiste en seleccionar una muestra aleatoria de términos de los títulos de la colección de documentos que deben ser clasificados. Así, posteriormente se agrupan los términos relacionados en temas comunes organizados en un esquema de clasificación. Este proceso requiere el conocimiento del dominio y el uso previsto de la colección. Los términos seleccionados y sus relaciones pueden ser considerados como un lenguaje del dominio específico utilizado para expresar las actividades en el ámbito de la biblioteca especializada.

Una peculiaridad de la garantía literaria en su aproximación a la representación temática está en analizar los procesos de clasificación sin preconceptos ni prejuicios. El método está centrado en los documentos o libros como entidad física de conocimiento y no teorías o ideas. La ordenación no debía ser concebida en función de preferencias ideológicas o visiones científicas. Otro elemento importante son los temas intrínsecos de cada libro o documento de naturaleza relativamente permanente. Estos temas deberían presentar perspectivas temáticas que pudieran servir como punto de acceso para los usuarios que por diversos motivos quisieran tomar contacto con ese documento, es decir, el motivo por el que un documento ha sido adquirido por la biblioteca o solicitado por un usuario.

Entre estas propuestas de clasificación u organización del conocimiento se pueden observar dos vertientes básicas de desarrollo. Una se apoya en el método deductivo denominado aproximación top-down, y otra que apela al método inductivo llamado aproximación bottom-up.

En el enfoque **top-down** los creadores siguen un proceso de división lógica del conocimiento desde lo más general a lo más particular. Si no están familiarizados con el área temática se apoyan en expertos en el dominio para revisar la estructura primaria y los conceptos recopilados. Este es el método que sugiere el análisis facetado para el desarrollo de estructuras. El enfoque top-down suele partir de las clasificaciones científicas y especializadas y de otras clasificaciones formales, encontrando en ese marco las divisiones tradicionalmente aceptadas de las macrodisciplinas y disciplinas. En estos casos la selección de los términos que se distribuyen a lo largo de las sucesivas divisiones está fundada en el valor aceptado de las clasificaciones formales preexistentes o de ciertos supuestos filosóficos.

El enfoque **bottom-up** parte del análisis de los términos o de los conceptos que aparecen en el documento o que se usan en la comunicación y en la práctica cotidiana de una comunidad de usuarios expertos. La colecta de términos puede ser exhaustiva o selectiva considerando los objetivos que se persiguen. Posteriormente, los términos candidatos se agrupan considerando su pertenencia a un tema, es decir, a un conjunto

homogéneo en función de al menos uno de sus atributos más distintivos. El enfoque bottom-up es respaldado por la garantía literaria.

Se ha sugerido que el análisis facetado tiene un enfoque top-down de organización del conocimiento y que parte de categorías abstractas que en un primer nivel se materializan en disciplinas y luego en sucesivas subdivisiones formales. Así mismo se ha comentado que la garantía literaria tiene un enfoque bottom-up de estructuración conceptual y que pretende extraer de los documentos los tópicos que se irán organizando conforme a sus atributos comunes.

La presunción de incompatibilidad de los enfoques bottom-up y top-down o análisis facetado y garantía literaria, es superada en el siguiente razonamiento: el análisis facetado desarrolla encabezamientos de facetas a partir de intereses cualesquiera; a esos encabezamientos se les aplica un principio de división en forma consistente, lo que da un número variable de focos. Los focos se ordenan dentro de cada faceta siguiendo criterios de ordenación denominados principios de secuencial útil, y la garantía literaria está incluida en estos principios, es decir, el análisis facetado y la garantía literaria se complementan. Por lo tanto, un análisis de dominios bottom-up asegura un proceso adecuado para recoger términos candidatos con una completitud aceptable. En completitud, un análisis top-down ajusta la estructura lógica y evita que queden lagunas en los esquemas.

Extrapolando estos principios a la ingeniería de requisitos, los requisitos de una ER pueden desgranarse con un método de garantía literaria en conjuntos menores de conceptos que mantienen entre sí relaciones estables y un hilo conductor. Así es posible con un análisis facetado agrupar estos conceptos y constituirlos en una arquitectura de puntos de vista.

2.2 Método para el desarrollo de una APV con un corpus de requisitos

En esta tesis doctoral se han aplicado propuestas de biblioteconomía enfocadas a la organización de conocimientos de libros o documentos. Sin embargo el contexto de esta memoria es la organización de conocimiento de requisitos. Por lo tanto, un requisito es considerado como la unidad fundamental comunicativa de conocimiento.

En el capítulo anterior se describió una Arquitectura de Puntos de Vista (APV) para el nivel de conceptualización de aspectos tempranos. La APV fue desarrollada con un **corpus de 6337 requisitos**. El corpus es una recopilación de **1658** requisitos de **proyectos privados realizados por PYMES**, **754** en **proyectos de fin de carrera** y **3925** en **proyectos académicos** desarrollados en el departamento de informática de la universidad Carlos III de Madrid. Los proyectos han sido realizados por analistas que pueden distinguirse entre trabajadores de PYMES, profesores docentes investigadores y estudiantes de ingeniería informática en último curso. Por lo tanto, se podría decir que en la creación del corpus han participado analistas junior y analistas séniores.

Los proyectos que forman el corpus de esta tesis doctoral son de la línea de producto de **sistemas de información**. Por lo tanto, la APV propuesta en esta tesis doctoral es factible para este tipo de aplicaciones. Sin embargo es posible reproducir el método desarrollado explicado a continuación para otras líneas de productos.

El enfoque de desarrollo de la APV en esta tesis doctoral se basa en la combinación del **análisis facetado** y la **garantía literaria**. En esencia se postuló una estructura de puntos de vista de alto nivel con un método top-down para posteriormente ser revisado y validado basado en un análisis bottom-up de los requisitos existentes.

Es interesante mencionar que los métodos top-down y bottom-up con las técnicas, metodología y herramientas actuales son susceptibles de ser semiautomatizados o incluso automatizados. Sin embargo, no ha sido trabajo de esta tesis doctoral, dejándolo para trabajos futuros.

Por otro lado, una ER tiene que ser una especificación o descripción completa de los aspectos importantes del sistema a crear o evolucionar. Algunos sugieren que para cumplir con este objetivo es necesario que para la creación de una ER se debe utilizar una plantilla estándar (Sommerville & Sawyer, 1999) pero las factorías de productos software tienen diferentes prioridades en sus contenidos.

Sin embargo, en su forma básica una ER parte de varios o de un solo Documento de Especificación Técnica (DET) textual que para su mejor entendimiento suele ser complementado con modelos gráficos. A menudo un DET está organizado conforme a una tabla de contenido en secciones predefinidas, las más conocidas suelen estar descritas en un formalismo típico de requisitos. Sin embargo, las demás secciones también son unidades de conocimiento fundamentales del DET. En este sentido, se deben tomar en cuenta todas las secciones desglosadas en el DET como posibles PV de alto nivel.

Entre los contenidos de los DET normalmente encontramos la sección de requisitos técnicos conocida como ER que es construida con un formalismo conocido. En dicha sección se suelen describir los requisitos bajo reglas de agrupación con un amplio consenso establecido. Dichos preceptos distinguen dos tipos de agrupación, los requisitos concernientes a la funcionalidad del sistema y todos los demás requisitos. En esta convención la agrupación de requisitos funcionales es aceptada por la gran mayoría de analistas por la facilidad de comprensión. En cambio, los otros requisitos, llamados no-funcionales en contraposición a funcionales, son una agrupación no tan fácil de comprender con puntos de vista contradictorios. Algunos dicen que imponen **restricciones** (Sommerville, 2005) a los funcionales, otros los visualizan como **propiedades** (Jacobson et al., 1999) del futuro producto software o como requisitos que indican **calidad** (Doerr, Kerkow, Koenig, Olsson, & Suzuki, 2005). Así mismo, normalmente en la agrupación de requisitos nombrados como no funcionales el contenido es redundante respecto a las otras secciones del DET. Por todo lo anterior, podríamos decir que las directrices para agrupar los requisitos no funcionales son ambiguas y subjetivas, siendo las buenas prácticas consensuadas la única directriz para la agrupación de los requisitos conocidos como no funcionales.

A este respecto, el estándar IEEE 830 (IEEE-Computer-Society, 1998b) y el estándar de ingeniería del software de la ESA PSS-05 (ESA-Board-for-Software-Standardisation-and-Control, 1994) han tenido una gran aceptación con sus temas propuestos para agrupar los requisitos no funcionales del DET (ver Tabla 7). Por otro lado, es sabido que proporcionar un conjunto inicial de temas candidatos en la construcción de una ontología simplifica considerablemente la tarea de construir sistemas de organización del conocimiento con una base de conocimiento por facetas. Estos PV de alto nivel proporcionan al constructor una ayuda para descubrir PV específicos.

En este método se considera que un PV es descubierto cuando el constructor se siente cómodo con un conjunto de términos relacionados con un PV y le proporciona un nombre que captura todos los términos agrupados. Por lo tanto, la definición de los PV es un proceso iterativo en el que un conjunto de términos se reagrupan en varias ocasiones. La forma de la agrupación también puede proporcionar pistas para la identificación de sinónimos, hiperónimos, hipónimos, pero sobre todo atributos de los términos específicos de un PV.

IEEE 830	
Contenidos textuales	**Requisitos**
Perspectiva del producto	Funcional
Características de los usuarios	Interfaces externas
Restricciones	Rendimiento
Suposiciones y dependencias	Base de datos
	Restricciones de diseño
	Atributos del sistema
	Otros requisitos

ESA PSS-05	
Contenidos textuales	**Requisitos**
Relación con proyectos en curso	Funcional
Relación con proyectos predecesores y sucesores	Rendimiento
Consideraciones del entorno	Interface
Relación con otros sistemas	Operacional
Restricciones generales	Recursos
	Verificación
	Pruebas de aceptación
	Documentación
	Seguridad
	Portabilidad
	Calidad
	Fiabilidad
	Mantenimiento
	Seguridad física

Tabla 7 Temas propuestos por estándares

2.2.1 Primer paso – Ámbitos

Por todo lo anterior el **primer paso** es la **consulta de expertos en el dominio**. En este caso se ha optado por seleccionar los temas propuestos por los estándares antes mencionados. Por lo tanto, es necesario unificar los temas propuestos por los expertos en el dominio en **temas de alto nivel**.

En este método la **Especificación de Requisitos** (ER) es parte de un **Documento de Especificación Técnica** (DET), sin embargo, la ER debe ser redundante a la semántica del DET. En este sentido, el DET está dirigido a los interesados en el sistema para una comprensión general y la ER está dirigida a los interesados en el desarrollo del sistema.

Por lo anterior, los temas considerados son los que agrupan las secciones del contenido textual y los subtemas en la sección específica de organización de requisitos. Este proceso cognitivo de unificación se logra buscando las relaciones producidas entre los temas propuestos que proporcionen una coherencia. La unificación de temas es coherente cuando cada tema puede ser comprendido en relación con la interpretación de otros temas relacionados.

En la aplicación de este método en los temas propuestos, inicialmente se detectó entre todos los temas considerados una dicotomía fundamental, puesto que unos agrupan especificaciones anteriores a la concepción de un producto software y otros agrupan especificaciones posteriores. Así mismo se observó que en la agrupación posterior a la

concepción del producto software surge otra dicotomía. Existe una agrupación de temas del futuro producto software y otra de su contexto operacional. Estos tres temas de alto nivel se denominaron ámbitos y fueron nombrados como:

- Ámbito del dominio especifico
- Ámbito del entorno operacional
- Ámbito del producto software

2.2.2 Segundo paso – Verificación de ámbitos

El primer paso del método se desarrolla a través de análisis facetado para obtener un conjunto de temas o puntos de vista de alto nivel llamados ámbitos. El **segundo paso** se desarrolla a través de la técnica de la garantía literaria. Por lo tanto, este paso comienza con la selección de una muestra de la colección de elementos de conocimiento o requisitos. En el caso de esta tesis doctoral todo el corpus de requisitos es considerado como una muestra. Posteriormente a un análisis lingüístico textual de cada uno de los requisitos del corpus fue posible agruparlos en alguno de los tres ámbitos propuestos, lo cual verifica la efectividad del primer paso.

Ejemplos de requisitos del ámbito del **dominio específico**:

RqDEa En la zona de pruebas del estadio se practican diferentes disciplinas deportivas de atletismo.

RqDEb Las disciplinas de atletismo se agrupan en carreras, lanzamiento, marcha, salto y combinada.

RqDEc El atleta es un deportista inscrito en alguna de las pruebas que se llevan a cabo en el estadio.

RqDEd La información mostrada en los paneles va dirigida a los espectadores que acuden al estadio para disfrutar las pruebas atléticas.

Ejemplos de requisitos del ámbito del **entorno operacional**:

RqEOa Los usuarios registrados en el sistema deben tener asociado uno de los siguientes roles: Administrador, Juez, Operador, Supervisor y Técnico.

RqEOb El usuario juez interactúa con el sistema a través de un dispositivo móvil.

RqEOc Los datos de los atletas deben ser cargados desde la base de datos de la Asociación Internacional de Federaciones de Atletismo (en inglés, International Association of Athletics Federations) que es el órgano de gobierno del atletismo a nivel mundial.

RqEOd Los técnicos pueden ser de limpieza, atención sanitaria y vigilante del orden público.

Ejemplos de requisitos del ámbito del **producto software**:

RqPSa Una vez solicitado el número de técnicos de apoyo el sistema software identifica a los técnicos específicos al tipo de incidencia más cercanos y sin incidencia asignada.

RqPSb El administrador podrá consultar al sistema software la lista de todas las incidencias con indicación de su estado, relacionadas con un técnico principal asignado.

RqPSc El sistema software deberá permitir al administrador dar de alta, modificar y dar de baja técnicos.

RqPSd Los sucesos indeseables de incendio, lesión de personas, suciedad extendida, actos violentos o extravío de personas serán considerados como incidencias.

RqPSe El sistema debe almacenar de cada usuario su nombre, apellido, DNI, alias, contraseña y localización geográfica.

RqPSf Las unidades de tiempo recogidas por el sistema en las diferentes pruebas de carrera deben contener horas, minutos, segundos y milésimas de segundo.

RqPSg Después de seleccionar el número de técnicos de apoyo necesarios se despliega una venta con el resumen de la incidencia y un botón para su cierre en el momento indicado.

RqPSh Cuando se pulsa el botón NOVEDAD se despliega una ventana con un cuadro de texto para introducir cualquier novedad y un botón con el texto MANDAR NOVEDAD.

RqPSi Algún técnico deberá haber recibido la alerta de asignación de emergencia principal en un tiempo no superior a 1 segundo desde que se inicia la gestión de incidencia.

RqPSj Los usuarios deben acceder al sistema por medio de un alias y una contraseña.

RqPSk El sistema debe realizar copias de seguridad de la base de datos al finalizar cada sesión en el servidor de respaldo corporativo.

RqPSl Si al acceder a la base de datos de la Asociación Internacional de Federaciones de Atletismo falla la conexión se debe mandar un mensaje del fallo a la interfaz del supervisor.

RqPSm Cuando al sistema no le sea posible cargar la información de algún atleta se hará uso del último respaldo existente.

RqPSn Oracle Database 11g versión 2debe ser el sistema de gestión de base de datos relacional a utilizar.

RpPSo Se dispondrá de tabletas Samsung galaxy note 10.1" para cada uno de los jueces.

RqPSp Se asegurará la comunicación entre el sistema y la base de datos IAAF con el protocolo SSH (Secure Shell) para cifrar la sección de conexión.

RqPSq Los artefactos de despliegue de la interfaz gráfica podrán instalar en los sistemas operativos Windows a partir de Windows xp y en sistemas operativos Android a partir de la versión 4.0

Estas tres agrupaciones de alto nivel son los tres ámbitos iniciales en donde inequívocamente es posible clasificar una colección específica de requisitos.

2.2.3 Tercer paso - Dimensiones

En el **tercer paso** se obtienen agrupaciones que permiten definir las dimensiones que componen cada uno de los ámbitos. En este paso nuevamente se utilizan los expertos en el dominio para proponer dimensiones en un determinado ámbito. Para conseguir este objetivo es necesario realizar otro análisis lingüístico textual de cada uno de los temas propuestos por los expertos en el dominio y de cada uno de los requisitos del corpus perteneciente a un ámbito específico.

Posterior al análisis lingüístico en el **ámbito de dominio específico** se identificaron los temas del **modelo de información de dominio** y **modelo del proceso de negocio**. Se observó claramente que en dichos temas era factible agrupar los requisitos pertenecientes a dicho ámbito. En el caso del modelo del dominio se detectó el peso que tenían los requisitos que proveen un vocabulario de información del dominio específico, por ejemplo los requisitos *RqDEa* y *RqDEb*. En el caso del modelo del proceso de negocio se encontraron requisitos que manifestaban conceptos con relaciones de jerarquía o composición que manifiestan el proceso de negocio anterior al producto software, por ejemplo los requisitos *RqDEc* y *RqDEd*. Estas dos dimensiones o temas de medio nivel fueron nombrados como:

- Dimensión de la información del dominio
- Dimensión del proceso de negocio

En el **ámbito de entorno operacional** se identificaron los temas del **contexto del producto** e **interesados en el sistema** (del inglés *stakeholder*). Los requisitos agrupados

en estos temas son los más sencillos de identificar; en el primer caso siempre se habla de sistemas con los que interactuará el producto software futuro, por ejemplo los requisitos *RqEOb* y *RqEOc;* en el segundo caso los interesados en el sistema contienen usuarios que utilizarán la aplicación e interesados en el desarrollo, los cuales son responsables de algunas de las primeras líneas de solución, por ejemplo los requisitos *RqEOa* y *RqEOd*. Estas dos dimensiones o temas de medio nivel fueron nombradas como:

- Dimensión del entorno de aplicación
- Dimensión de stakeholder

En el **ámbito del producto software** es donde se concentra el mayor número de propuestas de temas. Inicialmente se ha propuesto por expertos en el dominio del paradigma OA la existencia de la dicotomía de las dimensiones **dominantes** y de las dimensiones **trasversales** (Figura 25 Dimensiones del ámbito del producto software). Por lo tanto se agruparon los requisitos del ámbito del producto software en dos tipos de dimensiones: dominantes y transversales.

2.2.4 Cuarto paso – Dimensiones dominantes

Se ha demostrado en la sección anterior, a través de una encuesta, que la principal agrupación es la que reúne los requisitos que especifican **funcionalidad** útil al usuario del futuro producto software. Por lo tanto, los requisitos que referencian funcionalidad son considerados de una **dimensión del tipo dominante**. Al ser analizados lingüísticamente los requisitos considerados funcionales se encontraron términos comunes en esta agrupación que definían algunos tipos de funcionalidades específicas. Así, en lugar de una única dimensión funcional, fueron identificadas tres dimensiones dominantes:

- Dimensión de servicio
- Dimensión de consulta
- Dimensión de gestión básica

Se puede observar que un proceso iterativo identifica nuevas dimensiones dominantes para enriquecer la APV.

- La dimensión dominante de **servicio** es la que más requisitos agrupa con una alta cohesión, lo cual es debido a la dependencia que tienen los requisitos de esta agrupación.
- La dimensión dominante de **consulta** son requisitos independientes que al solicitar información específica al producto software no modifican el estado de este.
- La dimensión dominante de **gestión básica** al igual que la dimensión anterior son requisitos independientes entre ellos pero dependientes de comandos básicos que modifican el estado de la aplicación.

2.2.5 Quinto paso – Dimensiones transversales

Las **dimensiones del tipo trasversal** del ámbito del producto software son influenciadas por la gran cantidad de temas propuestos por los expertos en el dominio. Una lista inicial de estas agrupaciones basada en los estándares IEEE 830 y ESA PSS-05 se despliega a continuación:

IEEE 830

- **Interfaces externas**. Especifican características lógicas de la interfaz entre el producto software y sus usuarios; interfaces entre el producto software y los elementos de hardware; las interfaces de comunicación y el uso de otros productos software para la compatibilidad con otras aplicaciones.

- **Rendimiento**. Especifican los valores numéricos estáticos y dinámicos introducidos en el software o en la interacción humana con el software en su conjunto.
- **Bases de datos**. Especifican tipos de información utilizada, frecuencia de uso, capacidad de acceso, entidades de datos y sus relaciones, restricciones de integridad y retención de datos.
- **Restricciones de diseño**. Especifican restricciones de diseño impuestas por estándares, limitaciones de hardware, etc.
- **Atributos del sistema**. Especifican los factores que establecen la fiabilidad requerida, los que garantizan el nivel de disponibilidad, los que protegen el software del acceso accidental o malicioso, los que facilitan el mantenimiento y los relacionados con la facilidad de portar el software.

ESA PSS-05

- **Rendimiento**. Especifican valores numéricos para las variables medibles utilizadas para definir una función.
- **Interfaz**. Especifican las características de hardware, de software y de comunicaciones de elementos externos al sistema o componentes internos que deben interactuar o comunicarse.
- **Operacional**. Especifican cómo el sistema funcionará y cómo se va a comunicar con los operadores humanos.
- **Recursos**. Especifican los recursos disponibles para la producción y el manejo del software que son una restricción en el diseño.
- **Verificación**. Especifican restricciones del diseño cuando define características que se requieren para facilitar la verificación de las funciones del producto.
- **Pruebas de aceptación**. Especifican un tipo de verificación que limita el diseño.
- **Documentación**. Especifican el formato y el estilo de los documentos de especificación del proyecto.
- **Seguridad**. Especifican cómo se protegerá contra las amenazas a su confidencialidad, integridad y disponibilidad.
- **Portabilidad**. Especifican la facilidad con que se puede mover de un ambiente a otro.
- **Calidad**. Especifican los atributos del software que lo hacen apto para su propósito. Los más importantes son los de fiabilidad, relacionados con el mantenimiento y seguridad.
- **Fiabilidad**. Especifican la capacidad de un sistema para realizar sus funciones requeridas bajo condiciones establecidas por un periodo determinado de tiempo.
- **Mantenimiento**. Especifican la facilidad con que un sistema software puede ser modificado para corregir fallas, mejorar el rendimiento u otros atributos.

- **Seguridad física**. Especifican cómo reducir la posibilidad de daños que pueden derivarse de un fallo de software, identificando las funciones críticas cuyo fracaso afecta a las personas o los bienes.

Definidas las dimensiones dominantes en el ámbito del producto software, se procedió a la definición de las dimensiones trasversales. En este punto fue necesario un mayor esfuerzo en la unificación en los criterios de agrupación, dados los solapamientos obvios entre los diferentes temas propuestos de ambos estándares. Esto no quiere decir que no existan temas puros que sean considerados como el mismo concepto en diferentes propuestas.

Un tema inequívoco ofrecido por los dos estándares es el de **rendimiento,** donde se definirán puntos de vista que agruparán incumbencias relacionadas con los valores medibles en una actividad.

En la lista ofrecida en el estándar IEEE 830 tenemos el tema a*tributos del sistema* que se corresponde con la unión de una serie de temas con el estándar de la ESA PSS-05. Los temas de los que se habla son *fiabilidad, seguridad, seguridad física, mantenimiento y portabilidad*. Así mismo la ESA propone los temas independientes de *fiabilidad*, *mantenimiento, seguridad y portabilidad.* Aunado a esto se propone el tema *calidad* que describe los atributos del producto software *fiabilidad*, *mantenimiento* y *seguridad*. En este escenario, es observable cómo existe un solapamiento en estos temas de la ESA al relacionarlos con IEEE 830.

El tema seguridad (del inglés *security*) se agrupa con seguridad física (del inglés *safety*) en un mismo tema por las diferentes interpretaciones que se pueden deducir de estas dos acepciones. De esta forma, esta dimensión transversal se define con el tema **seguridad**.

En este proceso de unificación se detectó el subtema **portabilidad** dentro del tema *atributos del sistema* de IEEE 830, propuesto como tema independiente en la ESA con la misma conceptualización. En este tema se definirán puntos de vista de la facilidad de portar el producto software de un ambiente a otro.

Así al desambiguar los temas de *atributo del sistema* y *calidad* quedan los temas de **fiabilidad** y **mantenimiento** que son retomados un poco más abajo.

En el tema propuesto de *interfaces externas* por el estándar IEEE y el tema de *interfaces* propuesto por la ESA se refieren a los mismos tres subtemas de interfaces con diferentes objetivos, todos relacionados con el concepto de interfaz como un contrato para interaccionar.

En el subtema de *interfaces de hardware* se especifican los equipos hardware donde el software será ejecutado. Sin embargo, en el tema *recursos* propuesto por la ESA es necesario especificar los recursos disponibles para la producción y el manejo del software futuro. En esta línea podemos decir que hablan de lo mismo, por lo tanto, se propuso el punto de vista trasversal *recursos hardware.*

El subtema *interfaz de software* es definido en la ESA, como la especificación de la compatibilidad con otros programas en general, como pueden ser el compilador, el sistema operativo o lenguajes de programación. Por lo tanto es más claro definir este punto de vista como *software necesario.*

El subtema faltante del tema interfaz o interfaces externas es *interfaz de comunicación*. Este subtema se refiere principalmente a las *redes* y *protocolos de comunicación*. Sin embargo, en el tema *interfaz* propuesto por la ESA se comenta que las incumbencias en este tema también deben ser agrupadas en interfaces *internas* y *externas*, dependiendo de si las redes y protocolos de comunicación coinciden con los límites del sistema.

En los dos estándares los tres subtemas del tema de *interfaces* son considerados como *restricciones de diseño*; aunado a esto en IEEE 830 proponen el tema de restricciones de diseño. Este es el motivo por el que se propuso una dimensión transversal de **restricciones de diseño** donde está contenido claramente el tema **software necesario** como punto de vista. En esta línea, la **portabilidad** es considerada en los estándares como una restricción de diseño y por ende considerado como un punto de vista de dicha dimensión transversal. Así mismo nos encontramos con los temas propuestos anteriormente de **fiabilidad** y **mantenimiento**. Este último tema de *mantenimiento* se define como la facilidad con la que un sistema o componente puede ser modificado. La *fiabilidad* se define como la capacidad de un sistema o componente para realizar sus funciones necesarias en las condiciones establecidas. Se puede observar que los objetivos de estas agrupaciones son de muy alto nivel, por lo tanto, deberían guiar y proponer directrices pero no es así. Sin embargo, son puntos que restringen el diseño y son propuestos como puntos de vista.

Es interesante mencionar que las incumbencias que se agruparán en los aspectos creados bajo los puntos de vista de la dimensión de *restricción de diseño* son requisitos *apócrifos*. Son temas sin definición clara de su influencia en el modelado semántico de incumbencias. Agrupan incumbencias ambiguas sin posibilidad clara de ser verificadas. Sin embargo, existen otros subtemas clasificados como *restricción de diseño* pero que tienen la posibilidad de ser refinados para encontrar directrices con una tendencia a cambiar de agrupación.

Uno de estos temas es *recursos hardware* que al igual que los anteriores restringe el diseño software. Es un tema que modela los nodos que participarán en la topología del hardware en donde se ejecutará el producto software. Cuando se construye un sistema con gran capacidad de software, la principal atención del desarrollo se centra en diseñar el software. Sin embargo, para un ingeniero informático, la atención está en el hardware y el software del sistema, así como en el manejo del equilibrio entre ambos. En las etapas tempranas de desarrollo, mientras que los desarrolladores de software trabajan en el modelado lógico para la creación de futuros artefactos de código, los desarrolladores en sistemas trabajan con el modelado del hardware donde se ejecutarán los artefactos de código. Por lo tanto, cuando se habla de topología hardware se consideran un modelado que componen la vista lógica. En este escenario el tema *recursos hardware* tiene mejores resultados al agruparlo con los aspectos relativos a la *arquitectura lógica* del sistema.

Otros temas que restringen el diseño software son los temas **comunicación interna** y **comunicación externa** encontrados anteriormente. Sin embargo, la especificación de las *comunicaciones externas* tiene más coherencia modelando sus incumbencias en el ámbito de entorno operacional. Así mismo las incumbencias del punto de vista de *comunicación interna* especifican las relaciones que requieren los conceptos que se agrupan en el tema hardware del sistema. Por lo tanto al igual que *recursos hardware,* el punto de vita de *comunicación interna* presenta mejores resultados al agruparlo con los aspectos relativos a la topología hardware, modelando sus incumbencias en el ámbito de *arquitectura lógica*.

Un tema considerado sólo en el DET y no en ER dentro del estándar de la IEEE 830 es el de *interfaz de usuario*. Este concepto es considerado en el estándar de la ESA PSS-05 en el contenido de la ER con el tema *operacional*. A este respecto se ha considerado la dimensión de diseño trasversal de **interfaz de usuario**, donde se especifican las características lógicas de comunicación entre el producto software y el usuario.

Base de datos es un tema propuesto por el estándar IEEE 830 mencionado de forma sutil en el estándar de la ESA en el tema *interfaz*. Esta relación se debe a que en la actualidad, si un producto software requiere una base de datos, existen infraestructuras predefinidas

para su utilización. Por lo tanto podríamos decir que pertenece al tema de *software necesario*. Sin embargo, la base de datos también requiere definir la frecuencia de uso y capacidad de acceso, y en este caso serían puntos de vista de la *dimensión transversal de rendimiento*. Así mismo, la IEEE sugiere incluir las entidades *de datos con sus relaciones* pero estas ya han sido consideradas en el *ámbito del dominio específico*. Finalmente propone especificar los tipos de información utilizada por las funciones, tema a considerar identificado como **formato de datos**.

En este punto han sido considerados todos los temas propuestos por el IEEE 830, continuando con los temas faltantes ofrecidos por el estándar de la ESA.

El siguiente tema de la ESA fue *verificación*, donde la definición contiene la palabra que define, perdiendo consistencia en la propuesta del tema. Así mismo, en la ESA tenemos el tema *pruebas de aceptación*, el cual es definido como un tipo de verificación. Derivado de la ambigüedad en estos temas, no fueron considerados como dimensiones transversales. Sin embargo, el tema *verificación* es potencialmente un atributo importante de las incumbencias a modelar en la ER.

2.2.6 Sexto paso – Verificación de dimensiones

En este paso se realizó un segundo análisis lingüístico textual de cada uno de los requisitos pertenecientes al ámbito del producto software y considerados trasversales del corpus. En este segundo análisis lingüístico se requirieron mecanismos para evaluar la relevancia de los requisitos respecto a las dimensiones transversales propuestas. Un enfoque para relacionar las dimensiones con los requisitos es la ponderación de términos (del inglés *term weighting*) (Salton & Buckley, 1988). Los métodos estadísticos de ponderación de términos suponen que el comportamiento estadístico de un término dentro de un único requisito o un conjunto de requisitos, refleja la capacidad del término para representar el contenido del requisito y agruparlo en un tema. Una representación del contenido del requisito está caracterizada exhaustivamente cuando refleja con precisión y con profundidad sus múltiples temas. Por otro lado, la especificidad del término se refiere al nivel de detalle con el que un tema dado es representado por un único término. Un término que es específico de un tema puede diferenciar requisitos relevantes de una colección. Así mismo, términos específicos son de particular importancia cuando se construye a partir de un conjunto de puntos de vista de una misma dimensión. La ponderación de términos se desarrolla para aplicar la técnica de la garantía literaria con mayor profundidad.

En el quinto paso se aplicaron métodos de ponderación para agrupar los requisitos en las dimensiones transversales propuestas. Se procedió a organizar los requisitos del corpus en las diferentes dimensiones propuestas; sin embargo, se encontraron términos que sugerían crear dos nuevos temas de agrupación. Estas nuevas dimensiones transversales añadidas fueron las de **información** y **gestión de fallos**.

Anteriormente en la agrupación de requisitos de la dimensión de la información de dominio del ámbito del dominio específico se contemplaron los requisitos que especificaban entidades de información. Sin embargo, en las agrupaciones de las dimensiones trasversales del ámbito del producto software se identificaron requisitos relacionados con los datos o atributos específicos de entidades ya mencionadas anteriormente. Es necesario recordar que el estándar IEEE 830 describe la base de datos como el tema donde se agrupan los requisitos lógicos para cualquier información que será modelada en la base de datos. Por lo tanto los datos de información en las etapas tempranas podrían ser independientes de las entidades propuestas. Estos fueron

los motivos de añadir una dimensión transversal en el ámbito del producto software nombrada simplemente como **información**.

Un grupo de requisitos hacía referencia a fallos ajenos al producto software, indicando el comportamiento a tener en cuenta en caso de eventos predichos. En este caso *fallo* es un ejemplo de término ponderado, en sí mismo, motivo por el que se añadió la dimensión trasversal de **gestión de fallos**.

Este fue el método utilizado en el proceso de creación de una APV que nos permitirá la creación organizada de aspectos tempranos de un futuro producto software en tecnologías de la información.

2.2.7 Séptimo paso – Ámbito de la arquitectura lógica

En el último paso en la creación de una APV se obtienen agrupaciones que permiten definir las dimensiones del ámbito de la **arquitectura lógica**. En esta tesis doctoral se considera la dimensión arquitectura lógica donde se agrupan los PV bajo los cuales se modela el ámbito de la vista lógica de la arquitectura. En esta línea, en el séptimo paso se utilizan los principios ofrecidos por los estilos arquitectónicos. Para conseguir este objetivo es necesario comenzar tomado en cuenta el principio de **influencia de la localización tangible**.

En anteriores pasos se identificó la dimensión de **restricciones de diseño** en el ámbito del producto software. Esta dimensión en cierta medida es el preámbulo al séptimo paso, dado que se han identificado temas con agrupación de requisitos apócrifos que al refinarlos pasaron del ámbito de producto software al ámbito de la arquitectura lógica.

El tema **recursos hardware** modela los bloques de construcción de una topología de hardware y el tema **comunicación interna** modela las relaciones que requieren los conceptos hardware. Estos dos temas permiten entender la topología hardware que es la base del principio de la influencia de la localización tangible. En este sentido cuando se habla de *recursos hardware* y *comunicación interna* son considerados puntos de vista de la dimensión **hardware**.

El desarrollo de una aplicación comienza con el proceso de ingeniería de sistemas donde se capturan los requisitos totales del sistema, haciendo intercambios entre software y hardware para determinar un desglose de éstos. Posteriormente continúa el proceso de la ingeniería del software con el análisis y diseño aplicado a las partes del software. Así la ingeniería de requisitos comprende las tareas relacionadas con la determinación de las necesidades a satisfacer a un producto en ingeniería de sistemas o ingeniería de software. Considerar los límites exactos de los procesos a aplicar está fuera del alcance de esta investigación. Sin embargo, queda claro que las aplicaciones en sus etapas tempranas requieren de manera invariable elementos tanto de software como de hardware y el balance entre ellos. Así mismo, el objetivo de esta tesis doctoral es proporcionar una metodología para el modelado de los AT en el desarrollo de un producto software o aplicación.

Tomando en cuenta que cuando se desarrolla una aplicación con gran capacidad de software, la principal atención del desarrollo se centra en diseñar el software, es necesario considerar la dimensión **software** en el ámbito de la arquitectura lógica.

En el desarrollo teórico se encontraron tres puntos de vista generales en la clasificación de estilos arquitectónicos: *orientado a procesos*, *orientado a capas* y *orientado a la responsabilidad*. Los estilos agrupados en el tema orientado a procesos quedaron descartados dado que los principios de los estilos agrupados en este tema no son recomendados para el diseño OO. Sin embargo, los estilos arquitectónicos orientados a capas y a la responsabilidad

proporcionan los principios para facilitar una separación lógica de componentes. Dichos estilos aportan en mayor o menor medida principios genéricos de factorización lógica obtenidos de forma heurística a lo largo de la vida del estudio de la arquitectura software. En esta línea, se ha comentado cómo los conceptos de MVC, que son las agrupaciones de estilos arquitectónicos más utilizadas en los sistemas de información, están presentes de alguna u otra forma en los estilos orientados a capas.

La arquitectura lógica se basa en la idea de separación de conceptos lógicos para facilitar la tarea de asegurar que se cubren los AT mínimos. En este sentido, un principio importante en desarrollos de sistema de información es agrupar conceptos para modelar cómo el sistema recibe y presenta la información de las solicitudes de los usuarios, es decir, modela los conceptos dedicados a la interacción entre el sistema y sus actores. Por lo tanto, en la dimensión software del ámbito de la arquitectura lógica se sugiere el punto de vista **presentación**. Dicho tema está vinculado al tema *vista* de MVC y *presentación* del estilo arquitectónico cuatro capas.

Continuando con este enfoque, los desarrollos de sistemas de información requieren un almacén que permita guardar grandes cantidades de información de forma organizada para luego encontrarla y utilizarla fácilmente. Por lo tanto, es necesario un punto de vista **base de datos** que está relacionado con la capa de *infraestructura de persistencia de datos* del estilo de cuatro capas. El concepto define una capa compuesta por un conjunto de programas que manipulan un conjunto de datos almacenados en disco que permiten el acceso directo a ellos.

Así mismo se ha propuesto el punto de vista **funcional** que está vinculado a los componentes *control* en el estilo arquitectónico MVC y a la capa de *aplicación* en el estilo cuatro capas. Estos estilos parten del principio de que esta capa o componente está enfocado a controlar el modelo de la lógica de negocio. En un modelo de código en las etapas finales sería el responsable de la coordinación, la secuencia, la transacción y el control de los objetos en función de los servicios ofrecidos. Sin embargo, en las etapas tempranas, este punto de vista es responsable de agrupar conceptos organizados en aspectos temáticos basados en la funcionalidad lógica. En este contexto se puede observar cómo este tema está relacionado con los estilos arquitectónicos orientados a la **responsabilidad**, proporcionando los principios para modelar componentes lógicos con una conducta sobresaliente percibida desde el resto del diseño.

En esta propuesta no se propone una capa relacionada con el componente *modelo* del MVC o *modelo de dominio* en la propuesta tres capas, dado que esa información se encuentra plasmada en los otros ámbitos de la APV.

2.3 Conclusiones del método para el desarrollo de una APV

El corpus de requisitos utilizado en la creación de una APV con el método propuesto es de proyectos desarrollados en la línea de productos de **sistemas de información**. En consecuencia, la APV obtenida en esta tesis doctoral es potencialmente aplicable al modelado de los AT de cualquier producto software de sistemas de información. Sin embargo, el método puede ser aplicado para la misma u otras líneas de productos en diferentes factorías de software con sus correspondientes decisiones según el tipo de organización.

Se ha comentado que en el proceso de diseño del método para la obtención de una APV específica se encontró la necesidad de crear temas de alto nivel denominados como ámbitos en el primer paso para refinarlos en dimensiones en el tercer paso. Esto es una APV que satisface a todos los interesados cumpliendo con las restricciones explicitas del entorno.

Otra característica importante en el método es cómo los diferentes pasos aplicados en este método se complementan al ir alternando las propuestas de análisis facetado y garantía literaria. Es decir, en el proceso de creación de los temas en la APV se van verificando de forma iterativa.

Una visualización desarrollada con la herramienta Protégé del modelado de la APV obtenida se puede observar en la Figura 28.

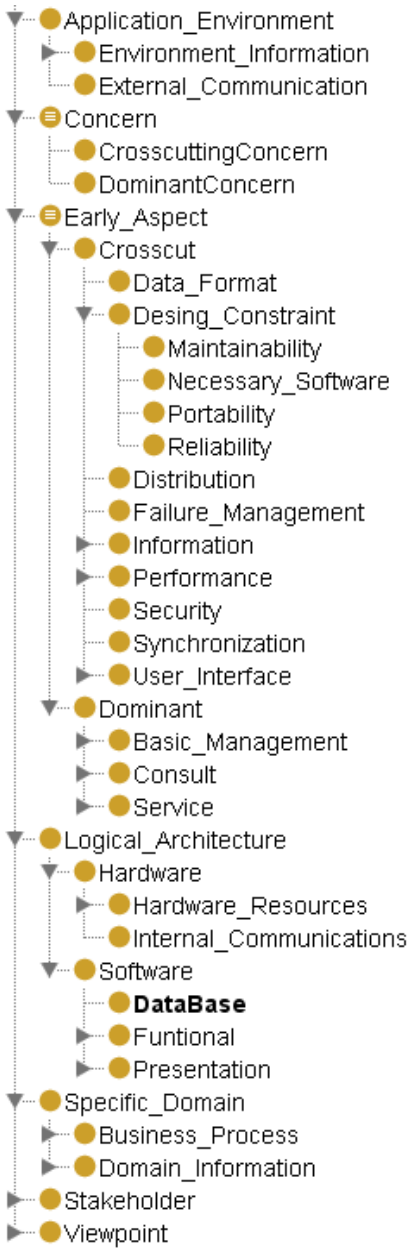


Figura 28 Arquitectura de puntos de vista propuesta

3 Metodología OCTA

A partir del desarrollo teórico expuesto en el capítulo anterior se ha desarrollado una metodología que define la construcción de modelos tempranos de un producto software en un proceso de desarrollo. Dicha metodología se ha denominado **"Organización de Conocimiento Temprano asistido por Aspectos"** (OCTA).

La metodología OCTA proporciona un conjunto de principios y heurísticas que intentan resolver la comprensión del conocimiento entre modelos para asegurar una

correcta modificación o evolución futura del producto software. Esta metodología no pretende sustituir el proceso de desarrollo software aplicado en alguna determinada organización. OCTA pretende dar soporte a la falta de flexibilidad al cambio inherente a los diferentes modelos tempranos en el desarrollo de un producto software.

3.1 Marco de herramientas de la metodología

En la actualidad, sin herramientas tecnológicas apropiadas, la aplicación de cualquier metodología o técnica de desarrollo suele fracasar, dado que la automatización supone generalmente un ahorro considerable en tiempo y dinero frente a la realización manual de los procesos. El modelado de Aspectos Tempranos debe especificarse con herramientas que permitan apoyar la metodología propuesta.

La metodología OCTA no pretende cambiar las bases actuales de los analistas en la especificación de requisitos. Sin embargo, una herramienta a tomar en cuenta es una de gestión de requisitos. La mayoría de estas herramientas cumplen con las características básicas: identificación unívoca de requisitos, clasificación de requisitos textual y gestión de atributos extrínsecos. Sin embargo, una eficiente herramienta de gestión de requisitos además debería poder: atomizarlos, categorizarlos, relacionarlos, reutilizarlos por separado o en conjunto y medir su calidad individual o global.

En el caso de las necesidades de esta metodología, el gestor de requisitos solo será usado como una herramienta de visualización, por lo tanto, es factible la utilización de cualquier gestor de requisitos típica. Sin embargo, es necesario que la herramienta trabaje con una base de datos fácil de exportar e importar para posibilitar la interoperabilidad con los modelos creados en la metodología OCTA.

En la experimentación de esta tesis doctoral se estuvo trabajando con la herramienta de gestión de requisitos **Visure Requirements**[38] que cumple con las características comentadas con cierta sencillez.

Al igual que la herramienta de gestión de requisitos se utiliza en la metodología OCTA como visualización es necesario utilizar herramientas de modelados de diseño para la visualización de diagramas complementarios. En consecuencia, es necesaria una aplicación, dentro del marco de las herramientas CASE, que cumpla con el estándar UML 2.x. Una característica deseable de una herramienta de entorno CASE, es que intente cubrir todo el ciclo de vida de desarrollo de software, desde la gestión de requisitos hasta el modelado y las pruebas. Esta característica permite la trazabilidad entre requisitos y otros modelos de representación, la cual la convertiría en una herramienta aceptable para trabajar en el modelado del nivel conceptual de aspectos tempranos. Sin embargo, estas herramientas suelen especializarse en diferentes niveles en diferentes etapas. Por lo tanto, buscando una herramienta que proporcione los medios para interaccionar con otras herramientas y permita modelar diagramas UML 2.x se optó por utilizar **Enterprise Architect**[39] que es de un uso sencillo con cierta estabilidad y rendimiento.

En los últimos años se han propuesto varios lenguajes para la definición de ontologías. El Lenguaje de Ontologías Web (OWL) es una recomendación presentada en 2004 por el W3C para la construcción de la Web Semántica, convirtiéndose en poco tiempo el

[38] http://www.visuresolutions.es/software-gestion-de-requisitos

[39] http://www.sparxsystems.es

lenguaje para la descripción semántica de ontologías más conocido. Por lo tanto, OWL ha sido el lenguaje seleccionado en esta tesis doctoral para crear ontologías que representan modelos de aspectos tempranos. En este sentido, es necesaria una herramienta que proporcione un entorno de edición de ontologías que ayude a crear ontologías y permita inferir conocimiento con OWL. Una herramienta de edición de ontologías de libre distribución que satisface estos requisitos es **Protégé**[40].

Este editor ha sido desarrollado principalmente por la universidad de Stanford con el apoyo de varias organizaciones gubernamentales y privadas. Protégé implementa un amplio entorno para el modelado, implementación, manipulación y visualización de ontologías en varios formatos. Otra de las características de Protégé es su posibilidad de ser ampliada por medio de una arquitectura de plug-in y de la API (del inglés *Application Programming Interface*) de Java para la construcción de herramientas y aplicaciones basadas en conocimiento.

La herramienta significativa en esta metodología es la que posibilita los medios para razonar con el modelado resultante de AT. Por lo tanto, es necesario un motor de razonamiento utilizado en el desarrollo de aplicaciones que hacen uso de representaciones ontológicas. En el caso del desarrollo de esta metodología se tomó en cuenta los razonadores compatibles con el editor de ontologías Protégé, optando por el razonador **FaCT++** por su estabilidad y velocidad.

3.2 Claves de la metodología OCTA

El modelado de AT implica el proceso recurrente de crear y registrar las incumbencias tempranas agrupadas en aspectos con el objetivo de modelar su conocimiento de forma clara. De esta forma, los modelos tienen la capacidad de comunicar a cada uno de los interesados en el futuro modelo software conocimiento específico. Es decir, conocimiento que tenga un significado claro para el cliente y los miembros del equipo de desarrollo, asegurando la correcta transferencia de conocimiento necesaria. A este respecto, el correcto modelado de AT en dicho proceso proporciona los medios para gestionar los efectos de las inevitables incumbencias cambiantes provocadas por la variabilidad de los deseos de los interesados en un proceso de desarrollo software. En este escenario, el modelado de aspectos e incumbencias tempranas tiene su mayor apoyo a través de la ingeniería ontológica.

El marco de referencia propuesto en esta tesis para la conceptualización de aspectos fomenta un conjunto de buenas prácticas, una de la cuales es modelar las etapas tempranas basadas en el modelo conceptual de Aspectos Tempranos (AT). En el modelo conceptual de AT, la **arquitectura lógica** está interrelacionada con una estructura de **especificación de requisitos**. En la metodología OCTA estas partes se modelan en una misma representación plasmando su trazabilidad. Esto permite identificar la repercusión de los cambios en incumbencias que representan requisitos atómicos relacionados con las incumbencias que representan la arquitectura lógica. Una directriz común y unificadora en el pensamiento OO es la conocida como "**en base a las responsabilidades**". En los enfoques OA es una directriz indispensable en la creación de AT. A través de la definición de las responsabilidades es posible representar la especificación de requisitos y la arquitectura lógica para modelar qué debe hacer el producto software sin decir cómo lo hará.

---

[40] http://protege.stanford.edu/

En este contexto, la APV proporciona una guía para el modelado de las responsabilidades a través de los PV de alto nivel denominados ámbitos y dimensiones para asegurar la creación de los PV de bajo nivel. El desarrollo de la metodología OCTA en esta memoria se realizó con la APV para productos software de información plasmada en la sección 2.3.

En el proceso de creación de una ER con lenguaje natural podríamos decir que la escritura de requisitos se compone de tres procesos mentales: planificar, redactar y examinar. **Planificar** es la representación mental de la información que contendrá el texto. **Redactar** es la transformación de las ideas en signos visibles y comprensibles para el lector. **Examinar** es la revisión para su reformulación consciente de todo lo planificado y escrito. Ahora bien, en la creación de una ER con la metodología OCTA, el esfuerzo que conllevan estos tres procesos no es mucho más, incluso en ocasiones menor, que el realizado en una representación típica. El motivo se encuentra en cómo las reglas de escritura son planteadas de forma diferente con el objetivo de una mejor representación del conocimiento. El resultado es un esfuerzo parecido en el proceso de planificación pero en la redacción está implicado el aprendizaje de las nuevas reglas de representación. Sin embargo el esfuerzo solo es reflejado la primera vez. En el proceso de revisión para su reformulación existe una reducción del esfuerzo debido a la aplicación de la metodología OCTA, dado que a través de sus directrices asegura una baja reformulación del conocimiento. Por lo anterior, la metodología OCTA no conlleva un mayor esfuerzo en el proceso de creación de un ER.

3.3 Caso de estudio "Estadio para eventos de atletismo"

Ahora bien, basado en el paradigma OA se optó por realizar un caso de estudio centrado en el proceso de modelado de los Aspectos Tempranos (AT) para exponer los principios y factores que entran en juego en la metodología OCTA. El caso de estudio elegido se encuentra en un contexto familiar, pero suficientemente rico de complejidad con problemas interesantes en las etapas tempranas que permiten concentrarse en cómo llevar a cabo la organización semántica de los AT, en lugar de centrarse en explicar el problema y el dominio.

A continuación se describe brevemente el caso de estudio de un futuro producto software "Estadio para eventos de atletismo" que fue desarrollado para demostrar las bondades de la aplicación de la metodología OCTA.

El objetivo principal del sistema a desarrollar es automatizar la gestión de información de un estadio para eventos de atletismo de alta competición.

Una parte del sistema estará encargada de informatizar los datos relacionados con las pruebas de atletismo y atletas, para difundir la información relativa a los eventos y atletas a través de cuatro pantallas led situadas en diferentes puntos del estadio. En lo relacionado con los datos obtenidos de cada prueba realizada por atletas existe la colaboración de jueces a través de un dispositivo móvil.

Una segunda parte del sistema se encargará de la gestión de incidencias producidas dentro del estadio con la asignación de los recursos necesarios para resolverlas. Entre los recursos se contempla la necesidad de técnicos especializados en asistencia sanitaria, vigilancia y limpieza, los cuales poseerán dispositivos móviles para el desarrollo de su logística.

A continuación se describen los pasos de la metodología OCTA que tendrán un enfoque iterativo en el desarrollo del modelado de aspectos tempranos. Es decir el modelo se crea de manera frontal y a continuación se regresará a las partes que se consideren en proceso de evolución, afinándolo y detallándolo.

3.4 Paso 1. Ontología común MMAspect

A partir de esta sección se ejemplificará un caso de uso para exponer los principios a cumplir en el desarrollo de modelado a través de la metodología OCTA.

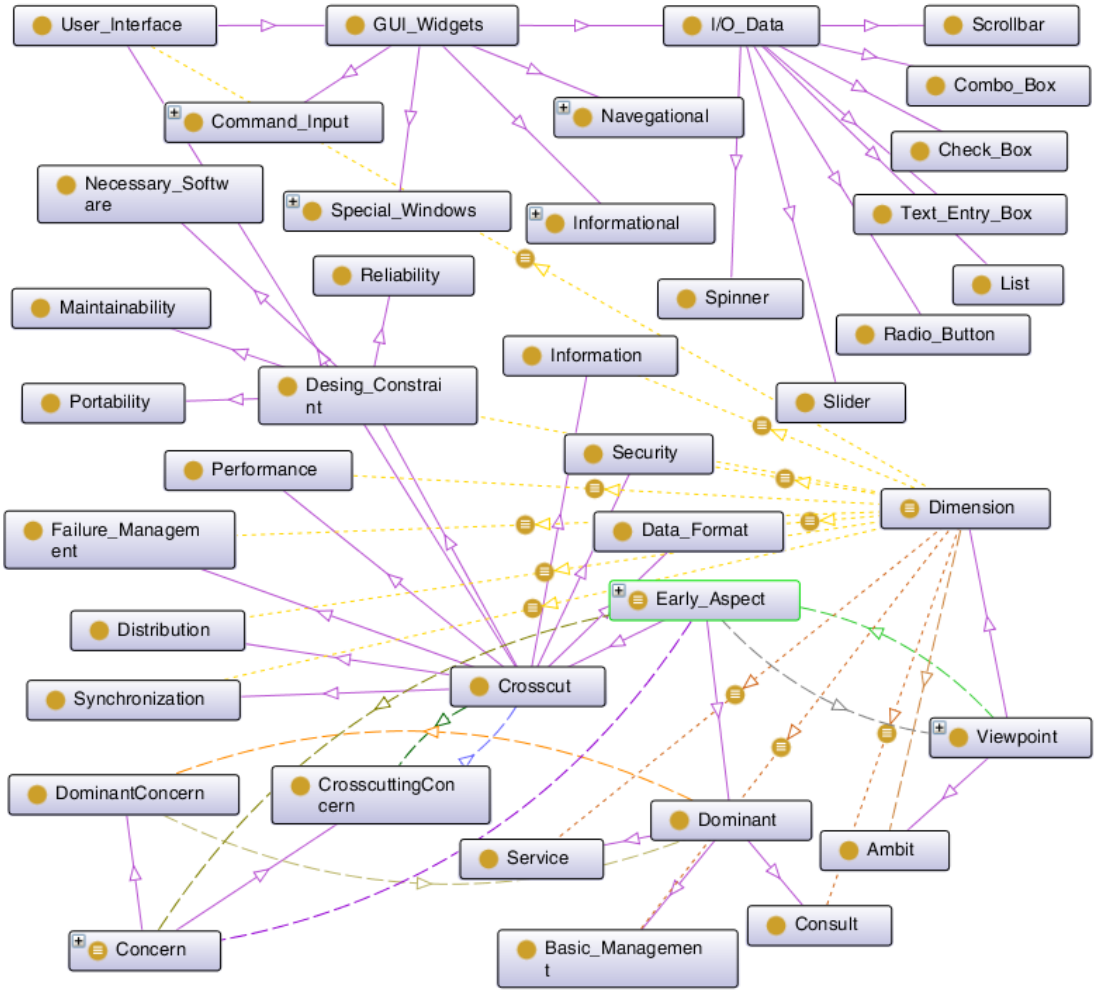


Figura 29 Propiedades de MMAspects

Un convencionalismo para mejorar la legibilidad adoptado en esta descripción de la metodología es incluir notas explicativas y teóricas dentro de recuadros.

La metodología OCTA está basada en una agregación de varias ontologías básicas que son partes de un todo.

En el paso uno es creada la primera **ontología básica MMAspect**, la cual es de las consideradas de **dominio**, dado que es usada para representar un meta-modelo de aspectos tempranos. El modelado de la ontología MMAspect debe contener los conceptos con sus relaciones y restricciones que plasman la semántica del modelo de referencia de AT (ver Figura 29), así como la estructura y entidades de la APV obtenidas con el Método para el desarrollo de una APV con un corpus de requisitos.

Es importante mencionar que en esta tesis doctoral las ontologías creadas antes del modelado de AT con la metodología OCTA fueron desarrolladas con nombres y definiciones de clases en idioma inglés con el objetivo de realizar experimentación con modelados de AT en lenguaje anglosajón.

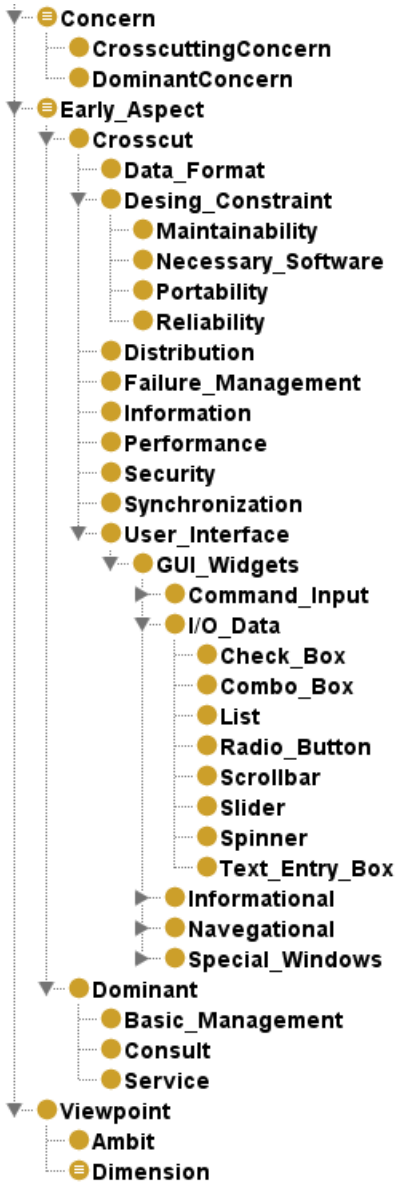


Figura 30 MMAspects en protégé

En el inicio de la creación de esta ontología se desarrolla la jerarquía de clases con la definición de sus propiedades en un proceso top-down, comenzando con la definición de los conceptos más generales con la subsecuente especialización de estos. La Figura 30 muestra cómo fueron registradas las clases o entidades *"Early _Aspect"*, *"Concern"* y *"Viewpoint"* que tienen existencia independiente. La clase *"Viewpoint"* contiene las subclases *"Ambit"* y *"Dimension"* dado que estos conceptos son considerados como Puntos de Vista (PV) pero de diferentes niveles de abstracción. *"Concern"* es una clase que no ha llegado a usarse en el caso práctico pero llegará a ser importante en otros la distinción entre *"CrosscuttingConcern"* y "*DominantConcern"* por la posibilidad de cambio constante en las incumbencias. La clase *"Early_Aspect"* contempla las subclases de *"Crosscut"* y *"Dominant"* para la descripción de la APV con todas sus dimensiones compuestas por PV.

Se puede observar que la ontología *"MMAspects"* tiene la intención de ofrecer un conocimiento reutilizable para cualquier otro DSOA en aplicaciones de sistemas de información. Por lo tanto, a veces no es necesario volver a desarrollarla, sin embargo, si es propuesta una modificación o una nueva APV es obvio su cambio pero con las mismas directrices.

3.5 Paso 2. Ontología del ámbito de dominio específico

La segunda **ontología** básica creada es de las consideradas de **dominio**, dado que puede llegar a ser reutilizada en un dominio específico de un producto software futuro.

En la metodología OCTA esta ontología modela el ámbito de *dominio específico* que está especializado en la dimensión de la *información de dominio* y en la de *proceso de negocio*. En el nivel conceptual de AT la *información de dominio* se enfoca en las entidades de información con propiedades o valores conocidos previamente. Podríamos decir que el modelado de la *información de dominio* contiene los elementos del negocio que se espera informatizar. Sin embargo, es necesario que al mismo tiempo se modelen los AT de la dimensión del modelo del *proceso de negocio* dado que el modelado independiente de las dos dimensiones provocaría duplicidad de conceptos. Los conceptos de la *información de dominio* se convierten en clases base para el entendimiento del *proceso de negocio*. Esto obliga al modelado del *proceso de negocio* a crear los conceptos complementarios de la *información de dominio* para su entendimiento actual o futuro. En este contexto, es recomendable partir del negocio existente en la captura de información, dado que se espera añadir los cambios de negocio cuando se quiera ampliar o modificar los beneficios de la informatización de los datos. De esta forma, en futuras evoluciones algunas entidades que se encuentran en el modelo de *información de dominio* pueden pasar a las del modelo del *proceso del negocio* y viceversa. Este formalismo permite no modelar ni más ni menos que lo necesario, teniendo siempre un modelo del ámbito de *dominio específico* consistente pero con posibilidades de su evolución en el tiempo a lo largo del desarrollo del futuro producto software.

> Cuando se habla de crear los modelos de una dimensión esto significa modelar AT bajo los PV vinculados a su respectiva dimensión. Se ha dicho que es posible crear varios AT desde un solo PV, sin embargo, se tiene la posibilidad de crear un número indeterminado de PV contenidos en una dimensión. En este contexto, algunas veces una dimensión contiene PV bajo los cuales se modela un solo AT. En estos casos se considera la misma entidad como el PV y su AT, pudiendo identificarlo como PV de bajo nivel.

En el caso de estudio de esta tesis doctoral las dimensiones de *información de dominio* y *proceso de negocio* son consideradas PV de bajo nivel, es decir, por cada dimensión se modela sólo un AT.

Ahora bien, en este paso el primer criterio en la creación de la ontología de *dominio específico* es la construcción de su forma básica, es decir, la creación de la estructura de clases. Siempre que sea factible, la concepción de las clases debe ser con una organización jerárquica de superclase a subclase o viceversa. Por ejemplo, de la Figura 31 consideremos las clases *carrera*, *combinada*, *lanzamiento*, *marcha* y *salto* que son subclase de *prueba*. Centrar la atención en la clase *carrera* de este modelado permite inferir que **todas las carreras son pruebas**, que **todas las futuras instancias de carreras son pruebas** y qué **carrera está incluida en la clase prueba**. En siguientes pasos esta ontología añadirá restricciones en una mayor ontología, lo cual permitirá realizar más inferencias.

Por otro lado, en esta misma ejemplificación de estructura de entidades de información construida bajo el PV *información de dominio*, también se identifica que las subclases de la entidad *prueba* requieren proporcionar la semántica para describir un rango de valores (ver Figura 31). El formalismo para su especificación no es único del modelado de esta ontología de dominio específico. Es un formalismo distintivo de la ingeniería ontológica, donde es conocido como **patrones ontológicos** (Gangemi, 2005).

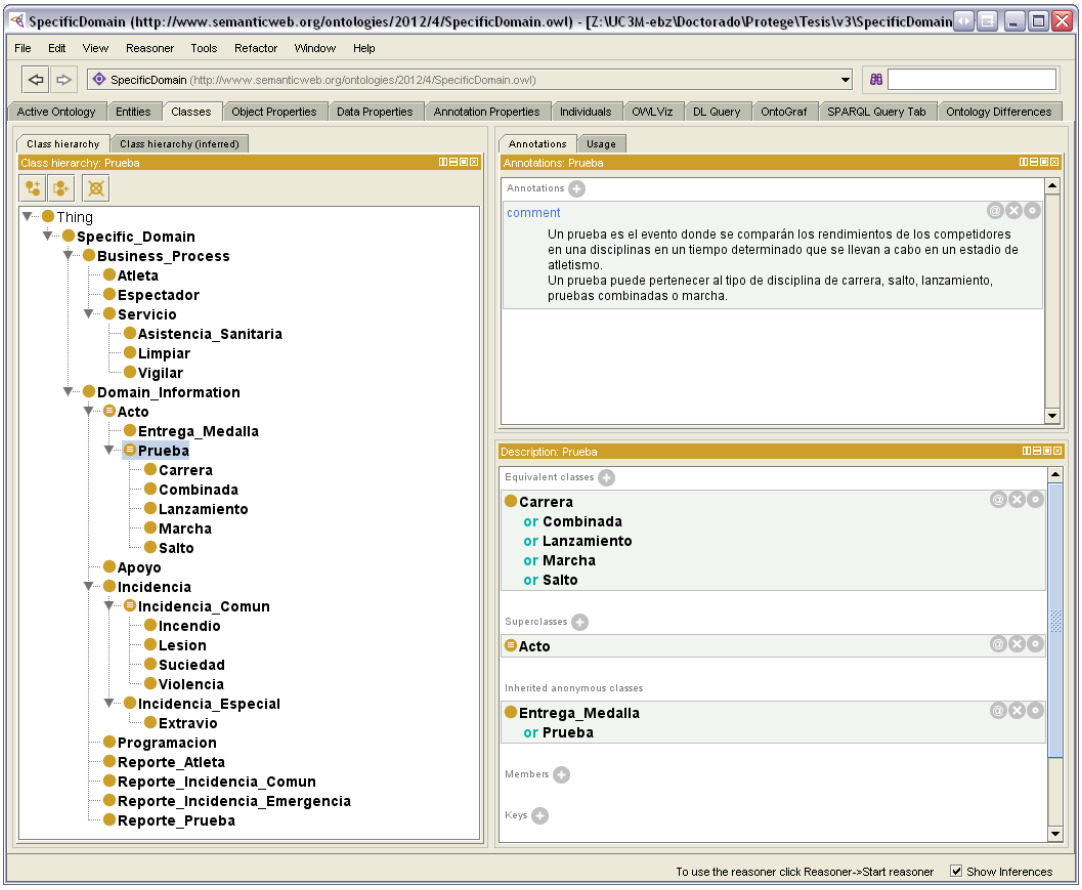


Figura 31 Dominio Específico

Un patrón ontológico es un patrón de diseño en la ingeniería ontológica, por lo tanto, son soluciones a los problemas de modelado que pueden surgir varias ocasiones en las ontologías. Es decir, son patrones de diseño reconocidos como soluciones probadas para problemas comunes de modelado.

Un patrón ontológico conocido entre los expertos en ontologías es el de **partición valorada** (del inglés *value partition*) que es un patrón para refinar las descripciones de clase. En este patrón lo primero es identificar la clase que representa una partición valorada, para a continuación asegurarse de que las subclases representan todas las opciones posibles de particiones valoradas.

En OWL se asume que las clases se pueden solapar, es decir, no podemos asumir que un individuo no es miembro de una única clase en particular, por lo tanto, con el fin de asegurar la separación de un grupo de clases deben hacerse **disjuntas** entre sí. Esto asegura que un individuo que se ha dicho que es miembro de una de las clases del grupo no puede ser miembro de cualquier otra clase de ese grupo.

En una estructura de clases padre e hijos, se dice que la clase padre es la clase que está cubierta y los hijos las clases que forman la cobertura. En el patrón partición valorada se requiere un **axioma de cobertura** (del inglés *covering axioms*); esto quiere decir que se requiere que la clase padre esté cubierta totalmente por sus clases hijo, es decir, la clase padre se manifiesta como una clase que es la unión de las clases hijo.

En la ejemplificación del caso de estudio en la ontología de *información de dominio* se creó una partición valorada a la entidad *prueba* para restringir el rango de valores posibles. En principio se crearon las subclases que posiblemente representan todas las opciones posibles de particiones valoradas, que en este caso han sido: *carrera, combinada,*

lanzamiento, marcha y *salto*. A continuación se especificaron disjuntas el conjunto de estas subclases en *"Disjoint classes"*. Se prosiguió proporcionando un axioma de cobertura para hacer la lista de los tipos de valor como una unión que se refiere únicamente a la clase *prueba*. Esto se logra añadiendo en *"Equivalent clases"* la especificación *Carrera or Combinada or Lanzamiento or Marcha or Salto*.

En este punto es pertinente aclarar algunos conceptos de OWL relacionados con Protégé para el correcto modelado de la ontología. Dentro de la lógica para la descripción semántica adoptada por OWL tenemos los conceptos de condición necesario y suficiente. Cuando una clase se describe usando las condiciones necesarias, entonces podemos decir que un individuo es miembro de esta clase cuando satisface las condiciones. Sin embargo, no podemos decir que cualquier individuo que satisface estas condiciones debe ser un miembro de la clase. No obstante, si la misma clase se define con las condiciones necesarias y suficientes, se puede decir que un individuo es un miembro de la clase cuando cumpla con las condiciones. Es decir, las condiciones no sólo son necesarias para ser miembro de la clase, sino también suficiente para determinar que algo que satisface estas condiciones es un miembro de la clase.

Las clases que tienen al menos un conjunto de condiciones **necesarias y suficientes** se conocen como **clases definidas**, y cualquier individuo que satisface la definición pertenecerá a la clase. Las clases que tienen sólo condiciones necesarias se conocen como **clases primitivas**. En Protégé las condiciones necesarias son simplemente llamadas **Superclases** y las condiciones necesarias y suficientes se llaman **clases equivalentes** (del inglés *equivalent classes*).

Una clase OWL se especifica en términos de sus superclases o clases equivalentes. Estas también pueden tomar la forma de **descripciones complejas** que se pueden construir usando descripciones de clases simples que se componen juntas usando operadores lógicos. En particular:

AND -Una clase formada utilizando el operador AND se conoce como una clase de intersección. La clase es la intersección de las clases individuales.

OR -Una clase formada utilizando el operador OR se conoce como una clase de unión. La clase que se forma es la unión de las clases individuales.

La aplicación del patrón ontológico **partición valorada** no es distintivo de la ontología de *"información de dominio"*. Es pertinente utilizar dicho patrón para refinar la descripción de cualquier clase. En el caso de la clase *prueba* es usado para describir los diferentes tipos de prueba que se pueden realizar en el *estadio* referenciado en el caso de estudio.

3.6 Paso 3. Ontologías del ámbito del entorno operacional

En el paso 3 se crean las dos ontologías básicas de aplicación del ámbito de "*entorno operacional"* especializada en la dimensión de *"entorno de aplicación"* y la dimensión de *"Stakeholder"*.

El modelo del entorno operacional describe o especifica cómo el producto o sistema software actual o futuro encaja dentro de un sistema más amplio. Por lo tanto, describe las entidades de su entorno para entender como interactuará el producto software con otras aplicaciones y roles. En este sentido, en una primera instancia se requiere considerar el modelo del producto software como una caja negra. Este criterio no es fácil cuando se observa entre los elementos que aparentemente conforman el producto software, cómo algunos constituyen un foco de atención externo. Es decir, existen aplicaciones que no serán desarrolladas dentro del producto software futuro pero que es

indispensable considerarlas en el desarrollo. En estos casos dichas partes del producto software son consideradas del *entorno de aplicación*. Un ejemplo de estas partes en el caso práctico es el software grabado dentro de los circuitos electrónicos de la pantalla LED que será adquirida. En consecuencia, el modelado de la ontología del *entorno de aplicación* contiene clases que representan el software del concepto y no el concepto en sí mismo. Así mismo, esta ontología debe modelar las entidades que representan los sistemas independientes que interaccionan con el producto software que claramente son dependientes de estos. Un ejemplo de estos sistemas es la base de datos de la asociación internacional de federaciones de atletismo, la cual requiere interaccionar con el producto software para proporcionar los datos de los atletas del caso de estudio *estadio*.

En la Figura 32 se puede observar la ejemplificación de la ontología del *entorno de aplicación* del caso práctico de esta tesis doctoral.

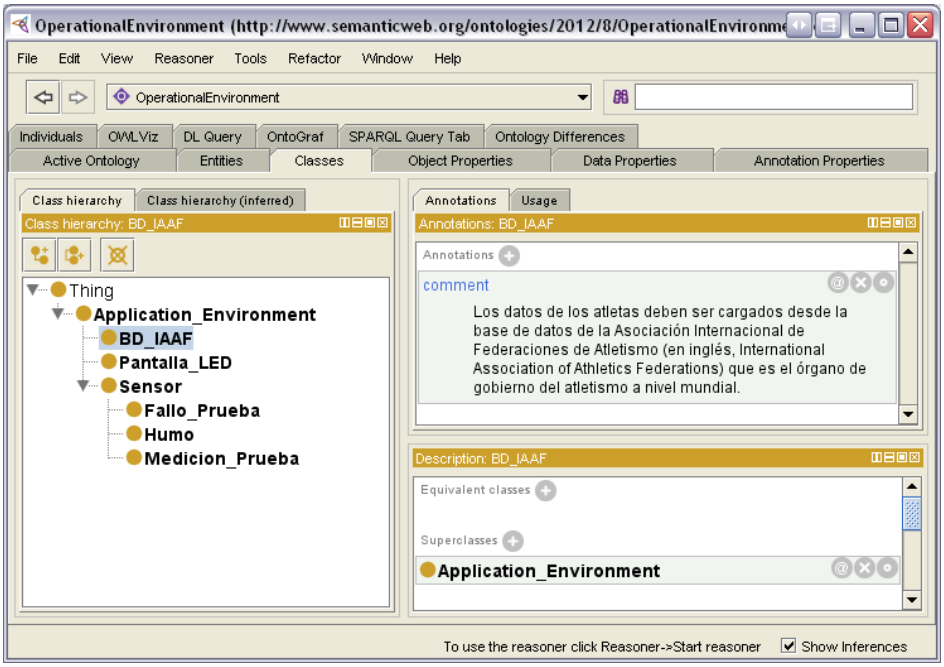


Figura 32 Entorno de aplicación

En este paso el ámbito de entorno operacional fue separado en dos ontologías por la necesidad de expertos muy diferentes en el desarrollo de cada uno. La primera ontología de aplicación del paso 3 es la de *entorno de aplicación* y la segunda la de *stakeholder*.

Stakeholder es un anglicismo en el área de ingeniería del software que es utilizado para referirse a las personas interesadas en el sistema. En el grupo de estas personas interesadas se distinguen las de acepción amplia y las de restringida. El sentido restringido se refiere sólo a aquellos roles o actores sobre los que el producto software depende para su funcionamiento o utilización, mientras que el amplio incluye a los que pueden afectar o que son afectados por el funcionamiento del producto software. En este sentido son más conocidos los stakeholder restringidos como usuarios y los amplios serían todos los demás.

En este contexto, la segunda ontología de aplicación del paso 3 *stakeholder* representa los roles necesarios para su funcionamiento o utilización pero también son tomados en cuenta los que afectan a su desarrollo e instalación. La ejemplificación de esta ontología se puede ver en la Figura 33.

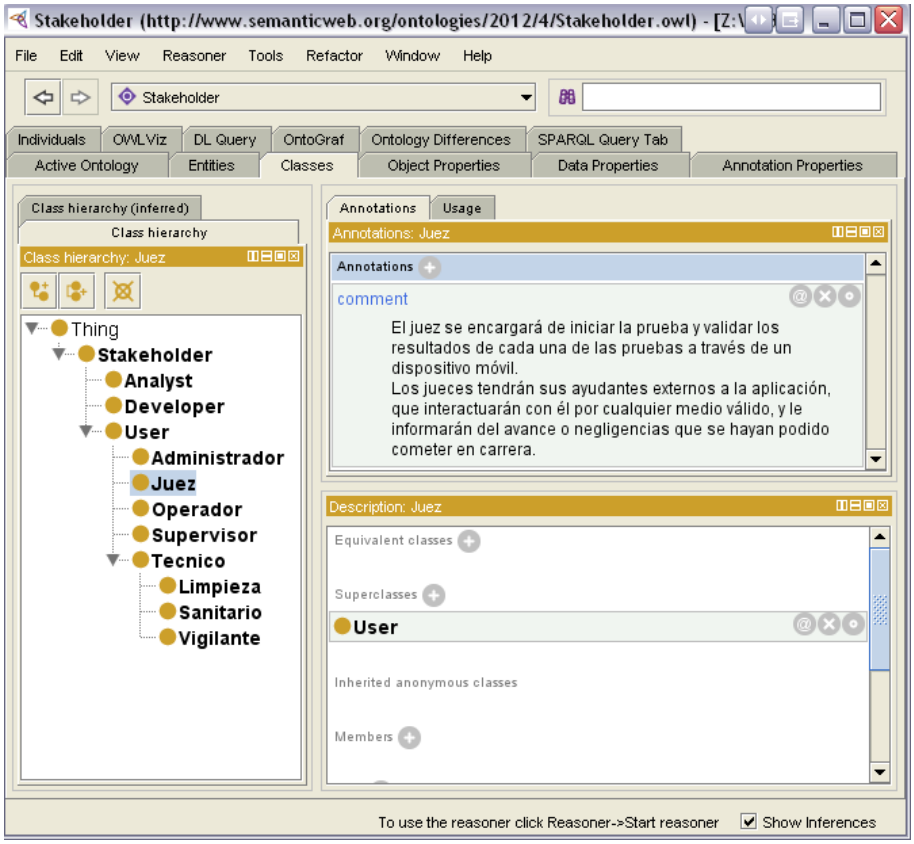


Figura 33 Stakeholder

En esta ontología, los roles especificados en clases representan entidades con atributos que serán añadidos más adelante. Los atributos representan campos que serán dotados de valor durante la ejecución del producto software finalizado.

3.7 Paso 4. Ontología de la arquitectura lógica

En este punto es pertinente recordar que el modelado semántico de los AT permite la gestión eficaz y eficiente de las incumbencias de una ER, pero también permite el control de las incumbencias de los modelos de la vista lógica de la arquitectura.

La última ontología básica de aplicación creada es la del ámbito de arquitectura lógica. En esta tesis doctoral se da por entendido que la vista lógica de la arquitectura representa un modelo semántico que será guía para el diseño posterior de los artefactos software. Este objetivo se consigue con la especialización de este ámbito en las dimensiones de hardware y software.

En la APV propuesta la dimensión de **hardware** está compuesta por los puntos de vista *recursos hardware* y *comunicación interna*. Los recursos hardware se enfocan a las entidades de equipos hardware disponibles y la comunicación interna agrupa incumbencias de los tipos de conexiones entre dichos nodos hardware. Los puntos de vista de la dimensión hardware se utilizan para modelar la topología del hardware sobre el que se ejecutará el sistema.

Es necesario que al mismo tiempo se modelen las incumbencias de la dimensión software por la necesidad de un balance entre las dos dimensiones. En este sentido, se recomienda modelar las incumbencias inicialmente con el equipo hardware existente o indispensable de comprar. Esto es con el fin de no modelar ni más ni menos de lo necesario, partiendo de un modelo de la demisión *hardware* consistente con

posibilidades de su evolución en el tiempo a lo largo del desarrollo del futuro producto software.

El modelado de la dimensión **software** del ámbito de la arquitectura lógica puede considerarse como un conjunto de servicios relacionados agrupados en puntos de vista que representan capas. La idea es modelar capas que sean cohesivas, delimitando claramente las diferencias entre ellas. Esto se logra aplicado los principios de los estilos arquitectónicos clasificados en orientación a capas. En esta tesis doctoral existe un énfasis importante en todo lo relativo a la gestión de la posible evolución del futuro producto software. Por lo tanto, uno de los objetivos de la arquitectura lógica es que se puedan realizar evoluciones con el menor impacto posible en el producto software. En el desarrollo de la APV en secciones anteriores se propusieron con base en estos conceptos los puntos de vista: **Presentación**, **Funcional** y **Base de datos**. En proyectos de sistemas de información estos puntos de vista están sujetos a muchos cambios y es muy importante poder modificar, construir y realizar pruebas sobre dichas capas lógicas de una forma fácil e independiente. Por lo tanto, un objetivo importante es crear puntos de vista en la dimensión de software del ámbito de la arquitectura lógica con el mínimo acoplamiento semántico entre ellos. Las capas aisladas son mucho menos costosas de mantener porque tienden a evolucionar a diferentes ritmos.

3.8 Paso 5. Agregación de ontologías

Esta metodología se descompone en un grupo de ontologías básicas para el modelado de AT porque la complejidad del todo es mayor que la suma de las complejidades de las partes. Otra ventaja es la gestión de recursos dado que cada ontología requiere de diferentes expertos en el dominio. En esta línea, las evoluciones para cada ámbito o dimensión son más fáciles de gestionar. Una ventaja más, que no se cumple en todos los casos, es la posibilidad de reutilizar las ontologías.

En este contexto, hay que componer las ontologías en un todo que en este caso es el modelo completo de AT. Esta acción es realizada con el apoyo de la herramienta *Protégé,* en la cual se **importan las ontologías** a una ontología contenedora que tiene el objetivo de contener las propiedades del modelo que unifica semánticamente el todo, es decir, el modelo de AT. En el caso de estudio "*Estadio",* a la ontología se le ha llamado **ámbito** (ver Figura 34). Las propiedades modeladas van a permitir ubicar y reconocer nuevo conocimiento.

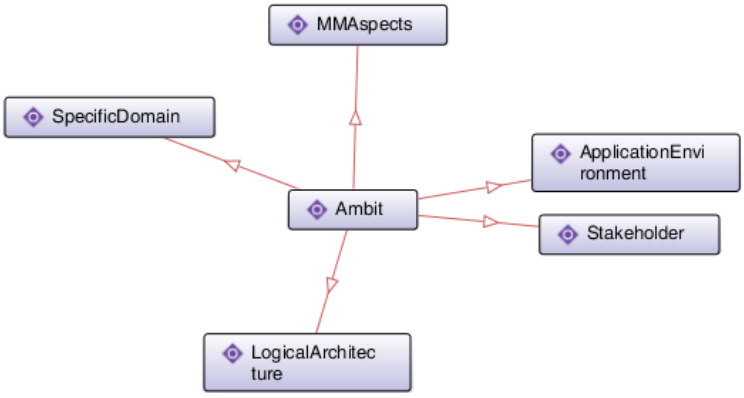


Figura 34 Ontologías del modelo de AT

Los pasos del 1 al 5 de la metodología OCTA sirven para la conformación de la estructura de la base de conocimiento que será conformada en AT que permitirá gestionar las incumbencias dominantes y transversales del futuro producto software.

3.9 Paso 6. Incumbencias dominantes – gestión básica

El objetivo principal del desarrollo de esta metodología es obtener un modelado del conocimiento organizado de los AT de un futuro producto software. En el contexto de esta tesis un modelo de AT está formado por una ER y el modelado de la arquitectura lógica.

A partir de ahora los pasos de la metodología OCTA están enfocados a modelar AT que agrupan incumbencias tempranas que representan requisitos típicos de una ER y capas de arquitectura lógica.

En ocasiones los requisitos se especifican con alto nivel de abstracción, generando muchas suposiciones. En estos casos se suele crear una estructura de requisitos en árbol, donde un requisito de alto nivel se convierte en el requisito contenedor o padre y una serie de requisitos contenidos o hijos surgen para detallar la especificación. Sin embargo, a veces los requisitos hijos se convierten en contenedores al necesitar descendientes hasta obtener los requisitos contenidos que especifiquen su semántica sin suposiciones.

Este formalismo heurístico ayuda a no olvidar información importante en la creación de una ER. Ahora bien, en principio estos últimos requisitos son **indivisibles semánticamente**, por lo tanto, a estos se les podría llamar **requisitos atómicos**.

Ahora bien, en el contexto de la metodología OCTA el modelado de los AT que representan la ER, debe reflejar una estructura en árbol pero basada en el paradigma OA. Es decir, los AT deben contener en la medida de lo posible **incumbencias dominantes o transversales** que representen **requisitos atómicos**, donde los requisitos contenedores son considerados aspectos.

El modelado de incumbencias o requisitos atómicos es un proceso que se puede realizar en una metodología clásica. Sin embargo, nunca se modelan claramente todos los **atributos** de los requisitos y la gestión efectiva de los atributos es la piedra angular del modelado del conocimiento de una ER.

En 7.1 del capítulo de desarrollo teórico se ha dicho que los **atributos** están agrupados en extrínsecos e intrínsecos. Entre estos tipos de atributo los que tienen una mayor repercusión en el modelado semántico son los **atributos intrínsecos**. Por este motivo en la ejemplificación de este caso práctico los atributos extrínsecos no han sido reflejados.

Ahora bien, para gestionar los **atributos** es necesario cumplir con todas las **propiedades** deseables en una ER.

En la sección 7 Calidad en la Especificación de Requisitos del capítulo de desarrollo teórico se ha propuesto un **marco de referencia base para unificar criterios de calidad en el desarrollo de una ER**. Este marco propone que la calidad está relacionada con el cumplimiento de algunas **directrices** que permiten formalizar el cumplimiento de las **propiedades** de calidad.

Dentro del marco de referencia propuesto las directrices de las propiedades **de completitud explícita desde el punto de vista individual y la propiedad de consistencia gramatical** se pueden aplicar en metodologías clásicas, así como en la metodología OCTA propuesta en esta memoria. Sin embargo, el cumplimiento de las directrices de **completitud explícita desde el punto de vista global y la coherencia**

semántica solo se pueden aplicar en metodologías que permiten gestionar la semántica, por lo tanto, estas directrices son las más importantes a cumplir porque tienen un mayor peso en la calidad de una ER.

Ahora bien, la calidad esperada de las propiedades base de completitud explícita desde el punto de vista individual y la propiedad de consistencia gramatical de un ER facilita el análisis de lingüística textual que permite cumplir con las propiedades base de completitud explícita desde el punto de vista global y la coherencia semántica con la metodología OCTA. Sin embargo, con la intención de desarrollar el Caso de estudio "Estadio para eventos de atletismo" como un análisis lingüístico de requisitos en el caso más complejo, no se han reflejado en el caso práctico las directrices de la calidad de las propiedades de **completitud explícita desde el punto de vista individual y la propiedad de consistencia gramatical**, y centrando el esfuerzo en mostrar cómo la metodología OCTA cumple con las directrices de las propiedades de **completitud explícita desde el punto de vista global y la coherencia semántica**.

En la metodología OCTA el modelado semántico de los AT se consigue con las relaciones entre incumbencias dominantes y transversales.

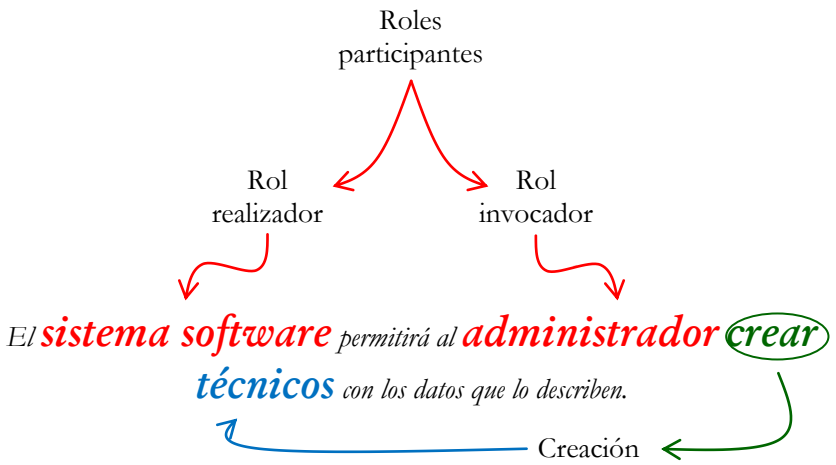
Existen dos principales tipos de propiedades que OWL representa como relaciones, **propiedades de objetos** y **propiedades de tipos de datos**. Las más importantes en la metodología OCTA son las propiedades de objetos, que se utilizan para vincular una incumbencia a otra incumbencia. OWL tiene también un tercer tipo de propiedad, **propiedad de anotación**; estas propiedades se utilizan en OCTA para añadir información. En esta metodología son utilizados para describir las incumbencias como un requisito típico en lenguaje natural para que un creador pueda expresarse sin limitaciones en la ontología que modela los AT.

En base a todo lo anterior, podemos decir que los siguientes pasos de la metodología OCTA, ejemplificada en el Caso de estudio "Estadio para eventos de atletismo", están enfocados a modelar requisitos atómicos y capas de arquitectura lógica con sus atributos intrínsecos a través de los bloques de construcción de una ontología. El modelado de AT debe cumplir con las directrices que permitan cumplir con las propiedades de calidad de completitud explícita desde el punto de vista global y la coherencia semántica.

Una actividad fundamental a desarrollar en todo momento de la aplicación de la metodología OCTA es la descripción en lenguaje natural de un típico requisito o entidad arquitectónica lógica en cada entidad que represente un aspecto o incumbencia a través de la propiedad de anotación.

Así, el paso 6 continúa con la extracción de la semántica de las incumbencias que modelan la funcionalidad dominante realizando un análisis lingüístico de la descripción en lenguaje natural de las descripciones creadas. A través de la ejemplificación se plasman los atributos intrínsecos más típicos encontrados en el desarrollo de la propuesta de la metodología OCTA. Ahora bien para el modelado de estos atributos cumpliendo con las directrices de calidad es a través de las **propiedades objeto** que permiten modelar relaciones en una ontología.

En la APV creada para esta metodología se encuentran tres PV propuestos para modelar aspectos que agrupan incumbencias dominantes. El primer PV propuesto por la APV es "*Basic Management*". Al desgranar los requisitos creados bajo este PV en la creación de modelos con una organización semántica, aparecen los primeros atributos. En esta línea, para la explicación de un análisis lingüístico en incumbencias de este tipo se ha seleccionado el requisito *"Crear Tecnico"*.



Esquema lingüístico 1 Atributos de incumbencia dominante en PV gestión básica

En este primer análisis aparece el atributo intrínseco más identificativo y recurrente en las incumbencias dominantes, nombrado en esta tesis doctoral como **roles participantes**. Este análisis se observa en el ejemplo del Esquema lingüístico 1, es apreciable cómo los roles participantes están compuestos por un rol que invoca y otro que en principio lo realizará. En el mismo requisito se observa que además del atributo de los *roles participantes* se encuentra que, derivado del término *crear,* aparece el atributo **creación**. El criterio para encontrar este tipo de atributos se basa en identificar la gestión simple que el rol invocador solicita, así como la instancia de entidad de información implicada, que el rol realizador cumplirá o delegará.

En OWL las propiedades describen relaciones binarias entre individuos, en la metodología OCTA un atributo intrínseco representa una propiedad. En esta línea, una clase se define por las relaciones entre individuos participantes y en OWL podemos describir dichas clases mediante el uso de restricciones. Por lo tanto, es necesario identificar las propiedades que serán utilizadas para la creación de las restricciones que definen la semántica de las clases de una ontología de AT modelada.

Ahora bien, en la metodología OCTA los aspectos se especifican principalmente a través de restricciones basadas en las relaciones binarias entre las incumbencias que participan. Por lo tanto, en esta metodología al utilizar la herramienta de modelado **Protégé** para la creación de ontologías se deben crear **propiedades de objeto** (del inglés *Object properties*). En este punto es importante comentar que es recomendable que los nombres de las propiedades deben sugerir las restricciones que imponen a los miembros de la incumbencia.

En OWL las restricciones se dividen en tres categorías principales: restricciones de cuantificación, restricciones de cardinalidad y restricciones de valor. Las restricciones más utilizadas son las restricciones de **cuantificación**, las cuales pueden clasificarse en restricciones existenciales y restricciones universales. El cuantificador existencial permite indicar la existencia de al menos un objeto. En protégé la palabra clave **"some"** se utiliza para designar a las restricciones existenciales. La restricción universal permite indicar la existencia de todos los objetos específicos. En este caso para designar a las restricciones universales la palabra clave en Protégé es **"only"**.

En el contexto de este contenido, la metodología propuesta en esta tesis doctoral recomienda crear las *propiedades de objeto* **"invoke"** y **"realize"** para la especificación de las partes del atributo *roles participantes*.

En esta ejemplificación, el atributo roles participantes relaciona la incumbencia *"DataBase"* que está ubicada en la dimensión *"Software"* del ámbito de la *"Logical Architecture"*, vinculada con el *rol realizador*. Ahora bien, en principio la incumbencia relacionada con el *rol invocador* se encuentra en la dimensión *"Stakeholder"* como *"Administrador"*. Sin embargo, en el marco de trabajo de esta metodología el rol invocador debe pertenecer al aspecto *"software"* del ámbito de *"Logical Architecture"*, por lo tanto, dicha incumbencia se encuentra en *"GUI_Administrador"*. Este criterio se debe a que la especificación de los roles participantes que relaciona estas incumbencias implica el razonamiento que justifica las partes lógicas de la arquitectura. Por lo tanto se requiere especificar un conjunto de condiciones **necesarias** y **suficientes** para hacer equivalentes las incumbencias *"Administrador"* y *"GUI_Administrador"*. Este paso es explicado en el paso Incumbencias de *"Logical Architecture"*.

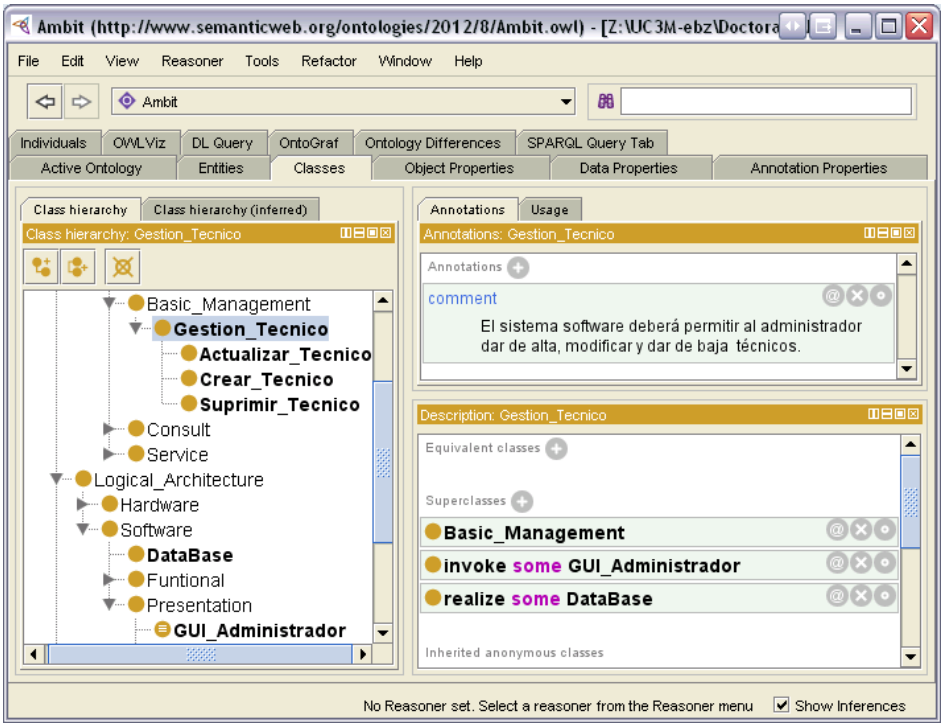


Figura 35 Restricciones de la incumbencia dominante "gestión técnico"

Ahora bien, en la Figura 35 se observa cómo *"Crear Tecnico"* es una incumbencia del aspecto dominante *"Gestion Tecnico"*, siendo complementado con las incumbencias dominantes *"Actualizar Tecnico"* y *"Suprimir Tecnico"*. La descripción en lenguaje natural del aspecto indica que los *roles participantes* son los mismos para todas sus incumbencias, por lo tanto, las restricciones relacionadas con las propiedades objeto *"realize"* e *"invoke"* se modelan únicamente en la clase padre para que todos los hijos hereden esta restricción.

Por otro lado, típicamente todas las incumbencias dominantes del PV de *"Basic Management"* hacen referencia a una entidad de información o dato especificativo de este. En el ejemplo del Esquema lingüístico 1 el término *técnico* refleja un concepto que implica una entidad de información modelada en la dimensión de *"Stakeholder"* del ámbito de *"Operational Environment"*.

Se ha dicho que al analizar lingüísticamente la incumbencia dominante *"Crear Tecnico"* entre los atributos intrínsecos encontrados se identifica uno que relaciona implícitamente los otros atributos encontrados inicialmente. Este atributo indica la acción que solicita el rol invocador que será servido por el rol realizador, haciendo referencia a una entidad de información. Por lo tanto, se requiere sugerir una propiedad objeto para la especificación del atributo *creación* que se observa en el Esquema lingüístico 1. La propiedad es nombrada como **"create"**, que permite la descripción de este atributo a través de la restricción *"create some Crear_Tecnico"* que especifica la descripción de la clase *"Tecnico"* de la ontología de *Stakeholder*. Así, la ontología es capaz de describir todos los individuos que tienen al menos una relación mediante la propiedad *"create"* a un individuo que es miembro de la clase *"Crear Tecnico"*, es decir, describe los individuos que son creados al realizarse la incumbencia dominante de *"Crear Tecnico"*.

La especificación del ejemplo de requisito plasmado en la Esquema lingüístico 1 se puede observar en la Figura 36.

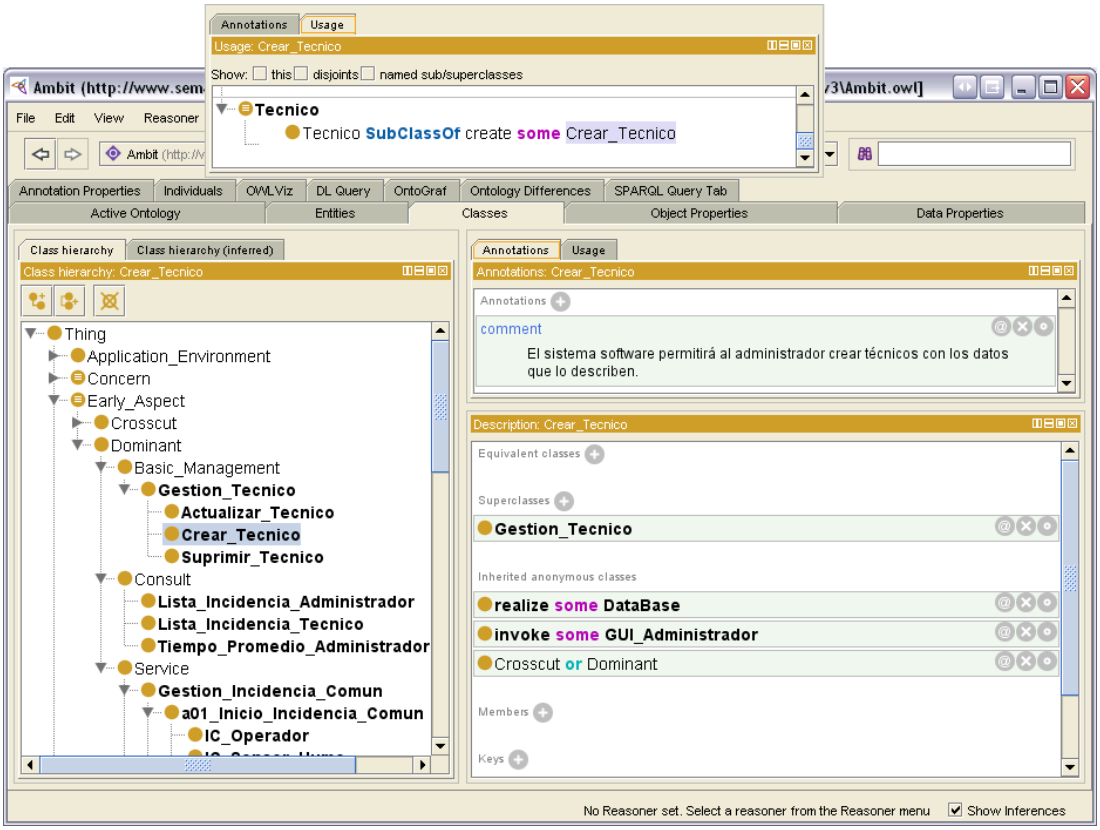


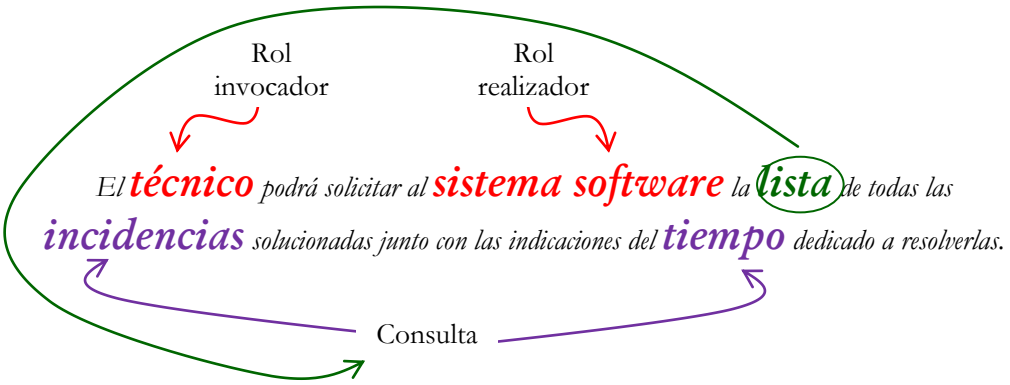
Figura 36 Restricciones de la incumbencia dominante "crear técnico"

En el caso del ejemplo de incumbencia dominante seleccionado *"Crear Tecnico"* se ha encontrado el atributo *crear*, sin embargo, este atributo es sólo uno de los atributos relacionados con "Basic management". Esto se puede observar en la descripción en lenguaje natural de las incumbencias dominantes *"Actualizar Tecnico"* y *"Suprimir Tecnico"*, donde surgen los atributos *actualización* y *supresión* respectivamente.

El segundo PV propuesto por la APV para agrupar aspectos dominantes es *"Consult"*. El desgrane de las incumbencias dominantes modeladas bajo este PV, proporciona criterios que se ejemplifican con el análisis del requisito *"Lista_Incidencia_Tecnico"*.

3.10 Paso 7. Incumbencias dominantes – Consulta

En el análisis mostrado en el Esquema lingüístico 2 se corrobora que el atributo intrínseco recurrente en las incumbencias dominantes es el de *roles participantes*. Así mismo, aparece nuevamente un atributo que indica la acción que se invoca y se realiza pero en este caso hace referencia a datos de información.



Esquema lingüístico 2 Atributos de incumbencia dominante del PV consulta

La especificación del ejemplo de requisito plasmado en el Esquema lingüístico 2 se puede observar en la Figura 37.

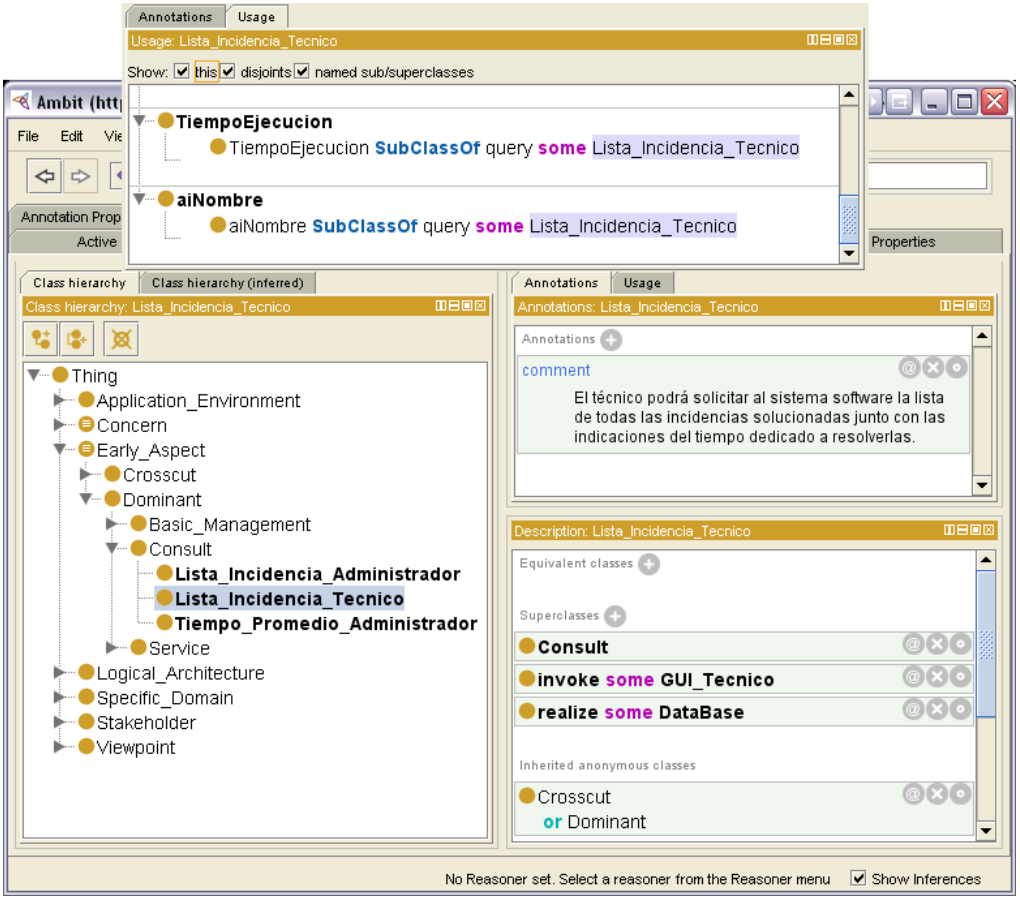


Figura 37 Restricciones de la incumbencia dominante "lista incidencia técnico"

En la incumbencia dominante *"Crear Tecnico"* creada desde el PV *"Basic Management"* se explicó la creación de una restricción que lo relaciona con la entidad *"Tecnico"*, la cual es capaz de referir semánticamente todos sus datos o información. Sin embargo, en la

incumbencia dominante *"Lista Incidencia Tecnico"* creada bajo el PV *"Consult"* requieren referir sólo un dato de la información de una entidad. En este ejemplo el término *lista* refleja la consulta necesaria de datos modelados en incumbencias transversales creadas bajo el PV *"Attribute"* en la dimensión *"Information"*. Por lo tanto, para el modelado de incumbencias que componen los AT dominantes a partir del PV *"Consult"* se recomienda la creación de propiedad de objeto **"query"** para la especificación del atributo propuesto **consulta**.

En la especificación de la incumbencia dominante *"Lista Incidencia Tecnico*" se requiere relacionar con la restricción *"query some Lista_incidencia_Tecnico"* la incumbencia transversal de información *"aiNombre"* que representa el atributo nombre de la clase *"Tecnico"*. Así mismo, con dicha restricción se especifica un atributo derivado agrupado en la incumbencia transversal *"Tiempo Ejecucion"*. Así, se describen los individuos consultados al realizar la incumbencia dominante *"Lista Incidencia Tecnico"*.

3.11 Paso 8. Incumbencias dominantes – Servicio

La última agrupación de aspectos dominantes propuesta por la APV se realiza bajo el PV "S*ervice"*. Los criterios que se aplican al modelado de incumbencias bajo este PV se explican con el análisis del aspecto *"Gestion Incidencia Comun"* que se describe con lenguaje natural como:

El sistema software se encargará de la gestión de incidencias comunes producidas dentro del estadio con la asignación de los recursos de personal necesarios para resolverlas.

En este punto es recomendable recordar que un aspecto está compuesto de incumbencias. Esto se refleja claramente en el aspecto *"Gestion Tecnico"* del PV *"Basic Management"* compuesto por tres incumbencias dominantes. Sin embargo, a veces surgen aspectos que sólo contienen una incumbencia dominante; este es el caso de la incumbencia *"Lista Incidencia Tecnico"* del PV *"consult"*. En el caso de los aspectos creados bajo el PV *"Service"*, en ocasiones algunos requieren un **anidamiento recursivo de aspectos** hasta especificarlos con **incumbencias dominantes** o **requisitos atómicos**. Este es el caso de la descripción textual del aspecto *"Gestion Incidencia Comun"*. Dentro de la disciplina de la IR podría ser un requisito aceptable, sin embargo, al ser una especificación de alto nivel de abstracción genera muchas suposiciones. Por lo tanto, es necesario un aspecto intermedio *"Inicio Incidencia Comun"*. La prestación de estos aspectos intermedios es especificar restricciones comunes de las incumbencias que lo componen.

Por otro lado, normalmente entre las incumbencias dominantes que se agrupan en aspectos creados bajo el PV "service" existen atributos que implican **coherencia semántica secuencial**. Por este motivo, la ejemplificación seleccionada es un grupo de incumbencias dominantes del aspecto *"Inicio Incidencia Comun"* relacionadas respecto a su coherencia semántica secuencial. Esto se hace con el objetivo de comentar los principales criterios a realizar en este paso de la metodología OCTA.

"Inicio Incidencia Comun"
El operador, los técnicos o los sensores de humo podrán iniciar una gestión de incidencia común en el sistema software.

Esta incumbencia o requisito fue creada con la intención de indicar de forma clara cuándo es posible separar la semántica. En este caso derivado de los tres diferentes roles invocadores en el requisito *"Inicio Incidencia Comun"* se generan subrequisitos para su especificación. Estas incumbencias se describen a continuación:

"IC Operador"
El operador inicia la gestión de una incidencia introduciendo los datos en el sistema software con el apoyo de su GUI.

"IC Sensor Humo"
Cuando los sensores de humo se activan el sistema automáticamente inicia la gestión de una incidencia de incendio.

"IC Tecnico"
Cualquier técnico inicia la gestión de una incidencia común introduciendo los datos en el sistema software con el apoyo de su dispositivo móvil.

Un criterio a seguir en los casos como el anterior es identificar la nueva incumbencia padre como un aspecto que contiene incumbencias o requisitos atómicos

"Alta Datos Incidencia"
El sistema software almacenará físicamente los datos de la incidencia en el momento en que se inicia una incidencia.

"Pregunta Equipo Bluetooth"
Una vez iniciada o rechazada una incidencia común el sistema software pregunta a todos los equipos de detección bluetooth la ubicación de todos los técnicos.

"Localizar Equipo Bluetooth"
Cuando los equipos de bluetooth reciben una petición de la ubicación de los técnicos, localizan a estos a través del detector bluetooth.

"Respuesta Equipo Bluetooth"
En el momento en que estén localizados los técnicos el equipo bluetooth manda la información al sistema software.

"Identifica Tecnico"
Una vez recibidas las ubicaciones de los técnicos el sistema software identifica un técnico adecuado al tipo específico de incidencia más cercano sin incidencia principal asignada.

"Alerta Asignacion"
Cuando el sistema software identifica un técnico específico le envía una alerta de asignación de incidencia.

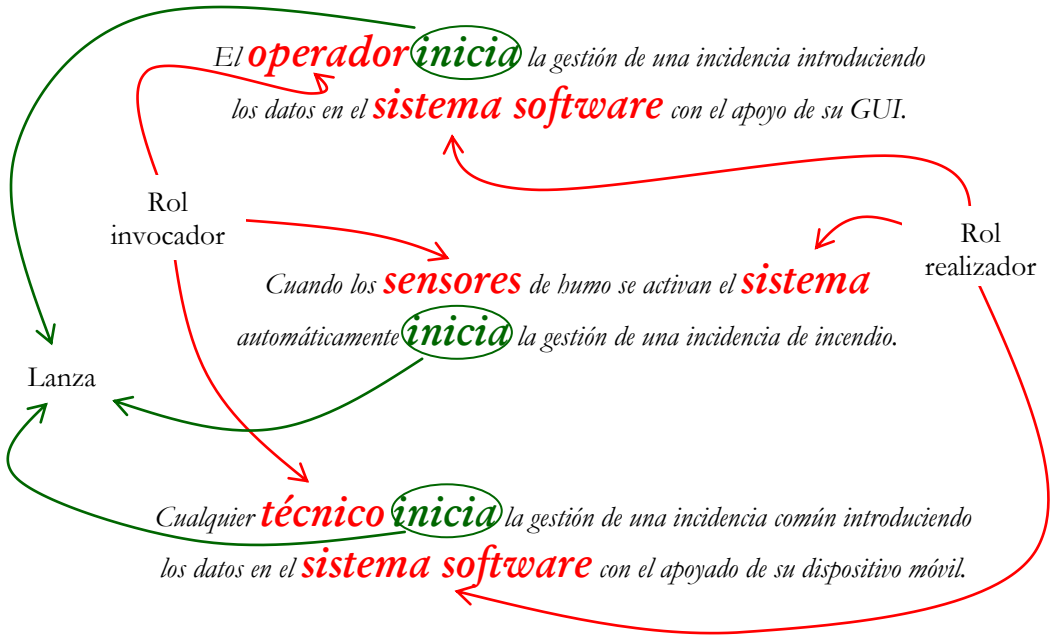
"Asignación Principal"
Al recibir una alerta de asignación principal de incidencia un técnico podrá aceptar o rechazar al sistema software la incidencia".

"Generar Reporte Incidencia"
"Cuando un técnico acepta una incidencia común el sistema software genera un reporte de incidencia".

"Envio Datos Incidencia"
"Cuando una incidencia ha sido aceptada por un técnico el sistema le envía un mensaje con el reporte de incidencia que le permita hacerse cargo de la incidencia".

"Modificar BD Asignacion Principal"
El sistema software indica en la Base de Datos el técnico asignado a la incidencia común en el momento que se genera un reporte de dicha incidencia.

"Necesidad Incidencia"
Cuando un técnico llega a la ubicación de la incidencia asignada avisa al sistema la necesidad de técnicos de apoyo.



Esquema lingüístico 3 Atributo "lanza" de las incumbencias dominantes de "Inicio Incidencia Comun"

Las primeras incidencias dominantes del aspecto *"Gestion Incidencia Comun"* se encuentran en el subaspecto *"Inicio Incidencia Comun"* compuesto por tres incumbencias dominantes: *"IC Operador"*, *"IC Sensor Humo"* y *"IC Tecnico"*.

Se puede observar en el Esquema lingüístico 3 que se realizó un análisis simultáneo a las tres incumbencias involucradas con el objetivo de observar la aparición del nuevo atributo **lanza**. El nuevo atributo es extraído de la identificación del término *inicio* en todas las incumbencias. Este término o algún otro sinónimo es indicativo para la identificación de una posible incumbencia dominante cabecera de un aspecto dominante creado bajo el PV *"Service"*. Estas incumbencias dominantes por lo general implican la referencia de un rol invocador a una entidad modelada en el ámbito de *"Operational Environment"* capaz de lanzar un servicio. De esta forma, la entidad puede estar modelada en la dimensión de *"Stakeholder"* desde cualquiera de sus PV o en la dimensión de "*Application Environment"* desde el PV *"Environment Information"*.

La entidad correspondiente debe especificar la responsabilidad de inicio, por lo tanto, es necesario crear la propiedad objeto **"launch"** para la especificación de las restricciones en las clases responsables. La especificación de las restricciones del atributo *lanza* del ejemplo de requisito plasmado en el Esquema lingüístico 3 se puede observar en la Figura 38.

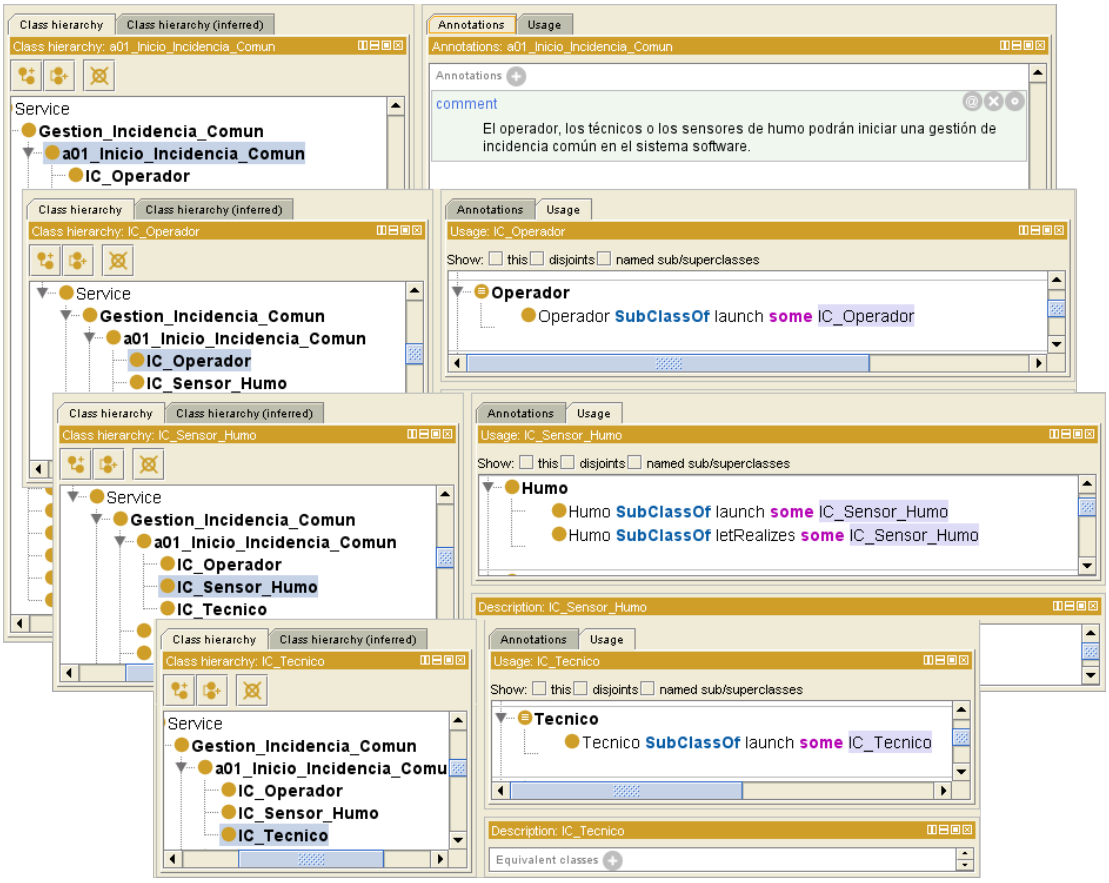
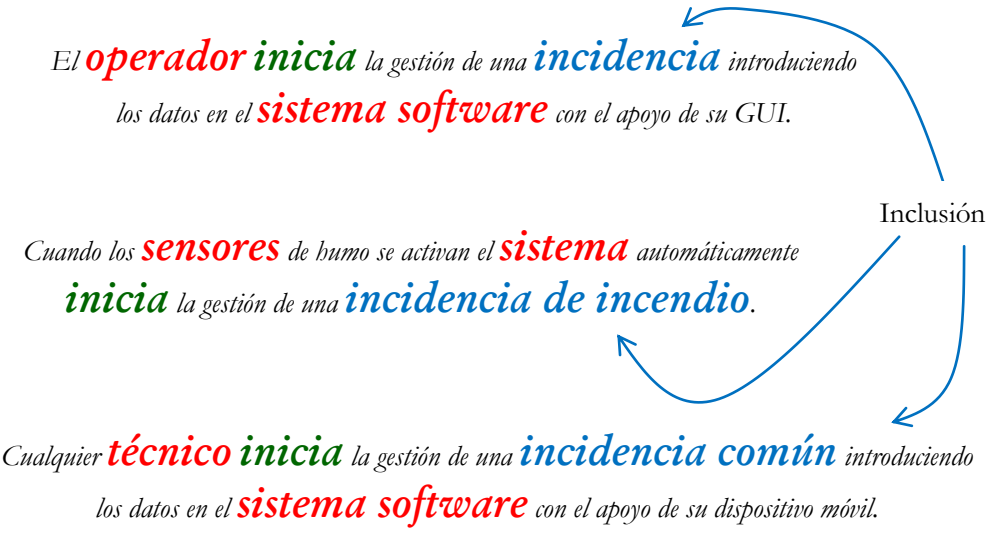


Figura 38 Restricciones "launch" de las incidencias del aspecto "inicio incidencia común"

Al analizar lingüísticamente un poco más las incumbencias dominantes que conforman el aspecto *"Inicio Incidencia Comun"*, las descripciones textuales marcadas en el Esquema lingüístico 4 hablan de la responsabilidad del rol invocador de la introducción de los datos que conforman una instancia de la entidad de información *"Incidencia Comun"*. Por lo tanto, aparece el atributo **inclusión** que propone la propiedad objeto **"include"** para construir la restricción *"Include some Incidencia_Común"* para especificar las incumbencias *"IC_Operador", "IC_Tecnico" y "IC_Sensor_Humo"*, como se puede ver en la Figura 39.



Esquema lingüístico 4 Atributo "inclusión" de las incumbencias dominantes de "Inicio Incidencia Comun"

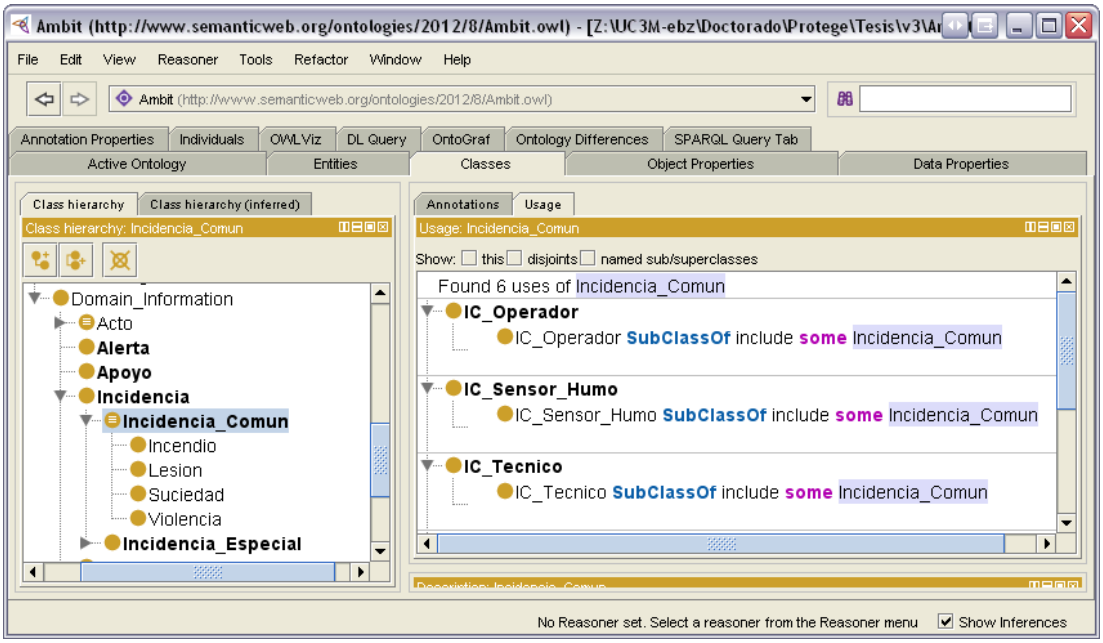


Figura 39 Restricciones "include" de las incidencias del aspecto "inicio incidencia común"

Sin embargo, la especificación de la incumbencia *"IC_Sensor Humo"* es diferente, dado que la creación del individuo se especifica como una *"Incidencia Comun"* del tipo *"Incendio"*. En las instancias de este tipo de entidades se involucran datos predefinidos por el mismo individuo responsable del inicio, por lo tanto, deberán estar presentes en el sistema software en el momento de su realización. No obstante, la especificación de este matiz en el modelado no es necesaria, dado que si la entidad responsable de inicio de una incumbencia cabecera es de la dimensión *"Stakeholder"* se da por supuesto que una persona introducirá los datos a través de una interfaz de usuario y en caso contrario los datos son responsabilidad de diseño del producto software. Por tanto, en esta ejemplificación la semántica de las incumbencias que refinan la identificación del responsable de iniciar un servicio implica una regla de inferencia.

Por otro lado, en párrafos anteriores se justificó la existencia del atributo *consulta* que está directamente relacionado con la lectura de atributos previamente creados en memoria no volátil o la lectura de atributos derivados de dichos datos existentes en la aplicación. En el caso del atributo de *inclusión*, está relacionado con la inclusión de atributos de una entidad de información que guarda estos en una memoria volátil. Este tipo de entidades de información suele ser consultada por un rol realizador para que las entidades sean utilizadas o creadas en memoria no volátil.

Esquema lingüístico 5 Atributos de incumbencia dominante "Alta Datos Incidencia"

En el Esquema lingüístico 5 se desgrana la segunda incidencia de la ejemplificación. Aquí se puede observar que al igual que las otras incumbencias dominantes, contiene atributos de roles participantes. Sin embargo, la descripción en lenguaje natural de este requisito sólo describe el *rol invocador* cuando existe una seguridad implícita de que *el rol realizador* es el sistema software. Por lo tanto, la restricción que especifica el atributo de *rol realizador* debe ser modelada en la ontología de AT para la correcta especificación de la incumbencia dominante *"Alta Datos Incidencia"*.

Así mismo, en esta incumbencia nuevamente surgen las restricciones relacionadas con incumbencias del ámbito de la *"Logical Architecture"*.

Ahora bien, en esta incumbencia dominante a través del término *guardará* surge nuevamente el atributo *creación,* identificado anteriormente como un atributo común en algunas incumbencias creadas en el PV *"Basic Management"*. Por lo tanto, la propiedad objeto necesaria para la especificación de la incumbencia dominante *"Alta Datos Incidencia"* es *"create"*. La especificación de la incumbencia se logra a través de la restricción *"create some Alta_Datos_Incidencia"* especificada en la descripción de la incumbencia transversal *"Incidencia Comun",* la cual ha sido modelada desde el PV *"Business Process"* de la dimensión *"Specific Domain"*.

Así, la ontología principal es capaz de describir todos los individuos que tienen al menos una relación mediante la propiedad *"create"* a un individuo que es miembro de la clase *"Incidencia Comun"*, es decir, describe los individuos que son creados al realizarse la incumbencia dominante de *"Alta_Datos_Incidencia"*.

En este análisis lingüístico también se observa que para ser realizada esta incumbencia está condicionada a la realización de otra u otras incumbencias dentro del mismo aspecto dominante. Esto implica una semántica complementaria, es decir, se describe que sólo es posible la realización de la incumbencia especificada cuando se cumple otra incumbencia predecesora pero al mismo tiempo se describe que solo posteriormente a la incumbencia predecesora se realiza la incumbencia especificada.

En este escenario, surge un nuevo atributo nombrado como **condición**. Ahora bien, para su especificación se requiere la propiedad objeto **"postCondition"** para la descripción de la incumbencia específica e indicar la incumbencia que condiciona su realización. Sin embargo, también es importante describir en la incumbencia predecesora que especifica el estado en que debe estar la aplicación antes de la realización de la incumbencia específica, es decir, especificar cuál es la incumbencia que representa precondición. Por lo tanto, también se requiere la propiedad objeto **"preCondition"** para complementar el atributo *condición*.

Por lo tanto, para la especificación de la incumbencia *"Alta Datos Incidencia"* de esta ejemplificación se crea la restricción *"postCondition some Inicio_Incidencia_Comun"* que es utilizada para describir la incumbencia dominante *"Alta Datos Incidencia"* y la restricción *"preCondition some Alta_Datos_Incidencia"* para ser utilizada en la descripción de la incumbencia dominante "*Inicio_Incidencia_Comun"*. En este caso para completar la especificación de la incumbencia dominante "*Inicio_Incidencia_Comun"* también se debe especificar una precondición de la incumbencia *"Pregunta Equipo Bluetooth"*. La especificación de los atributos plasmados en el requisito del Esquema lingüístico 5 se puede observar en la Figura 40.

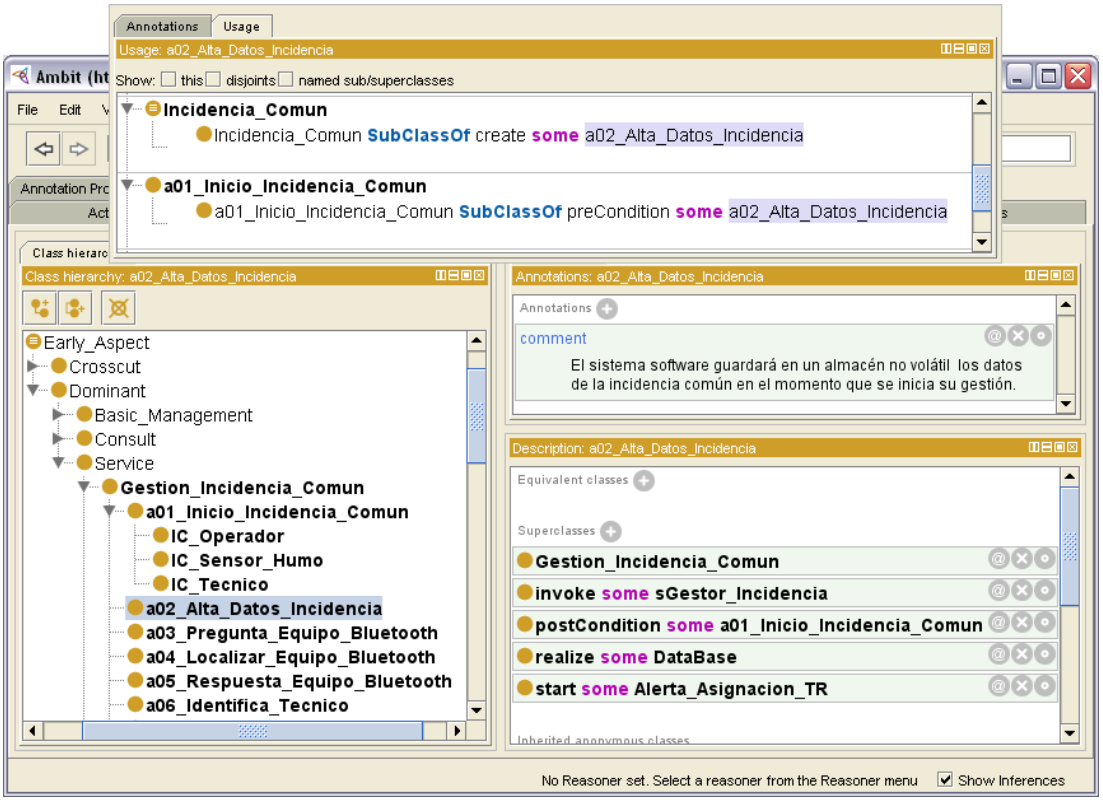


Figura 40 Restricciones de la incumbencia dominante "alta datos incidencia"

En el contexto de la semántica de las propiedades objeto del atributo condición se puede extraer que *"preCondition"* es la propiedad inversa de *"postCondition"*, lo cual permite inferir en ambos sentidos una de otra. La especificación de que una propiedad objeto es inversa de otra se especifica en la descripción de cualquiera de las propiedades objeto *"preCondiciones"* o *"postCondiciones"*. Así mismo, se puede deducir que las propiedades objeto de *"preCondiciones"* y *"postCondiciones"* utilizadas en la especificación de las incumbencias creadas desde el PV de *"Service"* constituyen una materia prima importante de las pruebas de verificación de incumbencias que debemos identificar para una mayor claridad del entendimiento del riesgo que existe en una evolución del producto software en las incumbencias dominantes que básicamente representan la funcionalidad esperada.

Esquema lingüístico 6 Atributos de incumbencia dominante "Pregunta Equipo Bluetooth"

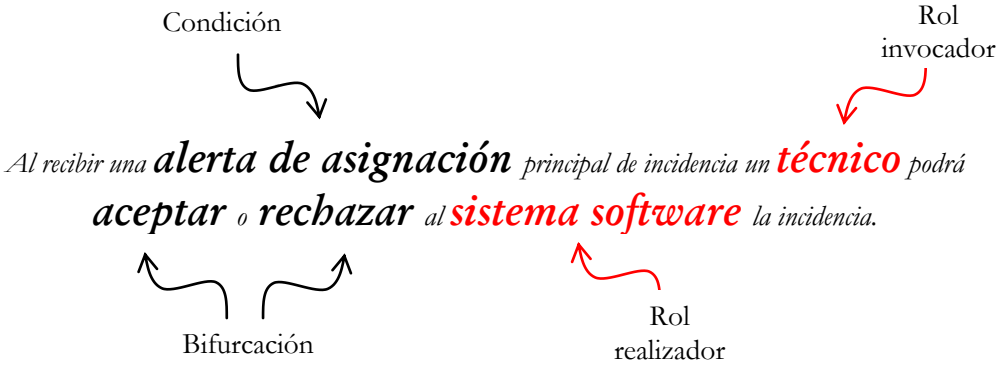
La siguiente incumbencia *"Pregunta Equipo Bluetooth"* es analizada en el Esquema lingüístico 6. Al desgranar el requisito atómico se puede apreciar la repetición de los atributos *roles participantes* y *condición*. Sin embargo, en esta ocasión la reaparición del

atributo *condición* contiene un matiz nuevo. En el análisis lingüístico de la incumbencia *"Pregunta Equipo Bluetooth"* se observan dos precondiciones posibles, es decir, existen dos posibles incumbencias que cambian el estado de la aplicación para que se pueda realizar la incumbencia. Por lo tanto, con la propiedad objeto *"preCondition"* se construye la restricción *"preCondition some Pregunta_Equipo_Bluetooth",* la cual es especificada en las incumbencias transversales *"Inicio Incidencia Comun"* y *"AP Rechazo"*. Así, la ontología de AT es capaz de describir los individuos que son precondición para la realización de la incumbencia dominante *"Pregunta Equipo Bluetooth"*.

En el mismo análisis el término *pregunta* identifica un atributo que en primera instancia podría identificarse como el anteriormente definido como *consulta*. Sin embargo, este atributo identificado en las incumbencias dominantes creadas desde el PV *"Consult"* se refiere a la consulta de información que se encuentra en un almacenamiento no volátil. En este caso el atributo indica una solicitud de información que aún no está recopilada, por lo tanto, aparece un nuevo atributo denominado **solicitud**. En la misma línea de los otros atributos se requiere sugerir una propiedad objeto para la especificación del atributo *solicitud*. En este caso la propiedad es nombrada como **"request".** La especificación de este atributo es a través de la restricción *"request some Pregunta_Equipo_Bluetooth"* con el objetivo de describir la clase *"aLocalizacionGeografica"*, que se puede observar en la Figura 42. Así, con esta especificación la ontología es capaz de describir los diferentes individuos que representan un estado de un atributo de la entidad de información *"Tecnico"*.

Siguiendo con la secuencia semántica condicional entre las incumbencias dominantes, con el objetivo de exponer los criterios dentro de la metodología OCTA, la siguiente incumbencia secuencial es *"Asignacion Principal"* que en lenguaje natural se enuncia: *Al recibir una alerta de asignación principal de incidencia un técnico podrá aceptar o rechazar al sistema software la incidencia*.

Esta incumbencia dominante es analizada en el Esquema lingüístico 7. Ahí aparecen nuevamente los atributos *roles participantes* y *condición*. Así mismo, se identifica el nuevo atributo **bifurcación**, el cual requiere modelar la semántica de diferentes posibilidades de estado en las que quedará la aplicación cuando sea realizada alguna de dichas incumbencias.



Esquema lingüístico 7 Atributos de incumbencia dominante "Asignacion Principal"

En párrafos anteriores se explicó la posibilidad de la aplicación de patronos ontológicos para dar solución a problemas de modelado que se han dado una y otra vez. Así, para solucionar el problema de modelar un atributo *bifurcación* se ha propuesto un patrón basado en el patrón *partición valorada*.

Patrón *bifurcación* es como se ha llamado al aplicado a esta ejemplificación. Lo primero es modelar una bifurcación para restringir el rango de valores posibles necesarios en la descripción de incumbencia *"Asignacion Principal"*.

Este objetivo es posible creando las subclases necesarias que representen las opciones de respuesta en la bifurcación, en este caso son *"AP_Aceptacion"* y *"AP_Rechazo"*. Así mismo, se requiere que dichas subclases sean disjuntas, así como proporcionar un axioma de cobertura para hacerlas como una unión que se refiere únicamente a su superclase. Es sabido que para la especificación de atributos se requieren propiedades objetos, en este caso son **"accept"** y **"refuse",** uno para cada subclase. Finalmente las propiedades deben ser limitadas a tener un valor único, es decir, es necesario indicar que son propiedades funcionales.

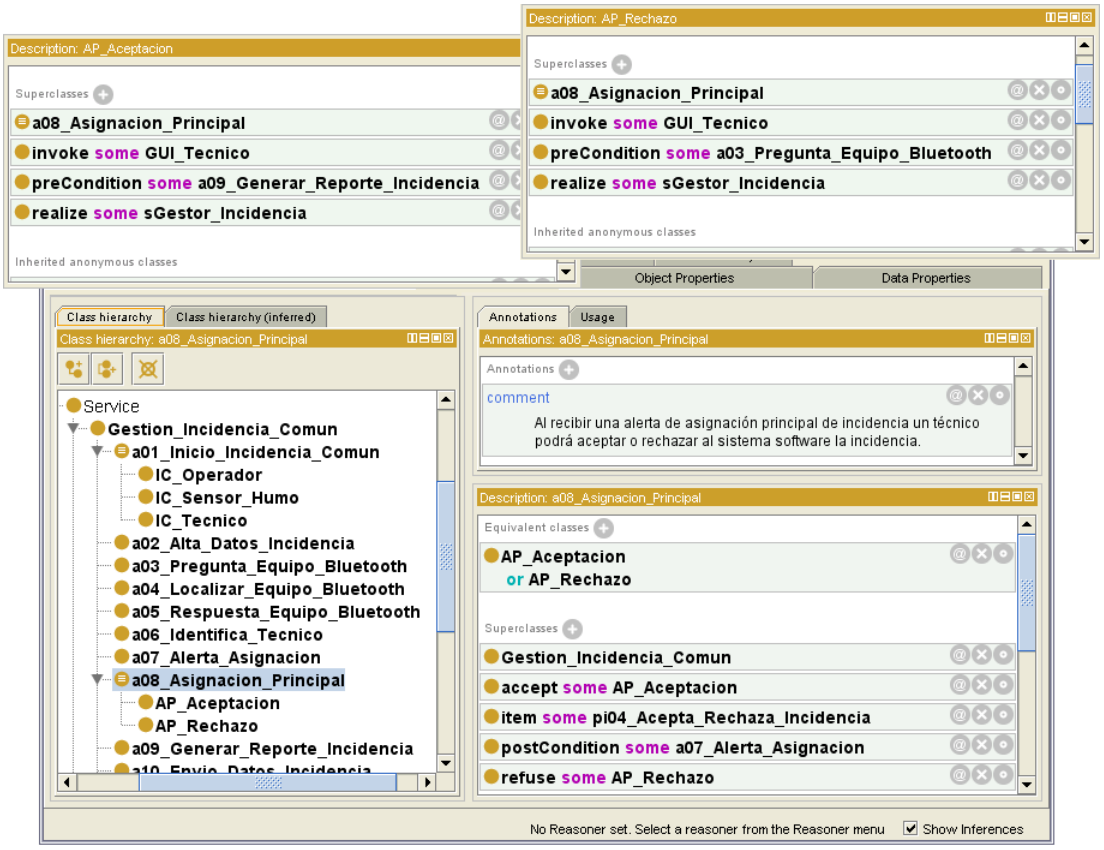


Figura 41 Restricciones de la incumbencia dominante "asignación principal"

En la entidad *"Inicio Incidencia Comun"* se identificó que representaba un aspecto anidado en el aspecto *"Gestión Incidencia Comun"*, el cual era necesario especificar con un grupo de incumbencias o requisitos atómicos. Este criterio no se cumple al crear subclases que representan incumbencias al aplicar el patrón bifurcación, dado que la finalidad es especificar las opciones de respuesta en la bifurcación. En este criterio las entidades hijas creadas de la entidad padre son consideradas como incumbencias. Por lo tanto, para su especificación es necesaria la creación de las restricciones *"acepta some AP_Aceptacion"* y *"rechaza some AP_Rechazo"* para describir la incumbencia dominante *"Asignacion Principal"*. Esta especificación se puede observar en la Figura 41, donde se describe la incumbencia general con sus incumbencias contenidas. Aquí mismo se puede observar la especificación descrita anteriormente de la incumbencia *"AP_Rechazo"*, utilizando la restricción *"preCondition some Pregunta_Equipo_Bluetooth"* para complementar el atributo *condición* de la incumbencia *"Pregunta_Equipo_Bluetooth"*. Así mismo, en la incumbencia

"AP_Acepto" está especificada la restricción *"preCondition some Generar_Reporte_Incidencia"* especificando el atributo *condición* de la incumbencia *"Generar Reporte incidencia"*.

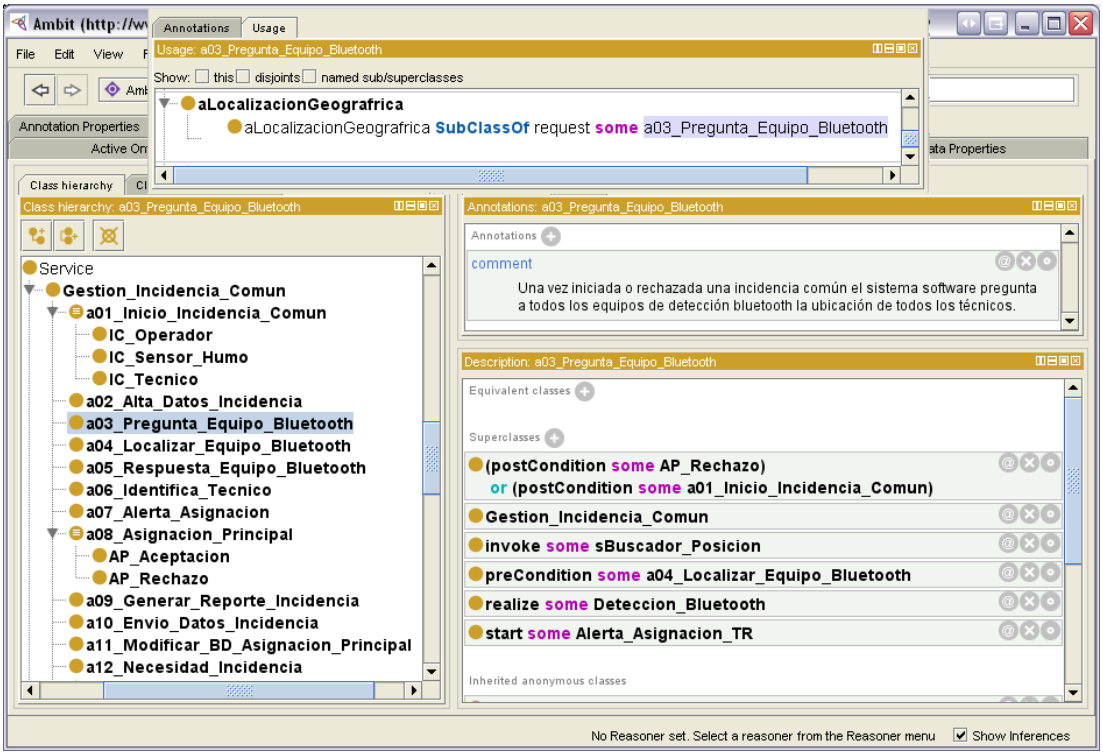


Figura 42 Restricciones de la incumbencia dominante "pregunta equipo bluetooth"

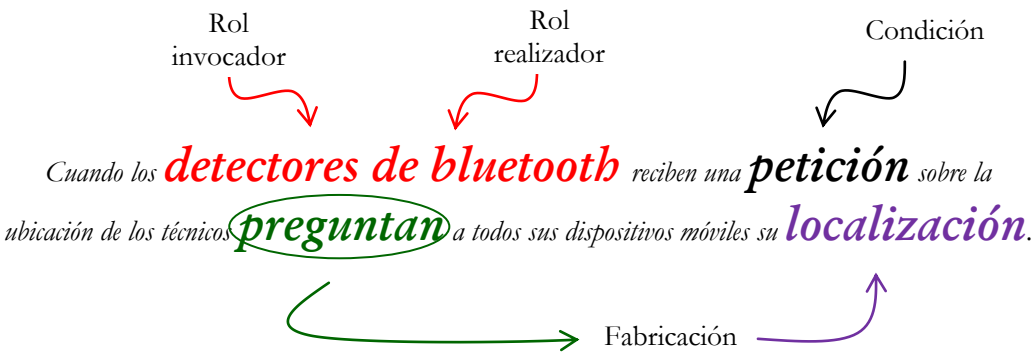
El atributo *condición* generó la secuencia semántica entre incumbencias que nos llevó a la incumbencia *"AP_Rechazo"*, sin embargo, la secuencia semántica también puede ser múltiple. Así es como de la incumbencia *"Pregunta_Equipo_Bluetooth"* además de la relación secuencial con *"Asignación Principal"* y con la incumbencia *"Inicio Incidencia Comun"*, también se tiene una relación secuencial con la incumbencia *"Localiza Equipo Bluetooth"*. Esta especificación se muestra en la Figura 42.

Aquí se puede observar cómo la especificación del atributo condición se realiza con la restricción *"preCondition some Localizar_Equipo_Bluetooth"*. Así mismo, con las restricciones *"postCondition some AP_Rechazo"* y *"postCondition some Inicio_Incidencia_Comun"* y utilizando el operador *unionOf* entre ellos representado en *Protégé* como *or*, se indica que para realizar la incumbencia *"Pregunta_Equipo_Bluetooth"* es necesario que se realice la incidencia *"AP_Rechazo"* o la incumbencia *"Inicio Incidencia Comun"*.

Los bloques de construcción de menor nivel de abstracción en la metodología OCTA son las incumbencias dominantes y las incumbencias transversales. Las incumbencias transversales cortan las incumbencias dominantes pero no existe secuencia semántica entre estas incumbencias, únicamente existe secuencia semántica entre requisitos dominantes, dado que son las únicas incumbencias que pueden tener la propiedad de cambiar el estado de la aplicación.

Con base en la secuencia semántica de la incumbencia *"Pregunta Equipo Bluetooth"*, la siguiente incumbencia a analizar es *"Localizar Equipo Bluetooth"*. Esta es analizada en el Esquema lingüístico 8, donde nuevamente se aprecian atributos recurrentes y uno nuevo. **Fabricación** es el nuevo atributo encontrado en esta incumbencia dominante a través del término *preguntan* que indica la acción de recopilar la localización geográfica de

los técnicos. *"aLocalizacionGeografica"* es un atributo que los individuos de *"Tecnico"* deben contener con un dato definido.



Esquema lingüístico 8 Atributos de incumbencia dominante "Localizar Equipo Bluetooth"

En la ejemplificación de este caso práctico, los equipos detectores de bluetooth son los encargados de obtener la localización geográfica a través de su tecnología, la cual detecta la magnitud de la señal bluetooth emitida por cada dispositivo móvil y en función de este dato calcula su ubicación.

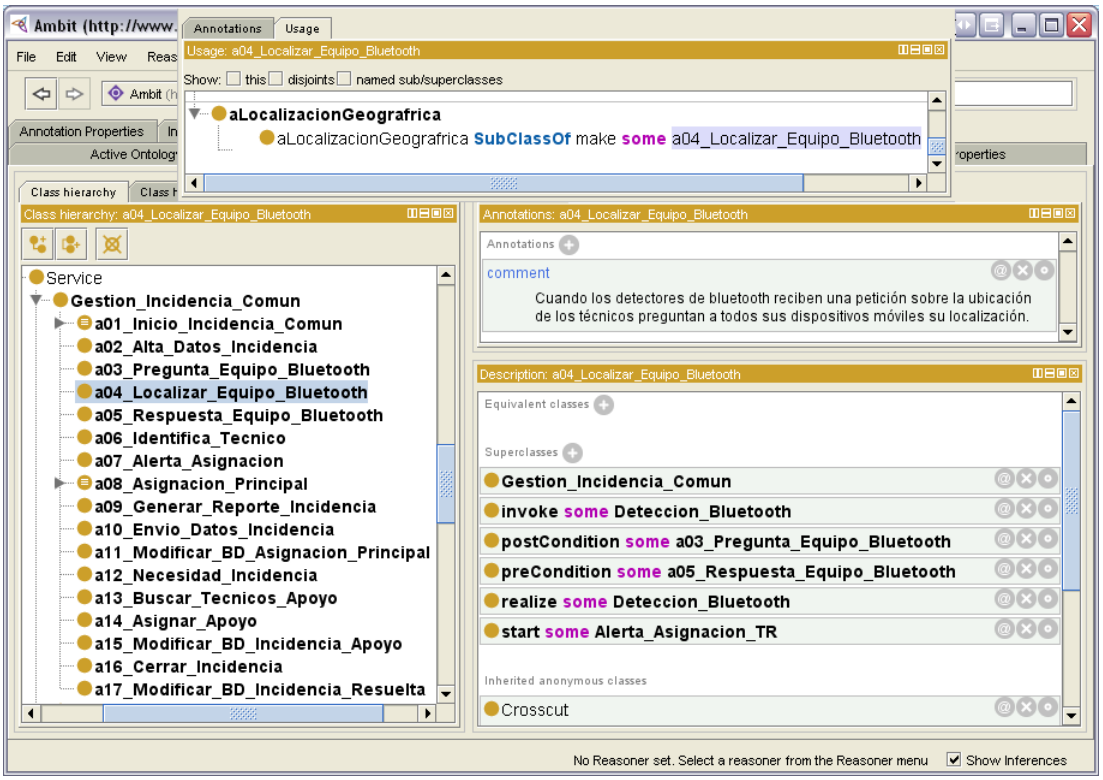
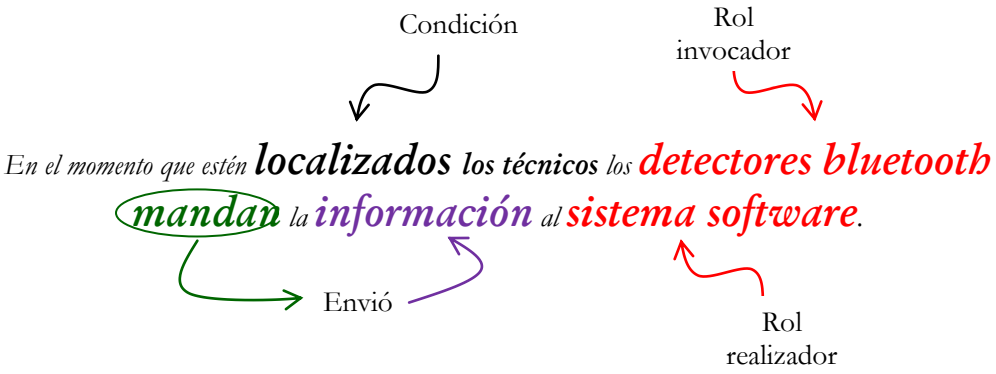


Figura 43 Restricciones de la incumbencia dominante "localizar equipo bluetooth"

En el análisis lingüístico de la incumbencia *"Localizar Equipo Bluetooth"* se observa cómo el *rol invocador* y el *rol realizador* son la misma entidad, donde la condición para iniciar esta recopilación de datos era que se lo solicitaran. Así, para especificar el atributo *fabricación* se propone la propiedad **"make"** para construir la restricción *"make some Pregunta_Equipo_Bluetooth"* que será descrita en la incumbencia transversal *"aLocalizacionGeografica"* que se puede ver en la Figura 43.

En este punto es oportuno manifestar la diferencia entre los atributos *"solicita"* y *"fabricación"* para evitar confusión en su modelado. El atributo *"solicita"* modela una

delegación de la responsabilidad que puede ser transmitida con el mismo atributo las veces que sean necesarias hasta que finalmente es realizada por la incumbencia que contiene el atributo *"fabricación"*.



Esquema lingüístico 9 Atributos de incumbencia dominante "Respuesta Equipo Bluetooth"

La siguiente incumbencia dominante analizada es la que se encuentra en el Esquema lingüístico 9 nombrada como *"Respuesta Equipo Bluetooth"*. Es clara la relación de secuencia semántica con la incumbencia *"Localizar Equipo Bluetooth"* a través de su atributo *condición*.

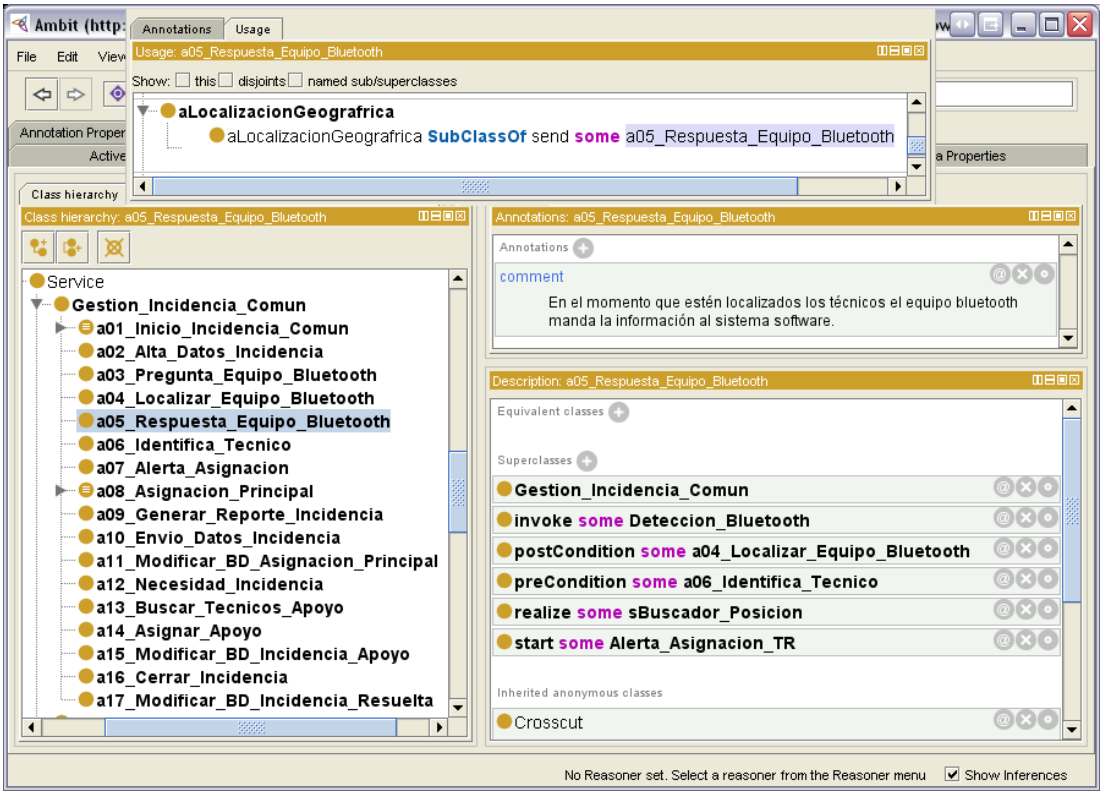
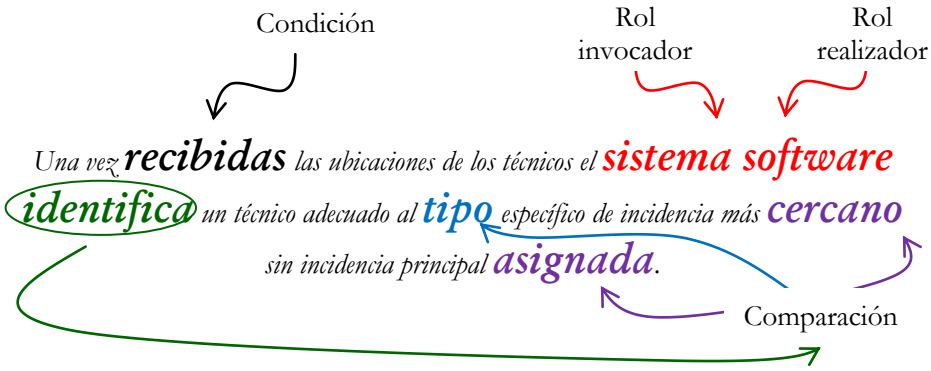


Figura 44 Restricciones de la incumbencia dominante "respuesta equipo bluetooth"

Aparece nuevamente el atributo de los roles participantes pero a diferencia de la incumbencia anteriormente analizada aquí los *roles participantes* son diferentes. Así mismo, aparece el nuevo atributo **envió** al cual se propone la propiedad **"send"** para su especificación a través de la restricción *"send some Respuesta_Equipo_Bluetooth"* descrita en la clase *"aLocalizacionGeografica"*. Esta especificación de la incumbencia dominante *"Respuesta Equipo Bluetooth"* puede ser observada en la Figura 44.

Se ha explicado la diferencia entre los atributos *solicita* y *fabricación*. A este respecto, aparece el atributo *envió* para complementar una trinidad de atributos. El modelado de uno de ellos implica la especificación de los otros dos, por lo tanto, esto proporciona una regla para la verificación de la coherencia semántica de una ER.



Esquema lingüístico 10 Atributos de incumbencia dominante "Identifica Tecnico"

En el Esquema lingüístico 10 se analiza la incumbencia dominante *"Identifica Tecnico"*, donde encontramos nuevamente cómo los *roles participantes* son la misma entidad. Así mismo, se identifica el atributo *condición* que proporciona la secuencia semántica.

Ahora bien, se concibe el nuevo atributo **comparación** con la propiedad objeto **"compare"** para la especificación de las restricciones. Al igual que los anteriores atributos *comparación* se puede especificar en diferentes ocasiones a lo largo de un modelado de una ER. Sin embargo, este atributo no puede ser repetido con el mismo nombre, dado que el problema a resolver se repite constantemente pero los datos a comparar son constantemente variados. Por lo tanto, se recomienda que dicho atributo sea enumerado de forma ascendente, tal como, *"compare1"*, *"compare2"*, *"compare3"*, etc. También es necesario que dichas propiedades objeto se agrupen como subpropiedades de una propiedad padre denominada "*compare"*. Este criterio de modelado permite identificar todas las entidades participantes del mismo tipo de atributo.

En la incumbencia analizada aparecen diferentes términos que identifican el atributo *comparación*. En esta línea, en base a criterios anteriores es factible convertir la incumbencia *"Identifica Tecnico"* en aspecto y crear nuevas incumbencias o requisitos atómicos que describan cada una de las diferentes comparaciones especificadas. Sin embargo, en esta ejemplificación se optó por dejarlo con su primera descripción para demostrar la libertad de esta decisión, cumpliendo únicamente con el criterio de tener la seguridad de gestionar la complejidad sin problemas.

En OWL se puede describir la clase con los individuos que tienen como máximo, por lo menos o exactamente un número determinado de relaciones con otros individuos o valores de tipo de datos. Las restricciones que describen estas clases se conocen como restricciones de **cardinalidad**.

Una restricción de **cardinalidad mínima** especifica el número mínimo de relaciones en que un individuo debe participar. Una restricción de **cardinalidad máxima** especifica el número máximo de relaciones que un individuo puede participar. Una restricción **cardinalidad exacta** especifica el número específico de relaciones en que un individuo debe participar.

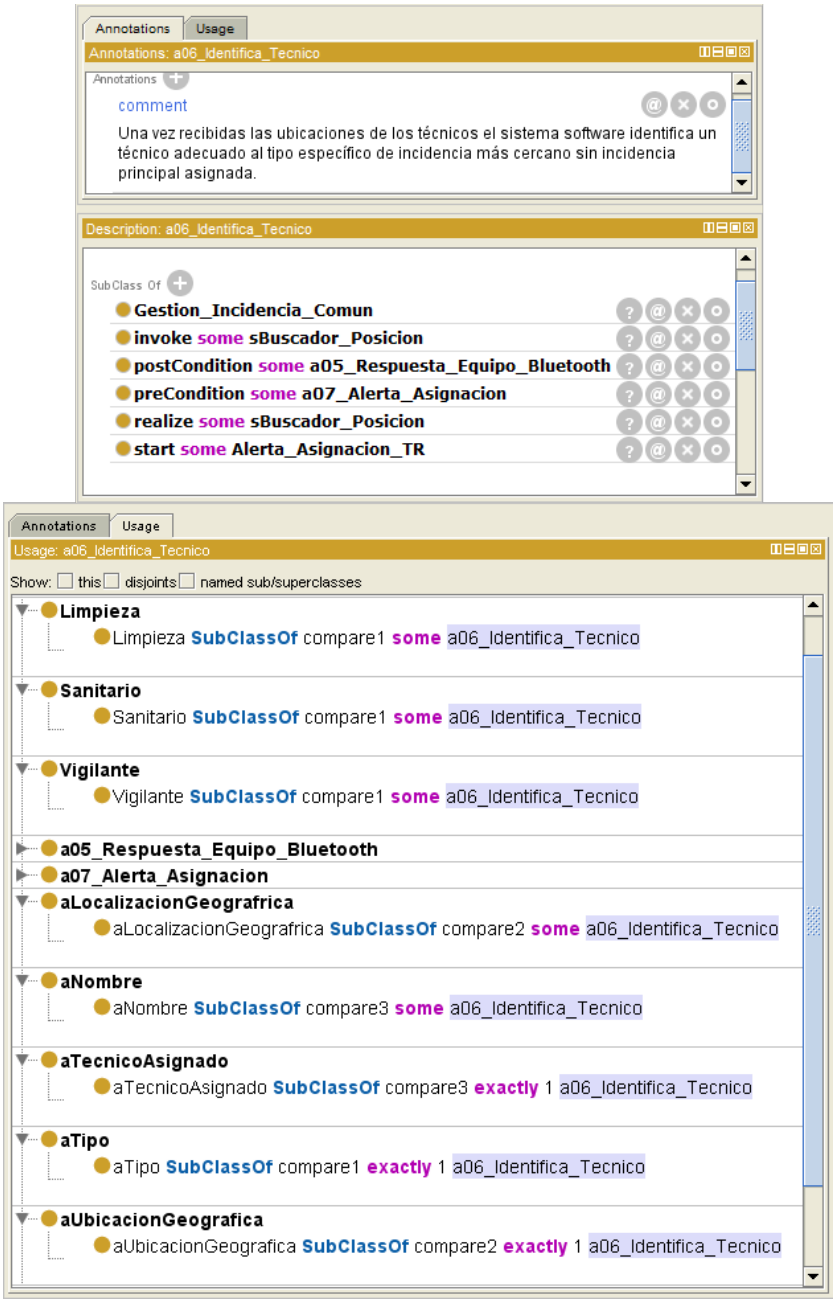


Figura 45 Restricciones de la incumbencia dominante "identifica técnico"

La Figura 45 muestra la descripción de la incumbencia *"Identifica Tecnico";* esta se inicia con la utilización de la propiedad *"compare1"* para la creación de la restricción *"compare1 some Identifica_Tecnico"* que especifica los tipos de técnicos *"Limpieza"*, *"Sanitario"* y *"Vigilante"*. Al mismo tiempo se crea la restricción *"compare1 exactly 1 Identifica_Tecnico"* que especifica la clase *"aTipo"*. En este caso la restricción utiliza una restricción de cardinalidad exacta para especificar que la comparación es de muchos tipos de técnicos localizados contra el tipo de incidencia común.

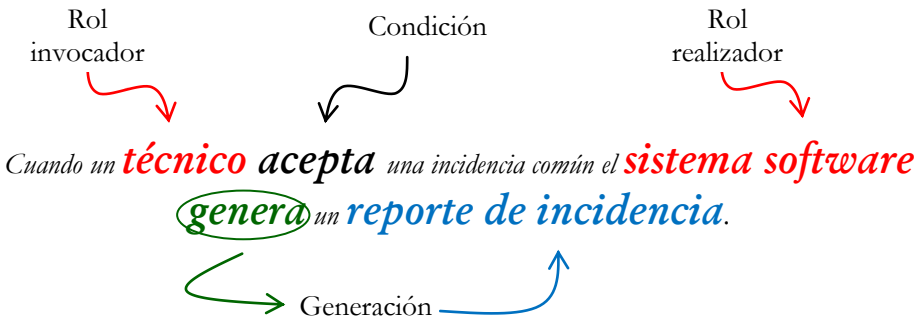
La siguiente propiedad creada *"compare2"* es utilizada para la creación de la restricción *"compare2 some Identifica_Tecnico"* para especificar la entidad *"aLocalizacionGeografica"*. Como en el caso anterior se crea otra restricción que en este caso es *"compare2 exactly 1 Identifica_Tecnico"* para especificar la entidad *"aUbicacionGeografica"* que permite modelar la semántica de la identificación del técnico más cercano a la incumbencia común en proceso. La tercera y última propiedad utilizada es *"compare3"* que permite crear la

restricción *"compare3 some Identifica_Tecnico"* para especificar *"aTecnicoAsignado"* y *"compare3 exactly 1 Identifica_Tecnico"* para la especificación de *"aNombre"* que en conjunto permite modelar la semántica de la identificación de un técnico que no tenga asignada una incidencia común.

Esquema lingüístico 11 Atributos de incumbencia dominante "Alerta Asignacion"

El siguiente requisito atómico se puede observar en el Esquema lingüístico 11 con la reaparición de los atributos *roles participantes*, *condición* y *envío*. Estos atributos ya han sido ejemplificados en criterios anteriores. Sin embargo, lo que se pretende mostrar en el desgrane de esta incumbencia es la forma en que la metodología OCTA encuentra nuevas entidades de información.

En pasos anteriores se crearon entidades de información que están justificadas con la existencia de atributos que los describen; ahora en la incumbencia *"Alerta asignación"* se puede observar cómo surgen entidades que hasta ese momento no existen porque no pueden ser especificadas con atributos. En este caso, a través del término *alerta* aparece claramente una entidad de información a la que el rol invocador envía al rol realizador. Posterior a la creación de esta nueva entidad en la ontología *"Domain Information"*, el atributo *envió* apoyado en la propiedad objeto *"send"* especifica la entidad *"Alerta"* a través de la restricción *"send some Alerta_Asignacion"*.



Esquema lingüístico 12 Atributos de incumbencia dominante "Generar Reporte Incidencia"

"Generar Reporte Incidencia" es la siguiente incumbencia de esta ejemplificación, la cual está analizada en el Esquema lingüístico 12. Aquí, nuevamente se observan los atributos de roles *participantes* y *condición*. Así mismo, aparece el nuevo atributo **generación** a través del término *generar*, que propicia una nueva propiedad objeto nombrada como **"generate"** para la descripción de la restricción *"generate some Generar_Reporte_Incidencia"* en la descripción de la clase *"Reporte Incidencia Comun"*. Así, la ontología general es capaz de describir los individuos que son generados al realizarse dicha incumbencia dominante. La especificación en Protégé de la incumbencia *"Generar Reporte Incidencia"* se puede observar en la Figura 46.

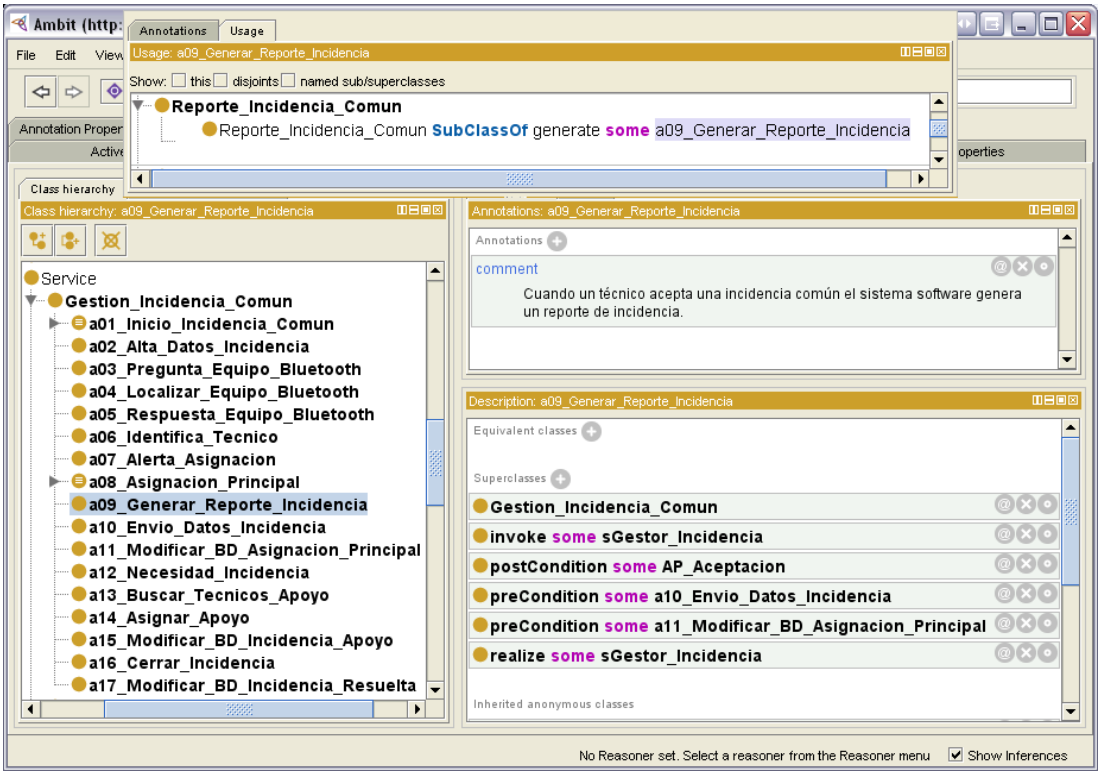


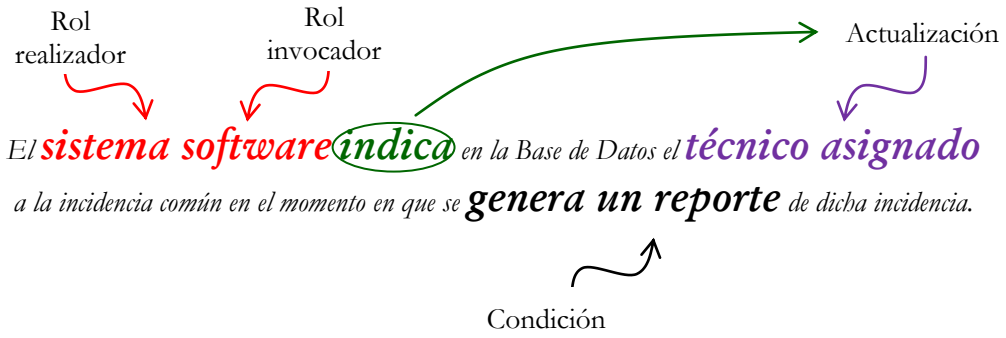
Figura 46 Restricciones de la incumbencia dominante "generar reporte incidencia"

Ahora bien, en el Esquema lingüístico 8 se vio que para la especificación de la incumbencia dominante *"Localizar Equipo Bluetooth"* se utilizó la propiedad objeto *"make"* utilizada en el atributo *fabricación* para la construcción de la restricción necesaria para la descripción de la incumbencia transversal *"aLocalizacionGeografica"*. En este caso, la creación de sus individuos requiere información fabricada por el software perteneciente al dispositivo hardware responsable, donde es previsible que el software exista previamente al inicio de la codificación del producto software prometido. Sin embargo, en el caso de la relación de la incumbencia *"Generar Reporte Incidencia"* con la entidad *"Reporte Incidencia Comun"* es diferente, en este caso, la creación de individuos requiere datos que serán generados con el futuro producto software. De esta forma, está contenida en el modelado la diferencia semántica de la responsabilidad de las diferentes partes del software.

Esquema lingüístico 13 Atributos de incumbencia dominante "Envio Datos Incidencia"

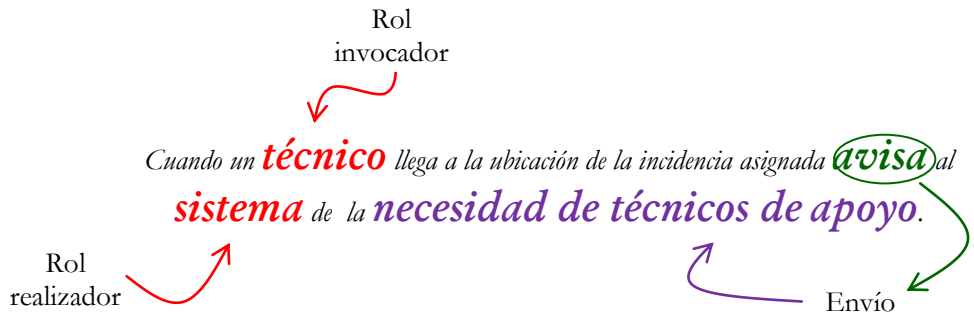
La siguiente incumbencia en esta ejemplificación de incumbencias dominantes del PV *"service"* es la nombrada como *"Envio Datos Incidencia"*. En el análisis del Esquema

lingüístico 13 se puede percibir cómo por primera vez todos los atributos y criterios de la metodología OCTA se repiten de entre los propuestos hasta este punto a través del caso práctico ejemplificado. En este análisis, es interesante puntualizar cómo el término *aceptar* manifiesta el atributo *condición*, sin embargo, es obvio que además de *aceptar* es necesario generar el reporte de incidencia común. Por lo tanto, la condición real es cuando se ha generado el reporte de incidencia común. En este sentido se puede observar que la metodología OCTA permite detectar descripción semántica errónea.



Esquema lingüístico 14 Atributos de incumbencia dominante "Modificar BD Asignacion Principal"

En cada esquema lingüístico de esta ejemplificación se explica algún nuevo atributo o criterio de modelado. En el caso del análisis de la siguiente incumbencia *"Modificar BD Asignacion Principal"* reflejada en el Esquema lingüístico 14 aparece el atributo **actualización**. Este ha sido mencionado en el modelado de aspectos bajo el PV *"Basic Management"* pero sin ejemplificarlo. La propiedad **"update"** permite la descripción de este atributo a través de la restricción *"update some Modificar_BD_Asignacion_Principal"* que especifica la clase *"aTecnicoAsignado"* para describir los individuos que son actualizados al realizarse la incumbencia dominante *"Modificar_BD_Asignacion_Principal"*.



Esquema lingüístico 15 Atributos de incumbencia dominante "Necesidad Incidencia"

Así, esta ejemplificación pasa al análisis de su última incumbencia dominante *"Necesidad Incidencia"*. En el Esquema lingüístico 15 se pueden observar atributos recurrentes que no serán nuevamente descritos. Sin embargo, esta incumbencia es aprovechada para puntualizar un nuevo criterio de análisis. En este requisito atómico se observa que aparentemente no existe el atributo *condición*, es decir, la descripción textual no indica en qué estado debe estar el sistema previamente para realizar esta incumbencia. La sencillez de este caso práctico permite comprender que después de recibir un técnico un reporte de incidencia común, el sistema pasa a un estado de espera a que se le indique la necesidad de técnicos de apoyo. Por lo tanto, la metodología OCTA permite identificar atributos que están implícitos pero que deberían estar explícitos. Así, el modelado de la

incumbencia *"Necesidad Incidencia"* especifica los atributos *roles participantes, envío* y *condición,* como se puede observar en Figura 47.

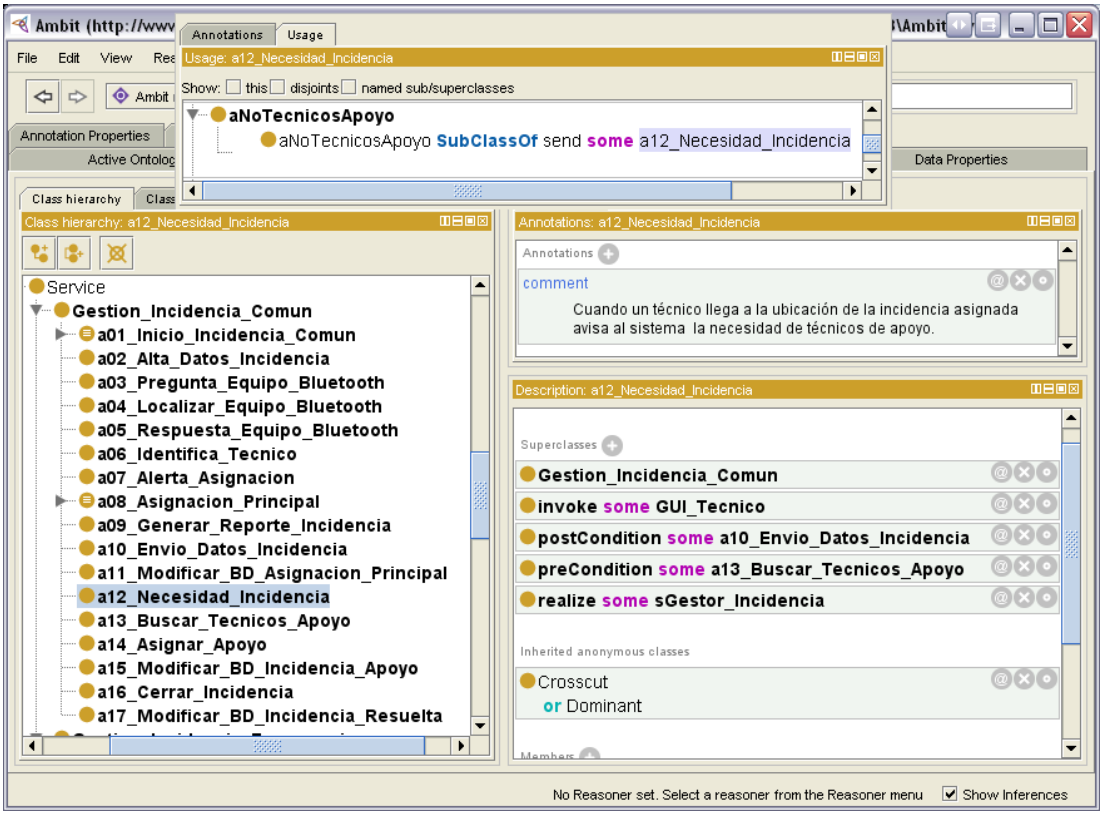


Figura 47 Restricciones de la incumbencia dominante "necesidad incidencia"

3.12 Paso 9. Incumbencias transversales – Información

Este paso comienza con el análisis y modelado de las incumbencias transversales que complementan las incumbencias dominantes. En la APV creada para la metodología OCTA se encuentran nueve PV propuestos para modelar aspectos que agrupan incumbencias transversales. Ahora bien, en este caso práctico se han ejemplificado los más representativos.

El PV propuesto por la APV que indiscutiblemente tiene que existir es *"Information"*. Las incumbencias de dicho PV son necesarias para la existencia de las entidades de información utilizadas en las incumbencias dominantes. Ahora bien, la estructura de información de una entidad se define por atributos. Algunos adquieren sus datos en forma estática pero otros obtienen sus datos a través de una operación que utiliza los atributos estáticos; este otro tipo de atributo es típicamente derivado. Por lo tanto, incumbencias transversales creadas bajo el PV de *"Information"* pueden estar agrupadas en *"Attribute"* y *"Derived"*. Se ha dicho que siempre tiene que existir este tipo de incumbencias transversales; así mismo, estas siempre describen una entidad de información. Por este motivo, la incumbencia *"Incidencia Común"* creada bajo el PV *"Domain_Information"* ejemplifica este tipo de incumbencias transversales.

Por otro lado, las incumbencias que representan entidades de información son descritas en su forma textual por sus atributos como se puede observar en Esquema lingüístico 16 donde se analiza la incumbencia *"Incidencia Común"* que representa una entidad de información.

*La **incidencia común** debe estar descrita por el **nombre**, **tipo** de incidencia común, **hora de alta**, **hora de inicio**, **hora de cierre**, **fecha**, **estado** de la incidencia, **descripción**, **ubicación geográfica** y **técnico asignado**.*

Esquema lingüístico 16 Incumbencia transversal "Incidencia Comun"

En este análisis es muy sencillo observar cómo la incumbencia que representa una entidad de información hace referencia a un determinado número de incumbencias transversales creadas bajo el PV de *"Information"*. Las incumbencias transversales correspondientes a la incumbencia *"Incidencia Comun"* que representan los atributos estáticos se describen con lenguaje natural a continuación:

aNombre. Cualquier incidencia debe ser reconocida con un nombre.

aDescripcion. Todas las incidencias de cualquier tipo deben tener enlazada una descripción textual de la misma.

aEstado. El total de las incidencias debe registrar su estado de atención por parte de un técnico.

aFecha. Las incidencias deben tener una fecha de creación.

aHoraAlta. Es necesario registrar la hora de alta de cualquier incidencia.

aHoraInicio. La hora de inicio de atención por un técnico de una incidencia debe registrarse.

aHoraCierre. La hora de cierre o resolución de cualquier incidencia debe registrarse.

aTipo. Una incidencia debe conocer el tipo al que pertenece.

aTecnicoAsignado. Una incidencia común debe conocer al técnico asignado para su resolución.

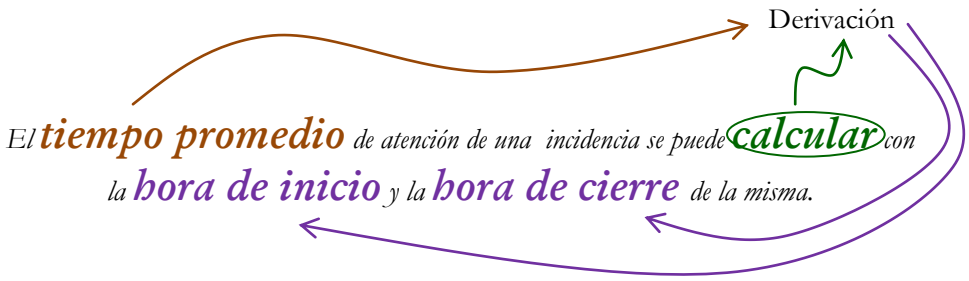
aUbicacionGeografica. La creación de una incidencia común conlleva la asignación de su ubicación geográfica.

La importancia de las incumbencias transversales de información requiere que su identificación sea fácil, por lo que para su modelado en la ontología de AT se optó por la convención de añadir una "a" antes del nombre de la incumbencia.

En este caso los requisitos atómicos o incumbencias transversales de *"Incidencia Comun"* fueron descritas en su totalidad. Sin embargo, en una ER típica estos requisitos atómicos no son completos, siendo necesario apoyarse en los requisitos apócrifos o en las incumbencias dominantes.

Ahora bien para especificar la relación entre la entidad de información y sus atributos se propuso la propiedad objeto **"compose"** para la creación de la restricción *"compose some Incidencia_Comun"* para la especificación de los atributos estáticos. Sin embargo, algunos atributos corresponden a todos los tipos de incumbencias, por lo tanto, en estos casos se requiere la restricción *"compose some Incidencia"*.

Hasta el momento la incumbencia seleccionada sólo ejemplifica las incumbencias que representan los atributos estáticos. En la Figura 37 se muestran las partes importantes de la especificación de la incumbencia dominante *"Lista Incidencia Tecnico"*, donde con la propiedad *"query"* se creó la restricción *"query some Lista_Incidencia_Tecnico"* para especificar la incumbencia dominante pero con la descripción de un atributo derivado reflejado en la incumbencia *"Tiempo Ejecución"*.



Esquema lingüístico 17 Incumbencia transversal "TiempoEjecucion"

El análisis del Esquema lingüístico 17 no se realiza para identificar atributos, en este caso es para extraer la semántica de derivación. Así, a través del término calcular se identifica qué datos son necesarios para derivar otro más complejo.

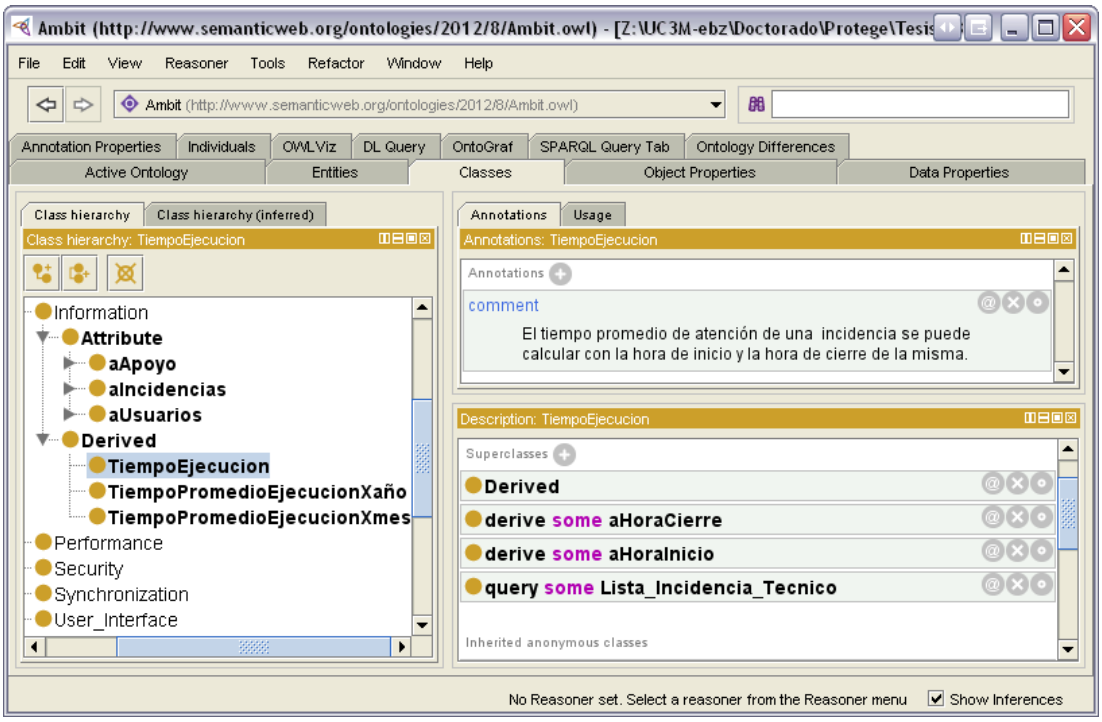
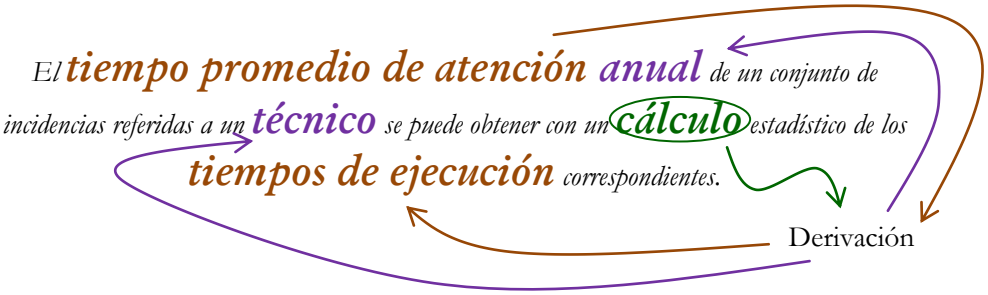


Figura 48 Especificación incumbencia transversal "tiempo ejecución"

En esta línea, se recomienda una propiedad objeto *"***derive"** para la creación de las restricciones que especifican la incumbencia transversal *"TiempoEjecución"* que se puede observar en la Figura 48.



Esquema lingüístico 18 Incumbencia transversal "TiempoPromedioEjecucionXaño"

En el análisis anterior de la incumbencia transversal *"TiempoEjecución"* se puede observar cómo para el cálculo del atributo derivado se utilizan incumbencias que representan atributos estáticos. Así mismo, pueden existir atributos derivados que utilizan otros atributos derivados para su cálculo. Este es el caso de la incumbencia transversal *"TiempoPromedioEjecucionXaño"* que es analizada en Esquema lingüístico 18. Esta incumbencia presenta una semántica de derivación compleja donde un atributo derivado requiere otro atributo derivado y dos atributos estáticos para su creación, cuya especificación se puede observar en la Figura 49.

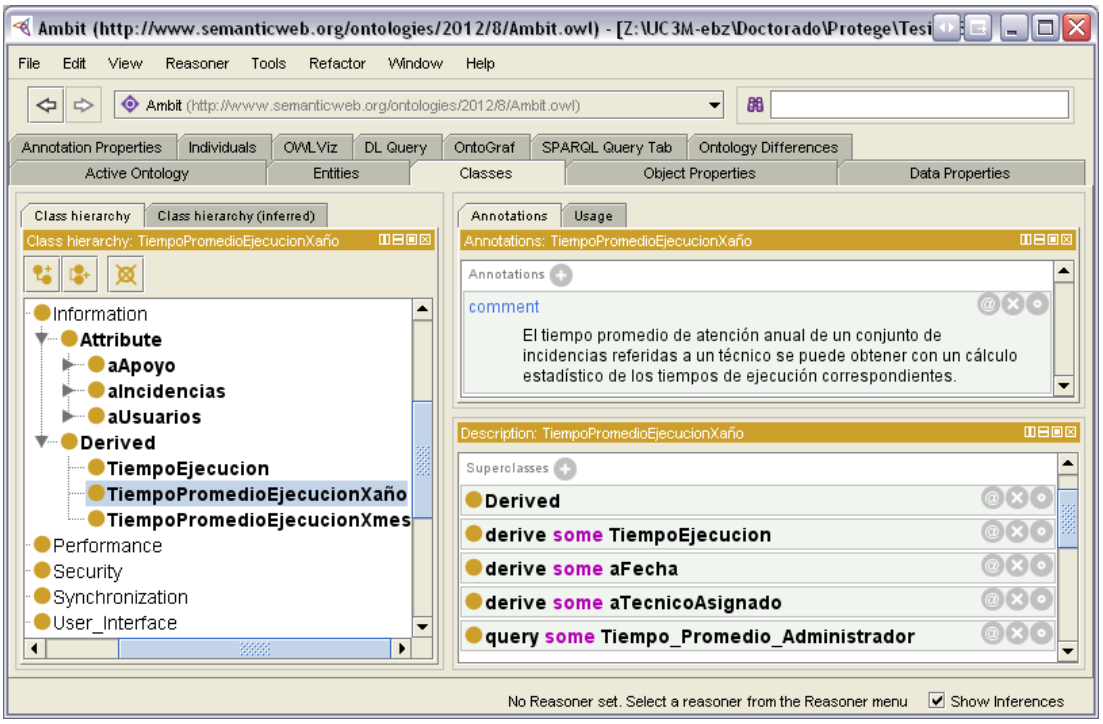


Figura 49 Especificación incumbencia transversal "tiempo promedio ejecución x año"

En las dos últimas ejemplificaciones se ha mostrado la especificación de incumbencias transversales que representan atributos derivados. En la Figura 48 y Figura 49 se pueden observar además de las restricciones creadas con la propiedad *"derive"* restricciones creadas con la propiedad *"query"*; esta última propiedad objeto fue creada para la especificación de las incumbencias dominantes creadas bajo el PV de *"Consult"*.

Esquema lingüístico 19 Incumbencia dominante "Tiempo_Promedio_Administrador"

Un ejemplo es la incumbencia dominante *"Tiempo Promedio Administrador"* que es descrita textualmente en el Esquema lingüístico 19. En esta incumbencia dominante se puede observar claramente que está agrupada bajo el PV *"Consult"*; sin embargo, el análisis textual de su requisito presenta características del atributo de comparación. El mismo

conflicto surge cuando en una ER clásica en lenguajes naturales estos requisitos atómicos representados con incumbencias dominantes están solapados con algunos requisitos atómicos que describen los tributos de entidades de información. Por lo tanto, en el modelado de la ontología de AT con esta metodología OCTA se recomienda modelar la semántica necesaria del cálculo de atributos derivados en las incumbencias transversales que los representan.

3.13 Paso 10. Incumbencias transversales – Rendimiento

En el paso 10 se ejemplifican las incumbencias transversales creadas bajo el PV de *"performance"*. Estas incumbencias representan los típicos requisitos de rendimiento en requisitos no funcionales en una ER clásica.

Esquema lingüístico 20 Incumbencia transversal de rendimiento "Localizar Tecnico TR"

En las incumbencias ejemplificadas anteriormente los términos de la descripción textual indican de una forma relativamente clara los atributos para el modelado de su semántica. Sin embargo, las incumbencias transversales como las de rendimiento, se presentan típicamente con términos de poca semántica adicional. Por ejemplo en el análisis de la incumbencia del Esquema lingüístico 20 se observa el término *sistema software* que aparentemente podría representar un atributo del tipo *rol participante* pero esta interpretación no representa una semántica importante. En este tipo de incumbencias la semántica importante es la que representa el tiempo que debe cumplir al ejecutarse una o un grupo de incumbencias dominantes codificadas en la etapa de aspectos finales. Por lo tanto, la semántica que hay que extraer es que incumbencias dominantes están involucradas en la especificación del respectivo rendimiento.

El análisis de la incumbencia *"Localicar técnico TR"* muestra claramente el atributo "**rendimiento"** que indica el tiempo específico a cumplir por una o un grupo de incumbencias. Este atributo es indicativo de que el requisito atómico descrito textualmente debe estar agrupado en el aspecto de rendimiento. Ahora bien, la identificación de las incumbencias dominantes que son cortadas por la incumbencia transversal es más compleja. El análisis de la incumbencia transversal *"Localizar tecnico TR"* no describe de forma clara las incumbencias dominantes participantes. Al resaltar en el Esquema lingüístico 20 una sentencia que aparentemente es indicativa de las incumbencias dominantes se torna compleja la extracción semántica. En este caso por el conocimiento del caso de estudio se puede extraer que las incumbencias dominantes que son cortadas por esta incumbencia transversal son: "*Pregunta Equipo Bluetooth", "Localizar Equipo Bluetooth"* y "*Respuesta Equipo Bluetooth"*. Así, para la especificación del atributo de *rendimiento* se propuso la propiedad objeto **"start"** para la creación de las restricciones expuestas en la Figura 50.

Ahora bien, la especificación no se limita a la descripción de las restricciones de la incumbencia transversal. La necesidad de inferencia de conocimiento en el modelo de AT en una ontología es en los dos sentidos, es decir, se requiere saber qué incumbencias dominantes se deben realizar en una incumbencia transversal de rendimiento y se requiere saber si una determinada incumbencia está relacionada con una incumbencia

transversal de rendimiento. En esta línea, utilizando la misma propiedad *"start"* se crea la restricción *"start some Localizar_Tecnicos_TR"* para describir las incumbencias dominantes participantes.

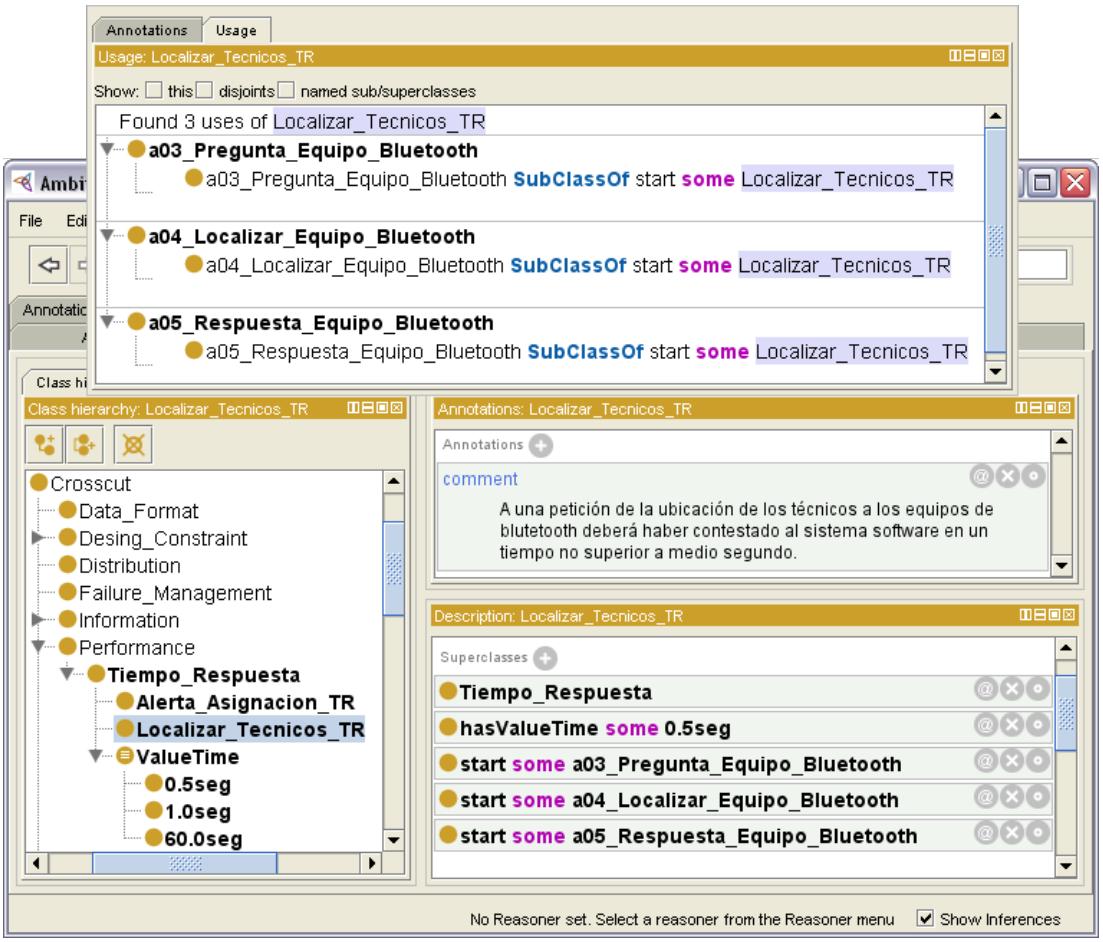


Figura 50 Especificación incumbencia transversal "localizar técnico TR"

En los párrafos del paso 2 se describió la utilización del patrón ontológico de partición valorada. En la especificación de las incumbencias transversales de rendimiento nuevamente se hace uso de este patrón. En este caso se crea una partición valorada llamada *"ValueTime"* para describir el tiempo que debe cumplir el grupo de incumbencias dominantes cortadas por la incumbencia de rendimiento específica. Así se restringe el rango de valores posibles de tiempo en una lista exhaustiva, en este caso, en los rangos de *"0.5 seg","1.0 seg"* y *"60 seg"*. Así mismo, se propone una propiedad de objeto funcional *"hasValueTime"* con el rango *"ValueTime"*. Por lo tanto, para la especificación de la incumbencia transversal *"Localizar_Tecnico_TR"* se crea la restricción *"hasValueTime some 0.5seg"*. Así mismo, es necesaria una propiedad inversa proponiendo *"isValueTimeOf"* para especificar en las clases de *"ValueTime"* los tiempos de las incumbencias de *"Tiempo_Respuesta"*.

3.14 Paso 11. Incumbencias transversales – Interfaz de usuario

Unas de las incumbencias más importantes en un producto software en la línea de sistemas de información son las creadas bajo el PV *"User Interface"*. Así en el paso 11 se ejemplifica este tipo de incumbencias transversales. En el caso práctico de esta tesis doctoral se visualiza el desarrollo de una Interfaz Gráfica de Usuario (GUI).

Ahora bien, para la ejemplificación de las incumbencias involucradas en este paso lo primero es crear una clase *"GUI Widget"* en la cual se modelan una serie de clases hija que agrupan los tipos de elementos de una GUI. Los elementos modelados en este caso práctico, mostrados en la Figura 51, cubren los extraídos del corpus utilizado, por lo tanto, dicha clasificación es factible de ser modificada.

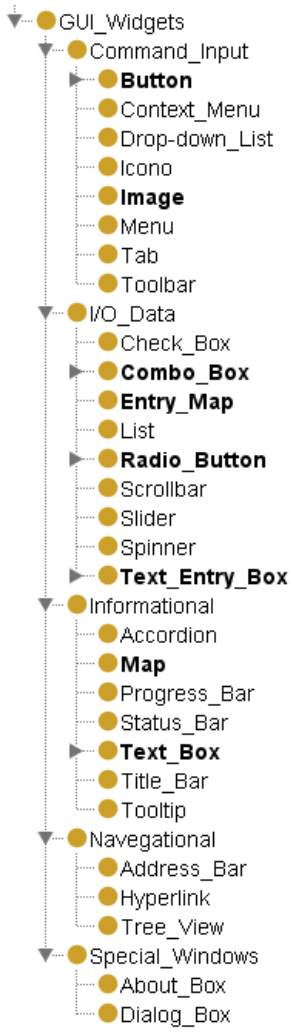


Figura 51 Tipos de elementos de una GUI

Muchos de los productos software desarrollados y en especial los productos de la línea de sistema de información están estrechamente relacionados con la creación de una GUI. En las etapas tempranas esta interfaz es descrita con la secuencia de actividades que un usuario podría realizar en un entorno visual. En muchas ocasiones estas descripciones visuales son confundidas con procesos funcionales. Sin embargo, en estos últimos existen transformaciones conceptuales y la descripción de las incumbencias transversales de una GUI sólo conllevan una transformación de formato.

En esta línea, la segunda acción es crear el aspecto *"Descripcion Vista"* bajo el PV *"User Interface"*. Este aspecto representa un contenedor de aspectos que agrupa incumbencias transversales que representan el funcionamiento de un concepto que puede ser una vista general o parte de una interfaz de usuario o diferentes partes de varias interfaces.

El criterio a seguir en la creación de aspectos bajo el PV de *"Descripcion Vista"* es agrupar las incumbencias que representan las actividades de un mismo concepto. Por este motivo, en esta ejemplificación se creó el aspecto *"Vista Gestion Incidencia"* donde se

describen las incumbencias que modelan las actividades que se pueden realizar en torno a gestión de incidencias. Como en otras ejemplificaciones se hizo una selección de incumbencias representativas del aspecto para exponer los criterios principales. La primera incidencia trasversal *"Vista Inicial"* es analizada en el Esquema lingüístico 21, donde se puede observar claramente cómo esta incumbencia describe una vista con tres imágenes que representan accesos a diferentes recursos.

Al iniciar una gestión de incidencias se despliegan tres posibilidades de selección que llevan a las actividades necesarias para el alta de una ***incidencia común****, para el alta de una* ***incidencia de emergencia*** *y para la consulta de* ***listas de incidencias*** *con atributos y estadísticas.*

Esquema lingüístico 21 Incumbencia transversal de descripción vista "Vista Inicial"

En este caso la incumbencia que representa el acceso a diferentes recursos se especifica con la creación de cada una de las incumbencias que representen las vistas de inicio de cada uno de los recursos; esto se puede observar en la Figura 52.

Ahora bien, esta vista contiene tres veces un elemento GUI, identificado como imagen de acceso que se especifica a través de la propiedad objeto **"isContained"** con la que se crea la restricción *"isContained exactly 3 Vista_Inicial"* que es descrita en la clase que representa el elemento gráfico *"image"*.

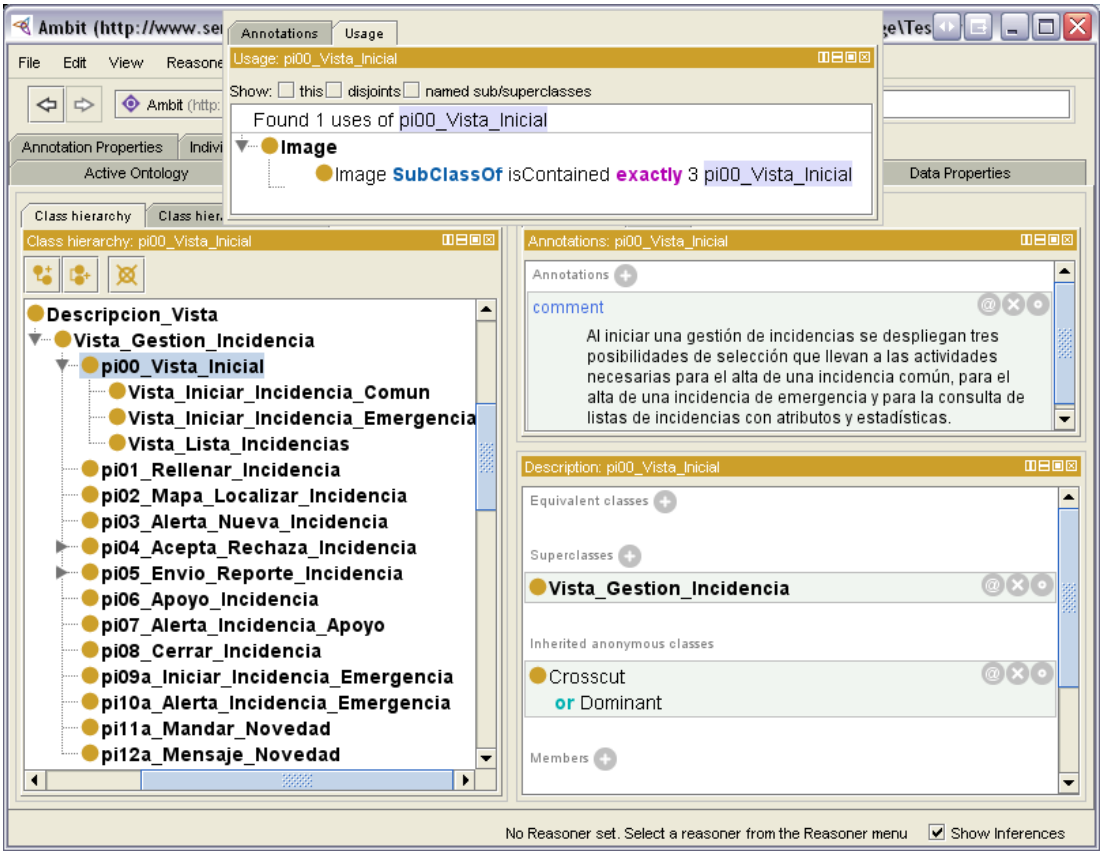


Figura 52 Especificación incumbencia transversal "vista inicial"

Por otro lado, es necesario aplicar el criterio que permita identificar la secuencia semántica de las incumbencias transversales de la interfaz de usuario. Por lo tanto, en

esta ejemplificación la incumbencia *"Vista Iniciar Incidencia Comun"* se especifica con la restricción *"preCondition some "Rellenar Incidencia"* y la incumbencia *"Rellenar Incidencia"* se especifica con la restricción *"postCondition some Vista_iniciar_Incidencia Comun"*.

Al iniciar una gestión de incidencia se despliega un formato con ***una caja de texto*** *para la* ***descripción*** *y* ***un botón de opción*** *para indicar el* ***tipo de incidencia****, así como* ***un botón*** *para* ***mandar*** *los datos introducidos.*

Esquema lingüístico 22 Incumbencia transversal de descripción vista "Rellenar Incidencia"

En el Esquema lingüístico 22 se analiza la incumbencia *"Rellenar Incidencia"*, donde se observan los elementos GUI contenidos. Al igual que en la incumbencia anterior esta semántica se especifica utilizando la propiedad objeto *"isContained"*. Ahora bien, el modelado de cada elemento de la interfaz gráfica contenido en la incumbencia *"Rellenar Incidencia"* también necesita especificar la semántica de su objetivo. En esta línea, en la clase *"Text Entry Box"* se crea una subclase *"TBentrada"* en la cual se crearán clases hija que representarán la información de entrada esperada en el elemento GUI, que en este caso es *"Descripcion"*. Para conseguir esta especificación se utiliza la propiedad objeto **"entrada"**, con la cual se crea la restricción *"entrada some Rellenar_Incidencia"*.

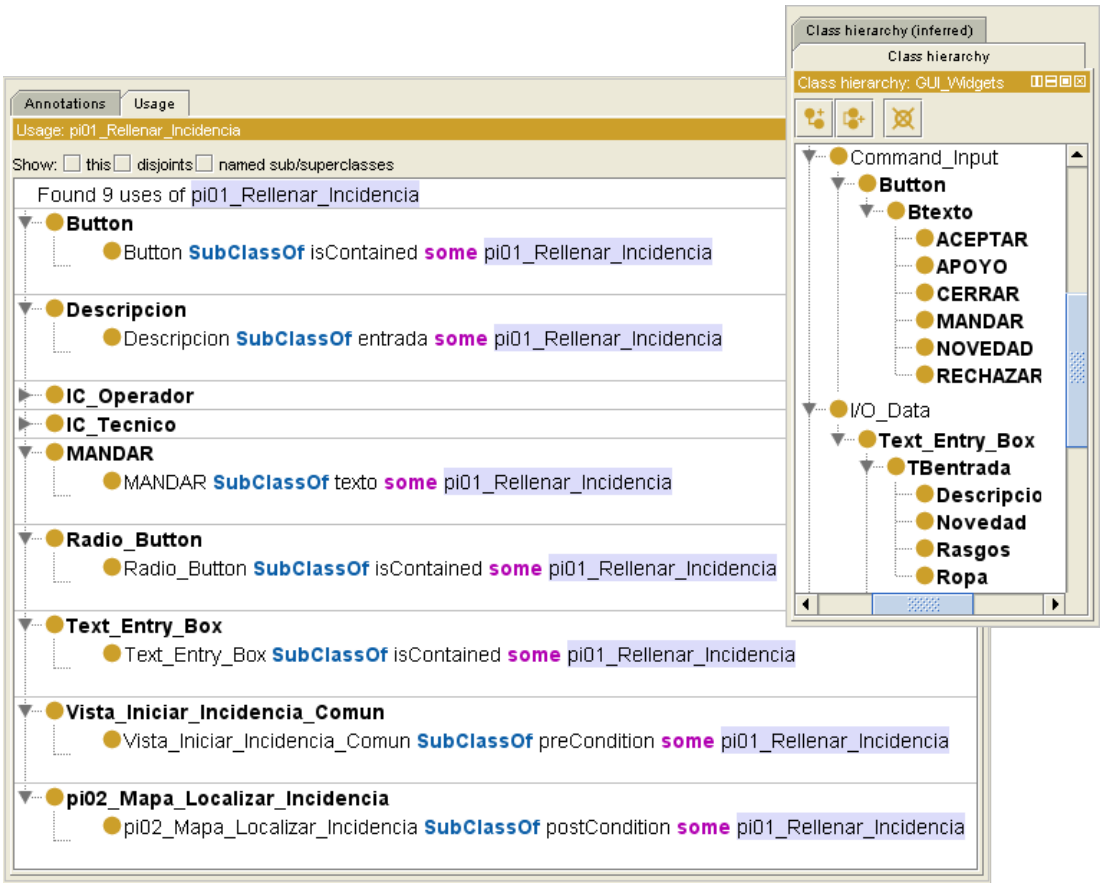


Figura 53 Restricciones para la especificación de la incumbencia "rellenar incidencia"

El mismo criterio se aplica con el elemento GUI *"Button"*; en este caso se crea una subclase *"Btexto"* en la cual se crearán las clases hija que representan el texto que mostrará el botón, que en esta incumbencia es *"MANDAR"*. En este caso se utiliza la propiedad **"texto"** para crear la restricción *"texto some Rellenar_Incidencia"*. La descripción

de la incumbencia transversal *"Rellenar Incidencia"* se puede ver en las restricciones mostradas en la Figura 53.

Ahora bien, en el modelado de las incumbencias transversales de *"User Interface"* no puede faltar la descripción de los enlaces con sus correspondientes incumbencias dominantes. Esta especificación se realiza de ida y vuelta entre las incumbencias contenidas en los aspectos creados bajo el PV *"Descripcion Vista"* que tengan una relación con alguna incumbencia creada bajo el PV *"Dominant"*. En esta ejemplificación se puede observar en la Figura 54 cómo con el apoyo de la propiedad objeto **"item"** se crean las restricciones necesarias para complementar las incumbencias involucradas. En esta especificación se puede observar cómo las restricciones *"item some IC_Operador"* y *"item some IC_Operador"* se utilizan para especificar la incumbencia *"Rellenar Incidencia"* del aspecto *"Descripcion Vista"*. Así se indica que al realizarse cualquiera de las incumbencias *"IC_Operador"* y *"IC_Operador"* se desarrollará la misma incumbencia transversal bajo el PV *"Descripcion Vista"*.

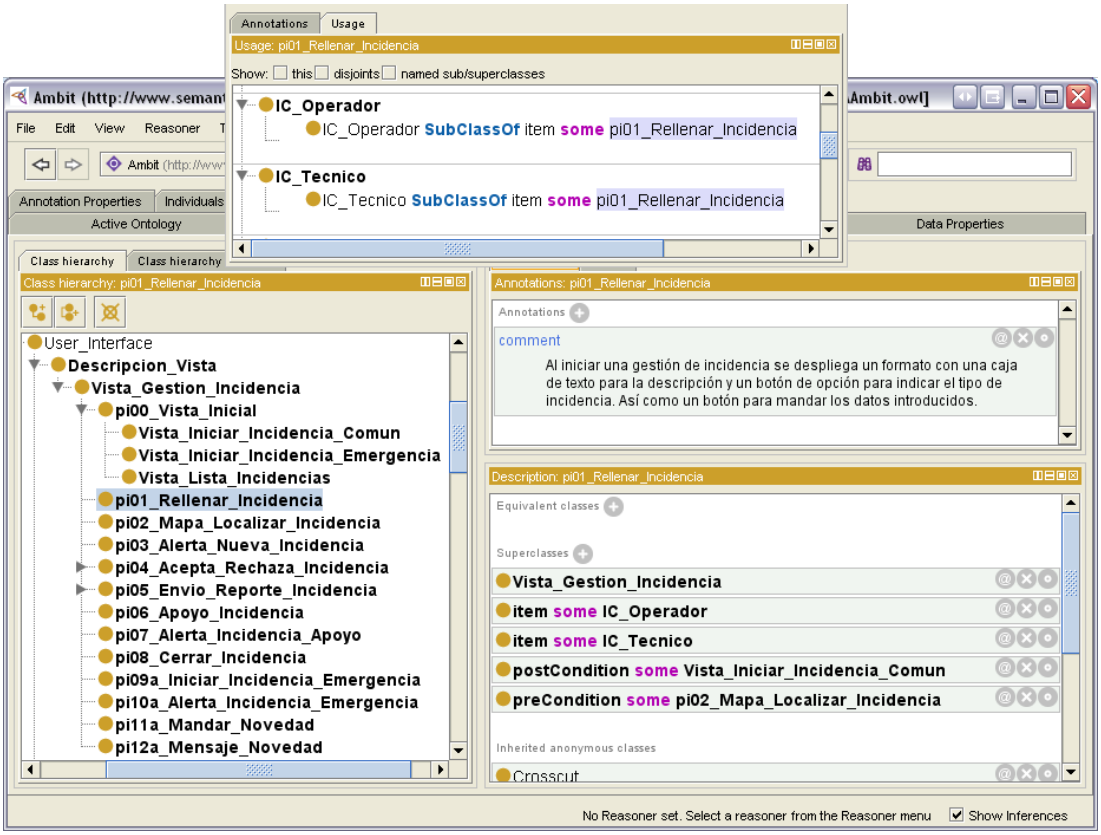


Figura 54 Especificación incumbencia transversal "rellenar incidencia"

3.15 Paso 12. Incumbencias transversales – Arquitectura Lógica

En este último paso se razona el modelado de las incumbencias transversales plasmadas en la arquitectura lógica. En la APV creada para la metodología OCTA se proponen las dimensiones: *hardware* y *software*. La dimensión *hardware* agrupa los PV *"Hardware Resources"* y *"Internal Communications"*. En este caso práctico *"Hardware Resources"* se ejemplifica con las incumbencias transversales *"Dispositivo Movil"* y *"Equipo Bluetooth"* los cuales representan requisitos que describen bloques de construcción de la topología hardware. Así mismo, en el PV *"Internal Communications"* se ejemplifica con las incumbencias *"Wifi"* e *"Intranet"* representando requisitos que describen características a cumplir en las conexiones de área local que requieren los conceptos hardware.

Por otro lado, en esta propuesta la dimensión más importante del ámbito de la arquitectura lógica es la de *software* donde son tangibles las primeras líneas de solución. El objetivo es crear un modelo de la arquitectura lógica con capas significativas del dominio de interés. La idea clave del modelo de una arquitectura lógica es un diccionario visual de abstracciones relevantes de los componentes o capas con responsabilidad. El razonamiento de la identificación de capas debe ser iterativo, es decir, se construye incrementalmente un modelo a lo largo de varias iteraciones en la fase de creación de la ontología de AT. En cada una, la arquitectura lógica se limita a lo modelado hasta ese momento y actual en estudio, en lugar de un modelo general que en las primeras iteraciones intenta capturar toda la semántica del producto software en desarrollo. Sin embargo, es normal obviar capas en el razonamiento inicial y descubrirlas más tarde.

Ahora bien, la idea de crear arquitecturas lógicas simples está implícita en la gran mayoría de los desarrolladores, convertida en un símbolo de elegancia y flexibilidad. Este deseo irreprimible se inscribe de modos distintos en cada línea de desarrollo de productos software. Sin embargo, la creación de arquitecturas lógicas limitada a la sencillez, no discrepa de la posibilidad de explotar capas con una organización modular recursiva, es decir, especificar en exceso una arquitectura lógica con muchas capas de grano fino.

En el desarrollo de la APV de esta tesis doctoral se ha trabajado la generalización y simplicidad en productos software de sistemas de información. En esta línea, el estilo arquitectónico MVC ha sido la base de la propuesta de los PV *"Functional"*, *"Presentation"* y *"DataBase"* que han sido utilizados en esta ejemplificación.

Las capas abstractas funcionales son las más complejas a considerar en términos de sus diferentes intenciones. En el caso de esta ejemplificación, eso significa capas relacionadas con *gestiones de incidencias*, *posicionamiento geográfico de entidades* o *mecanismos de control de tecnología*. Un ejemplo concreto en esta ejemplificación es considerar el componente o capa *"sGestor Incidencia"* bajo el PV *"Functional"* con la intención de establecer que tiene la responsabilidad de representar todas las funcionalidades posibles relacionadas con las incidencias. Así, en posteriores interacciones es posible añadir subcapas de grano más fino según las recomendaciones de los responsables del desarrollo del producto software. En el Paso 6. Incumbencias dominantes – gestión básica se creó la propiedad objeto *"invoke"* y *"realize"* para especificar el atributo de roles participantes. Este atributo es el que modela la semántica de la arquitectura lógica con los requisitos en el marco de trabajo de AT. Por lo tanto, es necesaria la creación de propiedades inversas que en este caso serían **"letInvokes"** y **"letRealizes"** para permitir inferir en ambos sentidos.

Ahora bien, otro PV a tomar en cuenta en el desarrollo de productos software de sistemas de información es *"Presentation"*. En este caso la pauta a seguir es identificar todas las interfaces de usuario a desarrollar. Por ejemplo en este caso la incumbencia transversal *"GUI_Tecnico"* que representa la interfaz de usuario gráfica de uno de los roles de usuario. Por lo tanto, es necesario relacionar esta incumbencia con la de *"Tecnico"* de la dimensión *"User"* de la ontología *"Stakeholder"* creada en el paso 3. En esta línea, se recomienda crear la propiedad objeto **"Interact"** para especificar con la restricción *"interect only GUI_Tecnico"* en condiciones necesarias y suficientes en la clase *"Tecnico"*. Así, es modelada la semántica de cómo un técnico interactúa con el producto software. Ahora bien, esta semántica requiere ser inferida en ambos sentidos, por lo tanto, se recomienda la propiedad inversa **"letInterects"** para crear la restricción *"letInteracts only Tecnico"* en condiciones de necesarias y suficientes en la clase *"GUI_Tecnico"*. En este tipo de modelado en ocasiones es necesario especificar en qué hardware estará instalada la interfaz de usuario para su uso. En este caso se recomienda

la propiedad objeto **"laid"** para crear la restricción *"laid some GUI_Tecnico"* y ser utilizada en la descripción de *"Dispositivo Movil"* modelado bajo el PV de *"Hardware Resources"*. Esta especificación modela que un técnico interactúa con la aplicación a través de una interfaz gráfica instalada en un dispositivo móvil.

Un producto software de sistemas de información es un conjunto de elementos los cuales interactúan en cuatro actividades básicas: entrada, almacenamiento, procesamiento y salida de información. Las capas funcionales modelan el procesamiento y la capa de presentación modela la entrada y salida de información. Por lo tanto, un elemento importante a modelar es el almacenamiento. El almacenamiento se compone de un banco de datos almacenados sistemáticamente para su posterior uso. En este sentido, es necesario modelar el gestor que permita almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Por lo tanto, los conceptos de banco de datos y gestor están representados en la capa *"DataBase"*, donde su especificación se hace a través de las mismas propiedades mencionadas en este paso 12.

3.16 Resumen del modelado con OCTA

En las anteriores secciones se han descrito los pasos a seguir en la metodología OCTA para el proceso de desarrollo de una ontología que modela los AT de un producto software. La descripción de la ejemplificación del Caso de estudio "Estadio para eventos de atletismo" se ha desarrollo de forma secuencial. Sin embargo, la aplicación de la metodología OCTA al desarrollo de cualquier producto software es un proceso concurrente dividido en dos fases.

La primera fase contempla del paso 1 al 4, en los cuales se toman las decisiones de organización del conocimiento creando la estructura del modelado de AT. En donde cada una de las ontologías de la estructura puede ser creada por diferentes grupos de expertos de forma concurrente. El paso 5 es el punto de agregación de todas las ontologías y su realización permite el paso a la siguiente fase contemplada desde el paso 6 hasta el 12, en los cuales se proponen los puntos de vista para la creación de aspectos y sus respectivas incumbencias tempranas. Al igual que la primera fase puede ser creada de forma concurrente por diferentes expertos.

Ahora bien, sabemos que las incumbencias en un modelado de AT pueden ser dominantes o transversales. Así mismo se ha dicho que los pasos de la segunda fase pueden ser concurrentes. En este escenario, es recomendable iniciar esta fase en el contexto de los pasos 6 al 8 para modelar primero los aspectos e incumbencias tempranas dominantes, permitiendo el modelado natural de las incumbencias transversales correspondientes al paso 9 en adelante.

Una característica clara del proceso de la metodología OCTA es el enfoque iterativo, el cual propone una comprensión incremental del modelado de AT a través de refinamientos sucesivos y un crecimiento incremental. Como parte del enfoque iterativo se encuentra la flexibilidad para adaptarse a la evolución del modelado. Así mismo permite que se identifiquen y se acometan los riesgos al principio del proceso.

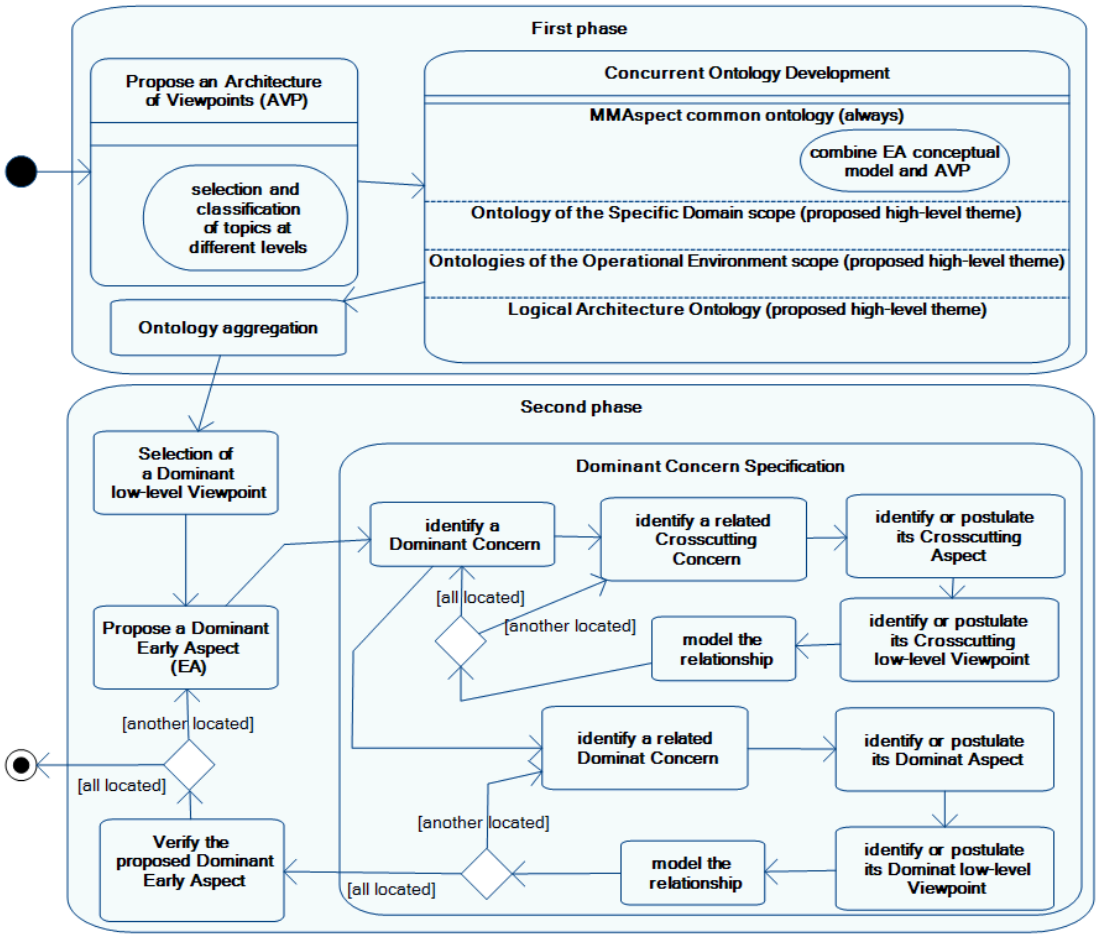


Figura 55 Arquitectura global del proceso de la metodología OCTA

Una arquitectura global del proceso de la metodología EKOA es mostrada en la Figura 55. El proceso inicia con la selección de un "punto de vista" para modelar "aspectos" que agrupan "incumbencias dominantes". Bajo este "punto de vista" se propone un AT dominante para identificar una de las "incumbencias dominantes" que lo componen. Las propiedades permiten identificar una a una las "incumbencias transversales" relacionadas. Cada una de estas son localizadas en su correspondiente AT transversal; en caso de no existir es necesario proponer uno nuevo. El mismo mecanismo se repite al buscar el "punto de vista de bajo nivel" del correspondiente "AT transversal". En este punto se modela la relación entre "incumbencias" a través de una "propiedad objeto". Una "incumbencia dominante" está relacionada con "incumbencias transversales", sin embargo, también está relacionada con "incumbencias dominantes". Por lo tanto, cuando se tiene identificada "la incumbencia dominante" a modelar también es necesario identificar "incumbencias dominantes" relacionadas. Esta "incumbencia" es localizada en su "AT dominante" correspondiente y de forma secuencial se identifica o propone el "punto de vista dominante de bajo nivel" al que pertenece. Esta información permite modelar la relación entre "incumbencias dominantes". En el mismo caso que en las "incumbencias transversales" se realiza el mismo proceso hasta completar la semántica de la "incumbencia dominante" modelada. Al completar un AT se inicia el proceso con el siguiente hasta constituir el "punto de vista dominante". El mismo proceso se aplica a cada uno de los "puntos de vista dominantes" hasta completar todos los puntos de vista propuestos en la APV. El resultado final es una representación organizada del conocimiento temprano de un producto software basado en AT.

Por otro lado, el desarrollo de la segunda fase de la metodología OCTA implica el modelado de los AT que representan el conocimiento de los requisitos típicos de una ER y el total de las capas de arquitectura lógica relacionadas. Este modelado semántico de AT se consigue con la especificación de las propiedades entre incumbencias tempranas dejando como actividad secundaria hacer una descripción en lenguaje natural para el entendimiento con los interesados en el sistema no expertos.

Este proceso tiene como actividad primaria modelar con OWL el conocimiento temprano. Existen dos principales tipos de propiedades que OWL representa como relaciones: "Object Properties" y "Data Properties". Las más importantes en la metodología OCTA son las "Object Properties" para modelar la semántica del modelado. El modelado del conocimiento temprano de un producto software asistido por AT conlleva la descripción de una incumbencia como un requisito típico en lenguaje natural para que un desarrollador pueda expresarse sin limitaciones. Esta información es añadida con la "Annotation Properties".

La especificación del conocimiento interno de los AT es a través del modelado de las propiedades entre incumbencias en base a las propiedades objeto. En este sentido, esta investigación ha proporcionado un grupo de "propiedades objeto" recurrentes que fueron encontradas a través del análisis heurístico de un corpus de 6337 requisitos de 90 diferentes proyectos software dentro de la línea de producto de IS. Es factible que en una organización surjan más propiedades para el modelado de las etapas tempranas con los criterios propios. En la Tabla 8 se puede observar el resumen de la propuesta de las propiedades objeto con una pequeña descripción de su propósito, organizados por los puntos de vista en donde son utilizados.

Punto de vista	Atributo	Propiedad Objeto	Descripción
ID Gestión Básica	Creación	create	Describe las instancias de entidad de información que son creadas al realizarse la incumbencia dominante
	Actualización	update	Describir los individuos que son actualizados al realizarse la incumbencia dominante
	Supresión	delete	Describir los individuos que son borrados al realizarse la incumbencia dominante
ID Consulta	Consulta	query	Cuando se requiere referir un dato guardado en un almacenamiento no volátil de una entidad de información.
ID Servicio	Lanza	launch	Para especificar la responsabilidad de inicio de rol capaz de lanzar un servicio
	Inclusión	include	Identificar la responsabilidad del rol invocador de la introducción de los datos que conforman una instancia de la entidad de información
	Condición	postCondition	Para describir la incumbencia condicionante para la realización de otra incumbencia
		preCondition	Para describir la incumbencia predecesora que especifica el estado en que debe estar la aplicación antes de la realización de la incumbencia

	Bifurcación	accept	Describen la semántica de diferentes posibilidades de estado en la aplicación del patrón bifurcación.
		refuse	
	Solicita	request	Cuando se requiere describir uno o varios datos no recopilados de una entidad de información para su delegación.
	Fabricación	make	Describir la incumbencia responsable de obtener uno o varios datos no recopilados de una entidad de información.
	Envío	send	Describir la incumbencia responsable de enviar los datos fabricados después de una solicitud.
	Comparación	compare	Para describir diferentes datos a cumplir por individuos de una entidad de información
	Generación	generate	Para describir la generación de un individuo de una entidad de información
IT Información	Composición	compose	Especificar la relación entre la entidad de información y sus atributos o información
	Derivado	derive	Especificar la relación entre la entidad de información y atributos derivados
IT Rendimiento	-	star	Indicar el tiempo específico a cumplir por una o un grupo de incumbencias
IT Interfaz de Usuario	-	isContained	Para especificar los elementos de la GUI que tiene una incumbencia de una vista completa de una interfaz
	-	entrada propiedad añadida	Para representar la información de entrada esperada en el elemento GUI
	-	texto propiedad añadida	Para representar el texto que mostrará el elemento GUI
	-	item	Para la descripción de los enlaces de ida y vuelta entre las incumbencias de interfaz de usuario con sus correspondientes incumbencias dominantes.
IT Arquitectura Lógica	Roles participantes	invoke	Describe la traza de la incumbencia entre la arquitectura lógica con los roles participantes compuestos por un rol que invoca y otro que realiza o delega
		realice	
		letInvokes	Inversa de invoke
		letRealizes	Inversa de realize
	Interactúa	interact	Identificar todas las interfaces de usuario a desarrollar en condiciones necesarias y suficientes
		letInteract	Inversa de interact
	Situado	laid	Describe en qué hardware estará instalada para su uso la interface de usuario

Tabla 8 Propiedades Objeto en la metodología OCTA

4 Evaluación de la metodología OCTA

En esta sección, se plantea un conjunto de procedimientos centrados en evaluar la metodología OCTA. La evaluación de una metodología para el modelado de las etapas tempranas de un producto software debe ser abordada en diversos pasos.

Verificar la estandarización y estructura del modelo de AT es el primero de los procedimientos que debe ser realizado, dado que sin este tipo de verificación, la evaluación de la metodología carecería de sentido, ya que no podría realizarse una valoración objetiva.

La validación de inferencia de modelado de AT es el segundo procedimiento que se tiene que tener en cuenta para valorar la metodología. Esta validación se basa en medir cuantitativamente la exactitud del modelado a la hora de consultar su conocimiento. Para esto, el experimento que se planteó se basa en la generación de un conjunto de cuestiones que serán consultadas a la ontología de AT.

Finalmente, se plantea el tercer procedimiento que permite evaluar la calidad de la metodología OCTA. En este punto se busca evaluar la flexibilidad de la especificación de AT a la hora de realizar cambios. La valoración de la flexibilidad de un modelo se logra con el diseño de un experimento donde se comparan diferentes especificaciones de etapas tempranas, uno desarrollado con la metodología OCTA y otros con métodos clásicos, permitiendo obtener resultados válidos y significativos estadísticamente.

4.1 Verificación de la estandarización y estructura

El primer procedimiento en la evaluación de la metodología OCTA fue verificar la estandarización y estructura del modelo de AT desarrollado del caso de estudio "Estadio para eventos de atletismo".

Se ha comentado que para la construcción de la ontología que modela los AT se utilizó el editor Protégé. Esto aseguró la estandarización del código generado asegurando la ausencia de errores sintácticos y semánticos del código OWL, así mismo proporcionó los medios para comprobar la presencia de todos los componentes modelados.

Ahora bien, para verificar la estructura organizativa del modelo de AT es necesario un motor de razonamiento, motor de reglas o razonador. Algunos de los razonadores más utilizados RacerPro, KAON2, PELLET, Hermit y FaCT++ fueron descritos en el estado de la cuestión. Todos se encuentran actualmente en un nivel muy alto de fiabilidad. Con este contexto, para la selección de uno de ellos para el desarrollo de esta tesis doctoral se tomó en cuenta la compatibilidad con el editor de ontologías Protégé, concluyendo que el razonador FaCT++ es una de las mejores opciones.

Ahora bien, aplicando los mecanismos del razonador FaCT++ a la ontología que representa el modelo de AT, por un lado se confirmó la ausencia de inconsistencias de partición, es decir, se verificó que las clases y las subclases, por grupos disjuntos, no contenían ni denominación ni propiedades comunes. Y por otro lado, se verificó la ausencia de errores de circularidad en la estructura jerárquica, es decir, no había ninguna clase definida como generalización y especialización de sí misma.

4.2 Validación de inferencias

Para la validación de inferencia, segundo procedimiento para la evaluación de la metodología OCTA, se desarrolló un experimento que se basa en medir cuantitativamente la exactitud del modelado semántico a la hora de responder a consultas de su conocimiento.

4.2.1 Diseño del primer experimento

El **primer paso** en este experimento es la recolección de un **corpus de preguntas** centradas en extraer la semántica de una especificación temprana de un producto software. Este corpus se obtuvo a través de la participación de 159 alumnos estudiantes del último curso de ingeniería informática de tres diferentes clases. En el desarrollo de los cursos de la asignatura de ingeniería informática se conformaban grupos de trabajo para la realización de un caso de estudio de diferente dominio (cocina robótica, carreteras limpias y agencia inmobiliaria). En el desarrollo del Documento de Especificación Técnica (DET) se les solicitaba proponer una selección de **cuestiones** centradas en **extraer la semántica** que percibieran como importante de solicitar a su especificación.

El **segundo paso** es la obtención de un banco de **patrones de preguntas** frecuentes del corpus de preguntas realizadas por los equipos participantes en el desarrollo de diferentes casos de estudio. La creación de estos patrones debe estar basada en los ámbitos y dimensiones de la APV aplicada en el desarrollo del caso de estudio bajo la metodología OCTA que se evaluará.

En el **tercer paso** se determinan las **respuestas** del banco de patrones creados en el segundo paso. En este caso la consulta se realizó manualmente a los modelos conocidos de la especificación desarrollada con la metodología OCTA al Caso de estudio "Estadio para eventos de atletismo".

En el **cuarto paso** es necesario transcribir estos patrones a un determinado lenguaje de consulta ontológica. Esto es necesario, dado que la finalidad de estas preguntas es extraer conocimiento de una ontología que modela los AT.

Ahora bien, para la realización de dichas consultas, se requiere un **motor de consulta** como Jena[41] o Sesame[42] que facilite y agilice la interrogación. El motor orientado a la interrogación de ontologías **DL Query** es un plugin estándar de Protégé que está basado en lógica descriptiva. El lenguaje de consulta de este motor está basado en la **sintaxis de OWL Manchester**[43]. Este lenguaje fue propuesto con el objetivo de proporcionar una sintaxis fácil de usar para OWL, sin utilizar símbolos DL. Plantearon que aunque la sintaxis debe ajustarse en la medida de lo posible con la especificación OWL, el objetivo principal es el de luchar por la legibilidad y una reducción en el tiempo para entender la información que se está representando. La sintaxis Manchester OWL toma ideas de la sintaxis abstracta de OWL pero con muchos menos detalles. Los símbolos matemáticos especiales de la sintaxis DL, tales como ∃, ∀, y ¬, han sido sustituidos por palabras clave más intuitivas, tales como *"some"*, *"only"*, y *"not"*. Un objetivo importante de la decisión de diseño era utilizar una notación fácil de entender con el fin de combatir directamente el problema de la mala interpretación de expresiones de clase por los no expertos en lógica.

Por estos motivos, para la **transcripción de los patrones** de preguntas en el **cuarto paso** de la experimentación del segundo procedimiento para la evaluación de la metodología OCTA se utilizó la sintaxis de OWL Manchester.

---

[41] http://jena.apache.org/

[42] http://www.openrdf.org/

[43] Anexo B

Se ha comentado, que el banco de patrones se construyó a partir de las cuestiones más frecuentes del corpus de preguntas relacionadas con la extracción semántica de las especificaciones creadas de diferentes casos de estudio. En este caso dichas cuestiones seleccionadas fueron transcritas, manualmente, de acuerdo con el vocabulario de la ontología que modeló los AT del Caso de estudio "Estadio para eventos de atletismo" desarrollado con la metodología OCTA. En la Tabla 9 se muestran las cuestiones seleccionadas con su transcripción.

C1	¿Qué usuario puede realizar una incumbencia de gestión básica X?
C1T1	User and interact only (Software and letRealizes some {Basic_Management X})
C2	¿Qué entidad de información de stakeholder puede crearse cuando es realizada una incumbencia de gestión básica X?
C2T2	Stakeholder and create some {Basic_Management X}
C3	¿Qué información se puede consultar cuando es realizada una incumbencia de consulta X?
C3T3	Information and query some {Consult X}
C4	¿Qué incumbencias de consulta pueden ser invocadas por un usuario X?
C4T4	Consult and invoke some (Presentation and (letInteracts only {User X})
C5	¿Qué usuarios pueden ser los realizadores de una incumbencia de consulta X?
C5T5	User and (interact only (Presentation and (letRealizes some {Consult X})))
C6	¿Qué roles de stakeholder tiene una GUI?
C6T6	Stakeholder and interact only Presentation
C7	¿Desde qué capa de presentación se invoca la incumbencia dominante X?
C7T7	Presentation and invoke some {Dominant X}
C8	¿Qué usuarios pueden interactuar con la capa de presentación X?
C8T8	User and interact some {Presentation X}
C9	¿Quién o qué puede lanzar una incumbencia contenedora de servicio X?
C9T9	Thing and launch some {Service X}
C10	¿Qué componente lógico software invoca y cuál realiza una incumbencia dominante X?
C10T10	Software and letInvokes some {Dominant X} or letRealizes some {Dominant X}
C11	¿Qué entidad de información se crea al realizar una incumbencia dominante X?
C11T11	Domain_Information and create some {Dominant X}
C12	¿Qué incumbencias son realizadas por la capa lógica X?
C12T12	Early_Aspect and some realize {Software X}
C13	¿Qué incumbencias son invocadas por la capa lógica X?
C13T13	Early_Aspect and some invoke {Software X}
C14	¿Qué incumbencia es una precondición de la incumbencia de servicio X?
C14T14	Service and preCondicion some {Service X}
C15	¿Qué incumbencias de dominio X tienen restricción de tiempo de respuesta?
C15T15	{Dominant X} and start some Tiempo_Respuesta
C16	¿Qué restricción de valor de tiempo tiene la incumbencia de rendimiento X?
C16T16	ValueTime and isValueTimeOf some Alerta_Asignacion_TR
C17	¿Qué incumbencias dominantes están relacionadas con la incumbencia de restricción de tiempo X?
C17T17	Dominant and start some {Tiempo_Respuesta X}

C18	¿Qué incumbencias de descripción de vista X de interfaz de usuario refleja la realización de la incumbencia dominante X?
C18T18	Descripcion_Vista and item some {Dominant X}
C19	¿Qué incumbencias dominantes están relacionadas con la incumbencia X de descripción de vista?
C19T19	Dominant and item some {Descripcion_Vista X}
C20	¿Qué elementos de GUI contiene una incumbencia X de descripción de vista?
C20T20	ElementoGUI and isContained some {Descripcion_Vista X}
C21	¿Cuál es el texto de los Botones de una incumbencia X de descripción de vista?
C21T21	Boton and texto some {Descripcion_Vista X}
C22	¿Cuál es el color del fondo de la caja de texto de una incumbencia X de descripción de vista?
C22T22	Text_Entry_Box and fondo some {Descripcion_Vista X}
C23	¿Qué atributos tiene un Usuario X?
C23T23	Attribute and compose some {User X}
C24	¿Qué información tiene una incumbencia de consulta X?
C24T24	Information and query some {Concult X}
Nota: El código de identificación es: C seguida de un número que indica la cuestión de la muestra. C acompañada de número y seguida de T y un número indica la transcripción a DL Query realizada respecto a la cuestión de la muestra.	

Tabla 9 Transcripción de patrones de las cuestiones seleccionadas en lenguaje DL Query

En este punto, es necesario puntualizar que el experimento planteado tiene como objetivo determinar la exactitud de las inferencias de la ontología que modela el Caso de estudio "Estadio para eventos de atletismo"; esta verificación se obtiene en el **quinto paso** de esta experimentación.

En el **quinto paso** se utiliza la funcionalidad del motor de consulta DL Query para realizar las consultas de las preguntas transcritas en el cuarto paso y extraer la inferencia de conocimiento de la ontología de AT del caso de estudio a evaluar.

Finalmente, en el **sexto paso** los resultados de estas inferencias se comparan con las respuestas previamente determinadas del banco de cuestiones de esta competencia. En este paso la validación se realiza al comprobar que las inferencias realizadas a partir del modelo declarado concuerdan con el significado esperado de las incumbencias contenidas en el modelo de AT.

4.2.2 Método de aplicación del primer experimento

En el quinto paso se valida la inferencia del modelado de AT; esto implica evaluar las respuestas de las consultas transcritas en el cuarto paso del experimento. Este análisis será realizado a través de métricas fundamentales de técnicas de evaluación de ontologías, precisión y exhaustividad (Brank, Grobelnik, & Mladenić, 2005). Exhaustividad (del inglés *Recall*) se calcula como la fracción de instancias correctas entre todas las instancias que de hecho pertenecen al conjunto relevante, mientras que la precisión (del inglés *precision*) es la fracción de instancias correctas entre aquellas que la inferencia cree que pertenecen al conjunto relevante.

La aplicación de estas métricas a la validación de inferencia del modelado de AT se tiene que plantear desde el punto de vista de los patrones de preguntas transcritos, buscando los parámetros de precisión y exhaustividad para cada consulta.

La precisión es el ratio entre el número de conceptos relevantes recuperados entre el número de conceptos recuperados, quedando la siguiente expresión:

$$\text{Precisión} = \frac{|\{\text{Conceptos Relevantes}\} \cap \{\text{Conceptos Recuperados}\}|}{|\{\text{Conceptos Recuperados}\}|}$$

De esta forma, cuanto más se acerque el valor de la precisión al valor nulo, mayor será el número de conceptos recuperados que no se consideren relevantes. Si por el contrario, el valor de la precisión es igual a uno se entenderá que los conceptos recuperados son todos relevantes.

La exhaustividad viene a expresar la proporción de conceptos relevantes recuperados, comparado con el total de los conceptos que son relevantes y existentes en la ontología, con total independencia de que éstos se recuperen o no; la expresión queda de la siguiente forma:

$$Exhaustividad = \frac{|\{\text{Conceptos Relevantes}\} \cap \{\text{Conceptos Recuperados}\}|}{|\{\text{Conceptos Relevantes}\}|}$$

Si el resultado de esta fórmula arroja como valor 1, se tendrá la exhaustividad máxima posible, y esto viene a indicar que se han encontrado todos los conceptos relevantes existentes en la ontología, siendo la recuperación de las respuestas a las cuestiones entendida como perfecta. Por el contrario en el caso que el valor de la exhaustividad sea igual a cero, se tiene que las respuestas obtenidas no poseen relevancia alguna.

Una vez las métricas han sido calculadas, cada una ofrece una interpretación que nos permite obtener información sobre la validación de inferencia del modelado de AT, según el valor que haya tomado dicha métrica.

La precisión mide el porcentaje de resultados obtenidos con el motor de consulta que son correctos. Es decir, es útil para comprobar errores de modelado en las propiedades objeto.

La exhaustividad mide la proporción de resultados obtenidos con la consulta que faltan. Es decir, es útil para detectar las propiedades objeto faltantes en el modelado.

4.2.3 Resultados de la aplicación del método del primer experimento

La fase de evaluación de resultados de este experimento tiene el objetivo de validar cuantitativamente la exactitud de inferencia semántica del modelado de las primeras etapas de un producto software desarrollada con la metodología OCTA.

El experimento que se planteó se basa en la consulta de la ontología del modelado de AT con una selección de un conjunto de preguntas de un corpus. Ahora bien, la obtención de la batería de preguntas a consultar debe ser una fase previa al modelado de una ontología (Grüninger & Fox, 1995). En el caso de la evaluación del Caso de estudio "Estadio para eventos de atletismo" desarrollado con la metodología OCTA, todas las preguntas fueron tomadas en cuenta para la creación de la ontología.

Con este escenario, en la fase de validación semántica se obtuvo un valor de **precisión igual a uno**, dado que todas las respuestas a las consultas fueron relevantes. Así mismo, se obtuvo un valor de **exhaustividad igual a uno**, dado que se encontraron todas las respuestas esperadas de la ontología. El hecho de obtener una precisión y exhaustividad máximas permite validar la inferencia semántica del Caso de estudio "Estadio para eventos de atletismo" desarrollado con la metodología OCTA asegurando el cumplimiento del objetivo del proceso.

4.3 Evaluación de la facilidad al cambio

En la introducción de esta memoria se planteó que el objetivo específico principal de esta tesis doctoral era la propuesta de una metodología para la organización del conocimiento de las etapas tempranas de un producto software OO que facilite la detección de conflictos y su evolución. A este respecto, la propuesta de la metodología denominada como OCTA propone una nueva forma de organizar y representar el conocimiento de las etapas tempranas de un desarrollo software. La especificación resultante hace posible la gestión efectiva de la evolución del producto software en cualquier momento del desarrollo de las etapas tempranas. Para comprender los beneficios de esta aportación es necesario ponerse en lugar del analista de mantenimiento, que tendrá que hacer frente a una corriente inacabable de peticiones de ampliación y modificación del producto software.

En este contexto, el tercer procedimiento se basa en una evaluación de la flexibilidad al cambio o evolución a diferentes especificaciones basadas en un mismo caso de estudio para demostrar cómo los modelos de AT desarrollado con la metodología OCTA son más eficientes que los modelos resultantes de un desarrollo clásico. Esta aserción se demuestra con la propuesta del segundo experimento "metodología OCTA vs desarrollo clásico" que permite obtener resultados válidos y significativos estadísticamente.

4.3.1 Diseño del segundo experimento

En el **primer paso** del experimento se deben **obtener dos especificaciones** efectivas de etapas tempranas de un producto software, del mismo caso de estudio a evaluar, desarrolladas bajo una **metodología clásica**.

El **segundo paso** del experimento se centra en **describir una modificación** o **evolución** a realizar en las diferentes especificaciones desarrolladas del mismo caso de estudio. Posteriormente, esta propuesta de modificación debe ser **aplicada de forma metódica** en las especificaciones desarrolladas bajo diferentes metodologías pero con el mismo caso de estudio. En este proceso se identifican todas las incumbencias involucradas en la modificación que están relacionadas semánticamente.

En el **tercer paso** las dos especificaciones técnicas desarrolladas con la metodología clásica y el modelo de AT desarrollado con la metodología OCTA son **modificados o evolucionados por un grupo de expertos analistas**, desconocedores del caso de estudio, basándose en la descripción de cambio propuesta en el paso dos para la especificación del producto software.

En el **paso cuatro** se contabilizan las incumbencias modificadas y añadidas por parte de los expertos analistas del proceso de evolucionar las especificaciones del paso tres. Posteriormente, se realiza una **evaluación** con el cruce de resultados estadísticos entre estos datos y los registrados previamente en el desarrollo minucioso de la propuesta de modificación por parte del árbitro en las tres especificaciones del mismo caso de estudio.

4.3.2 Metodología OCTA vs Desarrollo clásico

En la sección anterior 3.3 de este capítulo se describió el Caso de estudio "Estadio para eventos de atletismo", el cual fue la base para el desarrollo de la especificación de sus etapas tempranas en un modelo de AT a través de la metodología OCTA propuesta en esta tesis doctoral.

Ahora bien, para la realización del **primer paso** del segundo experimento, esta misma descripción básica fue entregada a 32 equipos conformados en promedio por cinco participantes con el perfil de estudiantes del último curso de ingeniería informática. Se solicitó a los equipos especificar las etapas tempranas de un producto software con un desarrollo clásico basándose en las líneas generales descritas del Caso de estudio "Estadio para eventos de atletismo". En esta actividad, se dejó total libertad de decisión al definir los límites del caso de estudio pero se pidió razonar y explicar estas decisiones de forma coherente al desarrollar la especificación. Algunos desarrollaron la primera parte, relacionada con los resultados de las pruebas de atletismo; otros la segunda, relacionada con la atención de incidencias en el estadio; y algunos otros, las dos partes. Se les propuso la opción de basar la organización de su DET en alguno o en la combinación de los estándares IEEE 830 y ESA PSS-05; sin embargo, sin ninguna obligación de utilizarlos.

Al finalizar del desarrollo de los trabajos fueron analizados desde distintos puntos de vista por cuatro profesores investigadores expertos en el tema. De entre todas las especificaciones, desarrolladas con una metodología clásica, se seleccionaron las dos soluciones mejor calificadas.

Ahora bien, para la evaluación del segundo experimento es necesario medir el esfuerzo requerido para la realización de una modificación controlada en los diferentes tipos de especificaciones tempranas de un mismo caso de estudio. Por lo tanto, en el **paso dos** de este experimento fue necesario definir un posible cambio en el Caso de estudio "Estadio para eventos de atletismo". La propuesta de modificación o evolución es descrita a continuación:

En el modelo actual del caso de estudio "Estadio para eventos de atletismo" cuando se atiende una incidencia por parte de un técnico responsable principal, después que el técnico evalúa la incidencia, avisa la cantidad de técnicos de apoyo. El sistema escoge a los técnicos y les avisa su rol en un equipo de apoyo. En principio estos técnicos se trasladan al lugar de la incidencia sin posibilidad de cuestionar la asignación.

En este contexto, se puede suponer un cambio previsible que permita que los técnicos asignados como apoyo rechacen o acepten la asignación, lo cual serviría para asegurar que siempre se atiende la incidencia con los recursos necesarios.

Definida la propuesta de modificación, fue aplicada de forma cuidadosa por el árbitro del experimento en cada una las tres especificaciones de las etapas tempranas desarrolladas con el Caso de estudio "Estadio para eventos de atletismo". El desarrollo de la modificación implicó identificar y modificar todos los requisitos involucrados en la modificación, así mismo registrar los requisitos añadidos para la correcta evolución de la especificación. Aunado a esto se identificaron las capas o componentes de la arquitectura lógica involucrados en el cambio.

En este escenario, en la modificación realizada en la especificación desarrollada con la metodología OCTA, la cuantificación de los requisitos modificados y añadidos fue de forma directa, dado que este modelado se desarrolla con requisitos atómicos. Sin embargo, para la cuantificación de incumbencias de la modificación realizada en cada una de las especificaciones realizadas con metodologías clásicas muchos de los requisitos contenían dos o más requisitos atómicos. Por lo tanto, para cuantificar los requisitos en cada caso fue necesario convertir algunos requisitos descritos de forma libre en requisitos contenedores. Ahora bien, en el modelado desarrollado con la metodología OCTA la identificación de incumbencias modificadas y añadidas para su cuantificación fue una actividad simple de realizar. Esto se debe al modelado natural de las relaciones semánticas entre requisitos y sus capas de arquitectura lógica participantes en una

ontología de AT. Sin embargo, el proceso de identificación de requisitos y capas arquitectónicas a cada una de las especificaciones desarrolladas con la metodología clásica conllevó añadir relaciones semánticas que debían haber estado antes de la modificación. En esta línea, el modelado de la semántica de las relaciones en cada una de las especificaciones desarrolladas con la metodología clásica solo fue reflejado en parte con el apoyo de una herramienta de gestión de requisitos que permiten la descripción de una simple traza.

Una vez aplicada y cuantificada la modificación a realizar en las especificaciones de las etapas tempranas del caso de estudio se continuó con el **paso tres** de este experimento. En este punto se solicitó la participación de seis expertos en análisis de software de diferentes factorías de software.

A continuación de la aceptación de los analistas participantes se les hizo la solicitud de realizar la modificación a las tres especificaciones de las etapas tempranas del Caso de estudio "Estadio para eventos de atletismo" en la que "**el sistema permita que los técnicos asignados como apoyo rechacen o acepten la asignación**". Ahora bien, primero se les hizo entrega de la descripción de la modificación. Sin embargo, la entrega de las diferentes especificaciones tuvo una secuencia controlada.

Es sabido que entre las tres especificaciones técnicas están dos con modelos realizados con una metodología clásica y una modelada con la metodología OCTA. Dado que las tres especificaciones técnicas son del mismo caso de estudio y en las tres se tenía que realizar la misma modificación en los modelos, los analistas podrían haber adquirido conocimientos del dominio del caso de estudio que podrían haber afectado a los resultados del experimento. Por lo tanto, con la intención de evitar que el aprendizaje del dominio del caso de estudio inicial tuviera desventaja en la efectividad del cambio con los posteriores se decidió proporcionar las especificaciones en un orden específico. La **primera especificación** entregada fue una de las desarrolladas con la metodología clásica, que se nombra a partir de ahora como **A**; al terminar el cambio se les entregó la especificación realizada con la metodología OCTA y finalmente se entregó la **segunda especificación** realizada con una metodología clásica, que se nombra a partir de ahora como **B**. Esta secuencia no evita que exista desequilibrio entre las dos especificaciones realizadas por la metodología clásica, pero equilibra la especificación desarrollada con OCTA contra las clásicas. En este sentido, lo esperado es que la realización de la modificación en la especificación con la metodología OCTA sea mejor en cualquier caso para demostrar la calidad de la metodología propuesta.

Ahora bien, para realizar una evaluación efectiva es necesario obtener porcentajes de métricas específicas, por lo tanto, son necesarias algunas reglas básicas a cumplir en la realización de la modificación en cada una de las especificaciones. Las reglas aplicadas en este experimento son descritas a continuación:

1. Si se va a cambiar la semántica de una incumbencia o requisito hay que comprender todo lo que afecta, es decir, identificar todas las incumbencias dominantes y transversales participantes.
2. Si un cambio a una incumbencia afecta alguna entidad de información, se tiene que estar familiarizado con todas las incumbencias que dependan de esa entidad de información.
3. Si un cambio afecta a la justificación de las capas de la arquitectura lógica, se tiene que decir si la organización de las capas ha cambiado. Si es así, se tendrá que entender el resto de incidencias que hagan referencias a las capas nuevas y modificadas.

Las reglas anteriores se refieren a lo que hay que cumplir en desarrollo, sin embargo, se indicaron dos reglas limitadoras:

4. El bloque a considerar como unidades son la incumbencia o requisito atómico. Por lo tanto, si en alguna especificación existen requisitos contenedores se contabilizarán como unidades los requisitos atómicos obtenidos de la transformación semántica.
5. Permitir una cantidad de tiempo limitado para resolver un cabio semántico. La cifra que se designó fue de 30 minutos.

Por otro lado, en la parte del experimento para la modificación o evolución de las especificaciones de los desarrollos clásicos A y B, a los seis expertos analistas se les proporcionaron los DTR con su respectivo archivo digital de la ER para ser utilizada en el gestor de requisitos Visure Requirements. Al estar familiarizados los expertos analistas con el funcionamiento de herramientas de gestión de requisitos no tuvieron problemas en el desarrollo de esta parte del experimento.

Ahora bien, en la parte del experimento para la evolución de la especificación desarrollada con la metodología OCTA a los mismos seis analistas se les proporcionó un archivo OWL para ser gestionado en Protégé a través del razonador FaCT++ y el interrogador de ontologías DL query. En este proceso se observó el desconocimiento de este tipo de herramientas, por lo tanto, fue necesario adiestrar a nivel básico a los expertos analistas en el editor de ontologías, el razonador seleccionado y en la utilización del interrogador de ontologías.

En el final del paso tres de este experimento con el conocimiento del cambio previsto fueron modificadas las especificaciones clásicas y la desarrollada bajo la metodología OCTA permitiendo a todos los expertos 30 minutos para la modificación de cada uno de los proyectos.

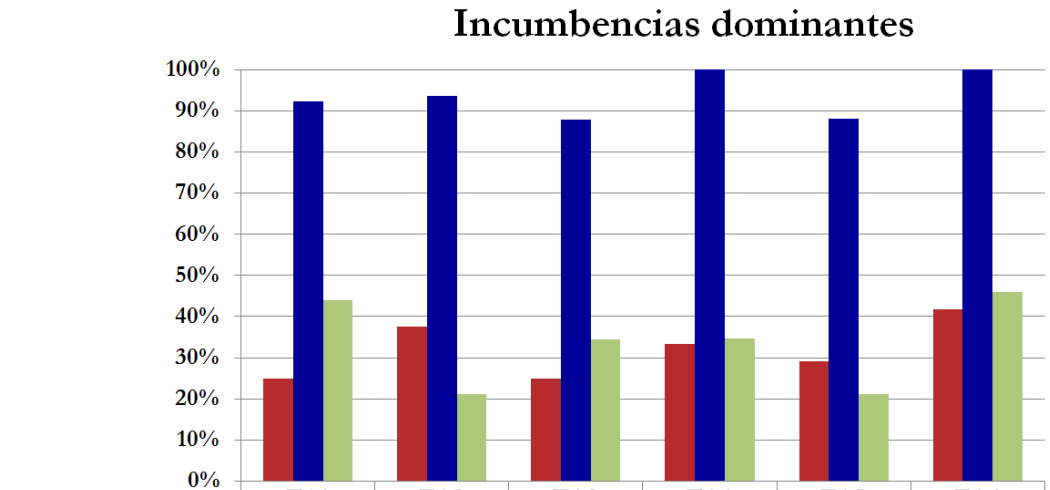
El **paso cuatro** fue el momento de la extracción, comparación y análisis de los resultados. La primera actividad fue la cuantificación de las incumbencias que fueron modificadas y añadidas correctamente en cada una de las especificaciones por parte de los expertos analistas. Ahora bien, se ha dicho que era necesario obtener métricas para grupos de incumbencias específicas para realizar una evaluación efectiva. Por lo tanto, las incumbencias fueron organizadas en temas específicos que anteriormente fueron la base para determinar las reglas que tenían obligación de cumplir los expertos analistas en la realización de la modificación en cada una de las especificaciones. Estos temas básicos son: incumbencias dominantes, incumbencias transversales, entidades de información, incumbencias relacionadas con las entidades de información y capas o componentes de la arquitectura lógica.

La primera métrica implica calcular el **porcentaje de incumbencias dominantes** que fueron modificadas o añadidas en el proceso de evolución de cada una de las especificaciones del mismo Caso de estudio "Estadio para eventos de atletismo" respecto a las incumbencias determinadas en el paso dos del experimento "metodología OCTA vs desarrollo clásico".

Ahora bien, dirigido a percibir mejor esta evaluación es necesario recordar que las incumbencias dominantes en su mayoría son requisitos atómicos, es decir, requisitos que no pueden ser separados semánticamente. Por lo tanto, el número de incumbencias dominantes en una especificación desarrollada con la metodología OCTA es mucho mayor que en los desarrollados con metodologías clásicas. Otro punto necesario a aclarar es que las incumbencias dominantes especificadas con la metodología OCTA

típicamente se encuentran dentro de los requisitos clasificados como funcionales en el desarrollo de una metodología clásica. Sin embargo, no todos los requisitos funcionales están dentro de la agrupación de las incumbencias dominantes. Es decir, la correlación de las incumbencias entre modelados desarrollados bajo diferentes metodologías no es directa ni trivial.

Con este contexto, la evaluación de los resultados estadísticos de la primera métrica expone que el modelado de incumbencias dominantes es el más numeroso. Esto hace que el proceso de modificación o evolución de las especificaciones desarrolladas con metodologías clásicas aparente un cierto grado de efectividad. Sin embargo, con este experimento se obtuvieron datos contundentes para determinar que es mejor la calidad de un modelado de incumbencias dominantes de un producto software desarrollado con la metodología OCTA a pesar del mayor número de incumbencias. Los resultados de esta evaluación se perciben de forma clara en la gráfica de la Figura 56.



	EA1	EA2	EA3	EA4	EA5	EA6
Clásico A	25%	38%	25%	33%	29%	42%
Metodología OCTA	92%	94%	88%	100%	88%	100%
Clásico B	44%	21%	34%	35%	21%	46%

Figura 56 Porcentaje de incumbencias dominantes modificadas o añadidas

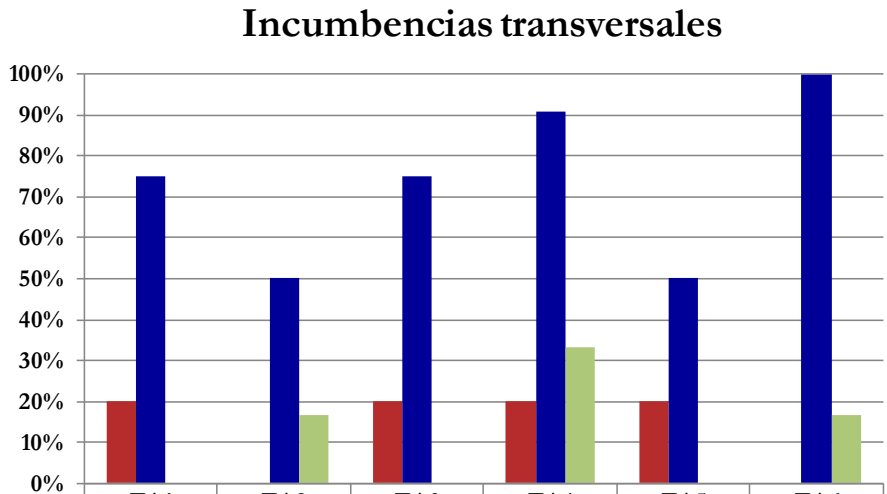
La segunda métrica implica calcular el **porcentaje de incumbencias transversales** que fueron modificadas o añadidas en el proceso de evolución de cada una de las especificaciones del caso de estudio respecto a las incumbencias transversales determinadas anteriormente.

En una correlación básica, las incumbencias transversales podrían representar los requisitos no funcionales típicos de una especificación desarrollada con una metodología clásica. Sin embargo, en estas especificaciones los requisitos no funcionales suelen ser el agrupamiento generalizado de todos los demás requisitos diferentes de los funcionales. En este sentido, solo se puede asegurar que todas las incumbencias transversales especificadas bajo la metodología OCTA están consideradas dentro de los requisitos no funcionales desarrollados con una metodología básica.

En este sentido, la metodología OCTA proporciona los medios para el modelado de las incumbencias transversales más recurrentes en productos software de sistemas de información. Sin embargo, es necesario aumentar constantemente los procedimientos necesarios para el modelado del crecimiento eminente de tipos de incumbencias transversales.

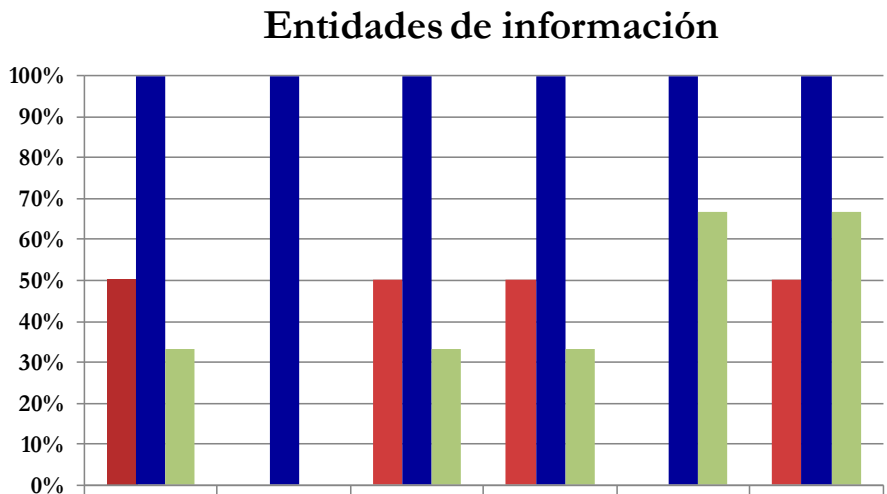
Ahora bien, la evaluación de los resultados estadísticos de la segunda métrica permite establecer el grado de calidad ante el proceso de modificación o evolución de las diferentes especificaciones de incumbencias transversales. En este caso se observó la mayor flexibilidad de la especificación de las incumbencias transversales con la metodología OCTA. En contraste, las especificaciones desarrolladas con una metodología clásica exponen un modelado de incumbencias transversales con escasa organización semántica.

En la Figura 57 se puede observar de forma clara la calidad en la especificación de incumbencias transversales desarrollada con la metodología OCTA.



	EA1	EA2	EA3	EA4	EA5	EA6
Clásico A	20%	0%	20%	20%	20%	0%
Metodología OCTA	75%	50%	75%	91%	50%	100%
Clásico B	0%	17%	0%	33%	0%	17%

Figura 57 Porcentaje de incumbencias transversales modificadas o añadidas

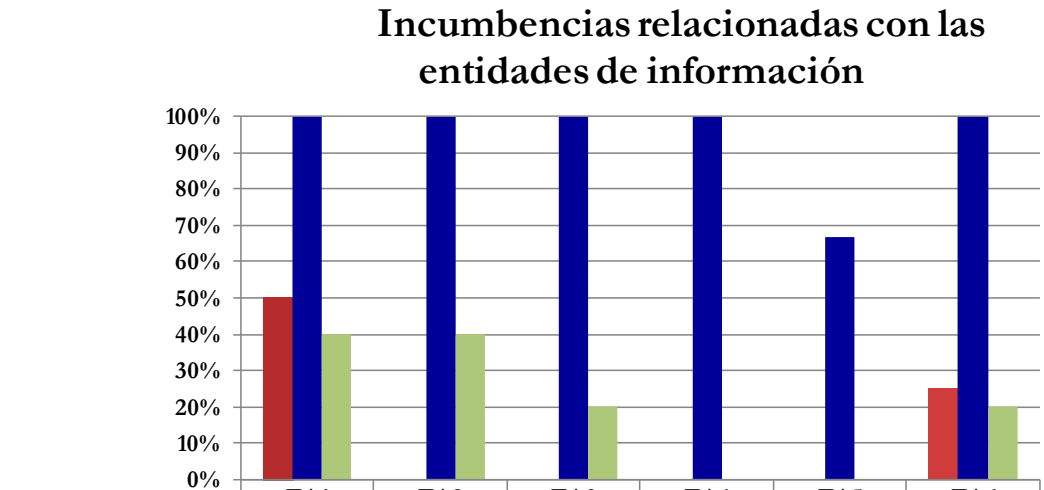


	EA1	EA2	EA3	EA4	EA5	EA6
Clásico A	50%	0%	50%	50%	0%	50%
Metodología OCTA	100%	100%	100%	100%	100%	100%
Clásico B	33%	0%	33%	33%	67%	67%

Figura 58 Porcentaje de incumbencias que representan entidades de información relacionadas con la evolución de la especificación

La tercera y cuarta métrica están relacionadas con la comparación de las incumbencias entorno a las **entidades de información** que fueron modificadas o añadidas en el proceso de evolución de cada una de las especificaciones del mismo Caso de estudio "Estadio para eventos de atletismo" contra las identificadas por el árbitro responsable del segundo experimento. La tercera métrica se centra en calcular los porcentajes de incumbencias modificadas o añadidas por los analistas que representan las entidades de información comparadas con las incumbencias esperadas. Se puede observar en la Figura 58 la flexibilidad que proporciona una especificación desarrollada con la metodología OCTA para la identificación de incumbencias que representan entidades de información.

La cuarta métrica se centra en calcular el porcentaje de las incumbencias modificadas o añadidas por los analistas que dependen de esa entidad de información (por ejemplo las incumbencias de atributo) contra las incumbencias identificadas por el árbitro. En la Figura 59 se puede percibir de forma clara la calidad que proporciona una especificación desarrollada con la metodología OCTA para gestionar las incumbencias que modelan el contenido intrínseco de las entidades de información.

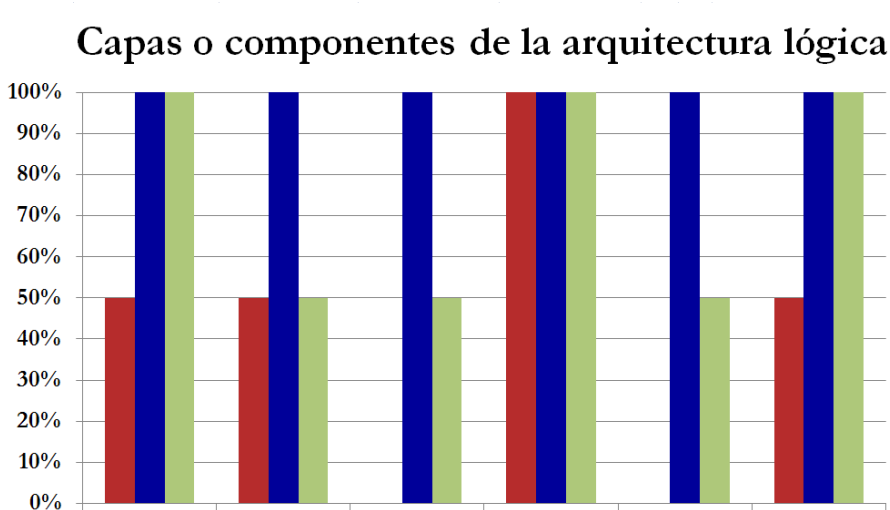


	EA1	EA2	EA3	EA4	EA5	EA6
Clásico A	50%	0%	0%	0%	0%	25%
Metodología OCTA	100%	100%	100%	100%	67%	100%
Clásico B	40%	40%	20%	0%	0%	20%

Figura 59 Porcentaje de incumbencias relacionadas con las entidades de información participantes en la evolución de la especificación

En el desarrollo de estas métricas se pudo percibir que una especificación desarrollada con una metodología clásica centra los esfuerzos en el modelado en los diagramas de apoyo, descuidando la relación semántica de estas incumbencias relacionadas con las entidades de información con la especificación de requisito o incumbencia.

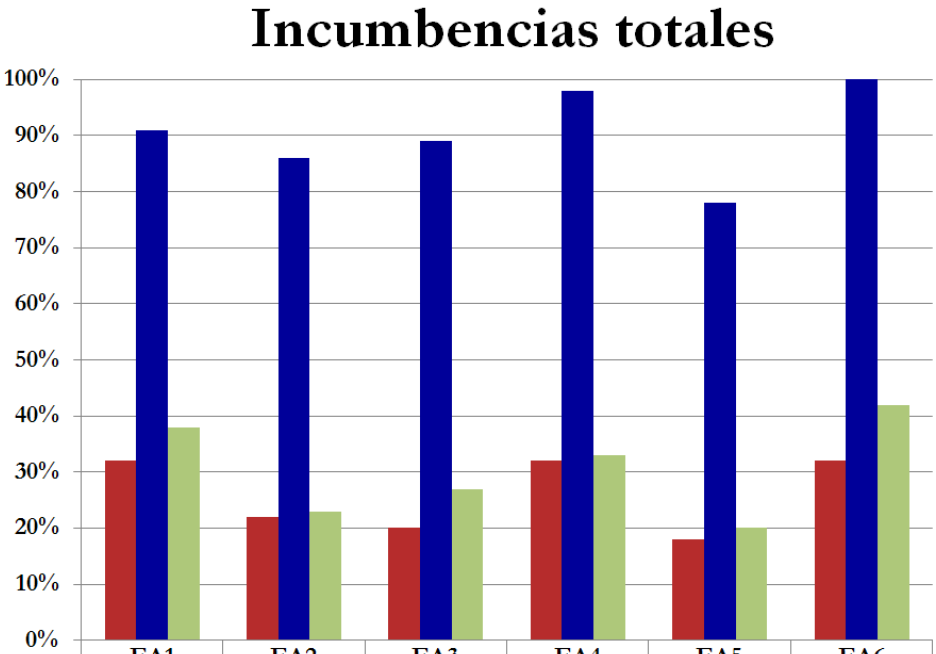
La quinta métrica implica calcular el porcentaje de incumbencias, que **representan las capas o componentes de la arquitectura lógica**, identificadas por los expertos analistas contra el porcentaje total determinado anteriormente por el árbitro responsable del experimento. En la Figura 60 se puede observar la efectividad total para la identificación de las capas o componentes modelados en la arquitectura lógica de un producto software con la metodología OCTA.



	EA1	EA2	EA3	EA4	EA5	EA6
Clásico A	50%	50%	0%	100%	0%	50%
Metodología OCTA	100%	100%	100%	100%	100%	100%
Clásico B	100%	50%	50%	100%	50%	100%

Figura 60 Porcentaje de incumbencias que representan capas o componentes de la arquitectura lógica relacionadas con la evolución de la especificación

El desarrollo de esta métrica expuso que un cambio involucra solo un pequeño grupo de capas de la arquitectura lógica. En la modificación propuesta al Caso de estudio "Estadio para eventos de atletismo" el número de capas es muy reducido, permitiendo cierta facilidad para la identificación de los elementos de la arquitectura lógica en cualquiera de las especificaciones. Sin embargo, en la evaluación se demuestra la efectividad total de una especificación desarrollada con la metodología OCTA.



	EA1	EA2	EA3	EA4	EA5	EA6
Clásico A	32%	22%	20%	32%	18%	32%
Metodología OCTA	91%	86%	89%	98%	78%	100%
Clásico B	38%	23%	27%	33%	20%	42%

Figura 61 Porcentaje de incumbencias totales modificadas y añadidas por los expertos analistas

Ahora bien, las cinco métricas aplicadas anteriormente fueron planteadas para la evaluación de la flexibilidad al cambio a las diferentes especificaciones del mismo caso de estudio proporcionadas a este experimento. El objetivo era analizar por temas específicos las incumbencias modificadas o añadidas por los expertos analistas en cada una de las especificaciones. En todos los casos la especificación desarrollada con la metodología OCTA reflejó una mayor flexibilidad al cambio. Con este escenario, se procedió a la evaluación global de todas las incumbencias involucradas en cada uno de los temas expuestos en las anteriores métricas.

En los resultados de la evaluación global de la Figura 61 se puede advertir de forma clara que la especificación desarrollada con la metodología OCTA permite con mayor eficiencia los cambios o evolución del modelado de un producto software en sus etapas tempranas.

4.3.3 Análisis del segundo experimento

Es interesante comentar que cuando se solicitó la participación de seis expertos en análisis de software de diferentes factorías de software solo se verificó que tuvieran experiencia en el desarrollo en las primeras etapas de productos software. En los inicios del desarrollo del experimento se percibió en todos los expertos analistas el conocimiento de la utilización de herramientas de gestión de requisitos. Sin embargo, también se observó el desconocimiento de herramientas editoras de ontologías, razonadores e interrogadores de ontologías. En esta línea, se ha comentado que fueron adiestrados en el manejo de este tipo de herramientas. En este proceso, se advirtió que dos expertos tenían conocimiento de los conceptos de ontologías a un nivel básico, los cuales desarrollaron con mayor eficiencia la evolución de la especificación desarrollada con la metodología OCTA. Sin embargo, no es un factor que afecte al resultado global del experimento.

Ahora bien, en la realización de la modificación controlada a las especificaciones A y B el primer problema al que se enfrentaron los expertos fue el entendimiento de la organización de requisitos que propusieron los equipos.

Las dos especificaciones proponían una agrupación de alto nivel de los clásicos temas de requisitos funcionales y no funcionales. En la primera especificación, que es nombrada como A, se organizaron los requisitos funcionales en los siguientes temas:

- Requisitos relacionados con la gestión de usuario
- Requisitos relacionados con las unidades de trabajo
- Requisitos relacionados con las incidencias
- Requisitos referidos al rol del operador y al alta de incidencias
- Requisitos relacionados con la asignación de incidencias
- Requisitos generales del sistema
- Requisitos relacionados con el rol del supervisor
- Requisitos de forma de datos

Los temas de los requisitos no funcionales fueron los siguientes:

- Requisitos de rendimiento
- Requisitos de interfaz
- Requisitos de operación
- Requisitos de recursos
- Requisitos de verificación
- Requisitos de documentación
- Requisitos de seguridad

- Requisitos de fiabilidad y disponibilidad
- Requisitos de mantenimiento
- Requisitos de seguridad frente a fallos

En la segunda especificación, que es nombrada como B, los requisitos funcionales se organizaron en los siguientes temas:

- Características de incidencias
- Tratamiento de incidencias
- Operaciones exclusivas del supervisor
- Operaciones de y sobre los administradores y supervisor
- Operaciones de y sobre los servicios
- Autenticación y Copia de seguridad
- Otras funcionalidades

Los temas de los requisitos no funcionales en la especificación B fueron los siguientes:

- Consumo de recursos
- Fiabilidad y disponibilidad
- Interfaz
- Restricciones
- Seguridad

Todos los expertos pensaron en una primera instancia que estas organizaciones sugerían centrarse en las agrupaciones de requisitos funcionales para localizar el tema en el que se ubicarían las nuevas incumbencias.

En el caso de la especificación B existe una descripción de los temas propuestos que aparentemente permiten ubicar rápidamente la agrupación pertinente. Sin embargo, se puede observar que el tema *"tratamiento de incidencias"* se cruza con *"operaciones de y sobre los servicios"*, donde *servicio* es definido como cualquier usuario del sistema encargado de la resolución directa de incidencias. Así mismo, si una instancia dada de alta por un supervisor también quiere decir que tenemos que tomar en cuenta la agrupación *"Operaciones exclusivas del supervisor"*. En este punto, al buscar la relación con los temas de la agrupación de requisitos no funcionales, la primera relación clara es con el tema de *"requisitos de interfaz"* por los cambios que podrían surgir en el diseño de la GUI. Sin embargo, también es lógico pensar que existe relación con los temas *"consumo de recursos"*, *"fiabilidad y disponibilidad", "seguridad"* y sobre todo con la agrupación generalizada de *"restricciones"*.

Por otro lado, los temas de la agrupación de requisitos funcionales de la especificación A no tienen ninguna descripción por lo que todos los analistas invirtieron más tiempo que en la especificación B en leer y entender todos los temas propuestos. Así mismo, todos los temas propuestos en la agrupación de requisitos no funcionales requirieron un mayor análisis.

El segundo problema al que se enfrentaron fue identificar qué entidades de información están involucradas en las incumbencias o requisitos atómicos que se modificarían, si la propia entidad conlleva un cambio y qué incumbencias estarían afectadas. En el caso de las dos especificaciones las entidades de información están diseminadas por el documento, generando entre los analistas una mayor inversión de tiempo.

Ahora bien, algunas entidades de información no están especificadas en la ER, sino que están plasmadas en las secciones complementarias del DTR. En este sentido, las entidades de información no son la única semántica que en el desarrollo de una

especificación clásica es descrita fuera de la ER. Por lo tanto, en el análisis desarrollado por los expertos significó leer prácticamente todo el DTR y gran parte de la ER.

El siguiente problema para los expertos analistas se encontró en la falta del modelado semántico en la ER de la información relacionada con la arquitectura lógica. En los casos de las especificaciones A y B el razonamiento de la arquitectura lógica no está justificado con la organización de requisitos. El único conocimiento relacionado se encuentra en la especificación B, en la cual se realizó una matriz incompleta de algunos requisitos con artefactos software.

En esta línea, es sabido que es recomendable en el desarrollo de un producto software que la ER debe ser la base para dirigir el diseño de la arquitectura software. En el caso de la metodología OCTA la correlación razonada de estas etapas tempranas es un conocimiento obligado a plasmar con algo más que una simple relación. Esto propició que todos los expertos analistas fueran capaces de identificar las capas o componentes de la arquitectura lógica participantes en la evolución en la especificación desarrollada con esta metodología.

Por otro lado, en las especificaciones A y B la relación entre requisitos es expuesta a través de matrices que relacionan los requisitos clasificados como funcionales con los no funcionales. Este tipo de relaciones especificadas implican poco modelado semántico, dejando este conocimiento plasmado en la descripción de los propios requisitos. Es decir, el conocimiento de las relaciones semánticas entre incumbencias no es claro. Por lo tanto, tuvieron muchos problemas para resolver los problemas de conflictos, acoplamientos y redundancias entre requisitos y las capas de la arquitectura lógica en las especificaciones desarrolladas bajo la metodología clásica. Sin embargo, la especificación con la metodología OCTA registra todas las relaciones semánticas entre todos los elementos del modelado. Con este contexto, los expertos analistas realizaban una actividad de modificación más efectiva con el modelo de AT.

Otra de las ventajas que proporciona un modelado de las etapas tempranas de un producto software con la metodología OCTA es la relacionada con la identificación de las incumbencias. Con la base de los patrones de preguntas desarrolladas en el primer experimento para la validación de inferencia semántica, es factible realizar una interrogación semántica secuencial efectiva al modelado de AT. Esto es posible por la organización del conocimiento que proporciona la APV aplicada. El entendimiento de la APV proporciona la guía para identificar a través de la interrogación semántica las incumbencias específicas involucradas en una modificación.

Por todo lo anterior, queda claro por qué en los resultados de la evaluación del modelado desarrollado a través de la metodología OCTA se observa una alta flexibilidad al cambio en un tiempo razonable. En contraparte, se visualiza un modelado pésimo para la localización de los efectos del cambio cuando la especificación de un producto software es desarrollada bajo una metodología clásica.

CAPÍTULO V
CONCLUSIONES Y TRABAJOS FUTUROS

Capítulo VI. CONCLUSIONES Y TRABAJOS FUTUROS

1 Conclusiones

Actualmente el paradigma OA se puede aplicar en todas las etapas de cualquier PDS. Sin embargo, la aplicación de los conceptos OA no son igual de efectivos en todas las etapas. Las etapas de codificación son las que acogen a la OA con mayor madurez pero las etapas tempranas tienen pocas propuestas fiables y eficientes. Uno de los posibles motivos es porque la comunidad de ingeniería del software del área de la OA no ha llegado a un consenso sobre cómo conceptualizar un aspecto en cada una de las etapas de un PDS. Por lo tanto, es necesario delimitar las diferentes conceptualizaciones de aspectos que se tienen en las distintas etapas a lo largo del ciclo de vida de un producto software. A este respecto, también es necesario tener clara esta conceptualización en cualquier PDS propuesto para la creación de productos software. La propuesta del **marco de referencia para la conceptualización de aspectos** ofrecida en esta tesis doctoral cumple con estos objetivos.

El marco de referencia propuesto se obtuvo centrándose en un exhaustivo análisis de los PDS más conocidos para evaluar su relación con las diferentes conceptualizaciones de aspectos propuestas en diferentes etapas de un DSOA por distintos grupos de investigación. El resultado fue un marco de referencia para la conceptualización de aspectos a lo largo de cualquier PDS con una división de las conceptualizaciones en tres niveles: aspectos tempranos, aspectos medios y aspectos finales. La definición de los límites de estas conceptualizaciones orienta a los diferentes expertos de un equipo de trabajo en un DSOA delimitando de forma clara sus responsabilidades. Así mismo, la delimitación conceptual permite utilizar lenguajes con las mismas restricciones. Por lo tanto, la transformación de los modelos creados en un mismo nivel conceptual guarda una trazabilidad que proporciona un viaje de ida y vuelta, afrontando cualquier cambio surgido en cualquier instante en un mismo nivel conceptual. Así mimo, este marco de referencia proporciona libertad a cualquier desarrollo de producto software para aplicar un DSOA en los tres, dos o solo un nivel conceptual.

Ahora bien, muchos estudios han demostrado la importancia que tienen las etapas tempranas para el proceso de desarrollo de software, especialmente la especificación y gestión de requisitos. En consecuencia la presente tesis se ha enfocado principalmente al primer nivel conceptual de aspectos. Este nivel conceptual son los **aspectos tempranos**, los cuales se han limitado desde la ingeniería de requisitos hasta las primeras líneas de solución de arquitectura, representando vistas lógicas independientes de la plataforma tecnológica. En este contexto, los límites entre aspectos tempranos y aspectos medios tocaron uno de los temas más conflictivos en la ingeniería del software, ¿cuáles son los límites entre el análisis y el diseño o el qué y el cómo o el problema y la solución? En esta tesis doctoral se plantea que posiblemente la solución está en la clásica visión de la arquitectura en la que se describen la vista lógica y la vista física. Sin embargo, en esta propuesta se demuestra cómo de cierta manera las líneas de investigación del área de la arquitectura software siguen arrastrando esta herencia. Por lo tanto, esta propuesta puede llegar satisfacer a un gran porcentaje de la comunidad de la ingeniería del software.

En esta línea, en los casos donde el comportamiento del negocio a automatizar está sujeto a muchos cambios y evoluciones es importante contemplar en sus desarrollos software una arquitectura lógica en las etapas tempranas, dado que en estos casos, disponer de una vista lógica que esté integrada de una forma desacoplada disminuirá el coste total de dichos cambios, y a medio plazo el coste total de la propiedad será mucho

menor, porque los cambios no tendrán tanto impacto. En definitiva, el tener todo el comportamiento lógico que puede estar cambiando encapsulado en áreas identificadas de nuestro software, disminuye drásticamente la cantidad de tiempo que se necesita para realizar un cambio; porque un cambio se realizará en un solo sitio y podrá ser convenientemente verificado de forma aislada, aunque esto por supuesto dependerá de cómo se haya desarrollado. En este sentido en la propuesta de esta tesis doctoral los viejos estilos arquitectónicos tienen un papel importante.

Por otro lado, la visión en general de diferentes grupos de investigación es que los productos software vistos desde la OA se componen de modelos dominantes y aspectos. Así mismo, se declara que los aspectos están compuestos de incumbencias transversales; sin embargo, no queda claro cómo están compuestos los modelos dominantes. Ahora bien, se ha dicho que la implementación en su conjunto de todas las incumbencias proporciona la necesidad deseada por los interesados en el sistema software; por lo tanto, los aspectos y los modelos dominantes se componen de incumbencias.

Ahora bien, las diferentes propuestas OA típicamente se centran en una etapa específica dentro de un proceso de desarrollo software sin tomar en cuenta las características de las diferentes conceptualizaciones de aspectos de las otras etapas. A este respecto, han surgido diferentes propuestas de metamodelos para el modelado de aspectos en cualquier etapa de un desarrollo OA. Sin embargo, en todas se han ignorado las características de la conceptualización de aspectos en las etapas tempranas.

Por lo tanto, la aportación de un entorno de trabajo para especificar los modelos en el primer nivel de conceptualización trae consigo beneficios cruciales en el modelado de un producto software. El **modelo conceptual de aspectos tempranos** plasmado en esta tesis doctoral proporciona una forma sencilla y elegante de identificar los conceptos necesarios en las etapas tempranas de un DSOA.

El desarrollo de la propuesta de un modelo conceptual de aspectos tempranos en este documento tiene el objetivo de proporcionar directrices para estructurar la especificación de las etapas tempranas de un producto software. Este modelo contiene varios bloques de construcción claramente identificados en cualquier propuesta de OA. Entre ellos se encuentra la **incumbencia** que representa la parte atómica de un producto software desarrollado bajo el paradigma OA. En el modelo se define como una agregación de incumbencias que componen un **aspecto**. Así mismo, en este modelo los **aspectos** son especificados en dos tipos principales: **aspectos tempranos trasversales** y **aspectos tempranos dominantes.** Esto quiere decir que los modelos dominantes del metamodelo de la POA en este modelo conceptual son tratados como aspectos pero del tipo dominante. En este contexto, surgen las instancias de incumbencia: **incumbencias dominantes** e **incumbencias transversales**. En el mismo modelo conceptual de aspectos tempranos se encuentra modelado el bloque de construcción **stakeholder** por la necesidad de reflejar los responsables de la **estructura** de incumbencias y el razonamiento que relaciona la estructura de incumbencias o requisitos atómicos con la **arquitectura lógica.** Por otro lado, la extracción de aspectos en algunos productos software ha sido etiquetada de interminable. Sugieren la posibilidad de obtener un producto software demasiado complejo con poca claridad en su modelado para su mantenimiento y evolución. Este problema ha sido solventado en el modelo conceptual de aspectos tempranos con el concepto de **punto de vista** que proporciona una guía para identificar los aspectos tempranos.

Ahora bien, el concepto o bloque de construcción más importante del modelo conceptual de aspectos tempranos es la **arquitectura de puntos de vista**. Esta es una

estructura en niveles anidados en **ámbitos** y **dimensiones**. Esta identificación de puntos de vista en diferentes niveles de abstracción proporciona una guía para identificar los aspectos tempranos más comunes entre la comunidad de la ingeniería del software que resuelvan mejor el desarrollo del producto software con calidad. Sin embargo, las líneas de productos software son varias e incluso dentro de un tipo de producto pueden surgir proyectos con características variables. Por lo tanto, esta tesis doctoral propone **un método para el desarrollo de una arquitectura de puntos de vista** basado en investigaciones de análisis facetado y garantía literaria pertenecientes al área de documentación. Esto no quiere decir que una arquitectura de puntos de vista no pueda ser reutilizada. En esta tesis se presenta el desarrollo de un sencillo caso de estudio en el primer nivel conceptual de aspectos. Este caso de estudio pertenece a la línea de productos software de **sistemas de información**, por lo tanto, la arquitectura de puntos de vista desarrollada en este documento puede ser reutilizada en un gran porcentaje en cualquier producto del mismo tipo.

Posterior a la propuesta de un marco de referencia para la conceptualización de aspectos a todo lo largo de un ciclo de vida de un producto software, se centraron los esfuerzos de investigación en la conceptualización de AT. Después de presentar un modelo conceptual de AT para crear modelados sistematizados se observó la importancia central de la ER.

Ahora bien, con la intención de identificar la calidad de una ER desarrollada con diferentes metodologías se percibió la necesidad de **un marco de referencia para unificar criterios de calidad en el desarrollo de una ER**. En el marco de referencia propuesto para medir la calidad de manera homogénea, cualquier tipo de ER se plasma con el desglose del concepto de atributo, identificando los grandes grupos de atributos extrínsecos y atributos intrínsecos. Los **atributos extrínsecos** son los que disponen de un enlace persistente con el requisito que describen pero con un enlace separado de los requisitos. Los **atributos intrínsecos** son los sensibles al contexto en que se utilizan, sincronizando su semántica en la descripción del requisito.

El otro concepto clave del marco de referencia para la creación de una ER es la **propiedad**. En esta propuesta se realizó un análisis de las diferentes propiedades propuestas en investigaciones reconocidas. Una distinción de las propiedades que influye en la manera de aplicar los criterios de calidad es que pueden desarrollarse de forma **individual y global**. Ahora bien, dado que la mayoría de los requisitos son especificados con lenguaje natural se tomaron en cuenta las líneas de una gramática para definir el grupo de propiedades base que hay que considerar en la creación de una ER. El primer grupo son las propiedades de **completitud explícita**, haciendo una distinción entre la aplicación individual, evitando ideas incompletas y la aplicación global en la que hay que asegurar que todos los requisitos explícitos estén descritos. La aplicación de la propiedad de completitud explícita de forma global es una propiedad que actualmente no tiene un método claro para la medición de su calidad. El segundo grupo son las propiedades de **consistencia gramatical**; estas parten de la idea de proponer directrices gramaticales para la definición de requisitos a la transferencia de conocimiento de calidad. El último grupo son las propiedades de **coherencia semántica**, que es aplicada solamente de forma global. Estas propiedades están estrechamente relacionadas con la organización semántica de los requisitos con el objetivo de desaparecer conflictos, evitar redundancias y gestionar de forma efectiva los acoplamientos entre requisitos.

En esta tesis doctoral existen cinco aportaciones destacables pero la más identificativa es la propuesta de una **organización de conocimiento temprano asistido por aspectos, es decir, la Metodología OCTA**.

Como se ha visto en el estado de la cuestión, en la actualidad existen algunas propuestas para la representación de modelos en etapas tempranas con un enfoque OA. Sin embargo, en estas propuestas en el proceso del modelado no existe relación directa entre la semántica de la especificación de requisitos y el razonamiento que propone las primeras líneas de solución lógica representada en la arquitectura. Este problema ha sido solucionado en esta tesis doctoral con las propuestas relativas a los aspectos tempranos. Pero no es la única carencia encontrada en las propuestas actuales de especificación del conocimiento de los modelos tempranos de un desarrollo software. Se observa lo poco claras que son a la hora de explicar cómo se representa la base de conocimiento, así como tampoco definen cómo se gestionará el conocimiento para realizar ingeniería de conocimiento.

En este contexto, se puede decir que este tipo de investigaciones deberían hablar de la creación de sistemas basados en conocimiento, los cuales están compuestos de una base de conocimiento en un lenguaje formal y un motor de inferencia, que es lo que permite sacar conclusiones y resolver problemas que se planteen. Esta ha sido la base por la que en esta tesis doctoral se proponen mecanismos de la ingeniería ontológica pertenecientes a la ingeniería de conocimiento para solventar el problema de la representación del conocimiento de aspectos con una organización definida.

La propuesta de este sistema basado en conocimiento comienza con la propuesta del lenguaje de marcado OWL. Esta decisión está basada en que dicho lenguaje ha sido influenciado por muchos años de investigación en lógicas descriptivas o DL para el modelado de ontologías. Las DL brindan un conjunto de constructores que permiten formar conceptos y roles complejos para describir bases de conocimiento formales que permiten hacer resolubles sus inferencias.

Ahora bien, todo lo anterior permitió definir los pasos de la metodología OCTA para encontrar y registrar los aspectos tempranos que comuniquen lo que se necesita realmente. A lo largo de los primeros pasos de esta propuesta se observa cómo el modelo de aspectos tempranos es una agregación de varias ontologías básicas que son parte de un todo. Esta propuesta permite que el desarrollo de una especificación de aspectos tempranos permita la separación del trabajo entre expertos en el dominio, permitiendo la reutilización del conocimiento de parte de algunas ontologías.

El primer paso es la creación de la Ontología MMAspect que tiene la intención de ofrecer un alto porcentaje de conocimiento reutilizable para DSOA de la misma línea de producto. Esto se debe a que gran parte de su creación se extrae del modelo de referencia de aspectos tempranos y de los términos de la arquitectura de puntos de vista. En el segundo paso se crea la ontología de ámbito de dominio específico en el cual se definen las entidades de proceso de negocio y dominio de información. En el tercero se crean dos ontologías del ámbito de entorno operacional nombradas como ambiente de aplicación y stakeholder. En este último caso están separadas porque son diferentes expertos los que posiblemente desarrollarían estos modelados. En el cuarto paso se crea la ontología de la arquitectura lógica en la cual se identificarán en una primera interacción entidades que reflejan en dimensiones reutilizables e incumbencias contenedoras que podrían ser reutilizadas en una misma línea de producto software. En este punto se realiza la agregación de las ontologías creadas obteniendo una estructura para la organización semántica de los aspectos tempranos. Por lo tanto, a partir de ese

momento comienza la creación de las incumbencias o requisitos atómicos dominantes y transversales.

La metodología OCTA es interactiva e incremental donde los expertos interaccionan en un mismo sentido para la especificación de un modelo de aspectos tempranos, es decir, la especificación de las etapas tempranas de un producto software.

Una característica resultante en una especificación realizada con esta metodología es la consistencia de requisitos. Es sabido que una especificación de requisitos es realizada con una estructura en árbol donde las sentencias de los requisitos en lenguaje natural tienden a ser redundantes. Sin embargo, con los modelos de aspectos tempranos creados con mecanismos de ingeniería ontológica es posible evitar en un gran porcentaje la inconsistencia.

Otra característica positiva en el modelado de aspectos tempranos es la generación de un glosario único. Usualmente en las especificaciones de requisitos actuales al ser realizadas en lenguaje natural las entidades descritas no tienen posibilidad de ningún control, dado que surgen varias acepciones para una misma entidad provocando confusión en la semántica de los aspectos tempranos.

El control de las características anteriores genera un modelado que facilita la extensión de la especificación. Incluso promueve la posibilidad de reutilización del conocimiento del dominio modelado en otras especificaciones.

Por otro lado, un objetivo principal en el modelado en el nivel conceptual de los AT es seguir la pista de las incumbencias cambiantes del producto software, de manera que siempre tenga un significado claro para los clientes y los miembros del equipo de desarrollo, es decir, en el modelado de aspectos tempranos también se requiere definir cómo se realiza la gestión de su semántica para actualizar los deseos cambiantes de los interesados en el producto software.

Ahora bien, retomando la idea de que el modelado de aspectos tempranos desarrollado con la metodología OCTA es la especificación de un sistema basado en conocimiento, es observable la condición de la generación de una serie de elementos de inferencia de conocimiento. Por lo tanto, surgió la necesidad de la utilización de un razonador, motor de razonamiento o motor de reglas. Este es una pieza software que es capaz de funcionar como un motor de inferencia, deduciendo conclusiones lógicas de un conjunto de axiomas. Así, después de un estudio de los diferentes razonadores fue seleccionado el FaCT++. Este razonador permite verificar la estructura organizativa del modelo de aspectos tempranos.

En este punto se realizó un experimento para la validación de inferencia a través de un banco de cuestiones para consultar la ontología que fue creada para el caso de estudio "Estadio para eventos de atletismo". Es interesante recordar que al desarrollar una ontología las decisiones se toman de acuerdo a los objetivos de la ontología que son extraídos de los expertos, estándares existentes en el dominio y las preguntas relevantes que se le harán al finalizar su creación. En este sentido, el banco de cuestiones del experimento permite fortalecer su organización semántica a través del desarrollo incremental.

En lo relativo al experimento de validación de la ontología de AT, la técnica aplicada fue la del motor orientado a la interrogación de ontologías DL Query que está basado en lógica descriptiva. Dicho motor propone un lenguaje de consulta basado en la sintaxis de OWL Manchester. El objetivo del experimento fue determinar la exactitud de las inferencias al extraer conocimiento de los aspectos tempranos modelados en la ontología, donde se obtuvieron unos resultados excelentes. Sin embargo, los pasos

seguidos al transformar cuestiones de acuerdo con el vocabulario del modelado desarrollado con la metodología OCTA en lenguaje DL Query permite realizar consultas a la ontología que modela aspectos tempranos que permite gestionar la semántica ante el cambio en las etapas tempranas de cualquier DSOA aplicando la metodología OCTA.

En los inicios de la investigación de esta tesis doctoral se plantearon cinco objetivos específicos para cumplir el objetivo general que plantea proporcionar los medios necesarios que permitan el modelado del conocimiento en las etapas tempranas de productos software OO a través de los conceptos del paradigma OA para facilitar una evolución más efectiva, evitar conflictos y mejorar la gestión de la trazabilidad entre incumbencias.

El **objetivo 1** plantea delimitar las diferentes conceptualizaciones de aspectos en las distintas etapas a lo largo de un PDS. El desarrollo del cumplimento de este objetivo se **concreto en una propuesta de un marco de referencia para la conceptualización de aspectos tempranos**. Este proporciona un medio para delimitar las diferentes conceptualizaciones de aspectos que se tienen en las distintas etapas a lo largo del ciclo de vida de un producto software en cualquier PDS.

El **objetivo 2** plantea proporcionar una forma sencilla y elegante de identificar los conceptos OA en las etapas tempranas de un DS. Este objetivo es conseguido con la propuesta en esta tesis de un **modelo conceptual de aspectos tempranos**. Este expone un entorno de trabajo para especificar los modelos en el primer nivel de conceptualización de aspectos.

El **objetivo 3** platea proponer un método para la creación de una APV eficaz que permita estructurar y organizar los aspectos para apoyar la evolución de un producto software. Este objetivo específico se concretó en la propuesta de **un método para el desarrollo de una APV**. En el desarrollo de la metodología se realizó una ejemplificación con un corpus de la línea de trabajo de sistemas de información. En consecuencia en el cumplimiento de este objetivo se proporciona una APV para la creación de productos software de sistemas de información.

El **objetivo 4** platea proponer directrices para verificar la calidad en una Especificación de Requisitos. En el desarrollo para el cumplimiento de este objetivo se buscaba proporcionar los medios para medir la calidad de una ER. El resultado fue **un marco de referencia para unificar criterios de calidad en el desarrollo de una ER**. Este marco también proporciona directrices que permiten la creación de métricas para la medición de localidad de una ER en una factoría de software.

El **objetivo 5** plantea crear una metodología que permita el modelado y organización del conocimiento de las tempranas de un producto software que facilite la gestión de los aspectos tempranos. Hasta ahora, las aportaciones resultantes del desarrollo de los otros objetivos pueden ser utilizadas de forma independiente para otras investigaciones. Sin embargo, el desarrollo del último objetivo específico de esta tesis doctoral requiere el conjunto de aportaciones de esta investigación. El resultado fue la propuesta de una metodología denominada **organización de conocimiento temprano asistido por aspectos, es decir, Metodología OCTA**.

En este contexto, el cumplimiento de estos objetivos específicos da validez a la hipótesis acotada de esta tesis doctoral.

Las conclusiones finales que se obtienen de esta investigación son que la aplicación de las propuestas de los conceptos del paradigma OA y los formalismos de la ingeniería ontológica han permitido realizar un tratamiento sistemático desde la especificación de

requisitos a las primeras líneas de solución de una arquitectura software. Así mismo, estas buenas prácticas de desarrollo promueven una organización y extracción del conocimiento, **mejorando la evolución**, **evitando conflictos** y **mejorando la gestión de la trazabilidad** entre incumbencias de un producto software en el primer nivel conceptual de aspectos.

Otras características observadas que proporcionan una mejor calidad a la especificación de AT fueron las siguientes:

- Una especificación de incumbencias o requisitos atómicos **intuitiva** para cualquiera de los expertos interesados en el sistema.
- Una organización de conocimiento sistemática simple y elegante.
- Una mayor facilidad para razonar sobre las decisiones al obtener las primeras líneas de solución plasmadas en la arquitectura lógica.
- Creación de modelos más simples y fáciles de mantener con un gran potencial para la reutilización.
- La aplicación de la metodología OCTA reduce los recursos para la evolución del producto software en el primer nivel conceptual de aspectos.
- Se demostró que este enfoque puede ser apoyado por herramientas comunes disponibles en el mercado, lo que significa que es factible.

2 Trabajos futuros

En el desarrollo de la propuesta planteada en esta tesis se han dejado pendientes algunos trabajos futuros. Uno de ellos es mejorar el entorno de trabajo de esta investigación para desarrollar con pocas diferencias la descripción de un modelo conceptual de Aspectos Medios (AM). En la propuesta de un **modelo conceptual de AM** es necesario plasmar una correlación de equivalencia con el modelo conceptual de AT. Esto es necesario para asegurar el viaje de ida y vuelta entre AM y AT con una correlación de modelos relativamente simple. En la actualidad diferentes propuestas del modelado en la etapa de Aspectos Finales (AF) tienen un alto nivel de resolución y aceptación. Por lo tanto, desarrollar un producto software OA definiendo los niveles conceptuales de AT y AM proporcionará las bases a la comunidad científica para avanzar en los trabajos futuros con propuestas relacionadas con las trasformaciones verticales entre niveles conceptuales y las reglas de modelado para lograr un DSOA con trazabilidad completa.

En esta línea, entre otros problemas se encuentra el de la gestión gráfica de la arquitectura lógica razonada a través de la organización semántica de requisitos atómicos y la gestión gráfica del nivel del diseño con vistas físicas. En esta línea, hasta el momento se tiene definido cómo se especificarán los AM y la arquitectura lógica de los AT en diagramas aspectuales creados con UML 2.x (Barra et al., 2004). Se proponen un conjunto de diagramas de UML que facilitan la comprensión del desarrollo, por ejemplo los diagramas de actividad aspectuales que representen los diferentes servicios o diagramas de componentes aspectuales que especificarían las capas lógicas. Estos elementos proporcionarían información suficiente para la búsqueda de la mejor organización de requisitos y arquitectura lógica. Sin embargo, a este planteamiento es necesario añadir un **mecanismo de trasformación de incumbencias entre los modelos** de arquitectura lógica de los AT y los modelos de AM quedando como el siguiente trabajo futuro.

Por otro lado, en el caso de estudio ejemplificado en esta tesis doctoral se han utilizado herramientas previamente desarrolladas y verificadas por un alto número de usuarios. Una de las aplicaciones utilizadas fue Protégé para la creación y edición de ontologías. Esta herramienta es de código abierto que crea archivos en código OWL. En la

ejemplificación, la ontología de aspectos tempranos fue respaldada en un archivo de código OWL para ser verificada su calidad de inferencia semántica a través del razonador FaCT++. Esta aplicación también es de código abierto que es capaz de razonar con una ontología codificada en OWL. Otra que se usó fue el motor de consulta DL Query que está como un plugin en Protégé. Estas herramientas fueron usadas para el desarrollo de la especificación de las etapas tempranas bajo la metodología OCTA. Sin embargo, para el análisis de la especificación de etapas tempranas desarrolladas con una metodología clásica se utilizó Visure Requirements como herramienta de gestión de requisitos. En el contexto de estas herramientas, se realizó una aplicación que interpreta el código OWL para transformarlo en una base de datos que puede interpretar Visure Requirements, así como una transformación inversa de la base de datos de Visure Requirements a código OWL que puede interpretar Protégé. La intención era incluirlo como un plugin en alguna herramienta de gestión de requisitos como Visure Requirements o DOORS para que fuera posible la especificación de aspectos tempranos con herramientas utilizadas en la actualidad por factorías software. Sin embargo, ha sido imposible hasta el momento conseguir la información de las interfaces que permitieran acoplar la funcionalidad de una metodología OCTA en una herramienta de gestión de requisitos. Por lo tanto, uno más de los trabajos futuros es conseguir el permiso de alguna empresa responsable de alguna herramienta de gestión de requisitos para lograr el **desarrollo de un complemento para especificar incumbencias o requisitos atómicos** con un desarrollo OA.

Sin embargo, un trabajo futuro que es inevitable plantear es la creación de **una herramienta de gestión de aspectos tempranos**. Dicha herramienta tiene que proporcionar los recursos que en la actualidad facilitan las herramientas típicas de gestión de requisitos pero respetando los conceptos del modelo conceptual de AT. Esta es la fase en la que se encuentra el desarrollo de la herramienta de esta tesis doctoral, que ha sido parado por falta de recursos. Junto a la posibilidad de proporcionar los recursos para una gestión de requisitos típica tendrá la posibilidad de aplicar la metodología OCTA pero con transparencia en lo relacionado con la creación de ontologías. Es decir, a través de una guía virtual se proporcionará apoyo para describir los requisitos pero seleccionando los atributos intrínsecos con los elementos correspondientes que modelen su conocimiento, añadiendo la posibilidad de modelar la semántica de las capas que compondrán la arquitectura lógicas y guardando su razonamiento.

Ahora bien, la metodología OCTA aporta los pasos para la creación de una especificación de AT. Sin embargo, uno de los deseos en el contexto de esta propuesta es llegar a **transformar especificaciones existentes que hayan sido desarrolladas con una metodología clásica en especificaciones de AT**. Es decir, transformar requisitos descritos con lenguaje natural a una ontología de aspectos tempranos especificada en incumbencias dominantes y transversales. En esta investigación se realizaron trabajos en corpus de requisitos existentes para la identificación de patrones en un formato relacionado con sus atributos intrínsecos. Sin embargo, ha sido insuficiente para obtener un proceso para la trasformación automática. El resultado actual del proceso es una trasformación con muchos errores con una inversión de tiempo muy alta sin resultados significativos. Por lo tanto, esta línea de investigación pasó a ser un trabajo futuro por la carga de investigación que conllevaba en torno al procesamiento de lenguaje natural.

Por otro lado, en esta tesis doctoral se ha propuesto una metodología que permite la creación de una APV con un esquema de clasificación capaz de expresar relaciones jerárquicas y relaciones entre conceptos entre diferentes jerarquías. Esto se logra con la clasificación de contenido semántico de análisis facetado y los principios de la garantía

literaria con una selección de muestras aleatorias de términos para agruparlos en temas comunes. Entre estas propuestas de clasificación u organización del conocimiento se pueden observar dos vertientes básicas de desarrollo. Una se apoya en el método deductivo denominado aproximación top-down y otra que apela al método inductivo llamado también aproximación bottom-up. A este respecto, se ha estado trabajando en el método top-down tratando de evitar los procesos informales con un carácter manual (Barra et al., 2011). Así mismo, ha quedado como trabajo futuro la automatización del método bottom-up, el cual tiene mayores posibilidades de éxito. La automatización conjunta de los dos métodos aportará el **apoyo tecnológico** para la **creación de una APV** que requieren las factorías de software para la aceptación de la metodología OCTA para el desarrollo de productos software.

Ahora bien, para el desarrollo de software en las factorías de software son aplicadas diferentes metodologías. Sin embargo, la mayoría son las clasificadas entre las metodologías ágiles. Entre las más populares en la actualidad se encuentra Scrum (Schwaber & Beedle, 2002) que obedece a las necesidades de proporcionar rapidez, calidad y reducción de costes. En Scrum es necesario indicar claramente las acciones a acometer en un ciclo iterativo con el objetivo de minimizar el esfuerzo y maximizar el rendimiento del desarrollo. En este escenario, se puede percibir cómo es posible acoplar la metodología OCTA para especificar las acciones a realizar con la semántica de AT, dejando claro que es un trabajo futuro importante en esta investigación.

Estos son los trabajos futuros de investigación más próximos pero al ser esta investigación una motivación general en la creación de software de calidad son posibles el lanzamiento de muchos más, derivados de la investigación realizada en esta tesis doctoral denominada "Metodología para la Organización del Conocimiento Temprano asistido por Aspectos - OCTA".

Glosario

1 Lista de términos

Incumbencia *(concern)*

Es una consideración específica o requisito que debe ser tomado en cuenta para satisfacer los propósitos de un producto software.

Incumbencia transversal *(crosscutting concern)*

Las incumbencias dispersas y enredadas en otros elementos del resto de la aplicación.

Aspecto

Es una unidad modular que agrupa incumbencias, examinadas y seleccionadas bajo puntos de vista específicos, diseminadas a través de los diferentes modelos creados en las diferentes etapas del ciclo de vida de un desarrollo software.

Componente

Término general para una parte de un producto software.

Capa

Un nivel jerárquico en la descomposición de un producto software.

Ontología

Es una especificación explícita formal de una conceptualización compartida. Así mismo, es entendida como un cuerpo estructurado de conocimiento, que no ha de ser considerada como una entidad natural que se descubre, sino como un recurso artificial que se crea.

Razonador

Es una pieza de software capaz de funcionar como un motor de inferencia, deduciendo conclusiones lógicas de un conjunto de axiomas

Punto de Vista

Define las reglas para extraer una o más incumbencias con una misma importancia para un conjunto de interesados en el sistema.

Arquitectura de Puntos de Vista

Son Puntos de Vista organizados que establecen los convenios específicos para la creación, organización y análisis de la estructura de aspectos tempranos.

Atributo de requisito

Propiedad de un requisito que identifica o cualifica su semántica.

Requisito atómico

Requisito indivisible semánticamente.

Requisito apócrifo

Requisitos que dependen de la experiencia del analista para considerarlos explícitos pero que podrían ser implícitos. Las definiciones de términos del dominio o del negocio son considerados dentro este tipo de requisitos.

2 Lista de acrónimos

OO	Orientado a Objetos
OA	Orientado a Aspectos
DS	Desarrollo Software
DSOA	Desarrollo Software Orientado a Aspectos
PDS	Proceso de Desarrollo Software
POA	Programación Orientada a Aspectos
LPG	Lenguajes de Procedimiento Generalizado
LOA	Lenguajes Orientados a Aspectos
CMMI	Capability Maturity Model Integration
IR	Ingeniería de Requisitos
AS	Arquitectura de Software
IROA	Ingeniería de Requisitos Orientada a Aspectos
ASOA	Arquitectura Software Orientada a Aspectos
OMG	Object Management Group
MOF	Metamodel Object Facility
UML	Unified Modeling Language
MDA	Model Driven Architecture
MDD	Model-Driven Development
PIM	Platform Independent Model
PSM	Platform Specific Model
IC	Ingeniería del Conocimiento
DL	Description Logic
RDF	Resource Description Framework
OWL	Web Ontology Language
XML	Extensible Markup Language
URI	Uniform Resource Identifier
ODM	Ontology Definition Metamodel
CASE	Computer Aided Software Engineering
AFD	Actividad Fundamental Disciplinar
PV	Punto de Vista
AT	Aspecto Temprano
APV	Arquitectura de Puntos de Vista
MVC	Modelo-Vista-Controlador
GUI	Graphical User Interface
ER	Especificación de Requisitos

DET	Documento de Especificación Técnica
OCTA	Organización de Conocimiento Temprano asistido por Aspectos

Referencias

AkŞit, M., Bergmans, L., & Vural, S. (1992). An object-oriented language-database integration model: The composition-filters approach. In *Proceedings of the ECOOP '92 European Conference on Object-Oriented Programming* (Vol. 615/1992, pp. 372–395). Utrecht, The Netherlands: Springer-Verlag. doi:10.1007/BFb0053026

Alexander, I., & Stevens, R. (2002). *Writing Better Requirements* (1st ed.). Addison-Wesley Professional.

Antoniou, G., & Harmelen, F. Van. (2004). *A Semantic Web Primer*. Cambridge, MA, USA: MIT Press. Retrieved from http://portal.acm.org/citation.cfm?id=975284&coll=GUIDE&dl=GUIDE&CFID=55393262&CFTOKEN=22443696

Baniassad, E., & Clarke, S. (2004). Finding aspects in requirements with Theme/Doc. In *Workshop on Early Aspects 2004 In conjunction with 3rd AOSD*. Lancaster, UK.

Baniassad, E., & Clarke, S. (2004). Theme: an approach for aspect-oriented analysis and design (pp. 158–167). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1317438

Baniassad, E., & Clarke, S. (2005). *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley Professional.

Baniassad, E., Clements, P. C., Araujo, J., Moreira, A., Rashid, A., & Tekinerdogan, B. (2006). Discovering early aspects. *Software, IEEE*, *23*(1), 61–70. doi:10.1109/MS.2006.8

Barbiero, D. (2004). Dictionary of Philosophy of Mind. Retrieved September 30, 2011, from http://philosophy.uwaterloo.ca/MindDict/tacitknowledge.html

Barker, R. (1994). *El modelo entidad-relación CASE*METHOD* (p. 256). Ediciones Díaz de Santos.

Barra, E., Fraga, A., & Llorens, J. (2007). The Symbiosis between View and Aspect. In *First Workshop on Aspects in Architectural Description In Conjunction with Sixth International Conference on Aspect-Oriented Software Development*. Vancouver, Canada.

Barra, E., Génova, G., & Llorens, J. (2004). An approach to Aspect Modelling with UML 2.0. In *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004*. Lisboa, Portugal.

Barra, E., Palacios, V., Sánchez-Cuadrado, S., & Moreiro, J.-A. (2011). Evaluación de Software Libre Para la Gestión de Archivos Administrativos. *El Profesional de La Informacion*, *20*(2), 206–213. doi:10.3145/epi.2011.mar.12

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice* (2nd ed.). Addison-Wesley Professional.

Bechhofer, S., Möller, R., & Crowther, P. (2003). The DIG Description Logic Interface. In *Proceedings of the International Workshop on Description Logics*. Rome, Italy.

Berners-Lee, T. (2000). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web* (1st ed.). HarperBusiness.

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, *21*(5), 61–72. doi:10.1109/2.59

Borst, W. N. (1997). *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. Universiteit Twente, Enschede. Retrieved from http://doc.utwente.nl/17864/

Brank, J., Grobelnik, M., & Mladenić, D. (2005). A survey of ontology evaluation techniques. In *In In Proceedings of the Conference on Data Mining and Data Warehouses ({SiKDD} 2005*.

Braude, E. J. (2000). *Software Engineering: An Object-Oriented Perspective* (1st ed.). John Wiley & Sons.

Brichau, J., & D'Hondt, T. (2005). Introduction to Aspect-Oriented Software Development. In *European Network of Excellence on Aspect-Oriented Software Development, August 2005*.

Brichau, J., Glandrup, M., Clarke, S., & Bergmans, L. (2002). Advanced Separation of Concerns. In Á. Frohner (Ed.), *Object-Oriented Technology* (pp. 107–130). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/3-540-47853-1_9

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture Volume 1: A System of Patterns* (1st ed.). Wiley.

Christel, M. G., & Kang, K. C. (1992). *Issues in Requirements Elicitation, Technical Report CMU/SEI-92-TR-012 ESC-TR-92-012* (p. 80). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania. Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/92tr012.cfm

Chung, L., Nixon, B. A., & Yu, E. (2000). *Non-functional requirements in software engineering*. Springer.

Clark, T., Sammut, P., & Willans, J. (2008). *Applied Metamodelling: A Foundation for Language Driven Development* (p. 241). London, UK: CETEVA. Retrieved from http://www.ceteva.com/home/Papers/Applied Metamodelling (Second Edition).pdf

Clemente, P. J., Hernández, J., & Sánchez, F. (2007). An MDA approach to develop systems based on components and aspects. In *Proceedings of the 2007 ACM symposium on Applied computing* (pp. 1033–1034). Seoul, Korea: ACM. doi:10.1145/1244002.1244226

CMMI Product Team. (2010). CMMI for Development, Version 1.3. Software Engineering Institute. Retrieved from http://www.sei.cmu.edu/reports/10tr033.pdf

CMMI_Product_Team. (2002). CMMI for Software Engineering, Version 1.1, Staged Representation. Pittsburgh,PA: Software Engineering Institute. Retrieved from http://www.sei.cmu.edu/reports/02tr029.pdf

Daconta, M. C., Obrst, L. J., & Smith, K. T. (2003). *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. Indianapolis, Indiana, USA: Wiley.

Dijkstra, E. W. (1976). *A discipline of programming* (p. 217). Prentice-Hall.

Doerr, J., Kerkow, D., Koenig, T., Olsson, T., & Suzuki, T. (2005). Non-functional requirements in industry - three case studies adopting an experience-based NFR method. In *13th IEEE International Conference on Requirements Engineering. Proceedings* (pp. 373–382). doi:10.1109/RE.2005.47

Elrad, T., Filman, R. E., & Bader, A. (2001). Aspect-oriented Programming: Introduction. *Commun. ACM*, *44*(10), 29–32. doi:10.1145/383845.383853

ESA-Board-for-Software-Standardisation-and-Control. (1994). ESA software engineering standards Issue 2. Noordwijk, The Netherlands: ESA Publications Division. Retrieved from ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS050.pdf

ESA-Board-for-Software-Standardisation-and-Control. (1998). Guide to applying the ESA software engineering standards to projects using Object-Oriented Methods. Darmstadt. Germany: ESA Publications Division.

Evermann, J., & Wand, Y. (2005). Ontology based object-oriented domain modelling: fundamental concepts. *Requirements Engineering*, *10*(2), 146–160. doi:10.1007/s00766-004-0208-2

Fernández-López, M., Gómez-Pérez, A., & Corcho, O. (2004). *Ontological Engineering*. London, UK: Springer.

Filman, R. E., Elrad, T., Clarke, S., & Aksit, M. (2004). Aspect-Oriented Software Development. Addison-Wesley Professional.

Fuentes, L., & Sánchez, P. (2006). A generic MOF metamodel for aspect-oriented modelling. In *Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD/MOMPES'06)* (p. 10 pp.–124). Potsdam, Germany: IEEE Computer Society. doi:10.1109/MBD-MOMPES.2006.1

Galofre, A. F. (2013). *El cambiante mundo de las organizaciones: Teoría, metodología e investigación* (1º ed; 1º .). Universitat Jaume I. Servei de Comunicació i Publicacions.

Gangemi, A. (2005). Ontology design patterns for semantic web content. In *The Semantic Web--ISWC 2005* (pp. 262–276). Springer.

GAO. (1979). *Contracting for Computer Software Development-Serious Problems Require Management Attention To Avoid Wasting Additional Millions* (p. 95). Washington, D.C., United States. Retrieved from http://www.gao.gov/products/FGMSD-80-4

Garcia, A., Batista, T., Rashid, A., & Sant'Anna, C. (2006). Driving and managing architectural decisions with aspects. *SIGSOFT Softw. Eng. Notes*, *31*, 6. doi:10.1145/1163514.1178646

Garlan, D., Allen, R., & Ockerbloom, J. (1995). Architectural mismatch or why it's hard to build systems out of existing parts. In *Proceedings of the 17th international conference on Software engineering* (pp. 179–185). Seattle, Washington, United States: ACM. doi:10.1145/225014.225031

Garlan, D., & Shaw, M. (1993). An introduction to software architecture. *Advances in Software Engineering and Knowledge Engineering*, *1*, 1–39. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.116.8569

Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., & Moreno, V. (2011). A framework to measure and improve the quality of textual requirements. *Requirements Engineering*. doi:10.1007/s00766-011-0134-z

Glass, R. L. (2003). *Facts and fallacies of software engineering*. Boston, MA: Addison-Wesley.

Graham, I., Henderson-Sellers, B., & Younessi, H. (1999). *The OPEN Process Specification*. Addison-Wesley Professional.

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina-Videira Lopes, Jean-Marc Loingtier, & John Irwin. (1997). Aspect-Oriented Programming (Vol. 1241/1997, pp. 220–242). Yväskylä, Finland: Springer-Verlag, Berlin Heidelberg. doi:10.1007/BFb0053371

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, *43*(5-6), 907–928. Retrieved from http://portal.acm.org/citation.cfm?id=219701

Grundy, J. (1999). Aspect-Oriented Requirements Engineering for Component-Based Software Systems. In *Requirements Engineering, IEEE International Conference on* (Vol. 0, p. 84). Los Alamitos, CA, USA: IEEE Computer Society. doi:http://doi.ieeecomputersociety.org/10.1109/ISRE.1999.777988

Grüninger, M., & Fox, M. S. (1995). Methodology for the Design and Evaluation of Ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95*. Montreal, Canada.

Guarino, N. (1998). Formal ontology in information systems. In G. N. (Ed.), *international conference on Formal ontology in information systems* (pp. 3–15). Trento, Italy: IOS Press, Amsterdam, The Netherlands.

Haarslev, V., & Möller, R. (2001). RACER System Description. In *Proceedings of the First International Joint Conference on Automated Reasoning* (pp. 701–706). London, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=648237.753964

Han, Y., Günter, K., & Cremers, A. B. (2005). Towards Visual AspectJ by a Meta Model and Modeling Notation. In *Proceedings of 6th International Workshop on Aspect-Oriented Modeling in conjunction with AOSD'05*. Chicago, Illinois, USA. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.3666

Hanson, S. J., & Bauer, M. (1989). Conceptual Clustering, Categorization, and Polymorphy. *Machine Learning*, *3*, 343–372. doi:10.1023/A:1022697818275

Harrison, W., & Ossher, H. (1993). Subject-oriented programming: a critique of pure objects. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications* (pp. 411–428). Washington, D.C., United States: ACM Press. doi:10.1145/165854.165932

Hayes, P. J., & Menzel, C. (2005). Simple Common Logic. In *Rule Languages for Interoperability*. Retrieved from http://www.w3.org/2004/12/rules-ws/paper/103/

Hayes, P., & Menzel, C. (2001). A Semantics for the Knowledge Interchange Format. In *In IJCAI 2001 Workshop on the IEEE Standard Upper Ontology*.

Horridge, M., & Bechhofer, S. (2009). The OWL API: A Java API for Working with OWL 2 Ontologies. In F. Hoekstra & P. Patel-Schneider (Eds.), *CEUR Workshop Proceedings* (Vol. 529).

Howe, D. (1994). Free On-Line Dictionary of Computing. Retrieved September 30, 2011, from http://foldoc.org/knowledge

Hulme, E. W. (1950). *Principles of Book Classification: Reprinted from the Library Association Record ...* Association of Assistant Librarians.

Hustadt, U. (2004). Reducing SHIQ− Description Logic to Disjunctive Datalog Programs (pp. 152–162). {AAAI} Press.

IEEE-Computer-Society. (1998a). IEEE Std 1233-1998 Guide for Developing System Requirements Specifications. New York, USA. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=741940

IEEE-Computer-Society. (1998b). IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements SpeciÞcations. New York, USA. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574&userType=inst

IEEE-Computer-Society. (2000). IEEE-1471 Standard. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.

ISO/IEC-IEEE. (2011). ISO/IEC FDIS 42010 IEEE P42010/D9 Systems and software engineering — Architecture description.

ITU-T/ISO/IEC. (1994). RM-ODP, ITU-T Rec. X.901-X.904, ISO/IEC 10746.

Jackson, M. (1990). Some complexities in computerbased systems and their implications for system development. In *CompEuro '90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering* (pp. 344–351). Tel-Aviv, Israel: IEEE Comput. Soc. Press. doi:10.1109/CMPEUR.1990.113645

Jackson, M. (2000). *Problem Frames: Analysing & Structuring Software Development Problems*. {Addison-Wesley} Professional.

Jacobson, I. (1992). *Object Oriented Software Engineering: A Use Case Driven Approach* (Revised.). Addison-Wesley Professional.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley Professional.

Jacobson, I., & Ng, P.-W. (2005). *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley Professional.

Jureta, I. J., Mylopoulos, J., & Faulkner, S. (2008). Revisiting the Core Ontology and Problem in Requirements Engineering. In *16th IEEE International Requirements Engineering, 2008. RE '08* (pp. 71–80). Catalunya, España. doi:10.1109/RE.2008.13

Kaindl, H. (1999). Difficulties in the Transition from OO Analysis to Design. *IEEE Software*, *16*(5), 94–102.

Kaiya, H., & Saeki, M. (2006). Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In *Requirements Engineering, 14th IEEE International Conference* (pp. 189–198). Minneapolis/St. Paul, MN. doi:10.1109/RE.2006.72

Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). Scenario-based analysis of software architecture. *Software, IEEE*, *13*(6), 47–55. doi:10.1109/52.542294

Kent, S. (2002). Model Driven Engineering. In Mi. Butler, L. Petre, & K. Sere (Eds.), *Proceedings of the Third International Conference on Integrated Formal Methods* (Vol. 2335, pp. 286–298). Turku, Finland: Springer-Verlag London,UK. Retrieved from http://portal.acm.org/citation.cfm?id=743552

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.-V., Loingtier, J.-M., & Irwin, J. (1997). Aspect-Oriented Programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97)* (Vol. 1241, pp. 220–242). Jyväskylä, Finland: Springer-Verlag, Berlin Heidelberg. doi:10.1007/BFb0053371

Kotonya, G., & Sommerville, I. (1992). Viewpoints for requirements definition. *Software Engineering Journal*, *7*(6), 375–387.

Krasner, G. E., & Pope, S. T. (1988). A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, *1*, 26–49. Retrieved from http://www.ics.uci.edu/~redmiles/ics227-SQ04/papers/KrasnerPope88.pdf

Kruchten, P. (2003). *The Rational Unified Process: An Introduction*. Addison-Wesley Professional.

Kwaśnik, B. (1992). The legacy of facet analysis. In R. N. Sharma (Ed.), *Ranganathan and the West* (pp. 98–111). Sterling.

Laddad, R. (2003). *AspectJ in action: practical aspect-oriented programming* (p. 516). Manning.

Lassila, O., & Mcguinness, D. (2001). The Role of Frame-Based Representation on the Semantic Web. *Linköping Electronic Articles in Computer and Information Science*, *6*(5).

Lieberherr, K. J. (1996). *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns* (p. 617). Boston, Massachusetts. Retrieved from http://www.ccs.neu.edu/research/demeter/book/aoos.PDF

Llorens, J., Morato, J., & Génova, G. (2004). RSHP: An información representation model based on relationships. In E. Damiani, J. Lakhmi C., & M. Madrvio (Eds.), *Soft computing in software engineering* (pp. 221–253). Berlin Heidelberg Germany: Springer.

Llorens, J., Velasco, M., Moreiro, J. A., & Morato, J. (1998). Características textuales como medida cualitativa de la información en la generación semiautomática de tesauros. *Procesamiento Del Lenguaje Natural*, *23*. Retrieved from http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/viewFile/3630/2103

Lopes, C. V., & Kiczales, G. (1997). *D: A Language Framework for Distributed Programming* (No. 97-010). Xerox Palo Alto Research Center.

Lopes, C. V., & Kiczales, G. (1998). Recent Developments in AspectJ, 398–401. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.6770

López Crespo, F., Amutio Gómez, M. A., Candau, J., & Mañas, J. A. (2006). *Magerit V.2: methodology for information systems risk analysis and management* (Ministry o.). Retrieved from http://administracionelectronica.gob.es/

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Introduction to WordNet: An On-line Lexical Database. *International Journal Lexicography*, *3*, 235–244. doi:10.1093/ijl/3.4.235

Minsky, M. L. (1969). *Semantic Information Processing*. The MIT Press. Retrieved from http://portal.acm.org/citation.cfm?id=1096479

Mizoguchi, R., Vanwelkenhuysen, & Ikeda, M. (1995). Task ontology for reuse of problem solving knowledge. In N. J. I. Mars (Ed.), *Towards very large knowledge bases: Knowledge Building and Knowledge Sharing* (pp. 46–57). Twente, Enschede, The Netherlands: IOS Press, Amsterdam, The Netherlands.

Moreira, A., Araújo, J., & Rashid, A. (2005). A Concern-Oriented Requirements Engineering Model. In O. Pastor & J. e Cunha (Eds.), *Advanced Information Systems Engineering* (Vol. 3520, pp. 55–100). Springer Berlin / Heidelberg. Retrieved from http://dx.doi.org/10.1007/11431855_21

Moreira, A., Rashid, A., & Araujo, J. (2005). Multi-Dimensional Separation of Concerns in Requirements Engineering. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering* (pp. 285–296). IEEE Computer Society. Retrieved from http://portal.acm.org/citation.cfm?id=1100638

Motik, B., Shearer, R., & Horrocks, I. (2007). Optimized Reasoning in Description Logics Using Hypertableaux. In *Proceedings of the 21st international conference on Automated Deduction: Automated Deduction* (pp. 67–83). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/978-3-540-73595-3_6

Nuseibeh, B., Kramer, J., & Finkelstein, A. (1994). A framework for expressing the relationships between multiple views in requirements specification. *Software Engineering, {IEEE} Transactions on*, *20*(10), 760–773. doi:10.1109/32.328995

Object_Management_Group-(OMG). (2006). Meta Object Facility (MOF) Core Specification, Version 2.0. Retrieved from http://www.omg.org/spec/MOF/2.0/

Object_Management_Group-(OMG). (2009). Ontology Definition Metamodel (ODM) Version 1.0. Retrieved from http://www.omg.org/spec/ODM/1.0/

Object_Management_Group-OMG. (2003). MDA Guide, Version 1.0.1. Retrieved from http://www.omg.org/cgi-bin/doc?omg/03-06-01

Object_Management_Group-OMG. (2010a). Unified Modeling Language, Infrastructure Specification, Version 2.3. Retrieved from http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/

Object_Management_Group-OMG. (2010b). Unified Modeling Language, Superstructure Specification, Version 2.3. Retrieved from http://www.omg.org/spec/UML/2.3/Superstructure/PDF/

OMG. (1989). Object Management Group. Retrieved September 30, 2011, from http://www.omg.org/

Ossher, H., & Tarr, P. (2000). Hyper/J: Multi-Dimensional Separation of Concerns for Java. In *Software Engineering, International Conference on* (Vol. 0, p. 734). Los Alamitos, CA, USA: IEEE Computer Society. doi:http://doi.ieeecomputersociety.org/10.1109/ICSE.2000.10154

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053–1058. doi:10.1145/361598.361623

Parnas, D. L. (1976). On the Design and Development of Program Families. *Software Engineering, IEEE Transactions on*, *SE-2*(1), 1–9. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1702332&tag=1

Parsia, B., & Sirin, E. (2004). Pellet: An owl dl reasoner. In *In Proceedings of the International Workshop on Description Logics* (p. 2003).

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *SIGSOFT Software Engineering Notes*, *17*(4), 40–52. doi:10.1145/141874.141884

Polanyi, M. (1958). *Personal Knowledge: Towards a Post-Critical Philosophy*. University Of Chicago Press.

Pressman, R. S. (2010). *Software engineering: a practitioner's approach*. New York: McGraw-Hill Higher Education.

R.A.E. (2010). *Nueva gramatica de la Lengua Espanola,* (p. 1049). Real Academia de la Lengua Espanola.

Ranganathan, S. R., & Gopinath, M. A. (2006). *Prolegomena to Library Classification: (Edition III)*. Ess Ess Publication.

Rashid, A. (2005). Early Aspects. Retrieved September 30, 2011, from www.early-aspects.net

Rashid, A., Moreira, A., & Araújo, J. (2003). Modularisation and composition of aspectual requirements. In *Proceedings of the 2nd international conference on Aspect-oriented software development* (pp. 11–20). Boston, Massachusetts. Retrieved from http://portal.acm.org/citation.cfm?id=643605

Rashid, A., Sawyer, P., Moreira, A. M. D., & Araújo, J. (2002). Early Aspects: A Model for Aspect-Oriented Requirements Engineerin. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering* (pp. 199–202). IEEE Computer Society. Retrieved from http://portal.acm.org/citation.cfm?id=731623#

Rising, L. (2000). *The Pattern Almanac 2000* (1st ed.). Addison Wesley Publishing Company.

Robertson, S., & Robertson, J. C. (2006). *Mastering the Requirements Process* (2nd ed.). Addison-Wesley Professional.

Ross, D. T., & Schoman, K. E. (1977). Structured Analysis for Requirements Definition. *Software Engineering, IEEE Transactions on*, *SE-3*(1), 6–15. doi:10.1109/TSE.1977.229899

Royce, W. W. (1987). Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings of the 9th International Conference on Software Engineering* (pp. 328–338). Los Alamitos, CA, USA: IEEE Computer Society Press. Retrieved from http://dl.acm.org/citation.cfm?id=41765.41801

Saeki, M., Horai, H., & Enomoto, H. (1989). Software Development Process From Natural Language Specification. In *11th International Conference on Software Engineering, 1989* (pp. 64–73). New York, NY, USA. doi:10.1109/ICSE.1989.714394

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, *24*(5), 513–523.

Schauerhuber, A., Schwinger, Wieland Kapsammer, Elisabeth Retschitzegger, W., & Wimmer, M. (2006). Towards a Common Reference Architecture for Aspect-Oriented Modeling. In *8th International Workshop on Aspect-Oriented Modeling, in conjunction with AOSD'06*. Bonn, Germany. Retrieved from http://www.wit.at/people/schauerhuber/publications/Schauerhuber_AOSD06-AOM-FINAL.pdf

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice Hall.

Sendall, S., & Kozaczynski, W. (2003). Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, *20*(5), 42–45. doi:http://doi.ieeecomputersociety.org/10.1109/MS.2003.1231150

Senlle, A., Ayuso, E. M., & Manchado, N. M. (2001). *ISO 9000: 2000 calidad en los servicios*. Gestión 2000.

Shapiro, S. C. (1969). A Net Structure for Semantic Information Storage, Deduction and Retrieval. In *Proceedings Second International Joint Conference on AI* (pp. 512–523). London, UK: The British Computer Society.

Shaw, M., & Clements, P. (1997). A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. In *COMPSAC '97 - 21st International Computer Software and Applications Conference* (pp. 6–13). Washington, DC, USA: IEEE Computer Society. doi:http://doi.ieeecomputersociety.org/10.1109/CMPSAC.1997.624691

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

Sommerville, I. (2005). *Ingenieria del Software*. Pearson Educacion.

Sommerville, I., & Sawyer, P. (1997). Viewpoints: principles, problems and a practical approach to requirements engineering. *Ann. Softw. Eng.*, *3*, 101–130. Retrieved from http://portal.acm.org/citation.cfm?id=590580

Sommerville, I., & Sawyer, P. (1999). *Requirements Engineering: A Good Practice Guide* (p. 391). John Wiley & Sons.

Souag, A., Salinesi, C., & Comyn-Wattiau, I. (2012). Ontologies for Security Requirements: A Literature Survey and Classification. In M. Bajec & J. Eder (Eds.), *Advanced Information Systems Engineering Workshops* (pp. 61–69). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-31069-0_5

Souag, A., Salinesi, C., Wattiau, I., & Mouratidis, H. (2013). Using Security and Domain Ontologies for Security Requirements Analysis. In *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual* (pp. 101–107). Japan. doi:10.1109/COMPSACW.2013.124

Stahl, T., & Voelter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. Wiley.

Standish Group. (2003). *CHAOS Chronicles v3.0*. West Yarmouth, Massachusetts, U.S.A. Retrieved from http://www.standishgroup.com/chaos/toc.php

Standish Group. (2010). *Chaos summary 2009*.

Stapleton, J., & Constable, P. (1997). *DSDM: Dynamic Systems Development Method: The Method in Practice*. Addison-Wesley Professional.

Stein, D., Hanenberg, S., & Unl, R. (2002). On representing join points in the UML. In *Aspect Modeling with UML workshop at the Fifth International Conference*

on the Unified Modeling Language and its Applications. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.154.8785

Tarr, P., Ossher, H., Harrison, W., & Sutton, J. S. M. (1999). N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 21st International Conference on Software Engineering* (pp. 107–119). Los Angeles, California, United States. doi:10.1109/ICSE.1999.841000

Taylor, R. N., Medvidovic, N., Anderson, K. M., Whitehead, E. J., & Robbins, J. E. (1995). A component and message based architectural style for GUI software. In *Proceedings of the 17th international conference on Software engineering - ICSE '95* (pp. 295–304). Seattle, Washington, United States. doi:10.1145/225014.225042

Tekinerdogan, B. (2004). ASAAM: Aspectual Software Architecture Analysis Method. In *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture* (pp. 5–14). Norway: IEEE Computer Society. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1310685&isnumber=29100

The-European-Cooperation-for-Space-Standardization. (2009). ECSS-E-ST-40C - Space engineering - Software. Noordwijk, The Netherlands: ESA Requirements And Standards Division.

Toval, A., Olmos, A., & Piattini, M. (2002). Legal requirements reuse: a critical success factor for requirements quality and personal data protection. In *IEEE Joint International Conference on Requirements Engineering, 2002. Proceedings* (pp. 95–103). doi:10.1109/ICRE.2002.1048511

Tsarkov, D., & Horrocks, I. (2006). FaCT++ description logic reasoner: system description. In *Proceedings of the Third international joint conference on Automated Reasoning* (pp. 292–297). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/11814771_26

Van den Berg, K., Conejero, J. M., & Chitchyan, R. (2005). *AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation*. Retrieved from http://purl.org/utwente/64104

Van Heijst, G., Schreiber, A. T., & Wielinga, B. J. (1997). Using explicit ontologies in KBS development. *Int. J. Hum.-Comput. Stud.*, *46*, 183–292. Retrieved from http://portal.acm.org/citation.cfm?id=250542#

Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: a guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on* (pp. 249–262). doi:10.1109/ISRE.2001.948567

Velasco, J. L., Valencia-Garcia, R., Fernandez-Breis, J. T., & Toval, A. (2009). Modelling Reusable Security Requirements Based on an Ontology Framework. *Journal of Research and Practice in Information Technology*, *41*(2), 119–133.

Retrieved from http://search.informit.com.au/documentSummary;dn=836483699251742;res=IELHSS

W3C Recommendation. (2004a). OWL Web Ontology Language Overview. (D. L. McGuinness & F. van Harmelen, Eds.). Retrieved from http://www.w3.org/TR/2004/REC-owl-features-20040210/

W3C Recommendation. (2004b). OWL Web Ontology Language Use Cases and Requirements. (J. Heflin, Ed.). Retrieved from http://www.w3.org/TR/webont-req/

W3C Recommendation. (2004c). Resource Description Framework (RDF): Concepts and Abstract Syntax. (G. Klyne & J. j. Carroll, Eds.). Retrieved from http://www.w3.org/TR/rdf-concepts/

W3C Recommendation. (2012). OWL 2 Web Ontology Language Document Overview (Second Edition). (W3C_OWL_Working_Grup, Ed.). Retrieved from http://www.w3.org/2012/pdf/REC-owl2-overview-20121211.pdf

W3C_Working_Group_Note. (2009). SKOS Simple Knowledge Organization System Primer. (A. Isaac & E. Summers, Eds.). Retrieved from http://www.w3.org/TR/skos-primer/

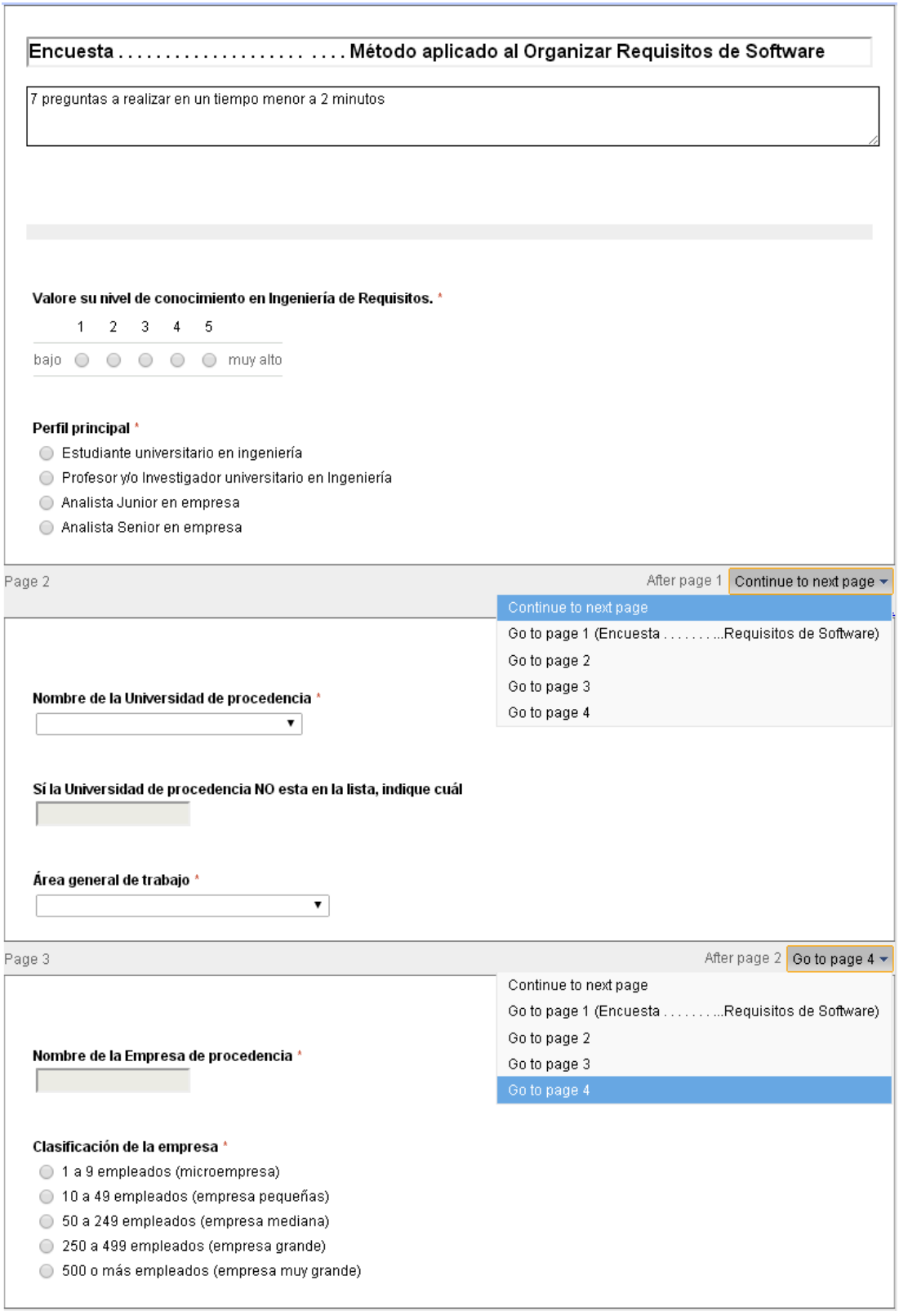
Weston, N., Chitchyan, R., & Rashid, A. (2009). Formal semantic conflict detection in aspect-oriented requirements. *Requirements Engineering*, *14*(4), 247–268. doi:10.1007/s00766-009-0083-y

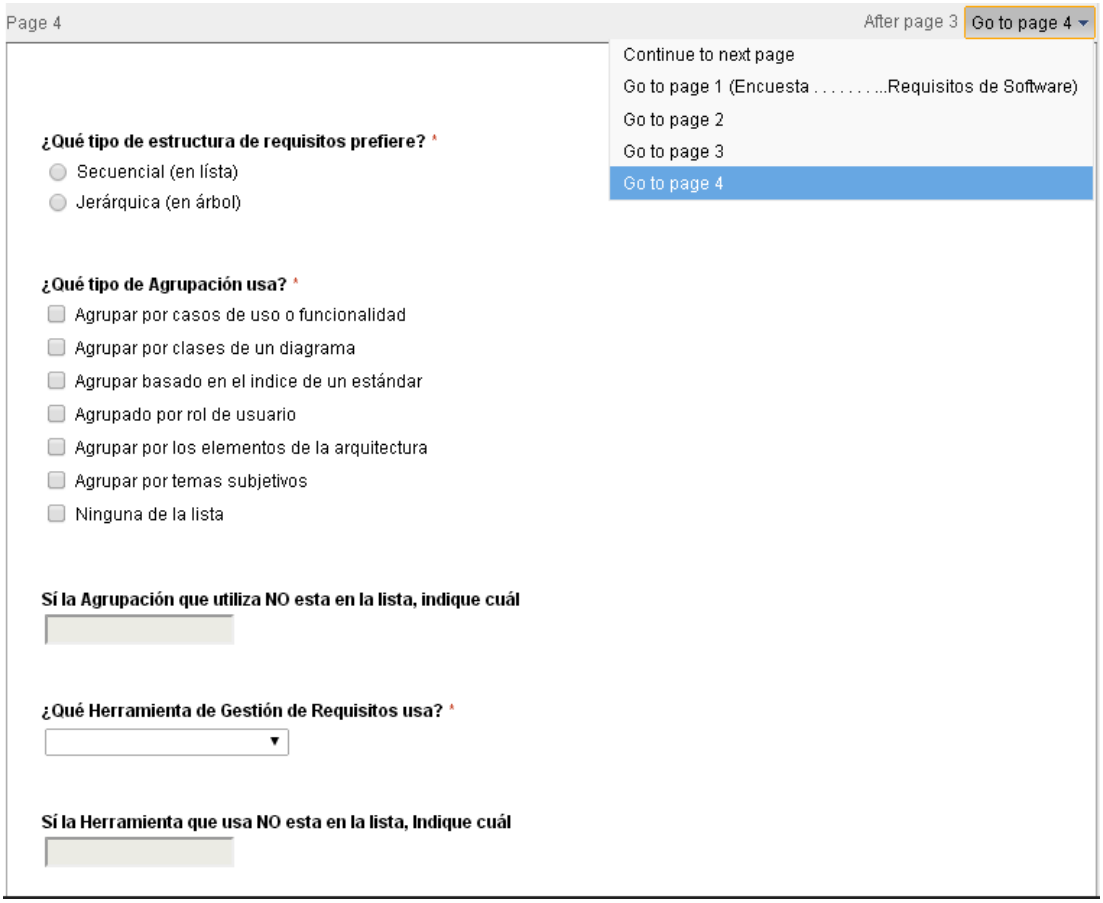
Wiig, K. M. (1997). Integrating intellectual capital and knowledge management. *Long Range Planning*, *30*(3), 323–405. doi:10.1016/s0024-6301(97)00020-4

Yu, Y., do Prado Leite, J. C. S., & Mylopoulos, J. (2004). From Goals to Aspects: Discovering Aspects from Requirements Goal Models. In *Proceedings of the Requirements Engineering Conference, 12th IEEE International* (pp. 38–47). IEEE Computer Society. Retrieved from http://portal.acm.org/citation.cfm?id=1022093

Zave, P., & Jackson, M. (1997). Four Dark Corners of Requirements Engineering. *ACM Trans. Softw. Eng. Methodol.*, *6*(1), 1–30. doi:10.1145/237432.237434

Anexo A
Encuesta sobre la organización de una ER


Encuesta . . . . Método aplicado al Organizar Requisitos de Software
7 preguntas a realizar en un tiempo menor a 2 minutos
Valore su nivel de conocimiento en Ingeniería de Requisitos. *
1 2 3 4 5
bajo muy alto
Perfil principal *
Estudiante universitario en ingeniería
Profesor y/o Investigador universitario en Ingeniería
Analista Junior en empresa
Analista Senior en empresa
Page 2
After page 1
Continue to next page
Continue to next page
Go to page 1 (Encuesta . . . .Requisitos de Software)
Go to page 2
Go to page 3
Go to page 4
Nombre de la Universidad de procedencia *
Sí la Universidad de procedencia NO esta en la lista, indique cuál
Área general de trabajo *
Page 3
After page 2
Go to page 4
Continue to next page
Go to page 1 (Encuesta . . . .Requisitos de Software)
Go to page 2
Go to page 3
Go to page 4
Nombre de la Empresa de procedencia *
Clasificación de la empresa *
1 a 9 empleados (microempresa)
10 a 49 empleados (empresa pequeñas)
50 a 249 empleados (empresa mediana)
250 a 499 empleados (empresa grande)
500 o más empleados (empresa muy grande)


Page 4
After page 3
Go to page 4
Continue to next page
Go to page 1 (Encuesta . . . .Requisitos de Software)
Go to page 2
Go to page 3
Go to page 4
¿Qué tipo de estructura de requisitos prefiere? *
Secuencial (en lísta)
Jerárquica (en árbol)
¿Qué tipo de Agrupación usa? *
Agrupar por casos de uso o funcionalidad
Agrupar por clases de un diagrama
Agrupar basado en el indice de un estándar
Agrupado por rol de usuario
Agrupar por los elementos de la arquitectura
Agrupar por temas subjetivos
Ninguna de la lista
Sí la Agrupación que utiliza NO esta en la lista, indique cuál
¿Qué Herramienta de Gestión de Requisitos usa? *
Sí la Herramienta que usa NO esta en la lista, Indique cuál

Anexo B
The Manchester OWL Syntax

La sintaxis Manchester OWL

Introducción

La sintaxis Manchester OWL es una sintaxis fácil de usar para descripciones OWL 2, pero también puede ser utilizada para escribir ontologías OWL 2 completas. La versión original de la sintaxis Manchester OWL fue creada para OWL 1; aquí está actualizada para OWL 2. La sintaxis Manchester es utilizada en Protégé 4 y TopBraid Composer®, concretamente para registrar y exponer descripciones asociadas a las clases. Algunas herramientas como Protégé 4 amplían la sintaxis para permitir una presentación incluso más compacta en algunas situaciones o para sustituir IRIs por valores de marcado.

La sintaxis Manchester OWL recoge información sobre nombres de manera estructurada, al contrario que RDF/XML, la sintaxis funcional para OWL 2, y la sintaxis XML para OWL 2. Esto está además más cerca de la sintaxis abstracta para OWL 1, que las sintaxis para OWL 2. Sin embargo, el análisis sintáctico de la sintaxis Manchester OWL hacia la especificación estructural OWL 2 es bastante fácil, tan fácil como identificar los axiomas dentro de cada entorno.

Ya que la sintaxis Manchester está estructurada, no puede manejar directamente todas las ontologías OWL 2. Sin embargo, hay una simple transformación que llevará cualquier ontología OWL 2 que no sobrecargue entre propiedades de objetos, datos y anotación, o clases y tipos de datos, en una forma que se pueda escribir en la sintaxis Manchester.

La Gramática

La sintaxis Manchester para ontología OWL 2 es definida utilizando una notación BNF, que es resumida en la tabla siguiente.

Construct	Syntax	Example
non-terminal symbols	boldface	**ClassExpression**
terminal symbols	single quoted	'PropertyRange'
zero or more	curly braces	{ **ClassExpression** }
zero or one	square brackets	[**ClassExpression**]
alternative	vertical bar	**Assertion** | **Declaration**
grouping	parentheses	**dataPropertyExpression**)

Tabla 10 La notación BNF utilizada en anexo C

Debido a que las listas separadas por comas tienen lugar en muchos lugares en la sintaxis, para ahorrar espacios la gramática tiene tres meta-producciones, una para listas no vacías, otra para listas de longitud mínima de dos, y otra para listas no vacías con anotaciones en ellas..

```
<NT>List ::= <NT> { ',' <NT> }
<NT>2List ::= <NT> ',' <NT>List
<NT>AnnotatedList ::= [annotations] <NT> { ',' [annotations] <NT> }
```

Los documentos en la sintaxis Manchester OWL forman ontologías OWL 2, constan de secuencias de caracteres Unicode y son codificados/cifrados en UTF-8.

La gramática de la sintaxis Manchester no muestra explícitamente un espacio en blanco. Los espacios en blanco están permitidos entre dos terminales o no terminales excepto dentro de **nonNegativeInteger**, **prefixName**, **IRI** y **literal**. El espacio en blanco es necesario entre dos terminales o no-terminales si su eliminación pudiera causar ambigüedad. Generalmente esto significa espacios en blanco necesarios excepto antes y después de la puntuación (pe., comas, paréntesis, llaves y corchetes).

El espacio en blanco es una secuencia de blanks (U+20), tabs (U+9), line feeds (U+A), carriage returns (U+D), y comentarios. Los comentarios son unas secuencias máximas de caracteres Unicode comenzando con una almohadilla # y sin contener una line feed o un carriage return. Los comentarios son reconocidos sólo donde el espacio en blanco está permitido, y además no dentro de los anteriores non-terminals.

IRIs, Integers, Literals, y Entities

Los nombres son IRIs (los sucesores de los URIs) y pueden incluso ser dados enteros o pueden ser abreviados de modo similar a como es en SPARQL. Los IRIs abreviados se componen de un prefijo opcional terminado en dos puntos, seguido de una parte local. Los prefijos en los IRIs abreviados no deben encajar las palabras clave de esta sintaxis. Los prefijos deberían comenzar con las minúsculas de forma que no desentonen con las palabras clave terminadas con dos puntos presentadas en futuras versiones de esta sintaxis. Las partes locales sin prefijo son ampliadas como si tuvieran dos puntos iniciales y no deben encajar ninguna palabra clave de esta sintaxis.

Esta sintaxis utiliza formas para valores de datos comunes, por ejemplo, strings y números, y formas cortas de algunos tipos de datos comunes, por ejemplo, integer. Éstos corresponden a las formas largas obvias.

```
fullIRI := an IRI as defined in http://www.ietf.org/rfc/rfc3987.txt, enclosed in a pair of < (U+3C) and > (U+3E) characters

prefixName := a finite sequence of characters matching the PNAME_NS production of SPARQL and not matching any of the keyword terminals of the syntax

abbreviatedIRI := a finite sequence of characters matching the PNAME_LN production of SPARQL

simpleIRI := a finite sequence of characters matching the PN_LOCAL production of SPARQL and not matching any of the keyword terminals of the syntax

IRI := fullIRI | abbreviatedIRI | simpleIRI

nonNegativeInteger ::= zero | positiveInteger

positiveInteger ::= nonZero { digit }

digits ::= digit { digit }

digit ::= zero | nonZero

nonZero := '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

zero ::= '0'


classIRI ::= IRI

Datatype ::= datatypeIRI | 'integer' | 'decimal' | 'float' | 'string'
```

```
datatypeIRI ::= IRI

objectPropertyIRI ::= IRI

dataPropertyIRI ::= IRI

annotationPropertyIRI ::= IRI

individual ::= individualIRI | nodeID

individualIRI ::= IRI

nodeID := a finite sequence of characters matching the
BLANK_NODE_LABEL production of SPARQL


literal ::= typedLiteral | stringLiteralNoLanguage |
stringLiteralWithLanguage | integerLiteral | decimalLiteral |
floatingPointLiteral

typedLiteral ::= lexicalValue '^^' Datatype

stringLiteralNoLanguage ::= quotedString

stringLiteralWithLanguage ::= quotedString languageTag

languageTag := @ (U+40) followed a nonempty sequence of characters
matching the langtag production from http://www.rfc-
editor.org/rfc/bcp/bcp47.txt

lexicalValue ::= quotedString

quotedString := a finite sequence of characters in which " (U+22)
and \ (U+5C) occur only in pairs of the form \" (U+5C, U+22) and \\
(U+5C, U+5C), enclosed in a pair of " (U+22) characters

floatingPointLiteral ::= [ '+' | '-'] ( digits ['.'digits]
[exponent] | '.' digits[exponent]) ( 'f' | 'F' )

exponent ::= ('e' | 'E') ['+' | '-'] digits

decimalLiteral ::= ['+' | '-'] digits '.' digits

integerLiteral ::= ['+' | '-'] digits


entity ::= 'Datatype' '(' Datatype ')' | 'Class' '(' classIRI ')' |
'ObjectProperty' '(' objectPropertyIRI ')' | 'DataProperty'
'('dataPropertyIRI ')' | 'AnnotationProperty' '('
annotationPropertyIRI ')' | 'NamedIndividual' '(' individualIRI ')'
```

Ontologías y Anotaciones

```
annotations ::= 'Annotations:' annotationAnnotatedList

annotation ::= annotationPropertyIRI annotationTarget

annotationTarget ::= nodeID | IRI | literal


ontologyDocument ::= { prefixDeclaration } ontology

prefixDeclaration ::= 'Prefix:' prefixName fullIRI

ontology ::= 'Ontology:' [ ontologyIRI [ versionIRI ] ] { import } {
annotations } { frame }

ontologyIRI ::= IRI

versionIRI ::= IRI
```

```
import ::= 'Import:' IRI

frame ::= datatypeFrame | classFrame | objectPropertyFrame |
dataPropertyFrame | annotationPropertyFrame | individualFrame | misc
```

Los prefijos 'rdf:', 'rdfs:', 'owl:', y 'xsd:' están predefinidos del siguiente modo y no pueden ser cambiados. Cada prefijo utilizado en un documento ontológico debe tener exactamente una declaración de prefijo en el documento ontológico.

```
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>

Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>

Prefix: owl: <http://www.w3.org/2002/07/owl#>
```

Propiedades y Expresiones datatype

```
objectPropertyExpression ::= objectPropertyIRI |
inverseObjectProperty

inverseObjectProperty ::= 'inverse' objectPropertyIRI

dataPropertyExpression ::= dataPropertyIRI


dataRange ::= dataConjunction 'or' dataConjunction { 'or'
dataConjunction }
         | dataConjunction

dataConjunction ::= dataPrimary 'and' dataPrimary { 'and'
dataPrimary }
         | dataPrimary

dataPrimary ::= [ 'not' ] dataAtomic

dataAtomic ::= Datatype
         | '{' literalList '}'
         | datatypeRestriction | '(' dataRange ')'

datatypeRestriction ::= Datatype '[' facet restrictionValue { ','
facet restrictionValue } ']'

facet ::= 'length' | 'minLength' | 'maxLength' | 'pattern' |
'langRange' | '<=' | '<' | '>=' | '>'

restrictionValue ::= literal
```

En la restricción datatype, las facetas y los valores de restricción deben ser válidos para el datatype, como en la especificación OWL 2, después hacer los cambios obvios para las facetas de comparación.

Descripciones

La notación infija para descripciones es ambigua como ya se dijo. Esta ambigüedad es resuelta normalmente con producciones posteriores en la gramática descriptiva siguiente, vinculando más firmemente; así, por ejemplo,

```
p some a and p only b
```

se analiza como

```
(p some a) and (p only b)
description ::= conjunction 'or' conjunction { 'or' conjunction }
         | conjunction
conjunction ::= classIRI 'that' [ 'not' ] restriction { 'and' [
'not' ] restriction }
         | primary 'and' primary { 'and' primary }
         | primary
primary ::= [ 'not' ] ( restriction | atomic )
restriction ::= objectPropertyExpression 'some' primary
         | objectPropertyExpression 'only' primary
         | objectPropertyExpression 'value' individual
         | objectPropertyExpression 'Self'
         |   objectPropertyExpression   'min'   nonNegativeInteger   [
primary ]
         |   objectPropertyExpression   'max'   nonNegativeInteger   [
primary ]
         |  objectPropertyExpression  'exactly'  nonNegativeInteger  [
primary ]
         | dataPropertyExpression 'some' dataPrimary
         | dataPropertyExpression 'only' dataPrimary
         | dataPropertyExpression 'value' literal
         |   dataPropertyExpression   'min'   nonNegativeInteger   [
dataPrimary ]
         |   dataPropertyExpression   'max'   nonNegativeInteger   [
dataPrimary ]
         |  dataPropertyExpression  'exactly'  nonNegativeInteger  [
dataPrimary ]
atomic ::= classIRI
         | '{' individualList '}'
         | '(' description ')'
```

Entornos y Miscelánea

```
datatypeFrame ::= 'Datatype:' Datatype
       { 'Annotations:'     annotationAnnotatedList }
       [ 'EquivalentTo:' annotations dataRange ]
       { 'Annotations:'     annotationAnnotatedList }

classFrame ::= 'Class:' classIRI
       { 'Annotations:'     annotationAnnotatedList
       | 'SubClassOf:'      descriptionAnnotatedList
```

```
       | 'EquivalentTo:'    descriptionAnnotatedList
       | 'DisjointWith:'    descriptionAnnotatedList
       | 'DisjointUnionOf:' annotations description2List }
       |   'HasKey:'   annotations   (   objectPropertyExpression   |
dataPropertyExpression )
                                        { objectPropertyExpression |
dataPropertyExpression }


objectPropertyFrame ::= 'ObjectProperty:' objectPropertyIRI
       { 'Annotations:'     annotationAnnotatedList
       | 'Domain:'          descriptionAnnotatedList
       | 'Range:'           descriptionAnnotatedList
       |                                          'Characteristics:'
objectPropertyCharacteristicAnnotatedList
       | 'SubPropertyOf:'   objectPropertyExpressionAnnotatedList
       | 'EquivalentTo:'    objectPropertyExpressionAnnotatedList
       | 'DisjointWith:'    objectPropertyExpressionAnnotatedList
       | 'InverseOf:'        objectPropertyExpressionAnnotatedList
       |  'SubPropertyChain:'  annotations  objectPropertyExpression
'o' objectPropertyExpression
                                                 {                'o'
objectPropertyExpression } }

objectPropertyCharacteristic ::= 'Functional' | 'InverseFunctional'
            |   'Reflexive'   |   'Irreflexive'   |   'Symmetric'   |
'Asymmetric' | 'Transitive'

dataPropertyFrame ::= 'DataProperty:' dataPropertyIRI
       { 'Annotations:'     annotationAnnotatedList
       | 'Domain:'          descriptionAnnotatedList
       | 'Range:'           dataRangeAnnotatedList
       | 'Characteristics:' annotations 'Functional'
       | 'SubPropertyOf:'   dataPropertyExpressionAnnotatedList
       | 'EquivalentTo:'    dataPropertyExpressionAnnotatedList
       | 'DisjointWith:'    dataPropertyExpressionAnnotatedList }

annotationPropertyFrame          ::=          'AnnotationProperty:'
annotationPropertyIRI
       { 'Annotations:'     annotationAnnotatedList }
       | 'Domain:'          IRIAnnotatedList
       | 'Range:'           IRIAnnotatedList
       | 'SubPropertyOf:'   annotationPropertyIRIAnnotatedList
```

```
individualFrame ::= 'Individual:' individual
       { 'Annotations:'   annotationAnnotatedList
       | 'Types:'         descriptionAnnotatedList
       | 'Facts:'         factAnnotatedList
       | 'SameAs:'        individualAnnotatedList
       | 'DifferentFrom:' individualAnnotatedList }

fact ::= [ 'not' ] (objectPropertyFact | dataPropertyFact)
objectPropertyFact ::= objectPropertyIRI individual
dataPropertyFact ::= dataPropertyIRI literal

misc ::= 'EquivalentClasses:' annotations description2List
       | 'DisjointClasses:' annotations description2List
       | 'EquivalentProperties:' annotations objectProperty2List
       | 'DisjointProperties:' annotations objectProperty2List
       | 'EquivalentProperties:' annotations dataProperty2List
       | 'DisjointProperties:' annotations dataProperty2List
       | 'SameIndividual:' annotations individual2List
       | 'DifferentIndividuals:' annotations individual2List
```

Asuntos Globales

La sintaxis Manchester tiene las mismas condiciones globales sobre las ontologías que las ontologías para OWL 2 en la especificación OWL 2, con el añadido de las restricciones de typing para ontologías OWL 2 DL, pero utilizando el entorno apropiado en lugar de las declaraciones.

Las condiciones globales de la sintaxis Manchester para ontologías OWL 2 DL son las mismas que para la especificación OWL 2 excepto por lo mencionado antes.