

Universidad de Castilla-La Mancha

Escuela Superior de Ingeniería Informática

Departamento de Sistemas Informáticos

Programa Oficial de Postgrado en Tecnologías Informáticas Avanzadas

Trabajo Fin de Máster

UN ENFOQUE INDUSTRIAL PARA LA GESTIÓN DEL CONOCIMIENTO ARQUITECTÓNICO

Julio de 2011

Alumna: Cristina Roda Sánchez

Directora: Dra. Dña. Elena María Navarro Martínez
Co-Director: Dr. D. Carlos Enrique Cuesta Quintero

Agradecimientos

En primer lugar, quiero darle las gracias a Elena Navarro, por haber depositado su confianza en mí desde el primer momento. Gracias Elena, por estar siempre ahí, por enseñarme tantas cosas, por animarme en todo momento, por descubrirme este apasionante mundo de la investigación. Sin tu apoyo y tu dedicación constante esto no habría sido lo mismo.

Gracias también a Carlos Cuesta, por creer y confiar en mí, a penas sin conocerme. Gracias Carlos, por tus comentarios cercanos y amables, que siempre levantan la moral y ayudan a seguir adelante.

Gracias a mis padres, Luis y Felisa, por su apoyo incondicional durante todos estos años de estudio y esfuerzo. Gracias por enseñarme que no hay que rendirse nunca y que hay que luchar por lo que uno quiere. Gracias por vuestro sacrificio continuo, para conseguirme un futuro mejor. Os quiero mucho.

Gracias a mis hermanos, Luis y Natalia, que siempre están ahí. Gracias por llenar la casa de alegría, bromas y risas, que siempre animan a seguir adelante. Os quiero mucho.

Gracias a mi pareja, Antonio, por todos estos años de amor, amistad, compañía y apoyo. Gracias Antonio por estar siempre ahí, cuando te necesito y cuando pienso que no te necesito; por ser crítico con mis pensamientos y mi trabajo; por apoyarme siempre y en todo momento. Te quiero mucho.

Finalmente, gracias a todos aquellos que me han ayudado durante este curso en la realización de esta tesis de Máster, especialmente a Dewayne Perry, por apoyar y confiar en mi trabajo; a Mario Plaza, por su ayuda con la estadística; y a Víctor López, por "prestarme" sus clases.

Gracias a todos.

Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En este documento, se recopila todo el trabajo desarrollado durante el periodo de realización del *Máster en Tecnologías Informáticas Avanzadas*. Este máster constituye la fase de formación del programa de doctorado en *Tecnologías Informáticas Avanzadas*. El curso es de carácter investigador, en modalidad semi-presencial, donde los idiomas de impartición son el español y el inglés, y los lugares de impartición son la Escuela Superior de Ingeniería Informática de Albacete y la Escuela Superior de Informática de Ciudad Real.

Durante este periodo se han cursado las siguientes seis asignaturas:

- Generación de Documentos Científicos en Informática
- Introducción a la Programación de Arquitecturas de Altas Prestaciones
- Sistemas Inteligentes Aplicados a Internet
- Calidad de Interfaces de Usuario: Desarrollo Avanzado
- Tecnología Software Orientada a Objetos
- Programación Internet con Lenguajes Declarativos Multiparadigma

Cada una de estas asignaturas tiene 5 créditos ECTS, cursándose un total de 30 créditos como introducción a la labor investigadora. Durante el primer cuatrimestre, se cursaron las tres primeras y, en el segundo cuatrimestre, las tres restantes.

Esta tesis de máster tiene la estructura que se detalla a continuación:

- En el capítulo 2, se describe cada una de las asignaturas cursadas, así como el trabajo realizado en cada una de ellas.
- En el capítulo 3, se describe el estado del arte del área en la que el alumno va a trabajar durante la realización de su tesis doctoral. Así mismo, también se describe la línea de investigación en la que se va a trabajar durante el doctorado.
- En el capítulo 4, se detalla el anteproyecto de la tesis doctoral, indicando la metodología y planificación de la misma.
- En el capítulo 5, se incluye el currículum vitae del doctorando.

Capítulo 2

Cursos de Doctorado

En este capítulo, se describe cada una de las asignaturas cursadas durante la realización del *Máster en Tecnologías Informáticas Avanzadas*, en el curso 2010/2011. Junto a cada asignatura, se incorpora una breve descripción de la misma y las distintas actividades realizadas para su evaluación.

2.1 Generación de Documentos Científicos en Informática

2.1.1 Descripción

La asignatura *Generación de Documentos Científicos en Informática* se centra en las buenas prácticas para la creación de documentos científicos, facilitando para ello tres módulos bien diferenciados:

- El primer módulo sobre *Metodología de investigación*, impartido por el doctor D. José Antonio Gámez Martín, presenta el contexto investigador en el cual se mueven los alumnos de doctorado, así como ciertas sugerencias para llevar a cabo su labor investigadora. También se explica cómo escribir y estructurar de manera adecuada un documento técnico y cómo realizar presentaciones apropiadas. Además, se mencionan los distintos medios de publicación de documentos científicos (revistas, congresos, etc.), así como los distintos índices que existen para evaluar la calidad investigadora de un determinado investigador.
- El segundo módulo sobre *Contraste de hipótesis*, impartido por el doctor D. Francisco Parreño Torres, presenta la estadística aplicada al ámbito investigador, que permite reforzar las conclusiones obtenidas de una manera formal. Para ello, se presentaron los distintos tipos de contrastes de hipótesis que se pueden realizar, así como algunos ejemplos concretos de los mismos. Además, se introdujo la herramienta estadística *R* y su módulo *R-Commander*, que permite realizar estos contrastes de una forma rápida y eficaz.
- El tercer y último módulo sobre *Composición de documentos y presentaciones con LaTeX*, impartido por el doctor D. Luis de la Ossa Jiménez, introduce la generación de documentos científicos en LaTeX, el cual se considera uno de los generadores más adecuados para diseñar presentaciones y editar artículos científicos con expresiones matemáticas complejas.

2.1.2 Trabajo realizado

Para esta asignatura se llevaron a cabo dos trabajos para poner en práctica todo lo explicado en la misma. El primero de ellos, propuesto por el doctor D. José Antonio Gámez Martín, consistió en calcular los índices *h*, *g* y *hg*, del grupo de investigación en el que estuviéramos involucrados, en este caso el grupo LoUISE. Además, este cálculo se realizó de dos formas distintas: formando un "pool" con todos los artículos de los integrantes del grupo y calculando el índice como si de un único autor se tratara; y considerando como "artículos" a los investigadores y como número de citas al índice de cada investigador y, posteriormente, calculando el índice total. Estos índices se midieron desde diferentes fuentes: *Scholar Google* y *Scopus*.

El segundo trabajo propuesto como trabajo final de esta asignatura se realizó en LaTeX y consistió en el planteamiento de dos problemas, uno para un test paramétrico y otro para un test no paramétrico, explicando, en primer lugar, en qué consistía cada método y, posteriormente, resolviendo cada uno de los

ejemplos planteados. Además, se realizó una presentación sobre el mismo para exponerlo, posteriormente, a los compañeros y profesores de la asignatura.

2.2 Introducción a la Programación de Arquitecturas de Altas Prestaciones

2.2.1 Descripción

La asignatura de *Introducción a la Programación de Arquitecturas de Altas Prestaciones* es impartida por los doctores D. Enrique Arias Antúnez, D. Diego Cazorla López y D. Juan José Pardo Mateo, y se estructura de la siguiente manera:

- La primera parte de la asignatura, impartida por el doctor D. Diego Cazorla López, intenta introducir la programación de arquitecturas de altas prestaciones, así como presentar las ideas básicas de la programación secuencial orientada a bloques. De esta manera, se tienen los conocimientos necesarios para realizar la primera práctica, la cual versa sobre la optimización secuencial de código.
- La segunda parte de la asignatura intenta presentar los modelos y las ideas básicas de la computación paralela. De esta forma, la parte teórica la imparte el doctor D. Enrique Arias Antúnez, mientras que la parte práctica, que versa sobre la programación paralela con MPI y BLAS, la imparte el doctor D. Juan José Pardo Mateo.
- Finalmente, la última parte de la asignatura intenta que el alumno aprenda a diseñar programas en arquitecturas paralelas, e introduce el software de programación de arquitecturas de altas prestaciones. De esta forma, la parte teórica la imparten los doctores D. Diego Cazorla López y D. Enrique Arias Antúnez, mientras que la parte práctica, que versa sobre la programación paralela con BLACS y PBLAS, sólo la imparte éste último.

2.2.2 Trabajo realizado

Para esta asignatura, se realizaron las tres prácticas anteriormente mencionadas, resolviendo los distintos problemas planteados y generando una memoria escrita para cada una de ellas.

También se llevó a cabo un trabajo final de la asignatura, el cual podía ser asignado por los doctores, o planteado por los propios alumnos. En este caso, el trabajo trataba de dar una visión particular sobre cómo llevar la computación paralela al ámbito de la ingeniería del software. Para dicho trabajo se debía presentar una memoria, así como una presentación para, posteriormente, exponerlo al resto de compañeros y profesores de la asignatura.

2.3 Sistemas Inteligentes Aplicados a Internet

2.3.1 Descripción

La asignatura de *Sistemas Inteligentes Aplicados a Internet* es impartida por los doctores D. Ismael García Varea, D. José Miguel Puerta Callejón y Dña. María Julia Flores Gallego. Esta asignatura se divide en tres unidades temáticas bien diferenciadas, donde cada una de ellas es impartida por uno de los doctores mencionados.

La primera unidad temática sobre *Modelado e Inferencia en Redes Bayesianas*, impartida por la doctora Dña. María Julia Flores Gallego, está compuesta por cinco temas que versan sobre las redes Bayesianas, desde el modelado hasta la inferencia. Además, esta unidad temática incluye una explicación sobre la utilidad de la herramienta *Elvira* para el caso concreto de las redes Bayesianas.

La segunda unidad temática sobre *Aprendizaje de Redes Bayesianas*, impartida por el doctor D. José Miguel Puerta Callejón, está constituida por tres temas que tratan sobre el proceso de aprendizaje en redes Bayesianas, tanto paramétrico como estructural.

La tercera y última unidad temática sobre *Metaheurísticas y Minería de Datos*, impartida por el doctor D. Ismael García Varea, está subdividida, a su vez, en dos partes claramente diferenciadas: Metaheurísticas (1) y Minería de Datos (2). Cada una de estas partes está compuesta por tres temas, que versan sobre algoritmos genéticos y de estimación de distribuciones, y sobre técnicas de minería de datos y evaluación de modelos de clasificación supervisada. Además, esta unidad temática se completa con una breve explicación acerca de la herramienta *Weka*, la cual está especialmente orientada a metaheurísticas y minería de datos.

2.3.2 Trabajo realizado

Para esta asignatura, se realizó una tarea opcional, propuesta por la doctora Dña. María Julia Flores Gallego, que implicaba la búsqueda de una aplicación concreta de redes Bayesianas.

Además, para cada unidad temática, se propusieron diversos ejercicios, de los cuales algunos eran de carácter obligatorio para poder aprobar la asignatura, mientras que otros eran de carácter opcional para subir nota. En este caso, se realizaron todos los ejercicios obligatorios, así como los ejercicios opcionales para subir nota.

2.4 Calidad de Interfaces de Usuario: Desarrollo Avanzado

2.4.1 Descripción

La asignatura de *Calidad de Interfaces de Usuario: Desarrollo Avanzado* es impartida por los doctores D. Pascual González López, D. Víctor Manuel López-Jaquero y D. Francisco Montero Simarro. Esta asignatura se divide en tres bloques de contenidos, donde cada uno de ellos es impartido por uno de los doctores mencionados.

El primer bloque temático sobre *Calidad de Interfaces de Usuario*, impartido por el doctor D. Pascual González López, está compuesto por tres apartados principales: una introducción sobre qué es la calidad y cómo distinguirla, además de algunas definiciones relevantes; una presentación de las distintas habilidades del ser humano utilizadas para percibir dicha calidad; finalmente, una descripción de diversos conceptos relacionados con la calidad de las interfaces de usuario.

El segundo bloque temático sobre *Diseño de Interfaces de Usuario Basado en Modelos y Adaptación*, impartido por el doctor D. Víctor Manuel López-Jaquero, versa sobre el diseño de interfaces de usuario basado en modelos, así como la ingeniería de adaptación de las interfaces de usuario. De esta forma, se tratan los siguientes ítems: modelos, qué son y qué representan; *Arquitectura Dirigida por Modelos* (*Model-Driven Architecture*, MDA); *Diseño de Interfaces de Usuario Basado en Modelos* (*Model-Based User Interface Design*, MB-UID); modelos en MB-UID; usiXML, un lenguaje de descripción de interfaces de usuario; transformación de modelos en MB-UID; adaptación; ISATINE, un proceso de adaptación; y, por último, especificación de reglas de adaptación.

El tercer bloque temático sobre *Calidad y Experiencia en el Desarrollo de Interfaces de Usuario*, impartido por el doctor D. Francisco Montero Simarro, versa sobre la calidad y el desarrollo centrado en el usuario, así como sobre la integración de experiencia y evaluación de interfaces. De esta forma, este módulo presenta el concepto de calidad en el ámbito de las interfaces de usuario, considerando los siguientes puntos: prerrequisitos y objetivos; presentación y definiciones; modelos de calidad centrados en la usabilidad; logro de la usabilidad; y, por último, técnicas y herramientas para evaluar la usabilidad.

2.4.2 Trabajo realizado

La evaluación de esta asignatura se realiza en base a la valoración de la actitud y trabajo llevado a cabo por los alumnos de manera individual. De esta forma, para superar la asignatura, se debía realizar un trabajo relacionado con la misma, ya fuera asignado por los profesores o elegido por el alumno. En este caso, se ha optado por proponer un tema, relacionado con el tercer bloque temático sobre *Calidad y Experiencia en el Desarrollo de Interfaces de Usuario*. Dicho trabajo describe diversos cuestionarios de usabilidad que se pueden utilizar para evaluar la calidad de un sistema software, con respecto a la usabilidad. Además, se propone una comparativa entre los distintos cuestionarios descritos, que permite seleccionar el cuestionario más adecuado para un estudio concreto.

2.5 Tecnología Software Orientada a Objetos

2.5.1 Descripción

La asignatura de *Tecnología Software Orientada a Objetos* es impartida por los doctores Dña. María Dolores Lozano Pérez, Dña. Elena María Navarro Martínez y D. Víctor Manuel Ruiz Penichet. Esta asignatura se divide en tres módulos bien diferenciados, donde cada uno de ellos es impartido por uno de los doctores mencionados.

El primer módulo es una *Introducción al Desarrollo Software Dirigido por Modelos*, impartido por la doctora Dña. Elena María Navarro Martínez, que versa sobre el desarrollo de software basado en modelos. Este módulo incluye contenidos teóricos como la introducción al manejo de las herramientas EMF (*Eclipse Modeling Framework*) y GMF (*Eclipse Graphical Modeling Framework*), y sesiones prácticas para introducir las transformaciones Modelo a Modelo y Modelo a Texto.

El segundo módulo sobre *Tendencias Actuales en el Desarrollo de Interfaces de Usuario*, impartido por la doctora Dña. María Dolores Lozano Pérez, versa sobre el desarrollo de interfaces de usuario basado en modelos, y consta de los siguientes puntos: introducción al desarrollo de interfaces de usuario, con la aproximación MB-UIDE (*Model-Based User Interface Development Environment*); entorno metodológico basado en modelos para el desarrollo de GUIs, y su aplicación a un caso de estudio; ejercicio práctico para

desarrollar un prototipo de interfaz de usuario basado en modelos; por último, MDA (*Model-Driven Architecture*) aplicado al desarrollo de interfaces para entornos ubicuos sensibles al contexto, y la herramienta desarrollada CAUCE CASE Tool. Nótese que este último punto fue impartido por el doctor D. Ricardo Tesoriero, dado que formaba parte de su tesis doctoral.

El tercer y último módulo sobre *Modelo de Procesos para el Desarrollo de Interfaces de Usuario Colaborativas*, impartido por el doctor D. Víctor Manuel Ruiz Penichet en el contexto de una sesión intensiva, versa sobre el trabajo desarrollado en la tesis doctoral del mismo, acerca del modelo de procesos orientado a tareas y centrado en el usuario para desarrollar interfaces para entornos Persona-Ordenador-Persona (*Human-Computer-Human*). También se introdujo la herramienta TOUCHE CASE Tool, desarrollada para este propósito.

2.5.2 Trabajo realizado

La evaluación de esta asignatura se realiza en base a la asistencia a clase y el seguimiento de los contenidos básicos, además de las lecturas de los artículos básicos y la realización de un trabajo resumen de las conclusiones. Finalmente, se debe elaborar y exponer un trabajo de investigación relacionado con la asignatura.

De esta forma, en el primer módulo, se realizaron diversas tareas, concretamente, ejercicios prácticos relativos a EMF y GMF, QVT (*Query/View/Transformation*), XPAND (lenguaje Modelo a Texto); y la lectura del artículo (Mattsson et al. 2008), exponiendo sus críticas principales a la propuesta MDD, así como las conclusiones y opinión personal sobre dicha propuesta.

En el segundo módulo, se llevó a cabo una sesión de puzle donde se crearon varios grupos, con el objetivo de aprender los distintos modelos existentes para desarrollar interfaces de usuario. También se realizó un ejercicio práctico para desarrollar un prototipo de interfaz para una tienda virtual de venta de libros online, a partir de los modelos de interfaz de usuario vistos anteriormente.

Finalmente, el trabajo final de la asignatura consistió en describir los distintos enfoques de *Desarrollo Software Dirigido por Modelos* (*Model-Driven Development*, MDD) aplicados al Conocimiento Arquitectónico, incluyendo, entre otros, la propuesta ATRIUM (Navarro & Cuesta 2008) y el modelado de variabilidad (Sinnema et al. 2006). Además, se comentó la experiencia sobre el uso de MDD en grandes proyectos industriales (Mattsson et al. 2007).

2.6 Programación Internet con Lenguajes Declarativos Multiparadigma

2.6.1 Descripción

La asignatura de *Programación Internet con Lenguajes Declarativos Multiparadigma* es impartida por los doctores D. Pascual Julián Iranzo, D. Ginés Damián Moreno Valverde y D. Francisco Pascual Romero Chicharro. Esta asignatura se divide en tres bloques bien diferenciados, donde cada uno de ellos es impartido por uno de los doctores mencionados.

La primera parte de la asignatura, impartida por el doctor D. Pascual Julián Iranzo, intenta dar a conocer los fundamentos y características de los lenguajes declarativos multiparadigma, que integran la programación lógica con el paradigma difuso. Además, se estudian los mecanismos operacionales y la semántica de algunas de las propuestas de integración, así como las técnicas de transformación de programas que permiten optimizarlos.

La segunda parte de la asignatura, impartida por el doctor D. Ginés Damián Moreno Valverde, versa sobre la programación lógica difusa (aproximación Multiadjunta) e Internet, dando a conocer las facilidades de un lenguaje declarativo concreto para la programación internet.

Finalmente, la tercera y última parte de la asignatura, impartida por el doctor D. Francisco Pascual Romero Chicharro, presenta el tópico de la búsqueda en internet desde una perspectiva declarativa.

2.6.2 Trabajo realizado

Para superar la asignatura, la asistencia a clase debía ser, como mínimo, del 80%, o bien, se podía elaborar y exponer un trabajo relacionado con los contenidos teórico/prácticos del curso. Adicionalmente, y para subir nota, el alumno podía optar por completar ambas actividades, es decir, la asistencia a clase y la realización y posterior exposición de un trabajo.

En este caso, se optó por la asistencia a clase y la exposición de un trabajo relacionado con la parte presentada por el doctor D. Ginés Damián Moreno Valverde. Dicho trabajo versaba sobre el desplegado de definiciones de conectivos en *Programas Lógicos Multi-Adjuntos* (*Multi-Adjoint Logic Programs*, MALP).

Capítulo 3

Estado del arte

En este capítulo se presenta el estado del arte relacionado con los puntos que se tratarán posteriormente en el anteproyecto de tesis presentado en capítulo siguiente. Primeramente, se describe el área de investigación en la que se va a trabajar. Posteriormente, se introducen cada uno de los temas relacionados con la futura tesis doctoral.

3.1 Introducción

El trabajo de investigación que presenta el alumno durante su doctorado, se enmarca dentro del área de *Arquitectura Software*. Durante varias décadas se ha resaltado la importancia de dicho artefacto en el proceso, no sólo de desarrollo, sino también de mantenimiento. Sin embargo sólo desde el año 2005 se ha empezado a dar relevancia a uno de los elementos claves de su descripción, las *decisiones de diseño*, haciendo aparecer un nuevo campo de investigación conocido como *Conocimiento Arquitectónico*. Ambos campos son brevemente introducidos en las secciones 3.2 y 3.3, respectivamente.

Tal y como se detalla en el capítulo 4, el anteproyecto de tesis doctoral versa sobre el desarrollo de nuevos métodos, técnicas y herramientas que mejoren el análisis de la red de conocimiento arquitectónica. Así, en la sección 3.4, se ofrece una visión sobre la utilización que actualmente se está realizando de las técnicas de Desarrollo Dirigido por Modelos en el campo del Conocimiento Arquitectónico. En la sección 3.5, se describen las técnicas que se están empleando actualmente para la visualización del mismo. Finalmente, en la sección 3.6, se realiza una introducción al análisis de redes sociales, a fin de facilitar la comprensión de cómo se pretenden explotar éstas para el análisis de la red de conocimiento.

3.2 Introducción a la Arquitectura Software

No hay una definición estándar y reconocida del término *Arquitectura Software* (*Software Architecture*, SA). Es por esto que se puede encontrar un amplio conjunto de definiciones, como las que se detallan seguidamente:

- (D. E. Perry & A. L. Wolf 1992) señalan que la Arquitectura Software es cuestión de elegir los elementos arquitectónicos, pero, además, se deben considerar las interacciones entre ellos: *"*La Arquitectura tiene que ver con la selección de elementos arquitectónicos, sus interacciones, y las limitaciones sobre dichos elementos y sus interacciones necesarias para proporcionar un marco en el que satisfacer los requisitos y servir como base para el diseño*"*.
- (Kruchten 1995) la define así: "La Arquitectura Software tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requisitos de desempeño de un sistema, así como requisitos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad".
- (D. Garlan & Shaw 1993) indican que ha surgido un nuevo tipo de problema: el diseño y la especificación de la estructura global del sistema. De esta forma, presentan su propia visión de la Arquitectura Software: *"*Más allá de los algoritmos y de las estructuras de datos de la computación; el diseño y la especificación de la estructura global del sistema emergen como un nuevo tipo de problema. Los problemas estructurales incluyen una gruesa organización y una estructura de

control global; protocolos para la comunicación, sincronización, y el acceso a los datos; asignación de funcionalidad a los elementos de diseño; distribución física; composición de los elementos de diseño; escalada y rendimiento; y selección entre alternativas de diseño".

- (D. Garlan & D E Perry 1995) tienen una perspectiva similar a los autores anteriores, definiendo la Arquitectura Software escuetamente como "la estructura de los componentes de un programa/sistema, sus interrelaciones, y los principios y directrices que rigen su diseño y evolución en el tiempo".
- El Instituto Nacional de Estándares Americano (Anon 2000) ofrece una definición extensa de Arquitectura Software que, además de englobar los aspectos que sugieren los anteriores autores, introduce el término de *patrones de diseño*: "La Arquitectura es definida, por la práctica recomendada, como la organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y el entorno, y los principios que rigen su diseño y evolución. Esta definición pretende abarcar una gran variedad de usos del término arquitectura, mediante el reconocimiento de sus elementos comunes fundamentales. Entre los principales, destaca la necesidad de entender y controlar dichos elementos del diseño del sistema, que capturan la utilidad del mismo, el coste, y el riesgo. En algunos casos, estos elementos son los componentes físicos del sistema y sus relaciones. En otros casos, estos elementos no son físicos, sino componentes lógicos. En otros casos, además, estos elementos son principios duraderos o patrones, que crean estructuras organizacionales duraderas. La definición pretende abarcar estos usos distintos, pero relacionados entre sí, fomentando al mismo tiempo una definición más rigurosa de lo que constituye la organización fundamental de un sistema dentro de los dominios particulares".

La mayoría de estas definiciones emergieron en la década de los 90, la cual, según afirmaban (D. E. Perry & A. L. Wolf 1992), sería la década de la Arquitectura Software. En este periodo, se utilizaba el término *arquitectura* (*architecture*), en lugar de *diseño* (*design*), de tal forma que se evocaran nociones de codificación, abstracción, estándares, formación (de los arquitectos software) y estilo.

En esta misma década, (D. E. Perry & A. L. Wolf 1992) indican algunos de los beneficios más importantes que supone la aparición de la SA como disciplina:

1. Arquitectura como marco para la satisfacción de requisitos.
2. Arquitectura como fundamento técnico para el diseño, y como base administrativa para la estimación del coste y la gestión de procesos.
3. Arquitectura como base efectiva para la reutilización.
4. Arquitectura como base para el análisis de la dependencia y la coherencia. Estos autores destacan los siguientes tipos de análisis:
   - Coherencia de las limitaciones del estilo arquitectónico.
   - Satisfacción de los estilos arquitectónicos por parte de una arquitectura.
   - Coherencia de las limitaciones arquitectónicas.
   - Satisfacción de la arquitectura por parte del diseño.
   - Establecimiento de dependencias entre la arquitectura y el diseño, y entre la arquitectura y los requisitos.
   - Determinación de las implicaciones de los cambios en la arquitectura o el estilo arquitectónico en el diseño y los requisitos, y viceversa.

Estos dos autores tenían la intuición de que había diferentes tipos de SA, pero no los formalizaron ni institucionalizaron, debido a la diversidad de tipos de arquitecturas que se habían definido hasta ese momento. Sin embargo, expusieron las distintas *disciplinas arquitectónicas* existentes, extrayendo los aspectos más relevantes que les permitieran desarrollar esa intuición que ambos sostenían sobre la SA.

La primera disciplina tratada fue la *Arquitectura Hardware* (*Hardware Architecture*). Había diferentes aproximaciones con respecto a este tipo de arquitectura, las cuales se distinguían según el aspecto hardware sobre el que hacían mayor hincapié. Por ejemplo, las máquinas con arquitectura RISC presentan una arquitectura hardware, la cual enfatiza la característica relacionada con el conjunto de instrucciones. Las máquinas multi-procesador, en cambio, son ejemplos de arquitecturas hardware que se centran en la configuración de las piezas arquitectónicas del hardware. (D. E. Perry & A. L. Wolf 1992) destacan dos interesantes características de este enfoque: "Hay un número relativamente pequeño de elementos de diseño; y la escalabilidad se consigue con la replicación de dichos elementos." Esto contrastaba con la SA, donde hay un número muy elevado de posibles elementos de diseño.

La segunda disciplina de la que se habla es la *Arquitectura de Red* (*Network Architecture*). Los dos aspectos arquitectónicos destacados de dicha disciplina respecto a la SA eran que "hay dos componentes – nodos y

conexiones; y sólo hay unas pocas topologías consideradas". Es cierto que esta Arquitectura de Red tenía ciertas similitudes con la SA. Sin embargo, en esta última, había un número excesivamente grande de posibles topologías y éstas, generalmente, no tenían nombre.

Por último, (D. E. Perry & A. L. Wolf 1992) consideran la *Arquitectura de Construcción* (*Building Architecture*), es decir, la Arquitectura propiamente dicha de construcción de edificios y estructuras. Esta disciplina proporciona los puntos de vista más interesantes para la SA: vistas múltiples, estilos arquitectónicos, estilo e ingeniería, estilo y materiales. Esto se debe a que un arquitecto de la construcción trabaja con el cliente por medio de diferentes *vistas*, donde se destaca algún aspecto particular de la construcción. Análogamente, el arquitecto software necesita un número de vistas diferentes de la SA para los distintos usos y usuarios. En el año 1992, concretamente, se consideraba que el arquitecto software trabajaba sólo con una vista: la *implementación*.

Además de relacionar la SA con el resto de disciplinas arquitectónicas, (D. E. Perry & A. L. Wolf 1992) destacan una propiedad que acompaña con frecuencia a la evolución de la SA: la *fragilidad creciente del sistema*, es decir, un aumento de la resistencia al cambio o, por lo menos, a los cambios con cierto impacto. Esto se debe, en parte, a los siguientes problemas arquitectónicos:

- *Erosión Arquitectónica* (*Architectural Erosion*), debida a las violaciones de la arquitectura. Estas violaciones, a menudo, conducen a un aumento de los problemas en el sistema y contribuyen al incremento de la fragilidad del mismo.
- *Desviación Arquitectónica* (*Architectural Drift*), debido a la insensibilidad sobre la arquitectura. Esta insensibilidad conduce más a una inadaptabilidad que a desastres, y resulta una falta de coherencia y claridad en la forma que, a su vez, hace que sea mucho más fácil violar la arquitectura, la cual ahora se ha vuelto más confusa.

Dados los problemas detectados respecto a la motivación y la caracterización de la SA, (D. E. Perry & A. L. Wolf 1992) indican los puntos que debemos ser capaces de satisfacer para realizar correctamente su especificación:

- Describir las limitaciones arquitectónicas hasta el nivel deseado.
- Separar la estética de la ingeniería.
- Expresar los diferentes aspectos de la arquitectura de una manera apropiada.
- Realizar un análisis de la dependencia y la coherencia.

El concepto de Arquitectura de Construcción que emplean (D. E. Perry & A. L. Wolf 1992) es el de la definición estándar: "El arte o ciencia de la construcción: especialmente diseñando y construyendo estructuras habitables". Aunque señalan que, quizá, es más relevante para sus necesidades una definición secundaria: "Una forma o estructura unificada o coherente". Esta última definición contiene el sentido de la arquitectura que infunde el modelo de SA de estos dos autores:

$$ARQUITECTURA\ SOFTWARE = \{Elementos,\ Estructura,\ Rationale\}$$

Este modelo considera a la *Arquitectura Software* como un conjunto de *elementos arquitectónicos*, que siguen una *estructura* determinada, de acuerdo a una motivación inicial (*rationale*). Además, se distinguen tres clases diferentes de elementos arquitectónicos: elementos de *proceso*, elementos de *datos* y elementos de *conexión*. Estos elementos se describirán en el apartado 3.2.1.

Estos mismos autores, comentan la gran importancia del *Estilo Arquitectónico* en una Arquitectura Software, el cual tiene como objetivo primordial encapsular las decisiones relevantes sobre los elementos arquitectónicos, y enfatizar las restricciones de los elementos y sus relaciones. El estilo arquitectónico proporciona visibilidad sobre ciertos aspectos de la arquitectura, de manera que las violaciones sobre dichos aspectos y la insensibilidad a ellos se hacen más obvias. Este concepto se presenta en mayor detalle en el apartado 3.2.2.

3.2.1 Elementos arquitectónicos

A pesar de la amplia gama de *Lenguajes de Descripción Arquitectónicos* (*Architecture Description Language*, ADL) definidos hasta la fecha (Nenad Medvidovic & Taylor 2000), hay algunos conceptos que resultan transversales a todos ellos. A continuación, y para facilitar una mejor comprensión de este trabajo, se van a presentar los conceptos más relevantes, es decir, aquellos elementos arquitectónicos que aparecen en el modelo arquitectónico.

3.2.1.1 Componentes

(Szyperski 1997) define el componente software como "una unidad de composición con interfaces especificadas contractualmente y dependencias de contexto explícitas, por tanto, un componente software puede ser desplegado tanto de forma independiente como compuesto con terceros". Así, podemos ver que los *componentes* (*components*) de un sistema son usualmente tratados como "cajas negras" sobre las que no se conoce casi nada, excepto la forma en la que se conectan con otros elementos arquitectónicos. Son la base para descomponer en módulos la funcionalidad de un sistema con un alto nivel de encapsulación. Esto es debido a que su interfaz está bien definida, describiendo claramente el servicio que solicita y/o presta.

3.2.1.2 Conectores

(Shaw 1994) describe a los conectores como "...el lugar de las relaciones entre componentes. Median las interacciones, pero no son "cosas" para ser conectadas (son, más bien, los que conectan). Cada conector tiene una especificación de protocolo que define sus propiedades. Estas propiedades incluyen reglas sobre los tipos de interfaces que son capaces de mediar, garantías sobre las propiedades de la interacción, reglas sobre el orden en el que ocurren las cosas, y compromisos sobre la interacción, como el rendimiento, etc.". Tal y como se deduce de la definición anterior, los *conectores* (*connectors*) están definidos en términos de la interacción entre componentes. De una forma similar a los componentes, los conectores interactúan con las otras partes del sistema por medio de las interfaces que describen los servicios que solicitan y/o prestan. Son los encargados de coordinar el proceso de los componentes que conectan, es decir, facilitan la separación de dos cometidos: el cálculo realizado por los componentes y la coordinación proporcionada por los conectores. Por esta razón, proporcionan a los componentes un acoplamiento flexible y mejoran su reutilización en sistemas diferentes.

3.2.1.3 Sistemas

Es frecuente, tal y como indica (Navarro 2007), que se deba proporcionar un nivel de abstracción mayor a fin de facilitar la comprensibilidad y la especificación de la descripción arquitectónica. Por esta razón, los mecanismos para describir los elementos arquitectónicos con diferente nivel de granularidad son siempre deseables. La mayoría de los ADLs han introducido la noción de *sistema* (*system*) como un componente complejo, es decir, un componente que está compuesto de otros elementos arquitectónicos. Esto facilita que el sistema a construir pueda ser descrito de una manera jerárquica. De esta manera, la especificación de la SA puede ser definida de una forma sencilla, facilitando la reutilización y la modularidad.

3.2.1.4 Puertos

Normalmente, indica (Navarro 2007), cada elemento arquitectónico posee interfaces. Una interfaz especifica el servicio, o conjunto de servicios, que proporciona y/o solicita. Las interfaces se utilizan a menudo para tipar los *puertos* (*ports*) de los elementos arquitectónicos. Un puerto actúa como el punto de interacción entre un elemento arquitectónico y el resto de la SA, siendo por tanto el encargado de preservar la vista de caja negra de cada elemento arquitectónico.

3.2.1.5 Conexiones

Las *conexiones* (*conections*), comenta (Navarro 2007), son utilizadas para limitar cuándo se permite una interacción entre los elementos arquitectónicos. Esto es así porque establecen el canal de comunicación entre los elementos arquitectónicos, conectando los puertos de los componentes y los puertos de los conectores. Sin embargo, si los conectores no son considerados como "ciudadanos de primera" en la especificación, entonces las conexiones sólo están establecidas entre componentes. Por lo general, estas conexiones se denominan *relaciones* (*attachments*).

3.2.1.6 Composiciones

Este tipo de relación se establece para permitir la comunicación entre los sistemas y los elementos arquitectónicos que lo componen (Navarro 2007). De esta manera, los elementos arquitectónicos con diferente nivel de granularidad son conectados, proporcionando una semántica composicional, según lo establecido. Por este motivo las *conexiones* no son usadas para este propósito. Estas relaciones, usualmente, reciben el nombre de *enlace* (*binding*).

3.2.1.7 Configuración

Adicionalmente, indica (Navarro 2007), la interconexión entre estos elementos tiene que ser incorporada para describir la estructura del sistema. Normalmente, esta estructura es conocida como *configuración* (*configuration*). Su definición es clave para determinar cómo será construido el sistema. El *Estilo Arquitectónico* (*Architectural Style*) proporciona cierta ayuda para definir esta topología, y seleccionar los elementos involucrados (véase apartado 3.2.2).

Además de todos estos conceptos, asegura (Navarro 2007), existen otros relacionados con la SA que pueden ser definidos también aquí. Por ejemplo, el concepto de *vista* (*view*), introducido por primera vez por (D. E. Perry & A. L. Wolf 1992), que ofrece al analista la facilidad de analizar una Arquitectura Software desde diferentes puntos de vista. También son relevantes los términos de *propiedad* (*property*) y *restricción* (*constraint*), usados para describir las semánticas asociadas a los elementos arquitectónicos o las restricciones de diseño, respectivamente.

3.2.2 Estilos arquitectónicos

La explotación de los *Patrones Software* (*Software Patterns*) es una tendencia actual en Ingeniería del Software. Representa una manera útil de reutilizar el conocimiento en el desarrollo software, especialmente, cuando el analista no tiene una base adecuada en el área. Por esta razón, su introducción implica una ventaja significativa en términos de costes, tiempo y calidad del producto final.

Los patrones han sido definidos en varios niveles. La propuesta más conocida en este sentido ha sido presentada por (Gamma et al. 1995). Describen un conjunto de patrones, a nivel de diseño, para gestionar la creación del objeto, componer objetos en grandes estructuras, y asignar responsabilidades a dichos objetos, en el contexto de los sistemas orientados a objetos.

Sin embargo, ¿cómo aplicar esta reutilización del conocimiento al especificar la Arquitectura Software? Pues bien, existen varias alternativas que exponen las ventajas de esta aplicación en esta etapa. Entre ellas, el uso de *Estilos Arquitectónicos* (*Architectural Styles*) es una aproximación útil y ampliamente extendida. (D. Garlan & Shaw 1993) los describen como sigue: "Un Estilo Arquitectónico define una familia de sistemas en términos de un patrón de organización estructural. Más concretamente, un Estilo Arquitectónico determina el vocabulario de componentes y conectores que pueden ser usados en instancias de dicho estilo, junto a un conjunto de limitaciones sobre cómo pueden ser combinados. Pueden incluir limitaciones topológicas sobre las descripciones arquitectónicas (por ejemplo, sin ciclos). Otras limitaciones, que tienen que ver con las semánticas de ejecución, pueden formar parte de la definición del estilo". Tal y como se deduce de su definición, los estilso arquitectónicos, ayudan a definir la *configuración* de un *sistema*, y a seleccionar los elementos involucrados. Este mismo concepto se puede encontrar con un nombre diferente: *Patrón Arquitectónico* (*Architectural Pattern*). Ambos términos se refieren a la solución recurrente descrita en el nivel arquitectónico.

No hay un marco común para identificar Estilos Arquitectónicos. Sin embargo, (Bass et al. 2003) han identificado un conjunto de rasgos que pueden ser usados para caracterizar un estilo arquitectónico:

- Un conjunto de tipos de elementos (por ejemplo, repositorio de datos, proceso, procedimiento) que realicen alguna función en tiempo de ejecución.
- Una disposición topológica de dichos elementos, indicando sus interrelaciones.
- Un conjunto de restricciones semánticas.
- Un conjunto de mecanismos de interacción que describa cómo se comunican, se coordinan, o cooperan los elementos, a través de la topología establecida.

Considerando estas características, (Bass et al. 2003) han identificado cinco tipos de Estilos Arquitectónicos, los cuales se describen brevemente a continuación:

- *Centrado en los Datos* (*Data-Centred*). Este tipo de estilo está relacionado con aquellos sistemas que explotan un repositorio central de información, para facilitar la comunicación y sincronización de sus múltiples componentes. Un ejemplo de este tipo es el *Estilo Pizarra* (*Blackboard Style*).
- *Flujo de Datos* (*Data Flow*). Este tipo trata sobre aquellos sistemas centrados en cómo los datos son procesados y transformado mediante el flujo establecido entre los diferentes componentes. *Tubería-y-Filtro* (*Pipe-and-Filter*) es un ejemplo claro de este tipo.
- *Llamada-y-Retorno* (*Call-and-Return*). Este tipo trata sobre cómo los sistemas complejos y heterogéneos deben ser descompuestos en partes que interactúan para facilitar su comprensión e implementación. El ejemplo más conocido y ampliamente usado de este tipo, es el *Estilo de Capas* (*Layered Style*).
- *Componentes Independientes* (*Independent Components*). Este tipo de estilo se centra en los sistemas cuyos componentes individuales intercambian mensajes para realizar el cómputo principal, pero mantienen su independencia. El *Estilo de Evento* (*Event Style*) es un ejemplo de este tipo.
- *Máquina Virtual* (*Virtual Machine*). Este tipo se preocupa sobre cómo ofrecen los sistemas una capa de abstracción que es explotada por la infraestructura de computación. El *Estilo de Intérprete* (*Interpreter Style*) es un ejemplo de este tipo.

Estos estilos han sido definidos de manera genérica, sin considerar un dominio o aplicación particular. Sin embargo, recientemente, otro tipo de estilo está empezando a tener una mayor relevancia, y son las *Arquitecturas Software Específicas de Dominio* (*Domain-Specific Software Architectures*, DSSA) (Mettala & Graham 1992). DSSA está basado en la idea de identificar elementos comunes que son compartidos por una familia de sistemas, de manera que los nuevos sistemas pueden ser creados instanciándolos.

3.3 Introducción a la Gestión del Conocimiento Arquitectónico

Según afirman (D. E. Perry & A. L. Wolf 1992), la especificación de los fundamentos arquitectónicos era un requisito temprano en el campo de la SA, que se remonta a uno de los primeros artículos sobre el tema. Sin embargo, (Navarro & Cuesta 2008) y (Navarro, Cuesta, et al. 2009) afirman que, incluso en los planteamientos iniciales, cuando la arquitectura se consideraba en gran medida como una forma de documentar parte del diseño del sistema, la *rationale* fue pronto olvidada. La razón fue, probablemente, que en esta temprana etapa, la *rationale* fue percibida como documentación textual y desestructurada, y concebida como el tipo de información de proceso que, a menudo, se consideraba como no importante.

Sin embargo, comentan (Navarro & Cuesta 2008) y (Navarro, Cuesta, et al. 2009), conforme los sistemas iban creciendo en tamaño y complejidad, la SA fue tomando mayor relevancia; no sólo como elemento crítico para el diseño, sino como proyecto que ayudaría a proporcionar una perspectiva global del sistema, y a explicar o describir el estado actual durante el proceso de desarrollo de software.

En este contexto, se carecía de una visión clara de porqué la arquitectura era como era. Los resultados de las decisiones de diseño en que se basaba la arquitectura, estaban implícitamente incorporados dentro de la misma, pero las *Decisiones de Diseño* (*Design Decisions*, DDs) no estaban todavía explícitamente documentadas, y de ahí que todas las suposiciones e información sobre el diseño y su evolución, es decir, el *Conocimiento Arquitectónico* (*Architectural Knowledge*, AK), estuvieran finalmente perdidos ("evaporados"). Esta pérdida de la información sobre las decisiones de diseño, decían (Jansen & Bosch 2005), conduce a una serie de problemas asociados a la SA:

- Las decisiones de diseño son transversales al sistema y se entrelazan entre sí.
- Las reglas y limitaciones de diseño son con frecuencia violadas debido a su desconocimiento.
- Las decisiones de diseño obsoletas no son eliminadas al no haber sido correctamente especificadas.

Como consecuencia de estos problemas, los sistemas desarrollados tenían un alto coste de cambio, y tendían a erosionarse rápidamente. Además, la reusabilidad de los componentes arquitectónicos era limitada si el conocimiento sobre las decisiones de diseño se evaporaba durante el propio proceso de diseño. Dichos problemas estaban causados por centrar la atención, durante el proceso de diseño de la SA, en los componentes resultantes, en lugar de considerar las decisiones que conducían a éstos. Aunque los efectos de la toma de decisiones estaban presentes en el diseño, las propias decisiones no eran visibles.

Recientemente, (Lago & Van Vliet 2005) estuvieron entre los primeros autores que enfatizaron la necesidad de recuperar y mantener dicha información. Su propósito no estaba específicamente limitado al nivel arquitectónico, sino que intentaban plasmar cada *suposición de diseño*, un término en el que englobaban las DDs arquitectónicas actuales. (Jansen & Bosch 2005) también veían la SA desde una nueva perspectiva, definiendo ésta como "la composición de un conjunto de decisiones de diseño arquitectónico". Siguiendo esta línea, la SA sería el resultado de las decisiones de diseño arquitectónico realizadas a lo largo del proceso de desarrollo. Al igual que estos autores, (Tyree & Akerman 2005) también comentaban que, en la mayoría de los procesos de desarrollo de la arquitectura, las decisiones no eran documentadas explícitamente, sino que estaban implícitas en los modelos que construye el arquitecto. Esta nueva perspectiva sobre la SA, reduciría la evaporización del conocimiento de la información sobre las decisiones de diseño, ya que, hoy día, éstas se han convertido en una parte explícita de la arquitectura. Por tanto, se puede concluir que, claramente, las decisiones de diseño deben representarse y reflejarse en los diseños de Arquitectura Software.

Siguiendo esta línea, (Jansen & Bosch 2005) definen lo que es para ellos una *Decisión de Diseño Arquitectónico* (*Architectural Design Decision*, ADD): "una descripción de un conjunto de añadiduras, substracciones y modificaciones de la arquitectura software, la rationale, y las reglas de diseño, limitaciones de diseño y otros requisitos adicionales que, parcialmente, satisfacen uno o más requisitos de una arquitectura dada". (Tyree & Akerman 2005) también se cuestionan cómo definir e identificar las *Decisiones Arquitectónicas* (*Architectural Decisions*). Comentan que, para probar la importancia arquitectónica de una decisión, un arquitecto debería hacerse la siguiente pregunta: ¿afecta esta decisión a una o más cualidades del sistema (rendimiento, disponibilidad, modificabilidad, seguridad, y así sucesivamente)? Si es así, entonces se debería tener en cuenta esta decisión, y documentarla completa y correctamente.

Por tanto, afirman (Jansen & Bosch 2005), si definimos la SA como un conjunto de ADDs, esto implicará dar un paso adelante en la resolución de los problemas relacionados con su especificación mencionados en el apartado 3.2. Por lo que se ayudará al arquitecto de la siguiente forma:

- Guardando la integridad conceptual de la SA.
- Permitirá realizar una exploración explícita del espacio de diseño, ayuda al arquitecto a prevenir errores evidentes.
- Se podrán desarrollar técnicas de análisis de la SA y de su proceso de diseño.
- Se mejorará la trazabilidad de las decisiones de diseño y sus relaciones con características, aspectos de diseño, intereses y entre ellas mismas.

Sin embargo, (Jansen & Bosch 2005) indican que hay ciertos requisitos que deben ser satisfechos para dar este gran paso:

- se debe cumplir que las *decisiones de diseño arquitectónico* sean de *primera clase* (*first class architectural design decisions*), de tal forma que se pueda describir una SA como un conjunto de decisiones de diseño;
- debe haber *cambios arquitectónicos explícitos* desde la conexión entre las entidades arquitectónicas de primera clase y las decisiones de diseño arquitectónico;
- debe existir un *soporte para la modificación, substracción y adición* de cambios, para tener suficiente expresividad;
- debe existir una *relación clara y bilateral* entre la SA y su realización;
- y los *conceptos arquitectónicos* deben ser de *primera clase*.

Según comentan (Tyree & Akerman 2005), los clientes quieren tener una clara comprensión sobre los cambios que deben ocurrir en el entorno, y se debe asegurar que la arquitectura se adapte a sus necesidades de negocio. Otros arquitectos quieren tener una clara y destacada comprensión sobre los aspectos clave de la arquitectura, incluyendo la *rationale* y las opciones consideradas por el arquitecto original.

Los enfoques arquitectónicos tradicionales, como el *Modelo de Referencia para el Procesamiento Distribuido Abierto* (*Reference Model for Open Distributed Processing*, RM-ODP) (Putman 2001), 4+1 (Kruchten 1995), o el *Proceso Racional Unificado* (*Rational Unified Process*, RUP) (Kruchten 2000), no satisfacen estos requisitos de una manera clara y sencilla. Estas aproximaciones fracasan en varios puntos, como la comunicación de los cambios, de las implicaciones, de la rationale y de las opciones; la facilidad de trazabilidad; así como en proporcionar una documentación ágil.

Para abordar estos problemas, (Tyree & Akerman 2005) creyeron que el enlace que faltaba era considerar como parte inherente del proceso las decisiones tomadas sobre la SA. Si elevamos las decisiones arquitectónicas a elemento de primera clase de la SA y a documento explícito, y las socializamos, se convertirán en una herramienta efectiva para ayudar a los involucrados en el proceso de desarrollo a entender la arquitectura. De esta forma, si un arquitecto no tiene tiempo para nada más, estas decisiones pueden proporcionar una dirección concreta para la implementación, y servir como una herramienta efectiva para la comunicación con los clientes y la administración.

A modo de resumen, y tal y como resaltan (Jansen & Bosch 2005), las decisiones de diseño arquitectónico juegan un papel importante en cuanto al diseño, desarrollo, integración, evolución, y reutilización de las arquitecturas software. (Tyree & Akerman 2005), a su vez, comentan que un documento sencillo describiendo las decisiones arquitectónicas clave, puede recorrer un largo camino para desmitificar las arquitecturas del pasado y del futuro. Además, aconsejan utilizar una plantilla para las decisiones arquitectónicas, ya que resulta de gran utilidad el proporcionar un lenguaje común para discutir las decisiones.

A continuación, se describen algunos de los marcos de especificación de AK, y cómo es posible interconectar dicho AK.

3.3.1 Describiendo el Conocimiento Arquitectónico

Como ya se ha comentado, la especificación de la rationale arquitectónica se ha convertido en un requisito temprano en el ámbito de la SA, remontándonos a uno de los primeros artículos sobre el tema (D. E. Perry & A. L. Wolf 1992). De esta forma, cuando el AK se empieza a documentar explícitamente, surgen distintos marcos de especificación para describir dicho conocimiento. A continuación, se presentan algunos de los marcos de especificación de AK más significativos.

(Lago & Van Vliet 2005) fueron de los primeros autores en destacar la importancia y la necesidad de mantener y recuperar este tipo de información, proponiendo una solución que no estaba limitada al nivel arquitectónico. De esta forma, incluían cada *suposición de diseño*, es decir, las ADDs actuales, como un módulo específico atómico de primera clase, además de hacer uso de la *trazabilidad* para recuperar esta información. Sin embargo, esta propuesta nunca fue diseñada para proporcionar una estructura a las ADRs complejas.

En este contexto, (Bosch 2004) fue el que comenzó a interesarse más por este tema. Decidió concretar el enfoque al contexto de la SA y sugirió explícitamente que el mantenimiento de esta información se tratara como *decisiones de diseño de primera clase*, introduciendo dicho término, ahora convertido en un concepto específico del área. Posteriormente, este mismo autor, junto a otros (van der Ven et al. 2006), observaron las consecuencias de esta elección prematura: separando la rationale en pequeñas porciones, la estructura resultante también se puede considerar como una red que relaciona toda la SA con el nivel más alto de la rationale, en una aproximación que puede ayudar a resolver ciertos problemas complejos.

Además, (Jansen & Bosch 2005) relacionaron esta idea con el proceso arquitectónico, considerando a éste como la evolución de un conjunto de decisiones de diseño. Asimismo, describieron un primer bosquejo sobre la estructura interna de una decisión de diseño, la cual estaría compuesta por requisitos, la rationale concreta, y un conjunto de restricciones y reglas de diseño definido libremente.

Algunos estudios iniciales proporcionaron sugerencias a esta estructura, como el de (Tang et al. 2006), que definía un marco para capturar AK, expresando las similitudes entre diferentes enfoques, y proporcionando un primer intento para reconciliarlos; o el de (Tyree & Akerman 2005), que proponía una *ontología* (o conjunto de conceptos) desde un punto de vista industrial temprano.

También el trabajo reciente de (de Boer et al. 2007) ha definido un *modelo central* (*core model*), señalando los puntos comunes entre las diferentes estrategias existentes para describir estas decisiones de diseño, y revelando un cierto alineamiento entre ellas. Por otro lado, una ontología ligeramente diferente es la que proponen (Kruchten et al. 2006), la cual divide el AK en tres elementos: *decisiones*, *suposiciones* y *contexto*.

Cabe destacar que uno de los principales intereses en el área es que estos enfoques tengan soporte en herramientas. De esta forma, existen principalmente dos aproximaciones para desarrollar herramientas que soporten gestión de AK. La primera de ellas consiste en describir las decisiones de diseño con algunas de las estructuras ya mencionadas y, posteriormente, enlazarlas con el modelo arquitectónico para proporcionar una perspectiva integrada (Jansen & Bosch 2005). La segunda aproximación también describe estas ADDs, pero se interesa más por la definición de plataformas, implementando estrategias para compartir este conocimiento entre los actores involucrados en el proceso de desarrollo. En este sentido, una de las propuestas más evolucionadas es EAGLE (Farenhorst, Lago & Van Vliet 2007). Una propuesta reciente ha dado pie a una tercera aproximación, la cual permite describir estrategias para recuperar el AK cuando éste no se proporciona inicialmente, abriendo así el potencial para reutilizar y/o integrar diseño heredados (Jansen, Bosch, et al. 2008).

La investigación actual también se centra en actualizar el concepto original y adaptarlo a diversos enfoques de ingeniería del software venideros y de vanguardia. Por ejemplo, (Falessi et al. 2008) presentan un enfoque inspirado en la *ingeniería del software basada en valores* (*value-based software engineering*) que permite decidir qué atributos realmente añaden valor a la definición de las ADDs. (Garcia et al. 2006) presentan otra propuesta que utiliza un *enfoque orientado a aspectos* (*aspect-oriented approach*) (Elrad et al. 2001) para abordar la descripción del AK, es decir, describir las ADDs como un aspecto separado. Una propuesta similar a la de Garcia et al. es ATRIUM (Navarro & Cuesta 2008) (véase sección 3.4.2.4), dado que también se basa en aspectos. Sin embargo, en ATRIUM los aspectos no se utilizan para describir ADDs, sino para otras cuestiones, aunque el proceso en sí mismo si puede considerarse orientado a aspectos.

(Harrison et al. 2007) utilizan patrones para capturar las ADDs. Estos autores argumentan que son suficientemente similares para permitir la definición de una relación directa, proponiendo también la reutilización de la estructura clásica de patrones para describir ADDs. Por otro lado, (Falessi et al. 2006) proponen un *modelo de decisión* dirigido por escenarios y basado en objetivos que permite elegir entre distintas alternativas y documentar la rationale de diseño correspondiente.

(Zhu & Gorton 2007) presentan una propuesta similar a ATRIUM, en la cual definen un perfil UML y lo relacionan con un modelo de requisitos no funcionales. Sin embargo, se diferencian en que, en el caso de ATRIUM, los requisitos sólo están asociados con ADDs, como parte del propio proceso de diseño. Otra propuesta que presenta similitudes con ATRIUM es la de (Sinnema et al. 2006), que sugiere la explotación de la variabilidad en la SA para soportar la definición de ADDs. Esto se soporta con ayuda de

su principal propuesta: COVAMOF, un marco y lenguaje para modelar la variabilidad en familias de productos.

Finalmente, la propuesta de (Mattsson et al. 2007) también es parecida a ATRIUM, y proporciona un marco arquitectónico básico. Estos autores proponen, además, la formalización de reglas de diseño, las cuales deberían ser aplicadas por el proceso MDD.

3.3.2 Interconectando el Conocimiento Arquitectónico

Según afirman (Navarro & Cuesta 2008) y (Navarro, Cuesta, et al. 2009), la forma más natural de pensar en una decisión es considerarla de forma aislada, separada del resto del sistema por un proceso de abstracción. Es la manera más fácil de documentarla, sobre todo cuando se proporciona como plantilla, definiendo sus atributos básicos. Un conjunto de dichas decisiones aisladas proporcionarían conocimiento básico sobre el diseño del sistema, cuando operaran en el mismo *sub-estado* (*substrate*), denominado *arquitectura del sistema*. Sin embargo, una vista así es necesariamente incompleta y parcial.

(Navarro & Cuesta 2008) y (Navarro, Cuesta, et al. 2009) señalan que las decisiones de diseño están conectadas, porque hacen referencia unas a otras, interactúan con y se afectan entre sí. De hecho, existe una compleja fábrica de relaciones rodeándolas. Primero, cada decisión se elige a propósito: puede ser rastreada hacia atrás, hacia algún objetivo que alcance, o los requisitos que satisfaga. Segundo, cada decisión es implementada por algún artefacto de diseño, algún elemento arquitectónico. Aparte de estas dos relaciones, cualquier decisión se relaciona con otras decisiones al mismo nivel, de formas diferentes. De esta forma, todo este "entresijo" de decisiones de diseño y sus relaciones con otros elementos del sistema es lo que se denomina *aspectos del conocimiento* (*knowledge assets*).

Las *Decisiones de Diseño* y *Rationales* pueden ser consideradas correctamente como las bazas básicas, que juntas describen nuestro conocimiento de la arquitectura (Navarro, Cuesta, et al. 2009). Sin embargo, para proporcionar una justificación coherente de todo el sistema, estas decisiones tienen que ir complementadas con la información de sus interrelaciones y conexiones mutuas. Por tanto, estas *Relaciones de Conocimiento Arquitectónico* (*Architectural Knowledge Relationships*, AKRs) se presentan como otro valioso elemento que ayuda a transformar el conjunto de decisiones de diseño en una red de conocimiento arquitectónico.

Algunos autores, como (Jansen & Bosch 2005) y (Jansen, Bosch, et al. 2008), describen la *red de decisión* usando una relación única (de dependencia), o quizá varias asimiladas. Incluso esta simple relación es útil y mucho más que un simple "conjunto" de decisiones, ya que muchos detalles dependen de la topología de la red; pero estos enlaces uniformes pierden una característica esencial: la dirección de la relación semántica. De hecho, hay relaciones positivas y negativas, que exponen las sinergias y divergencias dentro del diseño, respectivamente. Es por tanto obvio que su influencia se extiende a toda la arquitectura.

Muchos autores, (Akerman & Tyree 2006), (Erfanian & Shams Aliee 2008), (Farenhorst & de Boer 2006), (Kruchten et al. 2006), (Kruchten 2004) y (Babu T et al. 2007), reconocen la complejidad inherente a la gestión y combinación de las decisiones de diseño y, por tanto, abogan por un *enfoque ontológico* (*ontological approach*) para proporcionar una base sólida a este razonamiento. Estas ontologías son capaces de identificar categorías de DDs (como *ontocrises*, *diacrises* y *pericrises* (Kruchten 2004), que versan sobre los conceptos, características concretas o restricciones a todo el sistema, respectivamente), y de enumerar las propiedades básicas que describen el conocimiento contenido en las DDs y DRs. Y, por supuesto, también ayudan a definir las relaciones básicas entre las DDs y DRs, e incluso describen un metamodelo más completo para su especificación.

Sin embargo, la mayoría de las propuestas en este ámbito, incluso las inspiradas en un enfoque metodológico, se han centrado en la descripción de la estructura (interna) de las DDs. Esto incluye, además de las citas del párrafo anterior, muchas otras como (Jansen, Bosch, et al. 2008), (Tang & Han 2005) o (Zhu & Gorton 2007). Algunas de ellas se ocupan específicamente de la documentación como motivación para la descripción de una plantilla (Harrison et al. 2007); otras soportan su definición sobre una fuerte base empírica (Harrison et al. 2007); e incluso algunas de ellas hacen ambas cosas (Tyree & Akerman 2005); pero para la mayoría, las relaciones juegan un papel secundario, sin ningún papel en relevante.

La Tabla 1 resume la mayoría de los enfoques existentes sobre AKRs, proporcionando alguna equivalencia terminológica y algunas referencias sobre su definición o uso. (Kruchten 2004) proporciona la referencia más completa acerca de este tema. No sólo proporciona la definición de la mayoría de las AKRs existentes, sino también la manera en que se relacionan entre sí. Otras referencias, como (Akerman & Tyree 2006), (Farenhorst & de Boer 2006) o (Kruchten 2004), básicamente utilizan el mismo marco ontológico.

Tabla 1. Evaluación de AKRs (Navarro, Cuesta, et al. 2009)

Relación	Sinónimos	Referencias
Constrains	Implies, Refines	(Kruchten 2004)(Erfanian & Shams Aliee 2008)(Farenhorst & de Boer 2006)(Jansen & Bosch 2005)(Kruchten et al. 2006)
Forbids	Excludes	(Kruchten 2004)(Farenhorst & de Boer 2006)(Kruchten et al. 2006)
Enables		(Kruchten 2004)(Farenhorst & de Boer 2006)(Kruchten et al. 2006)
Subsumes	(**)	(Kruchten 2004)(Farenhorst & de Boer 2006)(Kruchten et al. 2006)
Conflicts with	(*)	(Kruchten 2004)(Farenhorst & de Boer 2006) (Kruchten et al. 2006)
Overrides		(Kruchten 2004) (Farenhorst & de Boer 2006)(Kruchten et al. 2006)
Comprises	Made of	(Kruchten 2004)(Farenhorst & de Boer 2006)(Kruchten et al. 2006)(Tang & Han 2005)
Bound To		(Kruchten 2004)(Farenhorst & de Boer 2006)(Kruchten et al. 2006)
Alternative	Alternate DD	(Kruchten 2004)(Erfanian & Shams Aliee 2008)(Farenhorst & de Boer 2006)(Kruchten et al. 2006)(Tang & Han 2005)
Related To		(Kruchten 2004)(Farenhorst & de Boer 2006)(Harrison et al. 2007)(Kruchten et al. 2006)
Traces From/To	Addresses, Implements (*)	(Kruchten 2004)(Erfanian & Shams Aliee 2008)(Farenhorst & de Boer 2006)(Harrison et al. 2007)(Kruchten et al. 2006)(Tang & Han 2005)(Tang et al. 2005)
Not Complies		(Kruchten 2004)(Farenhorst & de Boer 2006)(Kruchten et al. 2006)
Depends on	(**)	(Jansen & Bosch 2005)(Kruchten et al. 2006)

Entre las AKRs identificadas en la tabla anterior, las más populares son *Constrain*, que expresa la como una DD ayuda positivamente a la consecución de otra DD, y la *Alternative*, probablemente la más citada. Esta última relación proporciona la expresividad necesaria para describir una elección, evitando así la pérdida del conocimiento relacionado con ese proceso de elección en el proceso de desarrollo. (Tyree & Akerman 2005). Este tipo de relación es muy útil en el contexto de las *Líneas de Producto* (Sinnema & Deelstra 2007). Aunque los nombres de estas relaciones deben ser entendidos en términos de este contexto particular, es obvio que algunos de ellos pueden ser considerados dentro de un ámbito más amplio; éstos han sido marcados con un asterisco (*) en la Tabla 1. Estas relaciones se pueden ver como *"extendidas"* (*"extended"*), las cuales no serían AKRs estrictamente, ya que son capaces de relacionar elementos que no son DDs, según comentan (Navarro, Cuesta, et al. 2009).

(Navarro, Cuesta, et al. 2009) destacan que las relaciones marcadas con (**), están en una situación similar a las marcadas con (*), pero la diferencia en este punto es la generalización. Tanto la relación *Depends on* como *Subsumes*, pueden ser consideradas como versiones genéricas del resto. Como en otros muchos contextos, la relación de dependencia puede ser considerada como la relación básica, por definición, de manera que el resto de relaciones hereden de ella. Por tanto, cada AKR es también una dependencia, según afirman (Jansen, Bosch, et al. 2008). Por otra parte, la relación de *Subsumes* es normalmente considerada como la relación objetivo para las ontologías, según comentan (Baader et al. 2003), actuando como el cierre transitivo para las relaciones ontológicas. De hecho, puede ser utilizada para "aplanar" una estructura con varias AKRs en un único modelo con una sola relación facilitando la aplicación de un análisis básico.

Considerando todo lo anterior, (Navarro, Cuesta, et al. 2009) concluyen que las AKRs proporcionan un marco rico para capturar el conocimiento arquitectónico. Como ya se comentó anteriormente, esto nos permite capturar y representar dicho conocimiento con un número más reducido y sencillo de DDs.

3.4 AK Dirigido por Modelos

(Könemann & Zimmermann 2010) afirman que, hoy en día, el desarrollo de sistemas software se realiza de forma distribuida en equipos de trabajo, y los *modelos* mejoran la comunicación entre dichos equipos y ayudan a desarrollar sus sistemas. De esta forma, el *Desarrollo Software Dirigido por Modelos* (*Model-Driven Development*, MDD) (Object Management Group 2003) mejora la productividad al permitir utilizar dichos modelos para la documentación, discusión, e incluso para la generación de código. Diferentes trabajos, como (Navarro & Cuesta 2008) y (Hilliard et al. 2010) resaltan la importancia de emplear las ventajas que ofrece este nuevo paradigma en el área de la SA. Por ello, en los siguientes apartados, se presentan algunos de los enfoques más significativos que tratan de abordar el soporte al conocimiento arquitectónico como un proceso dirigido por modelos. Nótese que estas aproximaciones pueden ser de naturaleza genérica o específica de AK, en caso de ser aplicables a diversos tipos de conocimiento (ver apartado 3.4.1) o sólo a AK (ver apartado 3.4.2), respectivamente.

3.4.1 Enfoques genéricos

En este apartado, se presentan algunos de los enfoques genéricos MDD más significativos. Estos enfoques se pueden aplicar a diversos tipos de conocimiento, por lo que, en particular, también se pueden aplicar al Conocimiento Arquitectónico.

3.4.1.1 MEGAF

(Hilliard et al. 2010) presentan una infraestructura, llamada MEGAF, que permite la construcción de marcos arquitectónicos reutilizables e identifican tres posibles tipos de beneficiarios de este trabajo, como los *arquitectos ordinarios* y *sus organizaciones*; los *arquitectos sénior*; y los *investigadores*. MEGAF se basa en el concepto de *Megamodelado* (*Megamodeling*), que fue propuesto con el objetivo de soportar el modelado ampliamente, es decir, tratar con modelos, metamodelos, y sus propiedades y relaciones. Por tanto, el megamodelado ofrece la posibilidad de especificar enlaces semánticos entre modelos (y metamodelos) y navegar entre ellos. MEGAF permite a los arquitectos software crear nuevos marcos arquitectónicos con características tales como: mecanismos para almacenar vistas, puntos de vista, stakeholders y aspectos del sistema; mecanismos para definir correspondencias entre las vistas, puntos de vista, stakeholders, aspectos del sistema e incluso entre elementos arquitectónicos que forman parte de ellos; y chequeos de consistencia y completitud basados en las relaciones arquitectónicas y reglas definidas entre los elementos arquitectónicos.

Observando la Figura 1, se puede decir que MEGAF es un repositorio extensible de *puntos de vista* (*viewpoints*), *vistas* (*views*), *tipos de modelo* (*model kinds*), *modelos arquitectónicos* (*architecture models*), *aspectos del sistema* (*system concerns*) y *stakeholders*. Además, se pueden crear correspondencias y reglas de correspondencia entre elementos arbitrarios, que permiten al arquitecto expresar e imponer relaciones, tanto entre varios elementos en una misma descripción arquitectónica, como a través de distintas descripciones arquitectónicas (como para líneas de producto o sistemas de sistemas).

MEGAF, por tanto, proporciona funcionalidades que permiten a los arquitectos software crear su propio framework, seleccionando adecuadamente los artefactos previamente definidos y residentes en MEGAF.

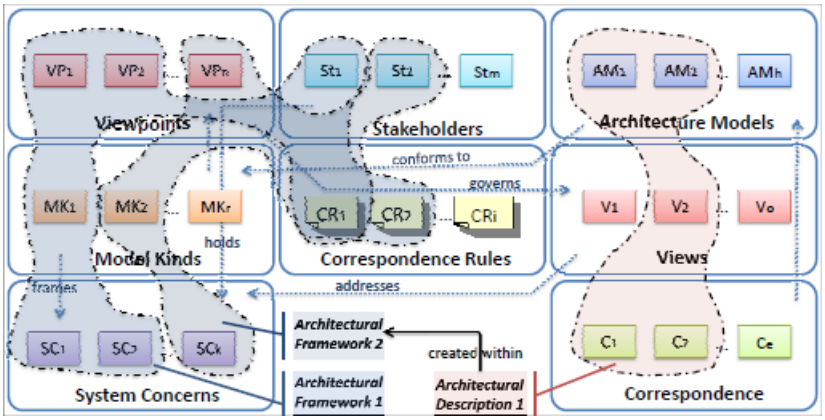


Figura 1. Visión general de MEGAF (Hilliard et al. 2010)

De esta manera, MEGAF podría ser utilizado para almacenar fácilmente el AK del sistema (ADDs y ADRs) en marcos arquitectónicos reutilizables, lo que permitiría definir las relaciones entre decisiones de diseño y sus rationales. Así, estos marcos arquitectónicos se podrían reutilizar en el caso de sistemas con AK similar, lo que conlleva un ahorro importante en cuanto a coste y tiempo.

3.4.1.2 Método de instanciación de patrones de diseño

Actualmente, afirman (Kajsa et al. 2011), los *patrones de diseño* representan una herramienta importante para los desarrolladores en el proceso de construcción de diseño de software, y proporcionan maneras particulares efectivas para mejorar la calidad de los sistemas software.

Hoy día, las herramientas CASE u otras herramientas de modelado proporcionan algún tipo de soporte a la instanciación de patrones de diseño, pero, a menudo, están basadas en copias simples de plantillas de patrones en un modelo con mínimas posibilidades para realizar modificaciones y con mínimo soporte a la integración de instancias en el modelo de aplicación de contexto.

La idea de estos autores enfatiza la colaboración entre el desarrollador y la herramienta CASE. De esta forma, asumen que no necesitan forzar al desarrollador a modelar o marcar explícitamente todos los participantes de patrones. Su objetivo es alentarlos sólo para sugerir la ocurrencia de instancia de patrón, mientras que el resto del proceso de instanciación es automatizado (véase Figura 2).

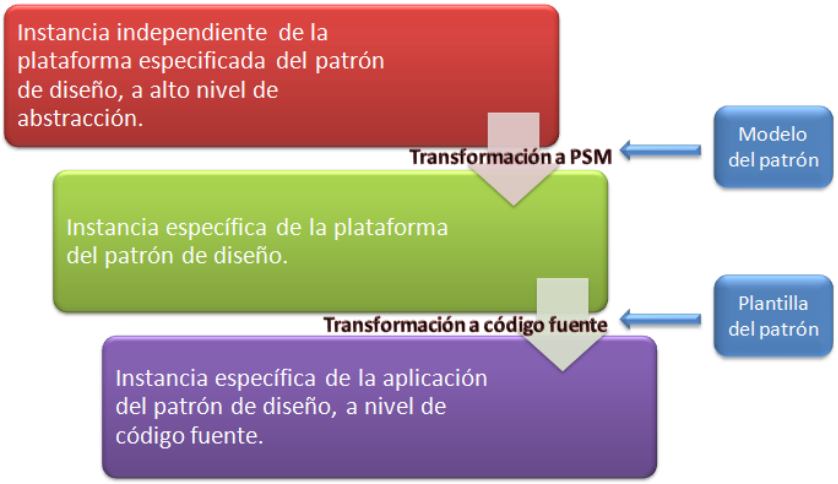


Figura 2. Propuesta del proceso de instanciación de patrones de diseño (Kajsa et al. 2011)

Por tanto, esta idea de utilizar patrones de diseño ayuda al analista a describir fácilmente la SA y su AK, dado que su uso implica la reutilización de una solución de calidad, transmitiendo así más información sobre las decisiones tomadas. De esta manera, los patrones pueden mejorar notablemente la solución propuesta.

3.4.2 Enfoques específicos de AK

En este apartado, se presentan algunos de los enfoques específicos MDD más significativos, aplicados exclusivamente al Conocimiento Arquitectónico.

3.4.2.1 Metamodelo de suposiciones

(Lago & Van Vliet 2005) definen las *suposiciones* (*assumptions*) como las razones para las decisiones de diseño que son, más o menos, arbitrariamente recogidas al vuelo, debido a la experiencia personal, los antecedentes, el conocimiento del dominio, los artefactos reutilizados, etc. Estos autores afirman que, cuando las suposiciones se reflejan de forma explícita, éstas proporcionan una orientación para la reutilización: las suposiciones se expresan de forma natural a nivel mundial. Por tanto, dichas suposiciones encajan perfectamente en la arquitectura software, dado que ayudan a ser conscientes, no sólo de las futuras capacidades evolutivas del sistema previsto, y de las cosas que pueden cambiar, sino también de las cosas que es mejor no cambiar. Así, las suposiciones pueden ser utilizadas como una herramienta para identificar los requisitos que queremos encontrar en un nuevo sistema, y para seleccionar los aspectos reutilizables que las satisfagan, permitiendo que el conocimiento se transfiera de una mejor manera, dando un mejor soporte a la evolución y el mantenimiento. Para este fin, la trazabilidad desde las suposiciones a los aspectos reutilizables, es necesaria. Además, dichas suposiciones se clasifican en tres tipos, en función de su fuente:

- *Suposiciones técnicas* (*Technical assumptions*). Se interesan por el entorno técnico en donde un sistema va a ser ejecutado: lenguajes de programación, sistemas de base de datos, sistemas operativos, software del middleware, etc.
- *Suposiciones organizacionales* (*Organizational assumptions*). Describen la organización como un todo, considerando su entorno social y sus principios.
- *Suposiciones de gestión* (*Managerial assumptions*). Reflejan las decisiones tomadas para alcanzar los objetivos de negocio.

(Lago & Van Vliet 2005) consideran que hay tres usos importantes para el modelado explícito de suposiciones:

- *Trazabilidad.* Se pueden trazar suposiciones a las soluciones de diseño e implementación, y viceversa.
- *Evaluación.* Las evaluaciones de la arquitectura software, a menudo, hacen uso de las solicitudes de cambio o los casos de uso que describen los futuros requisitos potenciales.
- *Gestión del conocimiento.* Se puede modelar explícitamente una arquitectura y sus suposiciones asociadas, de la misma forma que se puede modelar una arquitectura y sus variantes.

En base a estos tres usos del modelado explícito de suposiciones, estos autores propusieron un metamodelo que proporciona la base formal requerida para documentar suposiciones y sus relaciones con los aspectos arquitectónicos. Este metamodelo está representado en la Figura 3 cuya idea principal es la de describir cómo las suposiciones afectan a diferentes características.

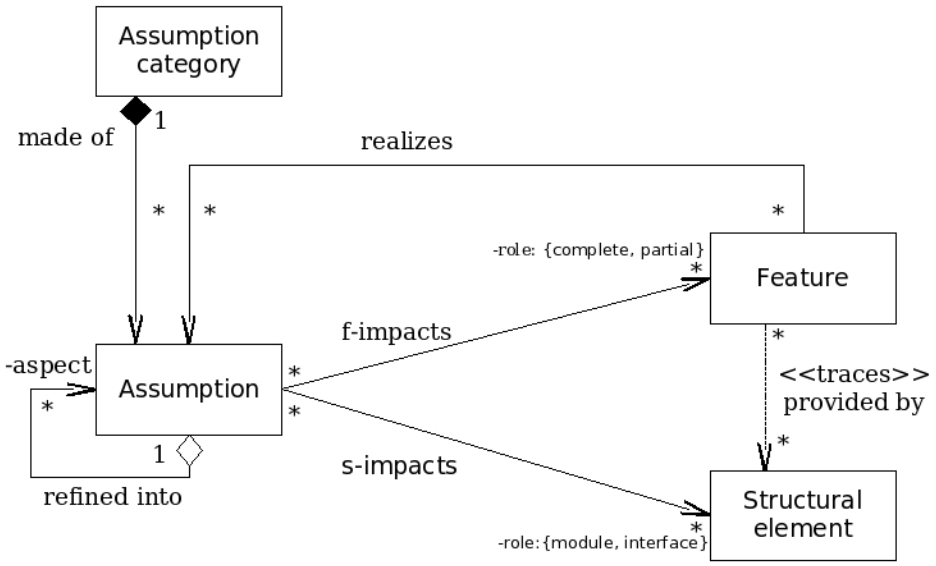


Figura 3. Metamodelo de suposiciones (Lago & Van Vliet 2005)

Una suposición es un aspecto transversal en una estructura objetivo (por ejemplo, un árbol de características o un modelo de componentes). Por tanto, en el árbol de características, los nodos hoja representan características concretas, y los nodos de nivel superior representan características abstractas.

Si nos fijamos en la Figura 3, una suposición dada afecta a una o más características concretas. Sin embargo, la asociación real se realiza con la raíz del árbol más pequeño donde todas sus características concretas se vean afectadas, es decir, una suposición se asocia con la característica abstracta "más pequeña" que cubre a todas las características afectadas. Por otro lado, la relación *realizes* enlaza los nodos hoja del árbol de características que están siendo afectados por una suposición dada.

3.4.2.2 Enlazando decisiones de diseño y modelos de diseño

(Könemann & Zimmermann 2010) comentan que una manera de documentar las decisiones de diseño en proyectos de desarrollo software es utilizar *sistemas de gestión de decisiones* (*decision management systems*). Las decisiones pueden ser específicas de un proyecto particular, o genéricas, y, por tanto, reutilizables en contextos similares. Las decisiones reutilizables, por ejemplo el uso de patrones de diseño para resolver una cuestión de diseño particular, pueden ser almacenadas como las mejores prácticas y reutilizadas en otros proyectos. Esto convierte a las decisiones de diseño en artefactos valiosos para expresar y compartir conocimiento de diseño.

Estos autores afirman que los sistemas de gestión de decisiones sólo son usados para documentar, analizar, y compartir conocimiento de diseño arquitectónico, y están aislados de los modelos actuales utilizados para desarrollo de software basado en modelos. Todas estas herramientas almacenan la información semi-formalmente, es decir, estructurada por decisiones. Las herramientas de modelado actuales, por otro lado, tienen capacidades limitadas o ninguna para documentar las decisiones de diseño. Por tanto, los modelos de diseño formales y las decisiones de diseño semi-formales están separados.

De esta forma, lo que proponen es que se traten las decisiones de diseño como artefactos de primera clase y explotarlas para integrar los modelos de diseño y la documentación semi-formal: un enlace explícito entre los elementos del modelo de diseño y las decisiones de diseño permitirá guardar los modelos de diseño de forma consistente con las decisiones tomadas.

Un metamodelo típico de decisiones de diseño encontrado en trabajo existente se presenta en la Figura 4, el cual puede ser utilizado por múltiples herramientas de gestión de decisiones: una decisión de diseño trata un *problema* de diseño particular (*Issue*), considerando una o más *soluciones* (*Alternatives*), y la *rationale* de porqué se ha seleccionado una alternativa particular (*Outcome*).

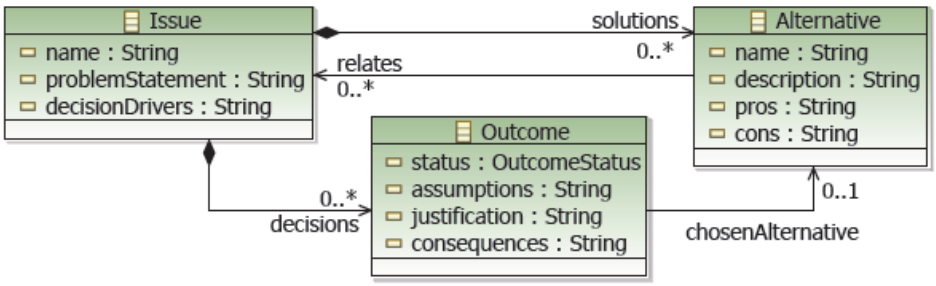


Figura 4. Un metamodelo típico de decisiones de diseño (Könemann & Zimmermann 2010)

3.4.2.3 Metamodelo de dominio UML

(Zimmermann et al. 2009) afirman que la falta de rigor en la captura de decisiones es una fuente posible de problemas de calidad de las arquitecturas software en construcción. Insuficientes incentivos, métodos, y herramientas para compartir decisiones, inhiben la reutilización activa de conocimiento y el intercambio entre profesionales en diferentes proyectos. Por tanto, explican los autores, definir plantillas, metamodelos y/o patrones de referencia, es un buen punto de partida para conseguir una captura de decisiones más sistemática y rigurosa. Sin embargo, esto no acaba con los impedimentos del mundo real, como son la carencia de beneficios no inmediatos, los problemas de presupuesto y planificación, y la falta de herramientas, para conseguir compartir de forma sostenible y mantenible el conocimiento arquitectónico.

El metamodelo sólo debería definir un pequeño conjunto de atributos obligatorios, de tal forma que los profesionales no se vean abrumados con la información a la hora de rellenar y estudiar los modelos de decisión. De esta forma, el metamodelo debe ser una máquina legible y traducible a otras especificaciones.

Estos autores, en su trabajo anterior, derivaron dicho metamodelo de propuestas anteriores y su experiencia práctica en la industria, y definieron tres pasos de procesamiento: identificación de la decisión, toma de decisiones, y aplicación de la decisión (Zimmermann et al. 2009). En la Figura 5, se presenta una versión actualizada de dicho modelo, que utiliza las clases UML para introducir las tres entidades principales: *ADIssue*, *ADAlternative*, y *ADOutcome*. Una instancia ADIssue informa al arquitecto de que se debe resolver un único problema de diseño arquitectónico; las instancias ADAlternative presentan posibles soluciones a dicho problema; y, finalmente, las instancias ADOutcome guardan la decisión tomada actual para resolver el problema, incluyendo su rationale.

De esta manera, estos autores extienden un metamodelo existente, como es el *Lenguaje de Modelado Unificado* (*Unified Modeling Language*, UML), formalizando las estructuras de datos en un modelo de decisiones arquitectónicas.

Por tanto, la aportación principal de (Zimmermann et al. 2009) es la formalización de conceptos en metamodelos y plantillas existentes, y extenderlos para ofrecer soporte a la reutilización y la colaboración.

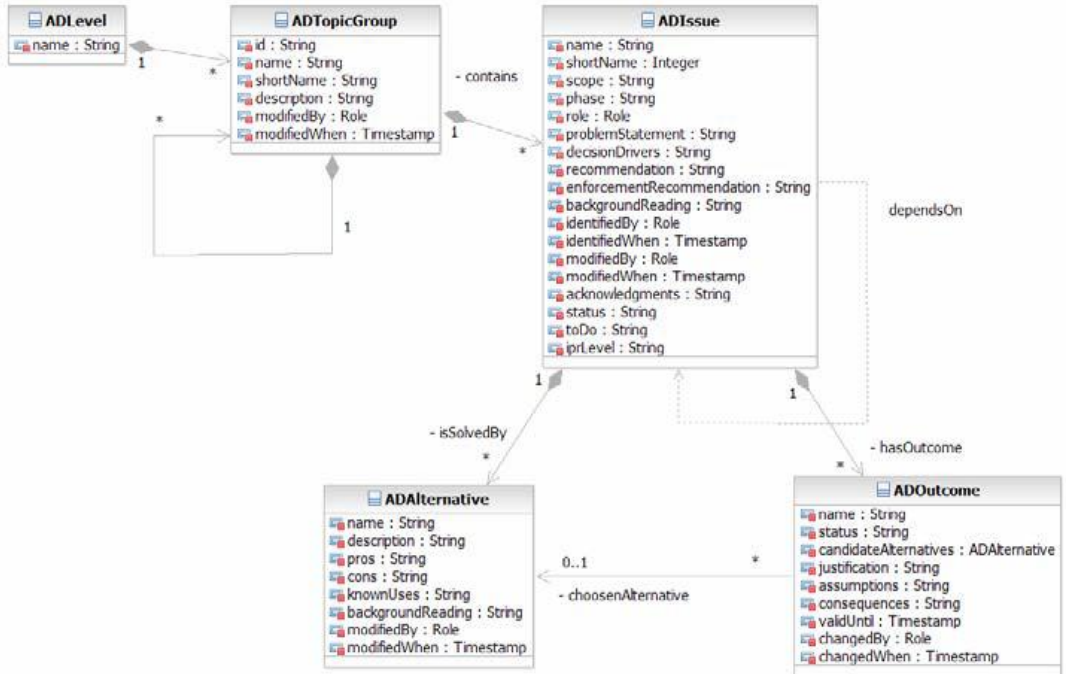


Figura 5. Metamodelo UML para la captura de decisiones arquitectónicas y reutilización (Zimmermann et al. 2009)

3.4.2.4 ATRIUM

(Navarro & Cuesta 2008) indican que hay razones convincentes sobre por qué es necesario incluir trazabilidad desde el principio hasta el final en un proceso de desarrollo software. Para llevar esto a cabo, los autores proponen la automatización de dicha trazabilidad, utilizando un enfoque dirigido por modelos: ATRIUM (*Architecture Traced from RequIrements by applying a Unified Methodology*).

La arquitectura ATRIUM (Navarro 2007), por tanto, fue descrita siguiendo un enfoque MDD, que consta de tres actividades principales, las cuales deben ser iteradas para definir y refinar los distintos modelos, permitiendo al analista discurrir tanto sobre los requisitos como sobre la arquitectura (véase Figura 6). A continuación, se describen brevemente estas actividades (Navarro & Cuesta 2008):

- *Definir objetivos* (*define goals*). Esta actividad permite al analista identificar y especificar los requisitos del sistema, definiendo el *Modelo de Objetivos de ATRIUM* (*ATRIUM Goal Model*), guiando al arquitecto desde los *objetivos* que el sistema debería alcanzar, hasta los *requisitos* que debería satisfacer, y las *operacionalizaciones* que describen las soluciones a los *requisitos* establecidos. Una *operacionalización* (*operationalization*) se define como una descripción de una solución arquitectónica, es decir, una elección de diseño arquitectónica para el sistema en construcción, de tal manera que se satisfagan las necesidades y expectativas de los usuarios. Se denominan operacionalizaciones porque describen el comportamiento del sistema para cumplir los requisitos, tanto funcionales como no funcionales. Por esta razón, cuando se están describiendo estas operacionalizaciones, se incluyen dos atributos clave: decisión de diseño y rationale de diseño. Así, las ADDs y las ADRs se introducen desde el comienzo del proceso de desarrollo software, en la fase de requisitos.
- *Definir escenarios* (*define scenarios*). Esta actividad se centra en identificar el conjunto de escenarios que define el comportamiento del sistema bajo ciertas decisiones arquitectónicas, descritas en el Modelo de Objetivos de ATRIUM como operacionalizaciones.

Sintetizar y transformar (*synthesize and transform*). Esta actividad ha sido definida para generar la proto-arquitectura (Brandozzi & D E Perry 2001) del sistema específico, es decir, sintetizar los elementos arquitectónicos que conforman el sistema, así como la estructura del sistema futuro, desde el Modelo de Escenarios de ATRIUM, obtenido de la etapa anterior.

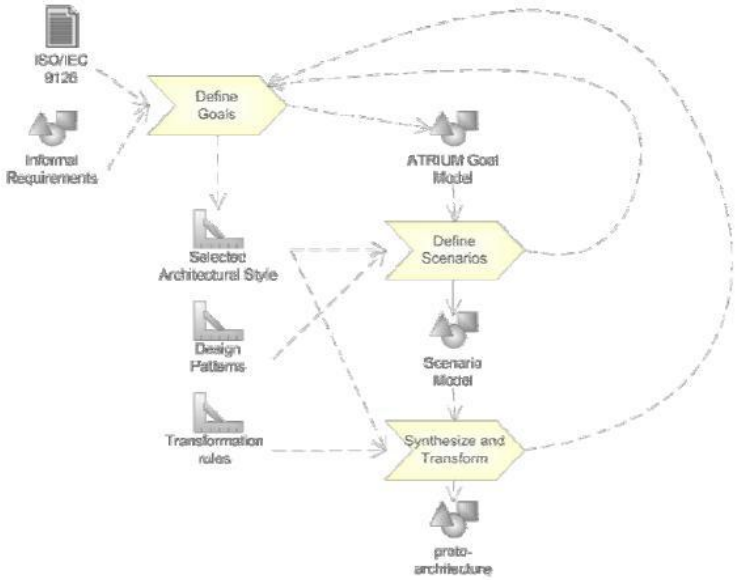


Figura 6. Esquema de ATRIUM (Navarro & Cuesta 2008)

Así, cada fase del proceso de desarrollo de software se describe estableciendo claramente sus metamodelos asociados, junto a los enlaces hacia delante y hacia atrás que existen entre ellos. Para tratar la descripción de las ADDs y las ADRs, esta propuesta explota estos enlaces aplicando ATRIUM por medio de su incorporación en cada metamodelo y el establecimiento de los mecanismos para su propagación a través de niveles de abstracción.

En resumen, esta propuesta de (Navarro & Cuesta 2008) proporciona una metodología para definir y gestionar explícitamente conocimiento arquitectónico, además de enfatizar la importancia de la trazabilidad, dado que proporciona la estructura básica para la rationale arquitectónica.

3.4.3 Conclusiones

Existen diversas alternativas que permiten soportar el AK como un proceso dirigido por modelos. A continuación, en la Tabla 2, se presenta una comparativa entre los distintos enfoques MDD vistos anteriormente, de tal forma que se puedan ver las ventajas y desventajas que presenta cada uno. Estos enfoques se han numerado de la siguiente manera: (I) MEGAF, (II) Instanciación de patrones de diseño, (III) Metamodelo de suposiciones, (IV) Enlazando decisiones de diseño y modelos de diseño, (V) Metamodelo de dominio UML, y (VI) ATRIUM.

Tabla 2. Comparativa - Enfoques MDD para el soporte de AK

Enfoque	Tipo	Basado en	Principales características
I	Genérico	Megamodelado	Permite crear marcos arquitectónicos reutilizables
II	Genérico	Patrones de diseño	Automatiza el proceso de instanciación del patrón de diseño
III	Específico	Suposiciones	Documenta suposiciones y sus relaciones con aspectos arquitectónicos
IV	Específico	Sistemas de gestión de decisiones	Enlaza los sistemas de gestión de decisiones con los modelos de diseño
V	Específico	UML	Extiende UML para soportar reutilización y colaboración
VI	Específico	Automatización de la trazabilidad	Define y gestiona explícitamente el AK

Como se comentó anteriormente, y a la vista de la Tabla 2, estos enfoques pueden ser de tipo *genérico*, es decir, aplicables a cualquier tipo de conocimiento (y, en particular, al AK), o bien *específicos* de AK, es decir, desarrollados exclusivamente para tratar con conocimiento arquitectónico. Es fácil pensar que los enfoques específicos son los más adecuados para tratar con AK, sin embargo, no siempre resultan alternativas más sencillas y satisfactorias que las genéricas. Esto es debido a que, en un enfoque específico, normalmente, se debe estar altamente familiarizado con el dominio y el contexto de aplicación, en este caso, el AK, para poder comprender y saber utilizar los distintos artefactos ofrecidos.

La Tabla 2 también muestra que cada una de las aproximaciones presentadas se basa en conceptos y sistemas diferentes, lo que indica la gran variabilidad existente entre las mismas. Esto nos hace pensar que esta área de investigación está en auge, que todavía queda mucho por hacer y que aún pueden surgir múltiples enfoques más, basados en otras alternativas.

Todos estos enfoques tienen en común una atractiva ventaja: la *reutilización*. Al tener como base un desarrollo dirigido por modelos, esta ventaja viene implícita en cada uno de estos enfoques, dado que es el principal artefacto ofrecido por MDD.

Por tanto, y a modo de resumen, tanto analistas, como desarrolladores, investigadores, arquitectos software o, incluso, clientes y usuarios del sistema final, se ven beneficiados cuando aplican estos enfoques para tratar con el conocimiento arquitectónico, dado que les permite ahorrar tiempo, esfuerzo y costes al tratar con arquitecturas (y AK) reutilizables, mantenibles y capaces de evolucionar.

3.5 Visualización en AK

De un tiempo a esta parte, muchos autores, (Biehl & Torngren 2010), (Kruchten et al. 2009), (Shahin, Liang & Khayyambashi 2010a) y (Shahin, Liang & Khayyambashi 2010b), han expresado la necesidad de capturar y utilizar las decisiones de diseño arquitectónico y la rationale de diseño como entidades de primera clase.

Como comentan (Shahin, Liang & Khayyambashi 2010a), visualizar y explorar las ADDs puede ayudar al entendimiento de las mismas y al razonamiento o motivación que hay detrás de éstas (*rationale de diseño*), especialmente en un entorno de desarrollo colaborativo y distribuido.

De esta manera, a continuación, se describe cuál ha sido la evolución en cuanto a la representación de las arquitecturas software, y cómo dicha representación ha ido, paulatinamente, incluyendo el AK de manera implícita. Seguidamente, se verán las distintas herramientas que permiten visualizar y capturar este AK.

3.5.1 Evolución de la representación arquitectónica

3.5.1.1 Primera fase

(Kruchten 1995) fue uno de los primeros que propuso la utilización de vistas arquitectónicas en su modelo *4+1*, a fin de proporcionar una visión del sistema desde diferentes perspectivas. De forma similar, Siemens (Hofmeister et al. 1999) desarrolló el método *Siemens Four-Views* (S4V), basado en las mejores prácticas para los sistemas industriales.

En 1995, Kruchten, Capilla y Dueñas propusieron vistas que ayudaban a los arquitectos a identificar todos los factores importantes que se podían usar para identificar los principales retos arquitectónicos y desarrollar estrategias de diseño para resolver problemas mediante la aplicación de una o más vistas (Kruchten et al. 2009). En la misma época, el *Instituto de Ingeniería del Software*[1] (*Software Engineering Institute*) propuso una clasificación basada en vistas y tipos de vistas que destacaban la importancia de documentar las decisiones de diseño (Clements et al. 2002).

(Rozanski & E. Woods 2005) definieron hasta seis puntos de vista que clarificaban los aspectos arquitectónicos más importantes o elementos de los sistemas de información que son relevantes para los stakeholders.

De esta forma, a mediados de los '90, la investigación arquitectónica se centraba en la descripción del diseño y el modelado, sin ponerse de acuerdo en cuanto a la notación de la representación arquitectónica.

3.5.1.2 Segunda fase

Según señalan (Kruchten et al. 2009), el periodo entre 1996 y 2006 trajo consigo técnicas complementarias en forma de *métodos arquitectónicos*, donde muchos de ellos derivaban de las prácticas industriales bien establecidas. Entre estos métodos arquitectónicos, se encuentran RUP, desarrollado por IBM; BAPO/CAFCR (método: *Business-Architecture-Process-Organization*, y sus vistas: *Customer*, *Application*, *Functional*, *Conceptual*, y *Realization*), desarrollado por Philips; S4V, comentado antes; ASC (*Architectural Separation of Concerns*), desarrollado por Nokia; ATAM (*Architecture Trade-off Analysis Method*), desarrollado por el Instituto de Ingeniería del Software; SAAM (*Software Architecture Analysis Method*); y ADD (*Attribute-Driven Design*).

Todos estos métodos tienen en común el uso de las decisiones de diseño que son evaluadas durante la construcción de la arquitectura. Desafortunadamente, las decisiones de diseño y su rationale todavía no estaban consideradas como entidades de primera clase, dado que carecían de una representación explícita.

La carencia de una representación de primera clase de la rationale de diseño en los actuales modelos de vistas arquitectónicas, lleva a la necesidad de incluir decisiones como cuestiones de primera clase que deberían estar plasmadas en la documentación arquitectónica tradicional, según afirman (Kruchten et al. 2009).

Uno de los beneficios de utilizar la rationale de diseño en la arquitectura es que evita procesos de recuperación de la arquitectura, que se utilizan mayormente para recuperar decisiones cuando un diseño de la arquitectura, de la documentación, o incluso de los creadores, ya no está disponible.

Por tanto, la fórmula *Conocimiento de la Arquitectura* = *Decisiones de Diseño* + *Diseño*, que propuso (Kruchten 2004), modernizaba la fórmula de (D. E. Perry & A. L. Wolf 1992) *Arquitectura* = {*Elementos, Estructura, Rationale*}, y considera a las decisiones de diseño como parte de la arquitectura.

3.5.1.3 Tercera fase

La investigación activa desde 2004 hasta 2008 ha producido un número significativo de aproximaciones para representar y capturar las ADDs, y ha definido nuevos roles y actividades para soportar la creación y uso de este Conocimiento Arquitectónico, según comentan (Kruchten et al. 2009).

Estos autores señalan que se han realizado dos estudios empíricos que informan sobre el valor de capturar y usar las decisiones de diseño y, además, proporcionan algunos resultados concretos:

- Las decisiones de diseño y rationales, consideradas tipos diferentes de conocimiento para representar y grabar la información de diseño, podrían no tener el mismo valor o importancia para todos los stakeholders. De esta forma, deberíamos decidir qué tipo de conocimiento encajaría mejor con cada tipo de usuario.

[1] http://www.sei.cmu.edu/

- El esfuerzo de capturar decisiones durante las etapas tempranas del desarrollo, realmente merece la pena sólo en fases posteriores de mantenimiento y evolución.

A partir de estos estudios, (Kruchten et al. 2009) proponen una guía para ayudar a los arquitectos a documentar las decisiones en sus arquitecturas:

1. Decidir qué ítems de información son necesarios para cada decisión de diseño, como el nombre de la decisión, descripción, rationale, pros y contras, estado y categoría. Después, decidir qué sistema de representación tratará mejor la grabación y organización de las decisiones, y seleccionar una estrategia para capturar los ítems.
2. Para cada decisión, definir enlaces a los requisitos que las motivan.
3. Si debemos evaluar decisiones alternativas, proporcionar mecanismos para cambiar el estado de la decisión y la categoría.
4. Si una decisión depende de decisiones previas, definir estas relaciones para soportar trazabilidad interna a través de ellas.
5. Una vez escogido un conjunto de decisiones significativas, enlazarlas a la arquitectura que resulta de dichas decisiones. Estos enlaces proporcionan la conexión a las vistas de la arquitectura tradicionales.
6. Después de realizar y capturar todas las decisiones, compartirlas a través de mecanismos de comunicación y documentación.

A continuación, se listan los beneficios que se esperan al capturar y documentar explícitamente las ADDs:

- Las decisiones mejoran la trazabilidad entre los artefactos de la ingeniería del software producidos durante el ciclo de vida del software.
- Capturar las dependencias entre decisiones soporta el análisis de impactos cuando añadimos, modificamos o borramos una decisión.
- Documentar las decisiones facilita nuestro entendimiento general de un sistema.
- Documentar las decisiones facilita la compartición del conocimiento y los procesos de evaluación, dado que los usuarios pueden repasar fácilmente la rationale de las decisiones pasadas.
- El aprendizaje de las actividades puede usar el conocimiento previo para evaluar a los arquitectos de software principiantes en sus carreras profesionales.

3.5.2 Herramientas que soportan la Lógica de Diseño

Existen diversas técnicas de visualización que permiten visualizar la *lógica de diseño* (*design rationale*), y se pueden utilizar durante la captura, representación o mantenimiento del AK. Cabe destacar que una de las suposiciones de este trabajo es que el AK es, per se, una base de conocimiento compuesta por ADDs y ADRs, y sus relaciones correspondientes que pueden ser utilizadas para entender mejor la SA de un sistema.

En este sentido, el AK se asemeja a una ontología (Gruber 1993), como otros autores ya han notado (De Boer et al. 2009). Esto nos permite utilizar como taxonomía de técnicas de visualización aquella propuesta por (Katifori et al. 2007), la cual distingue cinco técnicas de visualización, dependiendo de la forma de presentar la información, el método de interacción, o la funcionalidad soportada.

A continuación, se describen cada uno de estos tipos de representación, junto a sus herramientas disponibles, las cuales pueden clasificarse de acuerdo a dichos tipos. Nótese que nos vamos a centrar en herramientas bidimensionales, dado que son más cercanas a aquellas comúnmente utilizadas por los arquitectos software. Las subsecciones 3.5.2.1, 3.5.2.2 y 3.5.2.3 presentan herramientas de visualización específicas de AK, mientras que las subsecciones 3.5.2.4 y 3.5.2.5 presentan herramientas ontológicas. Hemos incluido las dos últimas subsecciones porque presentan la información de forma jerárquica, muy similar a las representaciones del AK en las herramientas específicas de AK.

3.5.2.1 Indented list

Este tipo de representación presenta el AK como texto plano en una vista de árbol, similar al explorador de Windows. La simplicidad de esta representación textual hace que no sea muy popular hoy día. Un sistema AK que utiliza esta representación es *Jerarquía de Procedimiento de Cuestiones* (*Procedural Hierarchy of Issues*, PHI) (R. McCall 1991), basada en el enfoque de (Kunz & Rittel 1970) para soportar un enfoque argumentativo para el diseño. Además, utiliza un proceso argumentativo adicional y se ocupa de un amplio rango de cuestiones de diseño. Algunas de las herramientas que soportan la metodología PHI son (Regli et

al. 2000): JANUS (Fischer et al. 1990), HOS (*Hyper-Object Substrate*) (Shipman & R. McCall 1997), y PHIDIAS (*Procedural Hierarchy of Issues/Design Intelligence Augmentation*) (Shipman & R. McCall 1997).

Sin embargo, estas herramientas no tienen soporte actualmente, por lo que es interesante considerar también herramientas ontológicas que ofrezcan una vista en árbol de explorador de Windows. Por ejemplo, *Protégé* (N. F. Noy et al. 2000) es un entorno de edición de ontologías y adquisición de conocimiento, donde las clases están representadas como nodos en un árbol indentado, retráctil y expandible, y las instancias se muestran en otra ventana. *KAON* (FZI 2011) es un entorno de gestión de ontologías de código abierto para aplicaciones de negocio, el cual incluye un juego completo de herramientas para gestionar y crear ontologías fácilmente. Además, también proporciona un marco para construir aplicaciones ontológicas. *OntoRama* (Eklund et al. 2002) es un cliente Java que permite a los usuarios navegar a través de una estructura (ontológica) basada en conocimiento, en disposición hiperbólica. Finalmente, *OntoEdit* (Sure et al. 2002) es un editor de ontologías que soporta una construcción de ontologías basada en metodologías.

3.5.2.2 Wiki

Como señalan (Farenhorst, Lago & H. Van Vliet 2007), una wiki para capturar AK permite a los diseñadores y arquitectos software colaborar y comunicarse fácilmente. Además, gracias a las capacidades de la wiki, la información se puede actualizar rápidamente, y los stakeholders siempre sabrán cuál es el estado actual del proyecto.

A continuación, se describen algunas herramientas wiki para visualizar AK. C-ReCS o *Sistema Colaborativo de Captura de Requisitos* (*Collaborative Requirements Capture System*) (M. Klein 1997) proporciona soluciones simples para manipular el conocimiento y guardar las decisiones de un amplio número de procesos de ingeniería del software. Este sistema es una herramienta computacional para el soporte colaborativo de captura de decisiones de diseño, en particular, desde las fases tempranas del ciclo de vida del producto.

La herramienta *PAKME* o *Entorno de Gestión del Conocimiento de la Arquitectura basado en Procesos* (*Process-based Architecture Knowledge Management Environment*) (Babar et al. 2006) está basada en la web y dirigida a proporcionar soporte colaborativo para la gestión del conocimiento en el proceso de arquitectura software. PAKME consiste, por tanto, en cuatro componentes: *componente de interfaz de usuario basado en la web* (*web-based user interface component*), *componente de gestión del conocimiento* (*knowledge management component*), *componente de búsqueda* (*search component*), y *componente informativa* (*reporting component*). De esta forma, tanto (Tang et al. 2010) como (Dutoit et al. 2006), indican que, correspondientes a estos cuatro componentes, las características de PAKME pueden ser categorizadas en cuatro servicios de gestión del conocimiento arquitectónico: *adquisición del conocimiento* (*knowledge acquisition*), *mantenimiento del conocimiento* (*knowledge maintenance*), *recuperación del conocimiento* (*knowledge retrieval*), y *presentación del conocimiento* (*knowledge presentation*).

La herramienta ADDSS o *Sistema de Soporte de Decisiones de Diseño de la Arquitectura* (*Architecture Design Decision Support System*) (Capilla et al. 2006) es un prototipo de investigación en curso basado en la web, que almacena, gestiona y documenta las decisiones de diseño (Dutoit et al. 2006) (Tang et al. 2010). Utiliza un enfoque flexible basado en un conjunto de atributos obligatorios y opcionales para caracterizar las decisiones de diseño. De ahí que proporcione una estrategia combinada de codificación y personalización, donde las decisiones están relacionadas con los requisitos y las arquitecturas. Además, aseguran (Tang et al. 2010), ADDSS proporciona una serie de ventajas, como trazabilidad hacia delante y hacia atrás, gracias a los enlaces entre los requisitos y las arquitecturas; también proporciona reutilización del conocimiento general en forma de patrones y estilos arquitectónicos.

La herramienta KA o *Arquitecto de Conocimiento* (*Knowledge Architect*) (Jansen, Vries, et al. 2008) (véase Figura 7b) permite capturar, gestionar y compartir Conocimiento Arquitectónico, utilizando un servidor y repositorio de Conocimiento Arquitectónico. Esta herramienta permite explorar el Conocimiento Arquitectónico mediante búsqueda y navegación a través de la web de enlaces de trazabilidad entre entidades de conocimiento, según afirman (Tang et al. 2010). Además, una de las ventajas que estos autores observan sobre KA, es que proporciona gestión genérica del Conocimiento Arquitectónico sin limitarse a meta-modelos específicos de dicho conocimiento.

3.5.2.3 Node-link and tree

Este enfoque proporciona una representación de nodos interconectados, en disposición *de arriba a abajo* (*top-down*) o *de izquierda a derecha* (*left-right*). Esta técnica permite a los usuarios expandir y retraer nodos y sus sub-nodos, de tal forma que el nivel de detalle de la información puede ser regulado. A continuación, se presentan algunas de las herramientas correspondientes a esta categoría.

La herramienta *Cuestiones, Opciones y Criterios* (*Questions, Options, and Criteria*, QOC) (Maclean et al. 1991), basada en el enfoque de (Kunz & Rittel 1970), es una notación semiformal utilizada para representar el espacio de diseño que rodea a un artefacto. Esta herramienta hace uso de tres elementos principales: *Cuestiones* (sujetos de diseño claves), *Opciones* (posibles respuestas a las cuestiones) y *Criterios* (valoración de las opciones).

Como su propio nombre indica, la herramienta SCRAM (*SCenario Requirements Analysis Method*) (Sutcliffe & Ryan 1998) es un método de análisis de requisitos basado en escenarios. Según afirman sus autores, es un método basado en cuatro técnicas para la captura y validación de requisitos: *uso de prototipos o demostradores de conceptos* (*use of prototypes or concept demonstrators*), *escenarios* (*scenarios*), *rationale de diseño* (*design rationale*) y *resumen de pizarra* (*whiteboard summary*).

Por otro lado, la herramienta SEURAT (*Software Engineering Using RATionale*) (Burge & Brown 2004) es una utilidad de plug-in del entorno de desarrollo Eclipse que captura y utiliza la rationale de diseño enlazando su código software. La herramienta *Sysiphus* (Bruegge et al. 2006) también se basa en la rationale, y es un conjunto de herramientas que ayuda en la captura de varios modelos del sistema para las actividades de desarrollo del mismo. Soporta las decisiones de diseño basadas en la rationale y las relaciona con los modelos del sistema, según comentan (L. Lee & Kruchten 2008).

La herramienta DRIMER o *Recomendación de Diseño y Modelo de Intención Extendido para la Reusabilidad* (*Design Recommendation and Intent Model Extended to Reusability*) (Peña-Mora & Vadhavkar 1997) permite la captura explícita de ADRs, durante el proceso de desarrollo software. La herramienta AREL o *Rationale de la Arquitectura y Enlace de Elementos* (*Architecture Rationale and Element Linkage*) (Tang et al. 2007) está basada en UML para ayudar a los arquitectos a crear y documentar diseños arquitectónicos, centrándose en las decisiones arquitectónicas y la rationale de diseño.

AREL captura tres tipos de conocimiento arquitectónico: *intereses de diseño* (*design concerns*), *decisiones de diseño* (*design decisions*), y *resultados de diseño* (*design outcomes*). Estas entidades de conocimiento están representadas como entidades UML estándar y enlazadas para mostrar sus relaciones, según indican (Dutoit et al. 2006). (Tang et al. 2010) definen los *intereses de diseño* como entradas que influyen en las decisiones de diseño, es decir, entidades que encapsulan conceptos como requisitos funcionales y no funcionales, y contextos de proyecto. En cuanto a los *resultados de diseño*, indican que están compuestos por los diseños resultantes, por ejemplo, clases, componentes, interfaz, y caso de uso.

El *Sistema de Información Basado en Cuestiones* (*Issue-Based Information System*, IBIS) (Kunz & Rittel 1970) es un enfoque basado en argumentación para la representación de la lógica de diseño, que consiste en tres conceptos básicos y simples: *Cuestiones* (*Issues*) que necesitan ser tratadas; *Posiciones* o ideas (*Positions*) que responden a las cuestiones; *Argumentos* (*Arguments*) que están compuestos por *Pros* (argumentos a favor) y *Contras* (*Cons*, argumentos en contra) de una determinada *posición*. La estructura básica de un sistema IBIS es la vista en forma de árbol (Shahin, Liang & Khayyambashi 2010a). Este sistema, señalan(Dutoit et al. 2006), fue uno de los primeros que aparecieron en el ámbito de la investigación sobre la lógica de diseño, a principios de los años 70.

A finales de los años 80, surgió el sistema *gráfico IBIS*: gIBIS (*graphical IBIS*), propuesto por (Conklin & Begeman 1988). Esta herramienta utilizaba el color y un servidor de base de datos relacional muy rápido para facilitar la construcción y exploración de las redes IBIS. Las propuestas IBIS y su sucesora gIBIS fueron aplicadas en proyectos a gran escala durante los años 70 y 80, comentan (Dutoit et al. 2006).

Hay múltiples herramientas que proporcionan soporte a la visualización de ADDs, pero ninguna de ellas proporciona soporte explícito para visualizar la rationale de ADDs, comentan . De esta manera, surge *Compendium*, una herramienta de código abierto de mapeo del concepto de hipertexto semántico, que está implementada conceptualmente basada en IBIS y soporta las notaciones IBIS gráficas (gIBIS), por lo que soporta argumentación para representar la rationale de diseño (Shahin, Liang & Khayyambashi 2010a) (Shahin, Liang & Khayyambashi 2010b).

(Shahin, Liang & Khayyambashi 2010b) emplean Compendium para visualizar y explorar las ADDs en un ejemplo de ADD concreto, el cual está basado en el modelo de *Decisión de Arquitectura Orientada a Servicios* (*Service-Oriented Architecture Decision*, SOAD). De esta forma, presentan la manera de mapear los conceptos del modelo SOAD, presentado por (Zimmermann et al. 2007), en nodos de Compendium, los cuales son elementos de visualización en dicha herramienta (véase Tabla 3).

SOAD Concepts	Compendium Nodes	
Architectural Decision (AD)		Decision
Role, Scope, EditorialInfo, Decision Drivers, Recommendation		Note
Problem Statement		Question
ADAlternative		Answer
Pros		Pro Argument
Cons		Con Argument

Tabla 3. Mapeo de conceptos SOAD en nodos de Compendium (Shahin, Liang & Khayyambashi 2010b)

Así, los usuarios pueden seguir fácilmente los enlaces entre las entidades rationale y una ADD en Compendium, y actualizar sus decisiones cuando el contexto de diseño cambie.

Estos mismos autores, (Shahin, Liang & Khayyambashi 2010b), realizan otra correspondencia, entre los conceptos IBIS y los *elementos principales de las ADDs* (*ADD major elements*). Esta correspondencia establece el enlace entre las herramientas basadas en IBIS, como Compendium, y las herramientas basadas en los elementos principales de las ADDs, desde que Compendium está basado en/extiende de los conceptos IBIS y todas las herramientas ADD tienen un consenso sobre los conceptos principales de las ADDs (véase Tabla 4).

ADD Major Elements	IBIS Concepts
Problem	Issue (question)
Solution	Position (idea)
Rationale	Arguments (Pros and Cons)

Tabla 4. Correspondencia entre parte de los elementos principales de ADD y los conceptos IBIS (Shahin, Liang & Khayyambashi 2010a)

Por tanto, aseguran (Shahin, Liang & Khayyambashi 2010a), tanto las ADDs SOAD, como las ADDs basadas en otros modelos de decisión de diseño, pueden ser visualizadas en Compendium.

A modo de conclusión, la visualización de ADDs utilizando Compendium puede ayudar a los arquitectos a mejorar el entendimiento de las ADDs y su rationale, lo que conduce a un mejor entendimiento del diseño arquitectónico; también permite promover la comunicación de la documentación de la arquitectura, mediante la visualización de alternativas, manejadores de decisión, y argumentos de ADDs; así mismo, Compendium permite acelerar la actividad de evaluación arquitectónica, facilitando la revisión y el análisis de impacto de los cambios de las ADDs, según afirman (Shahin, Liang & Khayyambashi 2010b).

Sin embargo, todavía existen ciertas limitaciones, como que no todos los elementos del modelo de ADD pueden ser correspondidos uno a uno por nodos de Compendium, o que la visualización de la rationale de diseño ayuda en el análisis arquitectónico y en la explicación, pero, en otras situaciones, los arquitectos podrían querer sintetizar y construir un diseño de la arquitectura basado en su entendimiento de la rationale.

En base al enfoque de (Kunz & Rittel 1970), surge otra herramienta, como extensión de la herramienta IBIS, según afirman (Dutoit et al. 2006): *Lenguaje de Representación del Diseño* (*Design Representation Language*, DRL) (J. Lee 1990). Este lenguaje permite construir grafos de decisión, que reflejan los pros y contras que evalúan las alternativas con respecto a los objetivos.

La herramienta *ARCHIUM* (Jansen & Bosch 2005) (véase Figura 7a) es una extensión Java que proporciona trazabilidad a través de un amplio rango de conceptos, como requisitos, decisiones, descripciones de arquitectura, y artefactos de implementación, que son mantenidos durante el ciclo de vida del sistema. Además, comentan (Tang et al. 2010), ARCHIUM soporta dos tipos de relaciones de trazabilidad: *relaciones formales* (*formal relationships*), que son enlaces de trazabilidad definidos en el metamodelo de ARCHIUM, el cual tiene semánticas bien definidas; y *enlaces informales* (*informal links*), que son referencias realizadas en las descripciones textuales de los elementos del modelo.

La suite de ARCHIUM contiene un compilador, una plataforma en tiempo de ejecución, y una herramienta de visualización. El compilador transforma los archivos fuente de ARCHIUM en modelos ejecutables para la plataforma en tiempo de ejecución. La herramienta de visualización utiliza dicha plataforma para visualizar y hacer accesible el conocimiento arquitectónico, que, como indican (Shahin, Liang & Khayyambashi 2010b), se presenta como una vista de componentes, donde se visualizan las ADDs y sus relaciones en un *grafo de dependencias* (*dependency graph*). De esta forma, los arquitectos de

software pueden seleccionar un componente o conector en una vista determinada, y ARCHIUM muestra las ADDs relacionadas en otra vista. En esta herramienta, una ADD es considerada como una "función de cambio" que modifica el diseño de la arquitectura.

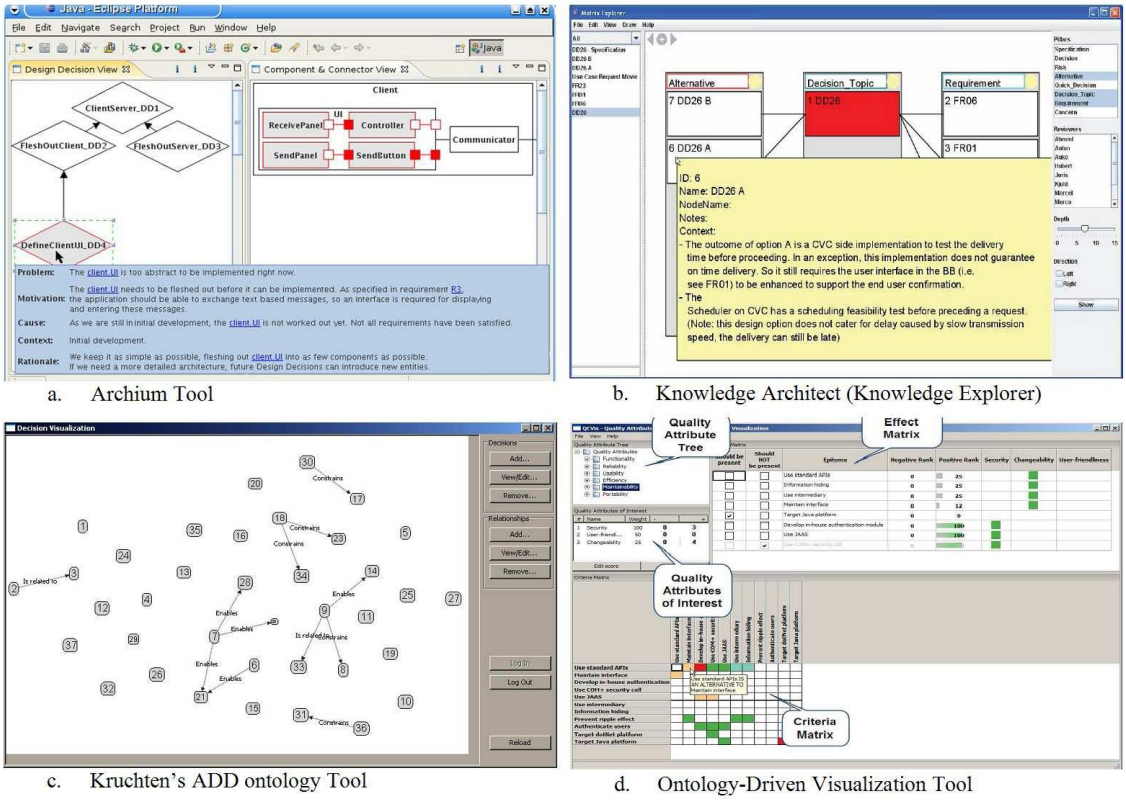
La herramienta *Kruchten's ADD Ontology tool* o *Herramienta de Ontología de las ADDs de Kruchten* (L. Lee & Kruchten 2008) (véase Figura 7c), como su propio nombre indica, está basada en la ontología sobre las decisiones de diseño arquitectónico de Kruchten. Sus autores indican que esta herramienta permite visualizar las ADDs separadamente de la arquitectura software en la que están referenciadas.

Esta herramienta facilita, tanto la exploración de decisiones, como el análisis detallado de las mismas, gracias a cuatro vistas (L. Lee & Kruchten 2008) (Shahin, Liang & Khayyambashi 2010b):

1. *Listas de decisión y relación* (*decision and relationship lists*). Esta vista es la más común en las herramientas de decisión. Lista las decisiones de diseño en una tabla, mostrando todos o una selección de los atributos de una decisión de diseño. En otra tabla, lista las relaciones de decisión. Los usuarios pueden crear, ver, modificar y eliminar las decisiones de diseño y sus dependencias.
2. *Vista de la estructura de decisión* (*decision structure visualization view*). Las decisiones de diseño y sus dependencias son visualizadas como un grafo dirigido. Cada decisión se muestra como un nodo, y la dependencia desde una decisión a otra como un arco directo.
3. *Vista cronológica de decisión* (*decision chronology view*). Permite ver la evolución en el tiempo de las decisiones de diseño, y proporciona la habilidad de determinar rápidamente las decisiones creadas o modificadas durante un intervalo de tiempo específico.
4. *Vista de impacto de decisión* (*decision impact view*). Proporciona una visualización de las decisiones que pueden ser afectadas potencialmente por un cambio en una decisión. Esta vista es muy valiosa cuando se van a realizar cambios radicales sobre un sistema, haciendo obvio el impacto de dichos cambios.

Además, esta herramienta adoptó el modelo de decisión de (Kruchten 2004), porque era un modelo sencillo, más centrado en el proceso, y el único que representaba explícitamente las relaciones de decisión.

Por último, la herramienta ODV o *Visualización Basada en Ontología* (*Ontology-Driven Visualization*) (De Boer et al. 2009) (véase Figura 7d) combina la potencia de las *listas de decisión y relación* introducidas por la herramienta Kruchten's ADD ontology, con la *vista de la estructura de decisión* de la misma herramienta, es decir, combina la simplicidad de la representación de la información tabular con la potencia de la inferencia ontológica de los atributos de decisión, típicamente utilizada por los auditores. De esta forma, se superaron las desventajas de ambas vistas.



a. Archium Tool b. Knowledge Architect (Knowledge Explorer)

c. Kruchten's ADD ontology Tool d. Ontology-Driven Visualization Tool

Figura 7. Capturas de pantalla de algunas herramientas de visualización de ADDs (Shahin, Liang & Khayyambashi 2010a)

3.5.2.4 Zoomable

Esta técnica de visualización es especialmente interesante en la manera en que trata las jerarquías. Presenta los nodos en los niveles más bajos de la jerarquía, anidados dentro de sus padres, con un tamaño menor que éstos últimos. Así, si queremos saber más sobre algunos nodos, tendremos que hacer zoom sobre los nodos hijos para expandirlos y hacer que sean el nivel de visualización actual (Katifori et al. 2007).

Como no existen herramientas de visualización específicas de AK, a continuación, se describen algunos sistemas de visualización de ontologías, relativos a esta categoría. *Grokker*[2] es un mapa gráfico de conocimiento, donde la información se representa gráficamente. Esta herramienta utiliza un mecanismo de clustering para presentar documentos como una serie de diagramas de Venn anidados (Rivadeneira & Bederson 2003).

Otra herramienta es *Jambalaya* (Storey et al. 2001), una integración o plug-in de SHriMP (*Simple Hierarchical Multi-Perspective*) (Wu & Storey 2000) con la herramienta ontológica Protégé (véase apartado 3.5.2.1). SHriMP es un entorno de visualización software multi-perspectiva, que combina técnicas de vista única y multi-vista para soportar exploración software. Finalmente, la herramienta *CropCircles* (Parsia et al. 2005) (Wang & Parsia 2006) presenta las jerarquías de clase en ontologías como árboles, de tal forma que los círculos representan nodos y cada círculo hijo está anidado dentro de su círculo padre.

3.5.2.5 Space-filling

Esta categoría presenta los nodos de forma jerárquica, utilizando todo el espacio de la pantalla, es decir, ajusta los nodos a dicha pantalla. Esta técnica no se considera muy interesante por dos razones: su falta de claridad, y su complicada conexión con la SA. Nótese que, para este tipo de representación, no existen ninguna herramienta específica de AK, por lo que, a continuación, se verán herramientas puramente ontológicas.

TreeMaps (Shneiderman 1992) es una herramienta para representar jerarquías o árboles que tengan cierto peso o magnitud en los nodos hoja, los cuales son rectángulos cuyo área es proporcional a algún atributo, como, por ejemplo, el tamaño del nodo. Otra herramienta es *SequoiaView* (Eindhoven 2002), donde la pantalla se subdivide de tal manera que los rectángulos se aproximan a cuadrados en la medida de lo posible. La última herramienta correspondiente a esta técnica es *Information Slices* (Andrews & Heidegger 1998) la cual representa estructuras jerárquicas utilizando uno o más discos semi-circulares, que representan múltiples niveles de la jerarquía.

[2] www.groxis.com

Algunos autores, como (Shahin, Liang & Khayyambashi 2010a), han comparado algunas de estas herramientas de visualización de ADDs comentadas anteriormente. En la Tabla 5, se muestra dicha comparativa ("+" indica "soportar", "-" indica "no soportar").

	Functionality					Architecting Support	
Existing Tools supporting ADD Visualization	Change impact analysis	Query support	Traceability support	Quantitative analysis	Collaborative architecting	Understandability of ADD	Modifiability of ADD
Archium	-	-	+	-	-	+	+
Knowledge Architect	+	+	+	+	+	++	++
Kruchten's ADD Ontology Tool	+	-	-	-	-	+	+
Ontology-Driven Visualization tool	+	-	-	+	-	+	+

Tabla 5. Comparativa entre diversas herramientas de visualización de ADDs (Shahin, Liang & Khayyambashi 2010a)

Como muestra la Tabla 5, dichas herramientas tienen algunas deficiencias. Por ejemplo, ARCHIUM y KA no pueden visualizar explícitamente relaciones de dependencia entre ADDs; la toma de decisiones colaborativa no está soportada en ARCHIUM, Kruchten's ADD ontology tool y ODV; y el soporte a la visualización de rationale (argumentos, pros, contras, y derivados de las decisiones) está ausente en la mayoría de las herramientas existentes. ARCHIUM, Kruchten's ADD ontology tool, y ODV no pueden visualizar alternativas, manejadores de decisiones y justificaciones de una ADD, según indican (Shahin, Liang & Khayyambashi 2010a). Éstas y otras deficiencias, sin embargo, pueden ser tratadas casi en su totalidad por la herramienta *Compendium*, antes comentada (véase sección 3.5.2.3).

Por tanto, y a modo de conclusión, el objetivo que se persigue actualmente, aseguran (Kruchten et al. 2009), es que todas estas herramientas de visualización sean mejoradas, adaptadas y mejor integradas, de tal forma que se eviten esfuerzos duplicados al capturar las decisiones de diseño. Esperanzadoramente, las decisiones de diseño y la rationale de diseño se han reconocido en el estándar ISO/IEC 42010[3].

3.5.3 Conclusiones

La percepción que tiene la comunidad de arquitectura software de que las decisiones de diseño arquitectónico son intangibles y difíciles de capturar y comunicar, está cambiando, como resultado de las investigaciones recientes, afirman (Kruchten et al. 2009).

De esta forma, muchos autores, como (Kruchten et al. 2009), (Biehl & Torngren 2010), (L. Lee & Kruchten 2008), (Shahin, Liang & Khayyambashi 2010a), (Tang et al. 2010) y (Shahin, Liang & Khayyambashi 2010b), sugieren que las decisiones de diseño arquitectónico claves se almacenen y documenten para poderlas reutilizar, reduciendo así el esfuerzo de modelar la arquitectura. Pero hay que tener en cuenta que no siempre vale la pena capturar y mantener todas las micro-decisiones que acontecen a lo largo de la vida de un sistema software (Kruchten et al. 2009).

Es por esto que muchos de los esfuerzos se deberán centrar, a partir de ahora, en mejorar las herramientas que soportan la rationale de diseño para adaptarlas e integrarlas mejor, y así *evitar esfuerzos duplicados* en la captura de estas decisiones, prestando especial atención a la poca diferencia que existe entre los requisitos software o la descripción de un patrón de diseño conocido, y la representación explícita de una decisión de diseño. En muchos casos, una decisión de diseño constituye una réplica del requisito que la motivó. Como resultado, el esfuerzo de capturar dichas decisiones, se considera duplicado, ya que los usuarios de dichas herramientas, a menudo, almacenan la misma información. Por tanto, habrá que proporcionar mecanismos adecuados, basados en potentes técnicas de localización y detección de duplicados, opinan (Kruchten et al. 2009).

A su vez, (Shahin, Liang & Khayyambashi 2010a) señalan que, aunque se han propuesto múltiples modelos de ADDs y se han desarrollado herramientas relacionadas con la captura de decisiones de diseño,

[3] Este estándar se puede encontrar en http://www.iso-architecture.org/ieee-1471/docs/ISO-IEC-FDIS-42010.pdf.

sigue existiendo la necesidad de visualizar y explorar estas decisiones, y su rationale subyacente. Además, estos mismos autores han enfocado su trabajo futuro, entre otros aspectos, en investigar sobre cómo la visualización de la rationale de las ADDs puede sistemáticamente *soportar y mejorar el proceso arquitectónico*, incluyendo la implementación arquitectónica y el mantenimiento; y *automatizar, parcialmente, el soporte a la visualización de ADDs*, por ejemplo, cuando los usuarios seleccionan una ADD documentada en el texto, la herramienta de visualización podría extraer automáticamente las entidades asociadas y visualizarlas.

(Tang et al. 2010) realizaron una comparativa entre cinco herramientas de gestión del conocimiento arquitectónico: ADDSS, ARCHIUM, AREL, Knowledge Architect y PAKME. De esta forma, han podido observar que todas las herramientas de conocimiento arquitectónico se centran en el *conocimiento razonado y contextual* (*reasoning and contextual knowledge*), algunas enfatizan el *conocimiento de diseño* (*design knowledge*), mientras que otras lo hacen sobre el *conocimiento general* (*general knowledge*). La diferencia de énfasis destaca los dos aspectos clave del conocimiento: soportar *actividades de diseño arquitectónico* (*architectural design activities*) y soportar *reutilización del conocimiento* (*knowledge reuse*). La *trazabilidad* está bien soportada por todas las herramientas de visualización que han estudiado, pero el soporte a la *integración y personalización del conocimiento* es limitado.

Además, (Tang et al. 2010) aseguran que los aspectos de gestión del conocimiento arquitectónico, como la *evolución de diseño* y los *patrones de diseño*, son las características principales de las herramientas de conocimiento arquitectónico. Sin embargo, las características de *compartición del conocimiento* todavía no están implementadas en las herramientas que estos autores han analizado, y sólo se menciona como capacidad futura.

Gracias a este trabajo comparativo, (Tang et al. 2010) han podido identificar algunas de las direcciones que podría tomar la investigación futura sobre herramientas de gestión del conocimiento arquitectónico:

- *Interoperabilidad de herramientas*, para permitir intercambio de conocimiento arquitectónico.
- *Herramientas de gestión del conocimiento arquitectónico inteligentes*, para proporcionar soporte automático a las actividades arquitectónicas (notificaciones, sugerencias, etc.).
- Definir un *modelo general de representación del conocimiento arquitectónico* que combine características de diferentes modelos de conocimiento arquitectónico.

Por tanto, todavía queda mucho por hacer en cuanto a técnicas y herramientas de visualización de AK, pero debe quedar claro que:

- Sino todas, al menos, las ADDs consideradas más importantes deben formar parte explícitamente de la arquitectura software, para poderlas reutilizar posteriormente.
- Aunque hay una gran variedad de sistemas que permiten visualizar este tipo de conocimiento, la mayoría presentan una clara falta de madurez.

A la vista de las características y deficiencias que presentan las herramientas de visualización de ADDs conocidas hasta el momento, nuestra propuesta de trabajo intentará proporcionar métodos más sofisticados para *visualizar la información* contenida en la red de decisión, y completar este conocimiento con la definición, e incluso la implementación local de varias métricas adecuadas, que también serán utilizadas para nuestro análisis detallado.

Para ello, el trabajo asociado a esta tesis doctoral comenzará analizando las fortalezas y debilidades de los distintos enfoques que han surgido en los últimos años para visualizar AK, dado que no existen análisis de este tipo. Así, los analistas dispondrán de una guía útil cuando tengan que tomar decisiones sobre la mejor alternativa para su proyecto (véase apartado 4.5.1).

Finalmente, la idea de esta tesis doctoral será la de contribuir en la mejora de la visualización (y, por consiguiente, de la comprensión) del AK, implementando algún tipo de mecanismo que permita visualizar las decisiones de diseño de una forma simple, intuitiva y muy gráfica, que permita expandir cada una de ellas sobre la red de decisión para visualizar entidades relacionadas con las mismas (nombre de la decisión, pros, contras, etc.).

3.6 Análisis de Redes

Hoy día, y cada vez más, reconocemos que nada sucede de forma aislada. La mayoría de los eventos y fenómenos están conectados, causados por, e interactuando con un gran número de piezas distintas de un puzle complejo. Vivimos en un pequeño mundo, donde todo está vinculado con todo. De esta forma, somos testigos de una revolución en la construcción y, por tanto, hemos llegado a comprender la importancia de las *redes* (Barabási 2002). "Las redes están en todas partes"-así comienza el primer capítulo del gran manual de referencia por excelencia en el ámbito del análisis de redes, escrito por (Barabási et al. 2006). Estos autores indican que el estudio de las redes ha tenido una larga historia en el campo de las

matemáticas y las ciencias, cuando, en 1736, el matemático Leonard Euler comenzó a interesarse por el enigma matemático ampliamente conocido como el *Problema de los Puentes de Königsberg* (véase Figura 8).

Figura 8. Problema de los Puentes de Königsberg (Barabási et al. 2006)

Este problema planteaba el siguiente dilema: ¿Existe algún camino único que atraviese los siete puentes exactamente una sola vez? Euler demostró que dicho camino no era posible. Esta demostración la llevó a cabo utilizando un *grafo*, es decir, un objeto matemático que consistía en puntos (vértices o *nodos*), y líneas (aristas o *enlaces*), lo que permitía abstraer todos los detalles del problema original, conservando su conectividad.

Esta comprobación de Euler es considerada como el primer teorema de la conocida *teoría de grafos*, la cual se ha convertido en el principal lenguaje matemático para describir las propiedades de las redes. De esta forma, esta *teoría de grafos pura* es elegante y rigurosa, pero no es especialmente relevante para redes derivadas del mundo real. Sin embargo, la *teoría de grafos aplicada*, como su propio nombre indica, se preocupa más por los problemas de las redes del mundo real, pero su enfoque está orientado hacia el diseño y la ingeniería.

Por el contrario, (Barabási et al. 2006) afirman que las investigaciones recientes en este ámbito se centran en redes que surgen de forma natural, como las *redes sociales*, dado que son redes de información como las redes de citas (o referencias) y la World Wide Web. Pero esto va más allá, y esta categoría es aún más amplia, incluyendo redes como las de transporte, las de energía y la Internet física, cuyo propósito es único y coordinado (transporte, reparto de energía y comunicaciones), pero que están construidas durante largos periodos de tiempo por multitud de agentes y autoridades independientes. Así, una *red social* se define como una estructura que refleja las relaciones entre las entidades sociales que la componen (Wasserman & Faust 1994).

Por su parte, el *análisis de redes sociales* es fuertemente empírico, pero tiende a ser descriptivo más que constructivo. Por tanto, aunque en el pasado, tanto la teoría de grafos como el análisis de redes sociales han tendido a tratar las redes como estructuras estáticas, trabajos recientes han demostrado que las redes evolucionan con el tiempo, siendo el producto de procesos dinámicos que añaden o eliminan nodos o aristas. No se puede, por tanto, ignorar el papel que juegan los participantes en la red ni los patrones de comportamiento seguidos por los mismos, según indican (Barabási et al. 2006).

Es por esto que, en las últimas décadas, la noción de una red social y los métodos de análisis de redes sociales han atraído considerablemente el interés y la curiosidad por parte de la comunidad de ciencias sociales y del comportamiento, según afirman (Wasserman & Faust 1994). Este interés viene determinado, por las nuevas facilidades ofrecidas a los investigadores de utilizar esta perspectiva de red, la cual permite responder a preguntas de investigación de las ciencias sociales y del comportamiento estándar, y aportando una definición formal precisa sobre aspectos del entorno estructural político, económico y social.

Existen diversos conceptos clave relativos al análisis de redes que resultan fundamentales a la hora de hablar de redes sociales. A continuación, se describen brevemente dichos conceptos, introducidos por (Wasserman & Faust 1994). En primer lugar, tenemos que hablar de actores. Un *actor* es una entidad social, es decir, una unidad social discreta individual, corporativa, o colectiva. De esta forma, en una red social, existen conexiones entre actores que conllevan una serie de implicaciones. Personas en un grupo,

departamentos de una corporación o agencias de servicios públicos en una ciudad son, todos ellos, ejemplos de actores. Además, la mayoría de las aplicaciones de redes sociales se centran en colecciones de actores en las que todos sean del mismo tipo, denominadas *redes de un modo* (*one-mode networks*).

Otro concepto importante es el de *vínculo relacional* (*relational tie*). Los actores están conectados con otros por medio de vínculos sociales, los cuales establecen enlaces entre parejas de actores. Algún ejemplo común de vínculos relacionales, en el ámbito de análisis de redes, son:

- *Evaluación de una persona a otra*, por ejemplo, expresando amistad, vinculación o respeto.
- *Interacción de comportamiento*, por ejemplo, hablar entre sí o mandar mensajes.
- *Conexión física*, por ejemplo, una calle, un río o un puente que une dos puntos.
- *Relaciones formales*, por ejemplo, de autoridad.
- *Relaciones biológicas*, por ejemplo, de parentesco o descendencia.
- *Etc.*

El término *pareja* (*dyad*) hace referencia, en el nivel más básico, al vínculo o relación que establece un vínculo relacional entre dos actores. Por tanto una pareja consiste en un par de actores y el (posible) vínculo(s) entre ellos.

A veces, resulta de interés estudiar las relaciones entre grandes subconjuntos de actores, dado que muchos métodos y modelos importantes de redes sociales se centran en dichos subconjuntos. De esta forma, surge el concepto de *tríada* (*triad*), un subconjunto de tres actores y el (posible) vínculo(s) entre ellos.

Otro concepto relevante es el de *subgrupo* (*subgroup*), definido como cualquier tipo de subconjunto de actores (pareja, tríada, etc.), y todos los vínculos que existen entre ellos. Pero el análisis de redes no es sólo colecciones de parejas, tríadas o subgrupos, sino que, en gran medida, su potencia recae en la habilidad de modelar las relaciones entre sistemas de actores. Un sistema consiste en vínculos entre los miembros de algún grupo (más o menos delimitado), donde un *grupo* (*group*) es la colección de todos los actores en donde los vínculos se van medir, es decir, un conjunto finito de actores, quienes por razones conceptuales, teóricas o empíricas son tratados como un conjunto finito de individuos sobre el que se realizan mediciones.

El término *relación* (*relation*) se define como la colección de vínculos de un tipo específico entre miembros de un grupo. Por ejemplo, un conjunto de amistades entre parejas de niños en una clase, o el conjunto de vínculos diplomáticos formales mantenidos por pares de naciones en el mundo. En cualquier grupo de actores, se podrían medir varias relaciones diferentes.

Finalmente, y una vez definido lo que es un actor, un grupo y una relación, es posible dar una definición más explícita de red social. Una *red social* (*social network*) consiste en un conjunto finito o conjuntos de actores y la relación o relaciones definidas sobre ellos. La presencia de información relacional es una característica crítica y de definición de una red social. De esta forma, los actores de una red social se representarán como nodos, mientras que las relaciones se corresponderán con aristas, según indican (Hanneman & Riddle 2005).

(Merelo 2006) también aporta sus propias definiciones al ámbito del análisis de redes. Este autor define una *red* como una forma abstracta de visualizar una serie de sistemas y, en general, casi todos los sistemas complejos. Indica, como ya es sabido, que estas redes están compuestas por *nodos* y enlaces entre ellos, llamados *aristas* (si son unidireccionales) o *arcos* (si son bidireccionales), donde la dirección de dichos enlaces se denomina *flujo* (Velázquez & Aguilar 2005). Merelo habla sobre las *redes sociales*, señalando que también son redes complejas, aunque usen una terminología ligeramente diferente: los nodos son *agentes*, porque hacen algo, mientras que las aristas o arcos expresan, habitualmente, una *relación social* tal como *conoce-a*, *es-amigo-de*, o *han-comido-espaguetis-juntos* (véase ejemplo de red social en Figura 9).

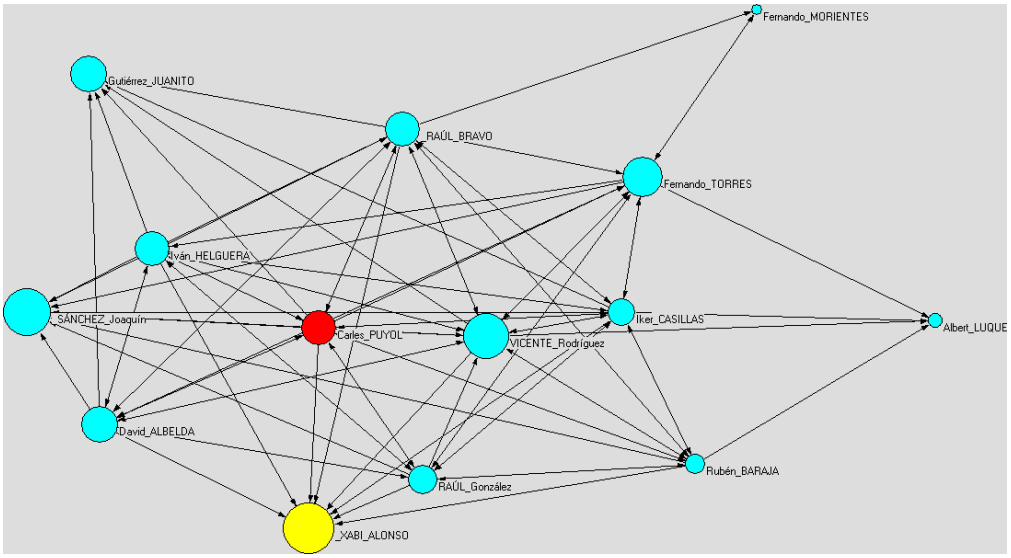


Figura 9. Red de pases de la Selección Española de fútbol en el encuentro España-Portugal de la Eurocopa 2004 (Merelo 2006)

Para llevar a cabo el análisis de una determinada red social, afirman (Hanneman & Riddle 2005), se suelen utilizar métodos formales para representar la información de dicha red, existiendo tres razones principales para ello:

- Las matrices y los grafos son compactos y sistemáticos. Resumen y presentan gran cantidad de información de una forma rápida y fácil; obligándonos a ser sistemáticos y completos cuando describimos patrones encontrados en las relaciones sociales.
- Las matrices y los grafos permiten tomar ayuda prestada de los ordenadores para analizar los datos, lo que resulta extremadamente útil cuando estamos trabajando con grandes cantidades de actores y de tipos de relaciones entre ellos.
- Las matrices y los grafos tienen reglas y convenios que nos permiten comunicarnos claramente. Además, estas reglas y convenios del lenguaje de grafos y de las propias matemáticas permiten ver cosas en nuestros datos que no podemos ver si describimos dicha información sólo con palabras.

Existen diversas herramientas que permiten visualizar las redes sociales y llevar a cabo el análisis de las mismas, por ejemplo, *UCINET* (Borgatti et al. 2002) o *NetDraw* (Borgatti 2002), los cuales son herramientas con características similares a las de otros programas compatibles con el sistema operativo Windows, permitiendo una fácil y rápida navegación a través de iconos de acceso directo, ventanas flotantes o su barra de menús. Estas dos herramientas son citadas por múltiples autores como (Hanneman & Riddle 2005) o (Velázquez & Aguilar 2005).

De esta forma, comentan (Wasserman & Faust 1994), el análisis de redes sociales puede verse como una ampliación o generalización de las técnicas analíticas estándares de datos y estadística aplicada que se centra, generalmente, en las unidades observacionales y sus características. Por tanto, lo que caracteriza a este tipo de redes es el uso de su información estructural y relacional para estudiar o probar teorías.

Así, para analizar redes sociales, existen métodos que proporcionan definiciones y descripciones formales sobre las propiedades estructurales de los actores, subgrupos de actores, o grupos. Estos métodos traducen los conceptos principales de las teorías sociales y de comportamiento en definiciones formales expresadas en términos relacionales, de tal forma que todos estos conceptos son cuantificados, considerando las relaciones medidas entre los actores de una red.

Por tanto, el análisis de redes sociales debe considerar la información sobre las relaciones entre las unidades, sin olvidarse de los atributos de los actores. Así, los conjuntos de datos de una red social compleja pueden contener información acerca de las características de los actores (como el género de las personas en un grupo, o el PIB de las naciones en el mundo), así como variables estructurales.

Estos métodos de análisis de redes sociales suelen ser adecuados para conceptos en determinados niveles de análisis. A continuación, se comentan brevemente los distintos tipos de métodos que distinguen (Wasserman & Faust 1994), dependiendo del número de actores que forman parte del análisis de la red.

Existen propiedades y métodos asociados sólo con *actores individuales*, que permiten conocer, entre otras cosas, cómo de destacado es un actor dentro de un grupo, cuantificado gracias a medidas como la

centralidad y el prestigio, expansividad a nivel de actor y parámetros de popularidad embebidos en modelos estocásticos, y medidas para roles individuales, como aislamientos, enlaces, etc.

También existen métodos aplicables a *parejas de actores* y las relaciones entre ellos, como aquellos de la teoría de grafos que miden, por ejemplo, la distancia y la accesibilidad del actor. Otros métodos están basados casi siempre en declaraciones teóricas sobre el balance y la transitividad, y postulan ciertos comportamientos acerca de *tríos de actores* y las relaciones entre ellos.

Otro tipo de métodos permiten encontrar y estudiar *subconjuntos de actores* homogéneos con respecto a algunas propiedades de la red. Por ejemplo, subgrupos cohesivos que contienen actores que están cerca unos de otros, o subgrupos de actores que presentan un comportamiento similar con respecto a ciertos parámetros del modelo, derivados de modelos estocásticos.

Finalmente, existen métodos y medidas que se centran en *grupos* enteros y todas las relaciones. Por ejemplo, las medidas teóricas de grafos, como la conexión y el diámetro, o las medidas a nivel de grupo de centralización, densidad y prestigio.

Capítulo 4

Anteproyecto

4.1 Introducción

Desde la aparición del área de Arquitectura Software (SA) se ha percibido que una parte importante de la especificación de la misma es la descripción del conocimiento arquitectónico, es decir, porqué la arquitectura ha sido especificada de una determinada forma. De hecho, (D. E. Perry & A. L. Wolf 1992) en uno de los artículos fundacionales del área, describen la SA de un sistema mediante la ecuación:

$$\textit{Arquitectura Software} = \{\textit{Elementos, Estructura, Rationale}\}$$

Este modelo considera la SA como un conjunto de *elementos arquitectónicos* (tales como componentes o conectores), que siguen una *estructura* determinada que determina sus posibles conexiones y distribución, de acuerdo a una motivación inicial (*rationale*).

Sin embargo, en las propuestas iniciales, cuando la SA se consideraba en gran medida como una forma de documentar parte del diseño del sistema, la rationale fue pronto olvidada. Probablemente esto fue debido a que en esta etapa temprana del área, la rationale fue percibida como documentación textual y desestructurada, concebida como el tipo de información de proceso que, a menudo, se considerada como no importante. Los resultados de las Decisiones de Diseño (DDs) en que se basaba la SA, estaban implícitamente incorporados dentro de la misma, pero las DDs no estaban explícitamente documentadas, ni tampoco lo que se ha denominado las Rationales de Diseño (DRs) del diseño, es decir, las justificaciones de porqué unas DDs eran tomadas y otras fueran desestimadas. Esté era el motivo por el que todas las suposiciones e información sobre el diseño y su evolución, lo que se ha denominado Conocimiento Arquitectónico, fueran finalmente perdidos. Esta pérdida de la información sobre las DDs, conduce a una serie de problemas relacionados con la SA (Jansen & Bosch 2005), tales como:

- Problemas imprevisibles cuándo el analista cambia la especificación sin conocer el porqué de su actual especificación, dado que las DDs son transversales en muchos puntos de la SA.
- Las reglas y limitaciones de diseño que se desean establecer en la especificación son violadas conforme ésta evoluciona.
- Las decisiones de diseño problemáticas no son eliminadas, dado que el arquitecto vuelve a tomarlas al no tener constancia de los problemas que éstas ocasionan.

Como consecuencia de estos problemas, los sistemas aumentan su coste de mantenimiento, tendiendo a erosionarse rápidamente.

No ha sido hasta el año 2005 cuando trabajos como los de (Jansen & Bosch 2005), (Lago & Van Vliet 2005) o (Tyree & Akerman 2005) cuando el Conocimiento Arquitectónico ha recuperado el foco de atención. A partir de dichos trabajos, la SA pasa a verse como el resultado de las diferentes DDs que se han ido tomando durante su especificación, es decir, pasan de aparecer de forma implícita en la arquitectura, por la propia especificación, a ser descritas de forma explícita. Este nuevo planteamiento ayudará al arquitecto (Jansen & Bosch 2005) a:

- Guardar la integridad conceptual de la SA;
- A prevenir errores evidentes por facilitar una exploración explícita del espacio de diseño;

- Analizar tanto la propia SA como su proceso de diseño;
- Mejorar la trazabilidad entre las DDs y características a satisfacer o aspectos de diseño.

Después de esta breve introducción, este capítulo se estructura como sigue. En la sección 4.2, se presenta el enunciado de la tesis doctoral, así como la hipótesis de la que se parte para su realización. En la sección 4.3, se indica la metodología a seguir para llevar a cabo dicha tesis. En la sección 4.4, se presenta la planificación inicial de la tesis doctoral y, finalmente, en la sección 4.5 se detallan las distintas tareas realizadas en el contexto de esta tesis doctoral.

4.2 Enunciado de la Tesis Doctoral e Hipótesis de Partida

La tesis doctoral que se plantea, pretende aportar una visión industrial a la explotación del conocimiento arquitectónico. La mayoría de las propuestas que se han planteado hasta la fecha en esta área, han estado centradas en la descripción de propuestas para su especificación, pero desafortunadamente no existen todavía trabajos suficientes que se hayan centrado en su explotación y las implicaciones que ésta supondría a nivel industrial. Aquí es donde se centra esta tesis planteada desde cuatro puntos de vista diferentes, identificados en la Figura 10, y que se detallan en los siguientes apartados.

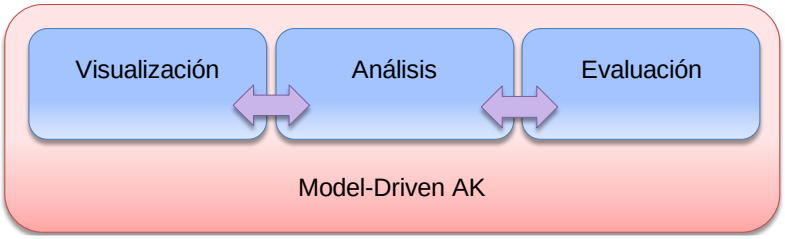


Figura 10. Ámbito de la tesis doctoral

4.2.1 AK Dirigido por Modelos

Actualmente, el *Desarrollo Dirigido por Modelos* (*Model-Driven Development*, MDD (Selic 2003)) está teniendo cada vez una mayor aceptación, no sólo entre la comunidad científica, sino también en la industria. Esto es debido a las ventajas que éste aporta en términos de *abstracción*, permitiendo a los *stakeholders* centrarse en el problema y no en la tecnología, y de *automatización,* facilitando la generación de forma automática/semi-automática del software para diferentes plataformas.

Por ello, su utilización en esta tesis doctoral aparece de forma natural. Actualmente, los directores de tesis han desarrollado una de las primeras propuestas en el área, presentada en (Navarro & Cuesta 2008), en la que se ofrece soporte al conocimiento arquitectónico en diferentes niveles de abstracción, elaborada mediante la extensión del proceso MDD denominado ATRIUM. A nivel de requisitos, se expresan como un elemento más de la especificación de forma que cada DD puede ser analizada de acuerdo a los requisitos que ayuda a conseguir. Gracias a las facilidades de transformación Modelo-a-Modelo utilizadas, las DDs tomadas pueden ser trazadas a nivel de la especificación arquitectónica para su utilización.

Sin embargo, las capacidades y facilidades que suponen la utilización de la aproximación MDD para la gestión de conocimiento arquitectónico todavía han de ser identificadas y evaluadas. Además, la incorporación de esta aproximación también pretende que sirva de marco integrador para las otras tres líneas de actuación, visualización, análisis y evaluación, tal y como se describe en los apartados siguientes.

4.2.2 Visualización en AK

Uno de los factores más importantes a considerar a fin de facilitar la explotación del conocimiento arquitectónico es la identificación y explotación de técnicas adecuadas de visualización del mismo. Sobre todo teniendo en cuenta el volumen de información que en determinados proyectos se puede llegar a manipular. Por ejemplo, en los proyectos desarrollados en colaboración con Eurocopter España se han llegado a manipular más de quinientas decisiones de diseño, sobre todo por la complejidad de los sistemas desarrollados.

En este sentido, otra de las aportaciones de esta tesis será estudiar y evaluar la viabilidad de utilizar las técnicas de visualización que de hecho ya se han utilizado en otro dominio directamente relacionado: las ontologías (Katifori et al. 2007). No hemos de olvidar que el conocimiento arquitectónico es por sí mismo una base de conocimiento constituida por DDs y DRs. Se deberá determinar qué técnicas son más apropiadas para facilitar el análisis del conocimiento arquitectónico, las necesidades de adaptación o de definición de nuevas propuestas.

Además, parte importante de dicha aportación será la componente humana de dicha visualización, es decir, estudiar dichas propuestas desde la perspectiva de la *usabilidad*. Se llevarán a cabo diferentes

experimentos con usuarios que nos permitan determinar qué propuestas facilitan, no sólo la definición del conocimiento arquitectónico, sino también la explotación del mismo.

4.2.3 Análisis de redes

El conocimiento arquitectónico, en la representación elegida, se desarrolla como un grafo que cubre todo el ciclo de vida, desde los requisitos en el modelo de objetivos hasta los componentes que constituyen la descripción arquitectónica final. La relación de trazabilidad, en combinación con la estructura creada por las DDs y sus interrelaciones, forma una estructura compleja, que en ocasiones se referencia con nombre de justificación (*rationale*) arquitectónica (Fenton & S. L. Pfleeger 1997)(D. E. Perry & A. L. Wolf 1992).

Aunque diversos autores han capturado esta información utilizando diversos formatos (Jansen & Bosch 2005)(Tang et al. 2006)(Tyree & Akerman 2005) uno de los aspectos más interesantes del enfoque aquí descrito es que la representación gráfica admite un análisis posterior, que resulta considerablemente más difícil utilizando estructuras más "textuales". De hecho, este es, junto a los aspectos de visualización ya mencionados, el principal motivo para preferir una representación gráfica frente a la puramente textual, más habitual. Este análisis hace posible, no sólo una evaluación preliminar de la estructura resultante, sino también la identificación de patrones, ya no en la arquitectura, sino en la propia red.

De hecho, en los últimos años, el análisis de redes, conocido con frecuencia como *Análisis de Redes Sociales* (*Social Network Analysis*, SNA), ha conocido una enorme difusión fuera de sus campos más tradicionales (matemáticas, economía), siendo aplicado de manera cada vez más frecuente a campos tan diversos como la física, la sociología y la computación (Newman et al. 2006). El análisis de redes estudia la estructura interna de las redes como grafos, localizando elementos y subestructuras, los roles que éstos adquieren, y su importancia en la estructura global. De este análisis se obtiene, tanto la identificación de elementos clave (conectores o *hubs*, puentes), como toda una serie de índices globales (grado de separación, grado de libertad, etc.) que determinan la estructura y propiedades de la red como un todo. De este análisis se puede deducir que una parte muy significativa de las características presentes en un sistema son de hecho subsumidas por la propia red.

SNA se ha aplicado ya en el campo de computación con objetivos muy diversos. En esta tesis se pretende abordar su aplicación al estudio de las redes de conocimiento arquitectónico, cuya complejidad resulta suficientemente significativa. Se pretende, mediante este análisis, identificar toda una serie de *patrones* que permitan determinar las subestructuras que dan soporte a una arquitectura eficiente, así como aquellas que puedan asociarse con ineficiencias o problemas en la misma.

En trabajos anteriores del equipo investigador (Navarro, Cuesta, et al. 2009)(Navarro et al. 2010) se ha trabajado en el estudio de la identificación de *patrones* y *anti-patrones* a nivel de la arquitectura, que permitan anticipar la detección de posibles problemas y su tratamiento semiautomático. Se pretende extender esta línea de investigación, no sólo a la estructura final de la proto-arquitectura (como se hace en la actualidad), sino también a la estructura de la red de conocimiento arquitectónico (que de hecho se modifica y extiende tras este análisis). Esto llevará, en última instancia, a identificar patrones y *anti-patrones* de conocimiento arquitectónico, que permitirán aplicar estas técnicas al propio *rationale*, obteniendo un grado de mejora en el *proceso* similar al que se ha alcanzado en el resultado del mismo.

En resumen, esto permitirá adoptar una visión estructural (Newman et al. 2006) del conocimiento arquitectónico, que sigue siendo compatible con las técnicas de la ingeniería ontológica, pero que permite manejar un nivel de abstracción más alto y, por ende, una complejidad menor. Todo ello, además, permitirá una mayor y más eficiente *reutilización* de este conocimiento, que es el objetivo último de todo el enfoque.

4.2.4 Evaluación de la red AK

Aunque presente desde hace décadas, en los últimos años se ha incrementado también el interés por la denominada *Ingeniería del Software Empírica* (Fenton & S. L. Pfleeger 1997), que introduce un énfasis explícito en la evaluación temprana de las técnicas y métodos de la Ingeniería de Software, con el fin de determinar, tanto su eficacia, como su eficiencia. Así, en lugar de afirmarse, de manera más o menos informal, que determinada técnica da resultado, se evalúa de manera explícita la aplicación de la misma y su impacto en la solución obtenida, determinando de manera precisa sus beneficios.

Existen numerosas técnicas y métodos para la evaluación de las arquitecturas de software (Clements et al. 2001), siendo las más conocidas aquellas basadas en escenarios, tales como los métodos SAAM (Rick Kazman et al. 1994) y ATAM (Rick Kazman et al. 1999), entre otros. Estas técnicas se centran en determinar la utilidad real de la arquitectura desarrollada, sin entrar a valorar aspectos genéricos de su estructura. Por otra parte, el enfoque tradicional de la ingeniería de software empírica se ha basado en el

desarrollo de métricas (Fenton & S. L. Pfleeger 1997) adecuadas al aspecto que se deseaba evaluar. De nuevo, y aunque con un desarrollo menor, existen ya diversas métricas arquitectónicas (Nakamura & Basili 2005) que pretenden evaluar diversos detalles de una arquitectura.

Sin embargo, los aspectos de evaluación apenas han sido abordados en lo que respecta al rationale (Burge et al. 2008) arquitectónico. Incluso las técnicas que centran el diseño en este enfoque (considerando que la justificación de una arquitectura es incluso más significativa que otros aspectos de la arquitectura) han considerado los aspectos de evaluación de una manera escasa (Burge et al. 2008).

Sin embargo, dada su naturaleza estructural, la representación del AK en forma de red hace factible su análisis estructural, como se ha indicado en el punto anterior. Del mismo modo, hace más sencillo el establecimiento de métricas que permitan tanto evaluar las propiedades globales de la red AK concreta, como realizar la comparación entre dos redes similares, o incluso la evolución de una única red.

En estos términos, resultan especialmente prometedoras, en primera instancia, las métricas basadas tanto en la complejidad métrica (Caseau et al. 2007), derivada a su vez de la complejidad ciclomática (muy utilizada a su vez en el propio campo de la arquitectura), como las basadas en la distancia estructural (Nakamura & Basili 2005). Estas métricas, aplicadas no ya a la arquitectura sino a la propia red de AK, permitirán una primera evaluación cuantificable del enfoque propuesto.

En última instancia, muchos de los métodos de evaluación a utilizar son susceptibles de aplicar técnicas de investigación operativa (en especial en la forma de optimización), que permitirían alcanzar un mayor grado de sofisticación en la evaluación final. No obstante, en una primera aproximación, se espera obtener resultados muy significativos incluso con las primeras técnicas propuestas.

4.2.5 Descripción del dominio de aplicación

Uno de los objetivos que siempre se han de procurar en cualquier tesis doctoral es la validación de las propuestas en entornos reales y, especialmente, industriales. Este es uno de los objetivos principales de esta tesis, tal y como se remarca en el título de la misma. Así, y gracias a la participación en diferentes proyectos industriales que el grupo LoUISE está llevando a cabo en colaboración con la empresa Eurocopter España, se podrán validar las diferentes propuestas que se planteen.

Eurocopter España, y concretamente el departamento de I+D situado en Albacete, lleva a cabo su actividad en el ámbito del desarrollo software para sistemas aviónicos. Este dominio se guía por un objetivo simple pero inflexible: evitar la pérdida de vidas humanas (Wils et al. 2006). Toda decisión que es tomada en este dominio ha de ser siempre analizada desde el punto de vista de la seguridad de las personas que se podrían ver afectadas por la misma. Es por ello que el Conocimiento Arquitectónico toma especial relevancia en este contexto, como así se ha constatado en los diferentes proyectos que se han desarrollado durante los últimos tres años.

4.3 Metodología

Para la realización de esta tesis doctoral, se plantea utilizar una metodología de trabajo *Investigación-Acción* (N. Padak & G. Padak 1994). Esta metodología identifica los siguientes cuatro tipos de roles (véase Figura 11):

- El *equipo investigador* que lleva a cabo el proceso de investigación. Este rol va a ser llevado a cabo tanto por el doctorando como por el grupo de investigación LoUISE en el que se integra (UCLM).
- El *objeto* bajo investigación, es decir, el problema que debe ser resuelto. En este caso, el objeto de investigación es el desarrollo de la gestión del conocimiento para su explotación industrial.
- El *grupo crítico de referencia*, el cual recibe los resultados de la investigación y participan en el proceso de investigación (aunque menos activamente que el investigador). Este rol va a ser llevado a cabo por la empresa Eurocopter a fin de validar los resultados aplicables de la investigación.
- El *beneficiario* de la investigación es quien espera explotar los resultados de la investigación aunque no tome parte en el proceso. En este caso, este papel es llevado a cabo por cualquier colectivo involucrado en el desarrollo de software.

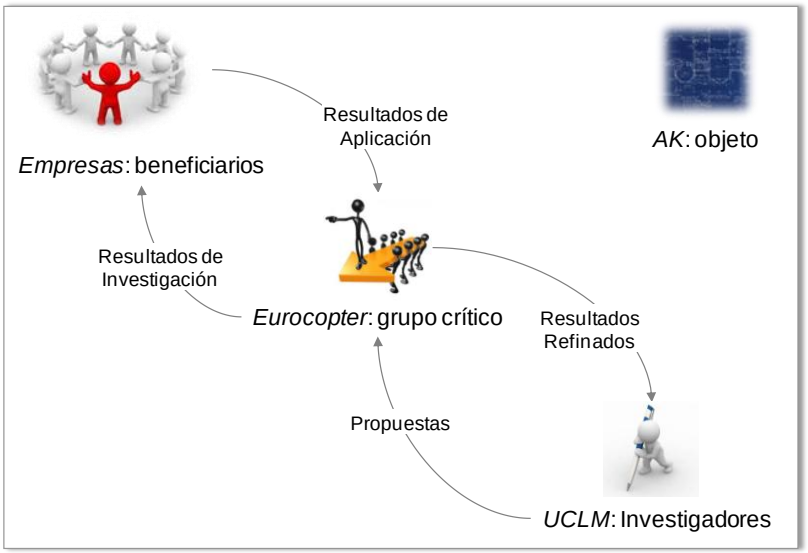


Figura 11. Aplicación de Acción-Investigación: Actores en el proyecto de tesis

4.4 Planificación

En función de la descripción del problema y de las cuestiones de investigación recogidas en este trabajo, identificamos la siguiente plantificación de tareas, representadas en la Figura 12:

- *Contexto*. Durante esta tarea se identificarán y analizarán trabajos relacionados presentados en foros nacionales e internacionales a fin de contextualizar apropiadamente la propuesta. Será está una actividad que se realice de forma reiterada a lo largo de toda la tesis doctoral.
- *MD-AK*. Esta tarea se centrará tanto en el estudio de MDD para AK como en el estudio de su utilidad para dar soporte a las actividades de visualización, análisis y evaluación. Por ello se realizará de forma reiterada a lo largo de toda la duración de la tesis.
- *Visualización*. Esta tarea está encaminada al estudio de técnicas de visualización del conocimiento arquitectónico. Se desarrollará principalmente, durante el primer año de tesis a fin de facilitar la realización de la tarea de análisis.
- *Análisis*. Esta tarea está orientada a la definición de nuevos mecanismos de análisis del conocimiento arquitectónico.
- *Evaluación*. Está tarea está definida para determinar nuevos mecanismos que permitan no sólo evaluar la calidad de la especificación arquitectónica sino de la propia red de conocimiento.
- *Soporte*. A lo largo de la presente tesis doctoral, se realizarán a su vez diferentes prototipos que ayuden a poner en práctica cada una de las aportaciones antes mencionadas, por ello se realizará en diferentes iteraciones a lo largo de esta tesis.
- *Validación*. A fin de determinar la validez de las diferentes propuestas planteadas, se llevarán a cabo diferentes prototipos utilizando la información que se ha recopilado a lo largo de estos tres años sobre el dominio de aplicación. Esta tarea se desarrollará de forma reiterada a lo largo de la tesis conforme se realicen avances de investigación.
- *Difusión*. A lo largo de todo el proceso de realización de la presente tesis doctoral, se realizarán diferentes publicaciones que permitan obtener el *feedback* necesario tanto de foros nacionales como internacionales. Además, también como parte de dicho proceso de difusión se realizará la escritura del documento de tesis.

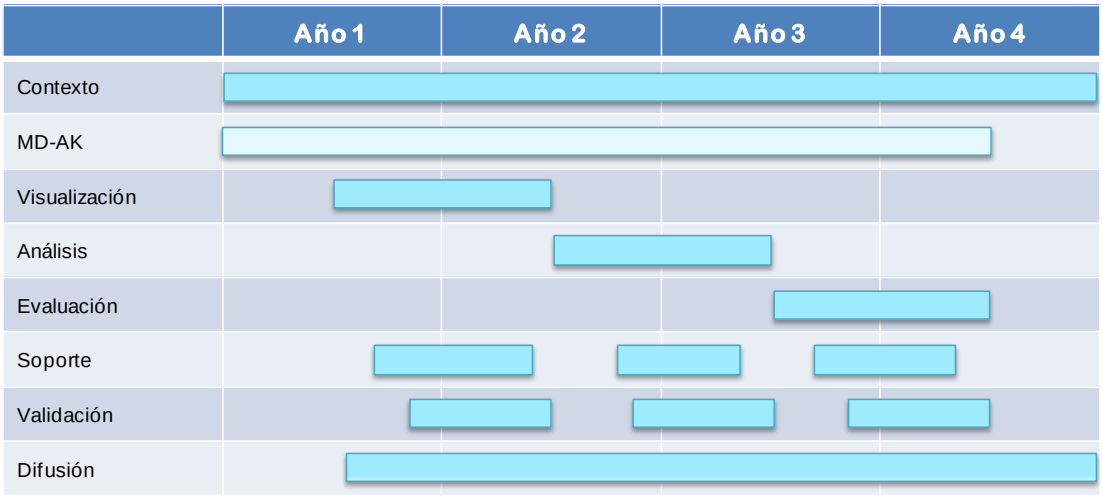


Figura 12. Planificación de la tesis

Estas actividades se llevarán a cabo de forma iterativa, siguiendo el ciclo que recomiendan (N. Padak & G. Padak 1994) de *planificar-actuar-observar-reflexionar*, permitiendo así dar soluciones más refinadas a lo largo de la realización de la presente tesis doctoral.

4.5 Tareas Realizadas

Durante el periodo lectivo de los estudios del máster de doctorado, se han abordado diferentes tareas: realización de trabajos relativos a las distintas asignaturas de dicho máster (véase capítulo 2); elaboración de diversos artículos científicos, relacionados con el estado del arte y el contexto en el que se enmarca esta tesis; así como la realización del presente Trabajo Fin de Máster.

A continuación, se describe el trabajo realizado en los distintos artículos científicos presentados, en los que se recogen, principalmente, resultados relativos a dos de las áreas principales de la futura tesis doctoral: AK dirigido por modelos (sección 4.5.1), técnicas de visualización (sección 4.5.2) y de análisis (sección 4.5.3).

4.5.1 Utilizando Técnicas de Transformación de Modelos para la superposición de Estilos Arquitectónicos

El trabajo presentado en este apartado, forma parte de la línea de trabajo relacionada con el AK dirigido por modelos, descrita en la sección 4.2.1. Este trabajo ha sido presentado como artículo científico en el siguiente foro:

- Elena Navarro, Carlos E. Cuesta, Dewayne E. Perry, Cristina Roda: *Using Model Transformation Techniques for the Superimposition of Architectural Styles*. Emerging paper in the 5th European Conference on Software Architecture (ECSA 2011), Essen, Germany, 13-16 September 2011 (aceptado).

Este trabajo se ha estructurado como sigue. En la sección 4.5.1.1, se presenta una breve introducción al mismo. En la sección 4.5.1.2 presenta una breve introducción al contexto metodológico en el que se ha elaborado esta propuesta, junto a un caso de estudio desarrollado para propósitos de validación. La sección 4.5.1.3 describe la manera en que nuestra propuesta introduce los estilos arquitectónicos en la generación de la proto-arquitectura (Brandozzi & D E Perry 2001), empleando técnicas de transformación de modelos. Finalmente, la sección 4.5.1.4 presenta el trabajo relacionado y la sección 4.5.1.5 expone las conclusiones.

4.5.1.1 Introducción

La especificación de la *Arquitectura Software* (*Software Architecture*, SA) siempre es un reto. Es un proceso de toma de decisiones que establece compromisos estrictos a nivel arquitectónico. Estos compromisos deben alcanzarse para elaborar una especificación capaz de satisfacer, tanto los requisitos funcionales, como los no funcionales. En este contexto, el uso apropiado de los *Estilos Arquitectónicos* (*Architectural Styles*) puede ser de gran ayuda para el proceso.

De acuerdo con (D. E. Perry & A. L. Wolf 1992), un estilo arquitectónico es algo que "abstrae elementos y aspectos formales a partir de varias arquitecturas específicas". Esta definición es deliberadamente abierta; puede ser utilizada para describir sistemas completos, o sólo un aspecto concreto de la arquitectura (D. E. Perry 1998), que puede estar compuesto por otros. Además, un estilo también puede especificar restricciones sobre dichos elementos y/o aspectos formales, (Shaw & David Garlan

1994) así como sobre el comportamiento del sistema. Por tanto, pueden afectar a la configuración de la arquitectura.

Sin embargo, a nuestro entender, existen muy pocas propuestas que claramente establezcan cómo pueden utilizarse los estilos arquitectónicos para describir la SA, ya sea de forma automática o semi-automática. Muchos artículos se han centrado en la clasificación de Estilos Arquitectónicos, o su evaluación; pero apenas se centran en su aplicación automática. Nuestra propuesta es, por tanto, un sistema que superponga automáticamente las restricciones obligatorias de un estilo arquitectónico sobre el sistema a construir.

4.5.1.2 Contexto de trabajo

Durante la descripción de la SA, el arquitecto debería ponderar el impacto de sus decisiones en el nivel arquitectónico y las relaciones que éstas tienen con otras decisiones y requisitos, antes de plasmarlas en el sistema. La metodología ATRIUM (Montero & Navarro 2009) proporciona soporte en este contexto. Ha sido diseñada utilizando el enfoque de *Desarrollo Dirigido por Modelos* (*Model-Driven Development*, MDD) y se compone de estas actividades:

- *Modelado de Requisitos*. Esta actividad permite al analista identificar y especificar los requisitos del sistema, definiendo el *modelo de objetivos de ATRIUM*, guiando al arquitecto desde los *objetivos* que el sistema debería alcanzar, hasta los *requisitos* que dicho sistema debería satisfacer, y las *operacionalizaciones* que describen las soluciones a los *requisitos* establecidos.
- *Modelado de Escenarios*. Esta actividad se centra en la identificación del conjunto de escenarios que definen el comportamiento del sistema bajo ciertas decisiones arquitectónicas, descritas en el modelo de objetivos de ATRIUM como *operacionalizaciones*.
- *Sintetizar y Transformar*. Esta actividad ha sido definida para generar la *proto-arquitectura* (Brandozzi & D E Perry 2001) de un sistema concreto. De esta forma, los elementos arquitectónicos que conforman el sistema se sintetizan, así como la estructura del sistema futuro, a partir del modelo de escenarios de ATRIUM.

Estas tres actividades deben ser iteradas con el fin de definir y refinar los distintos modelos, permitiendo al arquitecto reflexionar sobre los requisitos y la arquitectura. Una de las entradas de la actividad *Sintetizar y Transformar* es el Estilo Arquitectónico seleccionado. Se debe tener en cuenta que este Estilo Arquitectónico impone restricciones a la estructura y comportamiento de la descripción final. Por esta razón, esta actividad debe aplicar dichas restricciones de forma automática, ajustándose a ellas durante la generación de la arquitectura y, por tanto, evitando una tarea que puede resultar incómoda para el arquitecto y propensa a errores, si ésta se realiza a mano.

ATRIUM ha sido validado haciendo uso de un caso de estudio, el cual está asociado con el proyecto europeo *Environmental Friendly and cost-effective Technology for Coating Removal* (EFTCoR) (G3RD-CT-00794 2003). El objetivo de este proyecto es diseñar una familia de sistemas robóticos capaces de realizar operaciones de mantenimiento en el casco de los barcos. La *Unidad de Control de los Dispositivos Robóticos* (*Robotic Devices Control Unit*, RDCU) integra toda la funcionalidad requerida para gestionar el EFTCoR. Nos hemos centrado en la SA de este proyecto, debido a las estrictas restricciones que tiene que cumplir, en términos de seguridad, rendimiento y confiabilidad.

4.5.1.3 Utilizando Transformaciones Modelo-a-Modelo en Estilos Arquitectónicos

Como se presentó en la sección 4.5.1.2 anterior, los elementos arquitectónicos, su comportamiento, y la estructura de la proto-arquitectura (la salida de la actividad *Sintetizar y Transformar*) son sintetizados a partir del modelo de escenarios, considerando las restricciones impuestas por el estilo arquitectónico seleccionado.

Se han propuesto varios lenguajes para definir transformaciones *Modelo-a-Modelo* (*Model-to-Model*, M2M). Varios estudios, como el presentado por (Czarnecki & Helsen 2006), identifican las características que debería satisfacer un lenguaje de transformaciones M2M, para cumplir con el enfoque MDD. Considerando esto, se seleccionaron las Relaciones QVT (OMG doc. ptc/05-11-01 2005) como nuestro lenguaje de transformación M2M, dado que proporciona facilidades para gestionar el modelo fuente y el modelo objetivo; define una característica de *incrementalidad*, es decir, es capaz de actualizar el modelo generado, de acuerdo a los cambios en el modelo fuente; y ofrece *direccionalidad* y *trazabilidad*.

La característica *Organización de Reglas* de QVT ha sido utilizada para clasificar las distintas transformaciones. Concretamente, estas transformaciones han sido catalogadas como sigue:

- *Patrones de Generación Arquitectónica*. Describen aquellas transformaciones aplicables a la mayoría de los metamodelos arquitectónicos existentes, ya que se centran en la generación de componentes, conectores y sistemas.
- *Lenguajes* (*idioms*). Describen transformaciones de bajo nivel que son específicas de un metamodelo arquitectónico. Hemos hecho esta distinción para facilitar la generación de la proto-arquitectura, de acuerdo al metamodelo arquitectónico seleccionado por el arquitecto.
- *Estilos Arquitectónicos*. Estas transformaciones están orientadas a la aplicación de las restricciones impuestas por el Estilo Arquitectónico seleccionado durante la actividad *Modelado de Requisitos*.

Por tanto, utilizando el mismo modelo de escenarios, se pueden generar diferentes proto-arquitecturas, dependiendo del metamodelo arquitectónico seleccionado, mediante la aplicación de su lenguaje específico, y el Estilo Arquitectónico elegido.

Tabla 6. Cabecera de la transformación *ScenariosToArchmodelPRISMA*, la cual genera proto-arquitecturas PRISMA, a partir de los modelos de escenarios de ATRIUM

```
import archpatt;
import archAcrosetStyle;
transformation ScenariosToArchmodelPRISMA (scenarios: ATRIUMScenarios, archmodel: Archmodel)
    key archmodel::System {name};
    key archmodel::Component {name};
    key archmodel::Connector {name};
    key archmodel::Port {name, ArchitecturalElement};
    key archmodel::Attachment {name, System};
    key archmodel::Attachment {name, Architecturalmodel};
```

La Tabla 6 muestra la cabecera de la transformación encargada de generar proto-arquitecturas, a partir de los modelos de escenarios de ATRIUM. Como puede verse, los *patrones de Generación Arquitectónica* y los *patrones de Estilo Arquitectónico* se han importado. La Tabla 6 también ilustra la declaración de la transformación, por medio de un nombre (*ScenarioToArchmodelPRISMA*) y la identificación de los modelos candidatos. Existen dos modelos candidatos: *scenarios*, que representa un modelo candidato que se ajusta al metamodelo de Escenarios de ATRIUM; y *archmodel*, que representa un modelo candidato que se ajusta al metamodelo arquitectónico seleccionado para ser instanciado. Concretamente, hemos utilizado los metamodelos de *Escenarios de ATRIUM* y *PRISMA* (J. Pérez et al. 2006) para ejecutar las transformaciones. Además, la transformación puede definir *claves* para identificar unívocamente a los elementos y evitar instancias duplicadas.

A continuación, se describe en más detalle el proceso para llevar a cabo la superposición automática de Estilos Arquitectónicos, al tratar con las restricciones impuestas por éstos. Durante la actividad de *Modelado de Requisitos*, se selecciona el Estilo Arquitectónico; lo que significa que deben satisfacerse varias restricciones. La principal idea de esta propuesta es transformar estas restricciones en reglas de generación QVT, de tal forma que la proto-arquitectura generada es compatible con el Estilo Arquitectónico seleccionado.

El sistema EFTCoR utiliza un Estilo de Capas (Layered Style); concretamente, utilizamos ACROSET (Ortiz et al. 2005), una *Arquitectura Software Específica del Dominio* (*Domain Specific Software Architecture*, DSSA) que especializa el Estilo de Capas, identificando tres tipos de subsistemas (SUC, MUC y RUC). Este tipo de estilos especifica algunas restricciones, en términos de composición, al establecer que una capa sólo solicita los servicios proporcionados por la capa inferior.

Para generar la proto-arquitectura de EFTCoR, teniendo en cuenta las restricciones impuestas por el Estilo ACROSET, se definió una transformación (véase en Tabla 6 como *archAcrosetStyle*). Una transformación en QVT se define mediante un conjunto de *relaciones* que deben mantenerse para poder aplicar exitosamente la transformación. Una de estas relaciones es *ApplyingACROSET2Systems* (véase Figura 13).

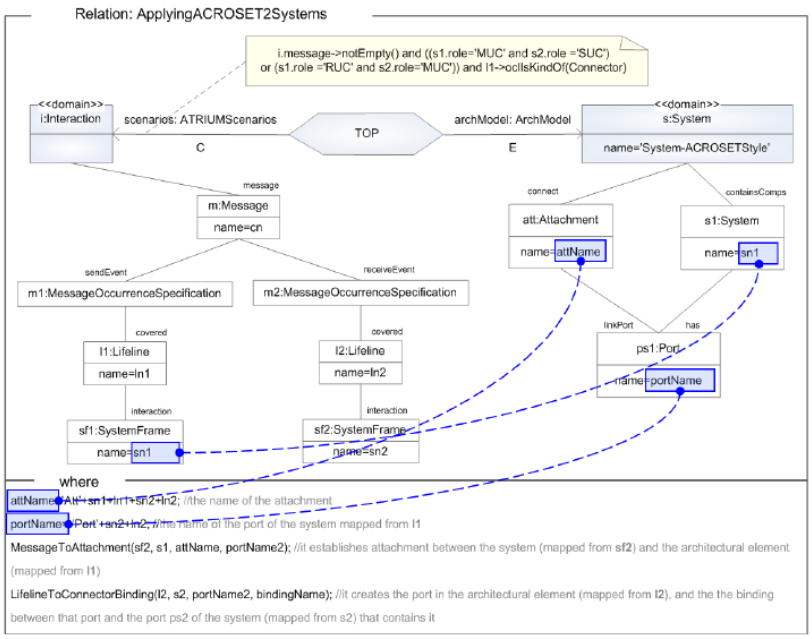


Figura 13. Relación QVT para superponer el Estilo ACROSET

La sintaxis gráfica de QVT ha sido utilizada para mejorar la legibilidad. El hexágono situado en el medio nos ayuda a representar la transformación, al identificar los modelos candidatos (en este caso, *scenarios* y *archmodel*) junto a sus metamodelos respectivos (*ATRIUMScenarios* y *Archmodel*). Cada flecha representa un dominio, etiquetado como *C* o *E* para determinar si la transformación es ejecutada en modo *checkonly* (sólo será comprobada si existe una condición válida que satisfaga la relación) o en modo *forzoso* (*enforce*, si la condición falla, el modelo objetivo será modificado para satisfacer la relación), respectivamente. Esto permite al arquitecto generar la proto-arquitectura o comprobar si surgen inconsistencias entre la proto-arquitectura y el modelo de escenarios.

Además, cuando se define una relación, podemos definir también cláusulas *where* y *when*. La cláusula *where* (véase Figura 13) especifica una condición que debe ser cumplida por todos los elementos involucrados en la relación, de tal forma que ésta pueda ser aplicada exitosamente.

Es necesario establecer los roles desempeñados por los elementos del modelo de escenarios, con respecto al Estilo Arquitectónico. Por esta razón, se define el atributo *role*, para que pueda ser utilizado por la expresión OCL en el dominio *scenarios* (véase de nuevo Figura 13), asegurando que la relación sólo se aplica cuando un ocurre una interacción entre elementos que pertenecen a las capas apropiadas. Esto significa que cualquier interacción entre otras capas no causará ningún tipo de generación en la proto-arquitectura.

La Figura 14 muestra un ejemplo del resultado de esta relación, cuando es aplicada sobre un Escenario de ATRIUM. En la parte superior de la figura, hay un escenario que establece cómo debería ser la colaboración entre los elementos arquitectónicos y ambientales, para satisfacer el requisito "REQ.6", de acuerdo a la operacionalización "OPE.13".

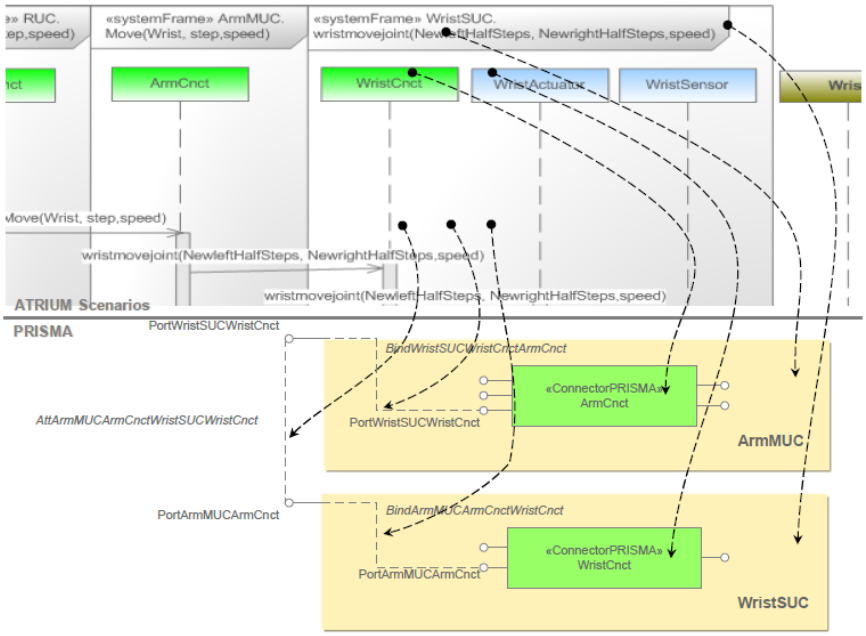


Figura 14. El Escenario de ATRIUM (arriba) y la proto-arquitectura generada (abajo)

Como se observa en la Figura 13, se establece una correspondencia entre *SystemFrame* y *System*, ya que ambos enlazan su atributo *name* con la misma variable. Esto significa que, cuando la Relación se aplica al Escenario de la Figura 14, un *System* PRISMA llamado *ArmMUC* se crea en el modelo PRISMA, dado que el *ArmMUC* del *SystemFrame* contiene la *Lifeline ArmCnct*. Tanto un *Puerto* (*Port*) como una *Relación* (*Attachment*) se resuelven en la cláusula *where*, y también se crean en el modelo arquitectónico. Estos dos elementos están relacionados para definir la conexión entre el sistema *s1* y el sistema *s2*, que serán creados utilizando la relación *MessageToAttachment* (véase Figura 15). Esta relación actúa de forma similar a *ApplyingACROSET2Systems* (véase Figura 13), generando el sistema *s2*, a partir del SystemFrame *sf2*, al coincidir sus nombres, y relacionando esto con el otro sistema generado, *s1*.

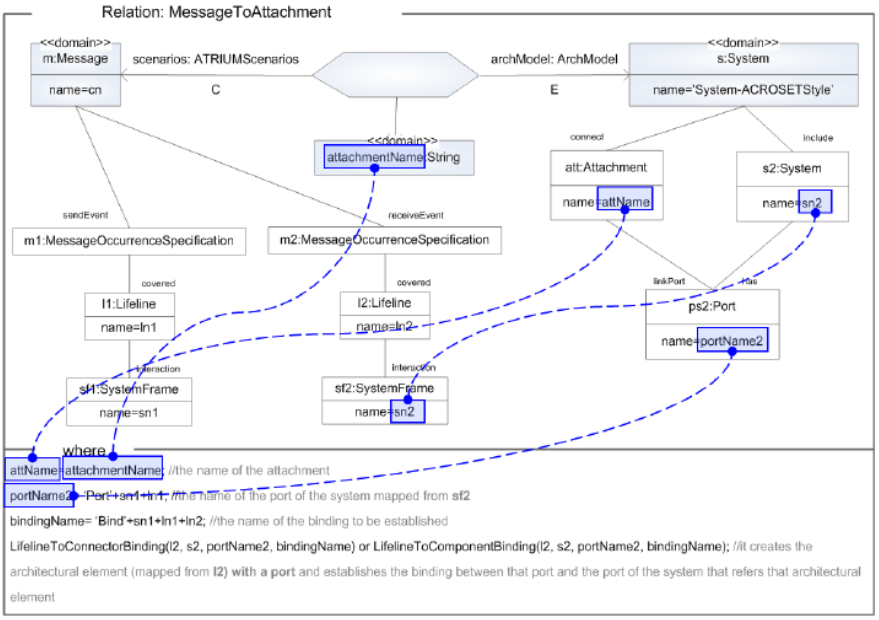


Figura 15. Relación QVT: Estableciendo conexiones entre Sistemas

En la cláusula *where* de ambas relaciones (*ApplyingACROSET2Systems* y *MessageToAttachment*), se especifica otra relación: *LifelineToConnectorBinding*. Esta relación ayuda a aplicar las restricciones al estilo ACROSET, esto es, la composición de los *Elementos Arquitectónicos*, pertenecientes a diferentes capas.

En resumen, hemos sido capaces de generar la proto-arquitectura sólo con un escenario ATRIUM. La característica de *incrementalidad* de las Relaciones QVT hace posible la actualización automática de la proto-arquitectura, dado que se definen escenarios nuevos o se modifican los existentes. Finalmente, cabe destacar que el proceso de desarrollo de ATRIUM está soportado completamente por la herramienta MORPHEUS (Navarro, Gómez, et al. 2009).

4.5.1.4 Trabajo relacionado

Los Estilos Arquitectónicos han recibido una atención considerable, tanto por parte de la academia como de la industria. El trabajo preliminar en este campo diseñó estructuras software especializadas en dominios concretos, como el control en aviónica o de misiles (Delisle & D Garlan 1989). Este trabajo inicial sobre las definiciones de Estilos Arquitectónicos es una de las piedras angulares de la SA.

Sin embargo, la mayoría de las propuestas sobre Estilo Arquitectónico han centrado su atención sobre temas recurrentes. La mayoría del trabajo ha sido orientado hacia la descripción de nuevos estilos arquitectónicos. Otros trabajos se han centrado en la clasificación de estilos arquitectónicos, mientras que otras propuestas han intentado definir primitivas para describir y/o componer estilos arquitectónicos (Mehta & N Medvidovic 2003)(U Zdun & P Avgeriou 2008). Pero, hasta lo que sabemos, ninguna propuesta ha prestado atención a la superposición automática de dichos estilos. Esta es la razón por la que, seguidamente, centramos nuestro análisis en propuestas sobre transformaciones de modelo, en el área de la SA.

Existen algunas propuestas que realmente intentan generar arquitecturas software. La propuesta de (De Bruin & Van Vliet 2003) describe un proceso para la generación de SA, tomando como entradas un rico grafo Característica-Solución y *Mapas de Casos de Uso* (*Use Case Maps*, UCM). Esta propuesta introduce los estilos arquitectónicos como una decisión en el espacio de soluciones, junto a fragmentos de decisión. Sin embargo, no tratan con la superposición automática del estilo arquitectónico, ni proporcionan detalles sobre cómo procede esta generación.

(Castro et al. 2002) han definido una metodología llamada TROPOS para guiar el proceso de especificación del sistema, a partir de los requisitos tempranos. Los requisitos se obtienen con el marco *i**. La metodología propone un proceso refinado, desde los requisitos hasta la SA. Sin embargo, el estilo arquitectónico (orientado a agentes) se aplica a mano.

El trabajo presentado por (Sánchez et al. 2010) es el único que utiliza una perspectiva generativa similar a la nuestra. Presentan un proceso que combina MDD con AOSD, para derivar Arquitecturas Orientadas a Aspectos a partir de modelos de Requisitos Orientados a Aspectos. Una vez que el conjunto de escenarios ha sido definido, se transforman en una Arquitectura Orientada a Aspectos mediante un conjunto de reglas de transformación, especificadas con QVT (OMG doc. ptc/05-11-01 2005). Sin embargo, esta propuesta no presta atención a la superposición de estilos arquitectónicos.

ATRIUM hace frente a muchas de las cuestiones exhibidas en estas propuestas. Los Estilos Arquitectónicos se seleccionan de acuerdo a las necesidades específicas del sistema, y se aplican automáticamente mediante transformaciones M2M. También ha sido diseñado para poder utilizar diferentes metamodelos arquitectónicos.

4.5.1.5 Conclusiones y Trabajo Futuro

ATRIUM pretende generar la proto-arquitectura del sistema por medio de transformaciones M2M. De esta forma, QVT emerge como la mejor solución para este propósito, ya que satisface la mayoría de los requisitos.

Al usar Relaciones QVT, se definen un conjunto de transformaciones para generar la proto-arquitectura, a partir del modelo de escenarios de ATRIUM. Proporciona varias ventajas. La primera está asociada con su aplicabilidad al conjunto de escenarios al completo. QVT soporta la definición de claves, que garantizan que el proceso de síntesis impida la creación de objetos duplicados. Además, no es necesario proporcionar el conjunto completo de escenarios para generar la proto-arquitectura. De hecho, la generación se puede crear con sólo un escenario. Gracias a la característica de *incrementalidad* de QVT, la proto-arquitectura generada puede actualizarse automáticamente cuando se definan nuevos escenarios.

Uno de los principales artefactos en la definición de ATRIUM es la trazabilidad. Se proporciona trazabilidad de arriba hacia abajo, ya que la proto-arquitectura se genera automáticamente al establecer las transformaciones adecuadas. La trazabilidad de abajo hacia arriba puede alcanzarse, ya que las Relaciones QVT crean una *Clase de Traza*, a partir de cada relación, de tal forma que se generan mapas de trazado. Esta habilidad es muy valiosa ya que se establece un mapeo entre cada elemento de la proto-arquitectura y sus elementos asociados en el modelo de Escenarios de ATRIUM.

Otro de los retos de nuestra propuesta es cómo establecer mecanismos para evaluar la proto-arquitectura obtenida. Actualmente, nos estamos centrando en cómo detectar defectos mientras se está generando la proto-arquitectura. Una detección temprana de dichos defectos mejorará significativamente el desarrollo, en cuanto a calidad y coste. La definición de un *modelo de evaluación* que describa los defectos potenciales a nivel de especificación, sería el primer paso en esta dirección.

4.5.2 Una evaluación empírica de técnicas de visualización para Conocimiento Arquitectónico

El trabajo presentado a continuación, forma parte de la línea de trabajo relacionada con el estudio de técnicas de visualización, descrita en la sección 4.2.2. Este trabajo ha sido presentado como artículo científico en dos foros:

- Cristina Roda, Elena Navarro, Carlos E. Cuesta y Dewayne E. Perry, *Técnicas de Visualización para Conocimiento Arquitectónico: una Evaluación Empírica*, XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2011), A Coruña, 5-7 Septiembre 2011 (aceptado).
- Cristina Roda, Elena Navarro, Carlos E. Cuesta y Dewayne E. Perry, *An Empirical Evaluation of Visualization Techniques for Architectural Knowledge*, IET Software (enviado).

Este trabajo se ha estructurado como sigue. En la sección 4.5.2.1, se presenta la clasificación de las técnicas de visualización utilizada en este trabajo, junto con la identificación de qué herramientas encajan en cada una de estas técnicas. En la sección 4.5.2.2, se presenta el material experimental utilizado para llevar a cabo la evaluación empírica. En la sección 4.5.2.3, se describe el experimento llevado a cabo, es decir, la evaluación empírica realizada. Finalmente, en el apartado 4.5.2.4, se presentan las distintas conclusiones y el trabajo futuro.

4.5.2.1 Técnicas de visualización para el Conocimiento Arquitectónico

Diversas técnicas de visualización pueden ser utilizadas para capturar, representar o mantener el AK. Queremos resaltar que una de las bases de nuestra investigación es que el AK es, per se, una sistema de conocimiento compuesto por ADDs y ADRs y sus correspondientes relaciones, las cuales pueden ser utilizadas para comprender y reflexionar sobre la arquitectura software de un sistema. En este sentido, se asemeja a una ontología (Gruber 1993), como otros autores ya han notado (De Boer et al. 2009). Esto nos ha llevado a utilizar como taxonomía de técnicas de visualización aquella propuesta por (Katifori et al. 2007), la cual distingue cinco tipos de representación diferentes, dependiendo de la presentación de la información, el método de interacción, o la funcionalidad soportada. En las siguientes subsecciones, se describe cada uno de estos tipos, junto a sus herramientas disponibles.

Este trabajo se centra en herramientas bidimensionales, principalmente porque resultan familiares, es decir, similares a las usadas comúnmente por los arquitectos. Las sub-secciones 4.5.2.1.1, 4.5.2.1.2, 4.5.2.1.3 presentan herramientas de visualización de AK, mientras que los apartados 4.5.2.1.4 y 4.5.2.1.5 presentan herramientas ontológicas. Hemos incluido las últimas dos técnicas porque presentan la información de una forma jerárquica, muy similar a las representaciones utilizadas por herramientas específicas de AK. Nótese que se prefieren las herramientas de AK, siempre que estén disponibles para una técnica de visualización particular; si no existen herramientas de AK disponibles, se prefieren las ontológicas frente a cualquier otro tipo.

4.5.2.1.1 Indented list

De acuerdo a este tipo de representación, el AK es representado mediante texto plano en una vista de árbol, similar al explorador de Windows, es decir, una lista indentada. La simplicidad de esta representación textual hace que este método no sea muy popular hoy día para representar AK. *PHI* (*Procedural Hierarchy of Issues*, (R. McCall 1991)) es un sistema AK que usa esta técnica de visualización, el cual extiende al sistema IBIS y presenta un enfoque de argumentación para resolver problemas. Algunas herramientas que soportan la metodología PHI, según (Regli et al. 2000), son: *JANUS* (Regli et al. 2000)(Fischer et al. 1990), *HOS* (*Hyper-Object Substrate*) (Regli et al. 2000), y *PHIDIAS* (*Procedural Hierarchy of Issues/Design Intelligence Augmentation*) (Regli et al. 2000). Sin embargo, actualmente, estas herramientas no tienen ningún soporte, por lo que se van a considerar herramientas ontológicas que ofrezcan una vista de árbol similar al explorador de Windows.

Protégé (N. F. Noy et al. 2000) es un entorno de edición de ontologías y adquisición de conocimiento (véase Figura 16). *KAON* (FZI 2011) es un entorno de gestión de ontologías de código abierto para aplicaciones de negocio. *OntoRama* (Eklund et al. 2002) permite a los usuarios visualizar una estructura (ontológica) de conocimiento en un diseño hiperbólico. *OntoEdit* (Sure et al. 2002) es un entorno de ingeniería de ontologías que combina el desarrollo de ontologías basadas en metodologías, con capacidad para colaboración e inferencia. De entre estas herramientas, se eligió *Protégé* para la evaluación de esta técnica de visualización, dado que es de código abierto y ha sido citada por otros autores, como (Jansen et al. 2009) y (Katifori et al. 2007), en el contexto de AK.

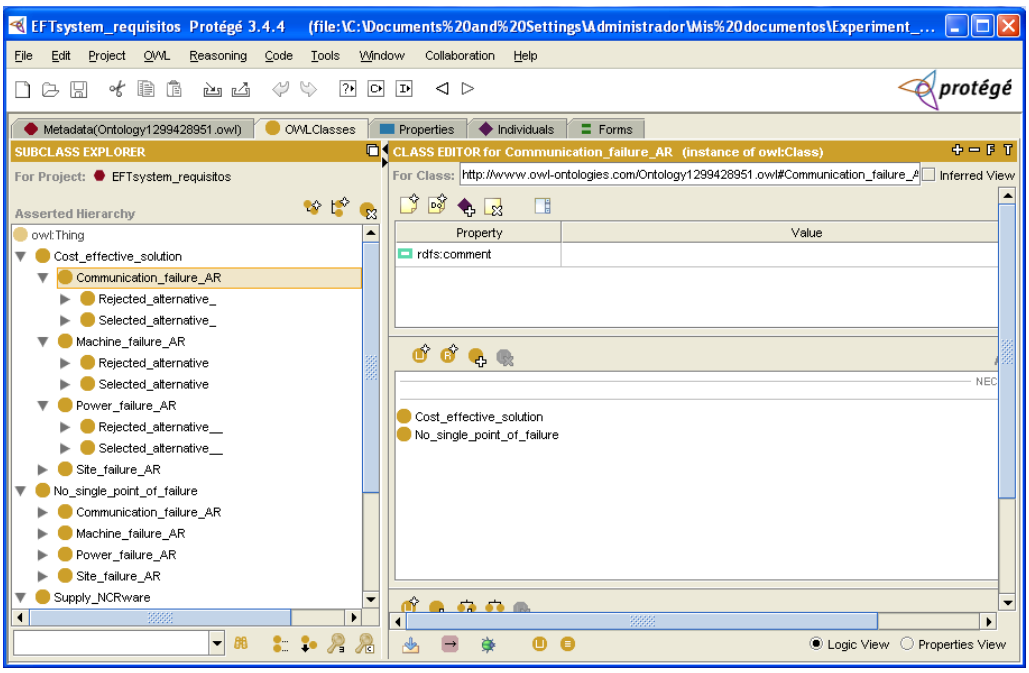


Figura 16. Protégé 3.4.4

4.5.2.1.2 Wiki

Como indican (Farenhorst, Lago & H. V. Vliet 2007), una *wiki* permite a los diseñadores y arquitectos colaborar y comunicarse entre sí fácilmente. Por tanto, la información puede actualizarse rápidamente, y los stakeholders pueden saber en cualquier momento el estado actual del proyecto. Algunas herramientas de este tipo son: *C-ReCS* (*Collaborative Requirements Capture System*, (M. Klein 1997)), sistema que soporta colaboración durante la captura de ADDs; *PAKME* (*Process-based Architecture Knowledge Management Environment*, (Babar et al. 2006)), herramienta basada en la web que soporta colaboración para gestionar el AK; *ADDSS* (*Architecture Design Decision Support System*, (Capilla et al. 2006)), herramienta basada en la web, al igual que PAKME, que permite gestionar y documentar ADDs (véase Figura 17); y *Knowledge Architect* (Jansen, Vries, et al. 2008), herramienta que proporciona mecanismos para la captura, gestión y compartición de AK, gracias a un servidor y repositorio de AK.

En este caso, ADDSS ha sido la herramienta seleccionada para llevar a cabo la evaluación presentada en la sección 4.5.2.3. Su selección frente a otras herramientas viene motivada porque es la más completa y proporciona un sistema de consultas que permite a los arquitectos encontrar fácilmente información.

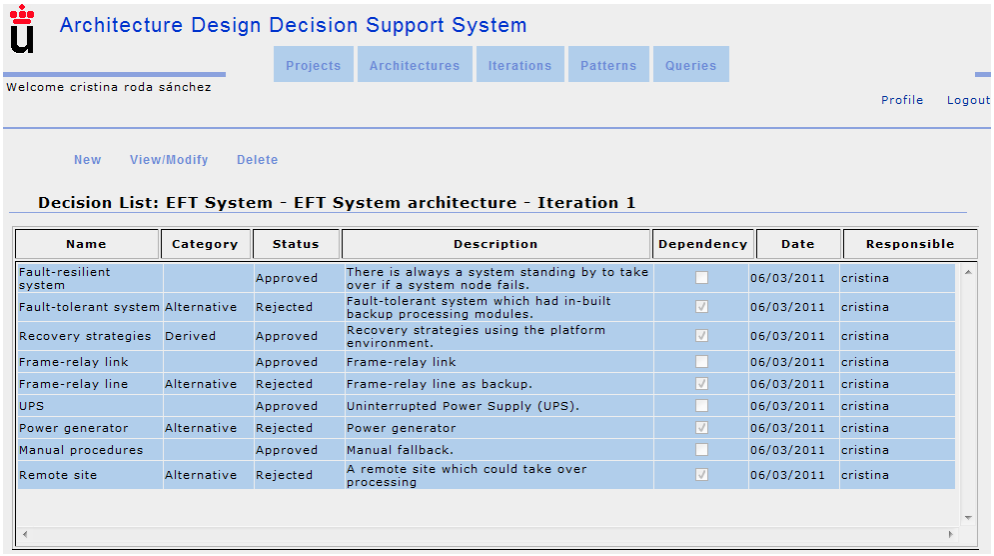


Figura 17. ADDSS 2.0

4.5.2.1.3 Node-link and tree

Este enfoque proporciona una representación de nodos interconectados, que permite a los usuarios expandir/contraer nodos, regulando el nivel de detalle de la información. Algunas herramientas

correspondientes a esta categoría, ordenadas alfabéticamente, son: *ARCHIUM*[4] (Jansen & Bosch 2005), una extensión Java que proporciona trazabilidad a través de un amplio rango de conceptos, como requisitos o decisiones; *AREL* (*Architecture Rationale and Element Linkage*, (Tang et al. 2007)), herramienta basada en UML para ayudar a los arquitectos a crear y documentar diseños arquitectónicos, centrándose en ADDs y ADRs; *Compendium*[5] (véase Figura 18), herramienta de código abierto, implementada en base a IBIS y con soporte a notaciones gIBIS; *DRIMER* (*Design Recommendation and Intent Model Extended to Reusability*, (Peña-Mora & Vadhavkar 1997)), herramienta que proporciona captura explícita de ADRs durante el proceso de desarrollo de software; *DRL* (*Design Representation Language*, (J. Lee 1990)), lenguaje que permite construir grafos de decisión, reflejando los pros y contras a la hora de evaluar alternativas con respecto a los objetivos; *gIBIS* (*graphical IBIS*, (Conklin & Begeman 1988)), herramienta de visualización de AK, sucesora de IBIS, que utiliza color y una base de datos relacional para facilitar la construcción y exploración de las redes IBIS; *IBIS* (*Issue-Based Information System*, (Kunz & Rittel 1970)), enfoque basado en argumentos para representar la rationale de diseño, basándose en tres conceptos básicos: *Cuestiones* (*Issues*) a tratar, *Posiciones* que responden a dichas cuestiones, y *Argumentos* compuestos por *Pros* (argumentos a favor) y *Contras* (argumentos en contra) de una *posición* concreta; *Kruchten's ADD Ontology tool* (L. Lee & Kruchten 2008), herramienta basada en la ontología de Kruchten para las ADDs, por lo que facilita, tanto la exploración como el análisis detallado de decisiones, gracias a cuatro vistas; *ODV* (*Ontology-Driven Visualization*, (De Boer et al. 2009)), herramienta que combina la potencia de dos de las vistas de la herramienta anterior; *QOC* (*Questions, Options and Criteria*, (Maclean et al. 1991)), enfoque basado en notación semi-formal para analizar el espacio de diseño, utilizando tres elementos principales: *Cuestiones* (*Questions*), o sujetos de diseño claves, *Opciones* (*Options*), o posibles respuestas a las cuestiones, y *Criterios* (*Criteria*), o valoración de las opciones; *SCRAM* (*SCenario Requirements Analysis Method*, (Sutcliffe & Ryan 1998)), método de análisis de requisitos basado en escenarios; *SEURAT* (*Software Engineering Using RATionale*, (Burge & Brown 2004)), plug-in del entorno de desarrollo Eclipse que captura y utiliza la rationale de diseño, al enlazar su código software; por último, *Sysiphus* (Bruegge et al. 2006), suite de herramientas basadas en la rationale que permite capturar varios modelos del sistema para actividades de desarrollo del mismo, y soportar ADRs, enlazándolas con dichos modelos mediante grafos.

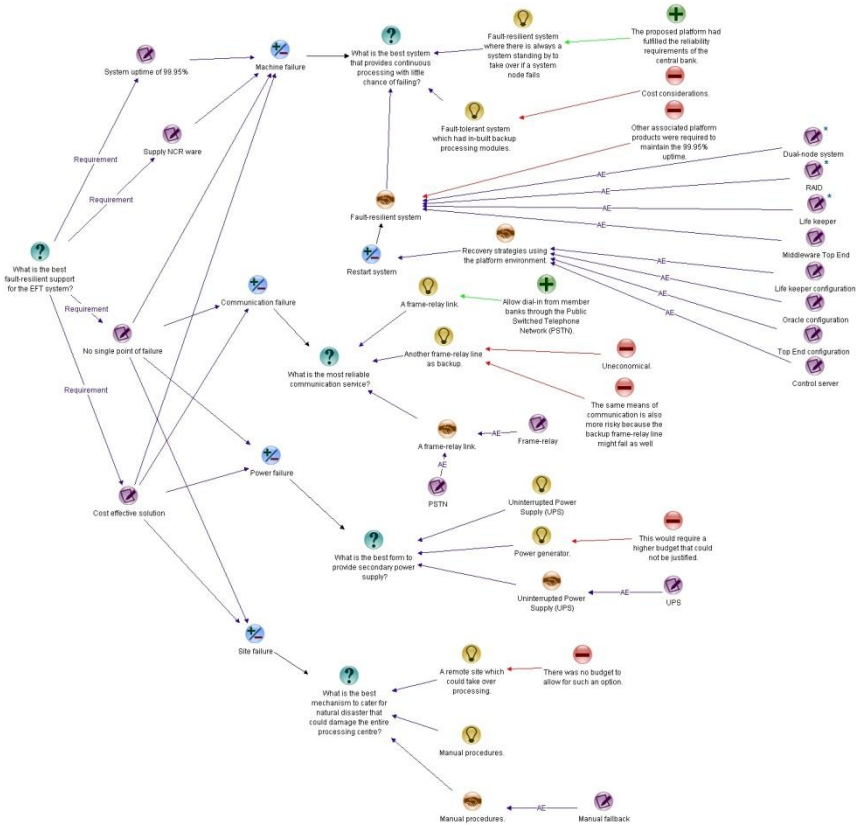


Figura 18. Compendium 1.5.2

Entre todas las herramientas analizadas, *Compendium* fue la elegida para representar a esta técnica de visualización y llevar a cabo la evaluación presentada en la sección 4.5.2.3, ya que proporciona soporte

[4] http://www.archium.net.

[5] http://compendium.open.ac.uk/institute/

explícito a la visualización de la rationale de las ADDs, presentando este AK en forma de grafo dirigido, lo que permite a los usuarios navegar y explorar fácilmente la red de decisión.

4.5.2.1.4 Zoomable

Esta técnica de visualización es especialmente interesante por la representación jerárquica que utiliza. Los nodos de los niveles más bajos de la jerarquía están anidados dentro de sus padres, con un tamaño menor que éstos. De esta forma, tendremos que hacer zoom en los nodos hijos para expandirlos y hacer que sean el nivel de visualización actual (Katifori et al. 2007). En este caso no existen herramientas específicas de AK en esta categoría por lo que se tuvieron en cuenta las siguientes herramientas de visualización de ontologías: *Grokker* (Rivadeneira & Bederson 2003), un mapa gráfico de conocimiento, donde la información se representa gráficamente; *Jambalaya* (Storey et al. 2001) un plug-in de *SHriMP* (*Simple Hierarchical Multi-Perspective*) (Wu & Storey 2000) integrado en la herramienta *Protégé* (véase Figura 19); y *CropCircles* (Parsia et al. 2005), que presenta las jerarquías de clases en ontologías como árboles.

En este caso, se ha seleccionado la herramienta *Jambalaya*, dado que tiene mayor soporte que el resto de herramientas y además está integrada en *Protégé*, herramienta seleccionada previamente.

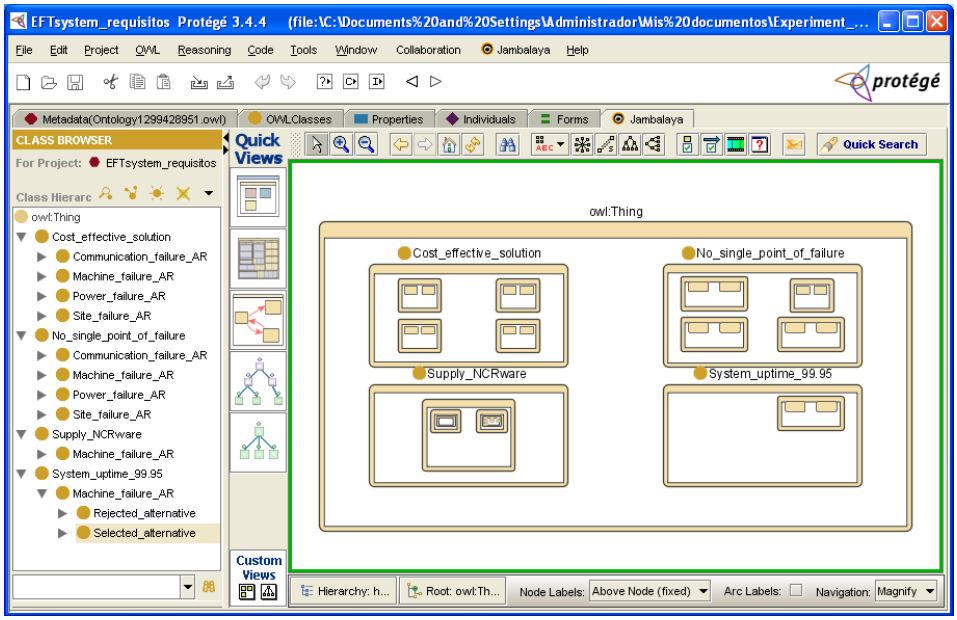


Figura 19. Pestaña de Jambalaya en Protégé 3.4.4

4.5.2.1.5 Space-filling

Esta categoría presenta los nodos de forma jerárquica, ajustándolos a la pantalla. No es considerada una técnica muy interesante por su falta de claridad y su complicada conexión con la arquitectura software. Nótese que no hay herramientas de visualización de AK específicas para esta categoría, por lo que se estudiaron las siguientes de tipo ontológico: *TreeMaps* (Shneiderman 1992), herramienta que permite representar jerarquías o árboles que tengan algún peso o tamaño en los nodos hoja, los cuales son rectángulos, cuyo área es proporcional a algún atributo, como el tamaño del nodo; *SequoiaView* (Eindhoven 2002), herramienta donde la pantalla se subdivide en rectángulos, aproximándose a cuadrados tanto como sea posible (véase Figura 20); *Information Slices* (Andrews & Heidegger 1998), herramienta que dibuja estructuras jerárquicas, utilizando discos semi-circulares, los cuales representan múltiples niveles de la jerarquía.

En este caso, *Information Slices* se descarta porque no tiene suficiente soporte. Con respecto a las otras dos herramientas, se selecciona *SequoiaView*, dado que tiene más soporte software y de mantenimiento que *TreeMaps*, es de libre distribución y proporciona una búsqueda de ficheros y un mecanismo de filtrado más eficientes.

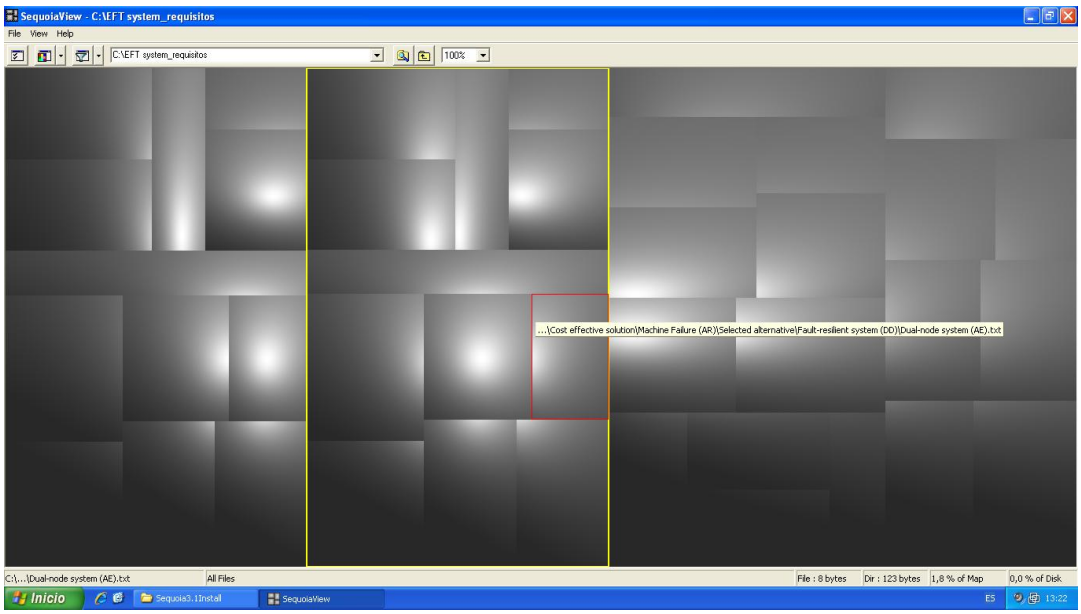


Figura 20. SequoiaView 1.3

4.5.2.2 Material experimental

Nuestra evaluación empírica utiliza la arquitectura software y el AK de un caso de estudio presentado en (Tang 2007) (Tang et al. 2007), asociado a un sistema de control financiero. La sucursal PBC-GZ es una rama del banco central responsable del control financiero y de pagos y liquidaciones interbancarias del centro financiero de Guangzhou y sus alrededores. Uno de sus sistemas es el *EFT* (*Electronic Fund Transfer*), que transfiere y liquida pagos de alto valor entre todos los bancos especializados y comerciales de sus alrededores. Este sistema tiene que servir a más de diez millones de personas en el sur de China, y trabaja como una pasarela que conecta a todos los bancos locales con la red de pagos nacional.

El diseño, desarrollo y prueba del sistema EFT llevó cerca de dos años, empleando treinta diseñadores y desarrolladores. Su diseño fue muy exigente dado que era necesario que fuera un sistema fiable, eficiente y seguro, debido a que es el núcleo principal del sistema financiero de la región. El principal problema que presentaba este sistema era que su diseño era difícil de entender para alguien externo al equipo de desarrollo original, a pesar del hecho de que su diseño fue ampliamente especificado. Por esta razón, se decidió capturar el AK (ADDs y ADRs) para que cualquier persona pudiera interpretar el diseño del sistema EFT.

El AK del sistema EFT ha sido utilizado según se indica en la sección 4.5.2.3 por un conjunto de arquitectos noveles seleccionados para llevar a cabo la evaluación de las diferentes alternativas para la visualización de AK. Por tanto, con ayuda de estas personas, podremos analizar fortalezas y debilidades de las herramientas de visualización de AK, seleccionadas en la sección anterior, que ilustran las ADDs y ADRs del sistema EFT.

4.5.2.3 Evaluación empírica

En este trabajo, se presentan múltiples casos de estudio exploratorios (Yin 2002), en los que cada arquitecto software debe realizar algunos cambios sobre la arquitectura del sistema EFT, utilizando las diferentes herramientas seleccionadas previamente para ayudarles en la elaboración de dichos cambios. Después de haber completado con éxito el conjunto de cambios, cada arquitecto evaluó la utilidad de cada una de las herramientas de visualización.

Nótese que, para presentar esta evaluación empírica, se ha seguido la estructura que propone (Wohlin et al. 2000) para llevar a cabo un experimento, dado que resulta una guía clara para este propósito. Además, hemos prestado especial atención a los consejos proporcionados por (B. A. Kitchenham et al. 2002).

4.5.2.3.1 Definición

4.5.2.3.1.1 Definición del objetivo

En general, el objetivo de este estudio es evaluar qué técnica de visualización es la más efectiva utilizando AK a la hora de realizar una serie de cambios arquitectónicos (esto es, toma y retoma de ADDs) a una SA existente, usando las distintas herramientas de visualización para representar el AK de dicha SA.

Así, siguiendo la plantilla GQM (*Goal/Question/Metric*) de Basili y Rombach, vamos a definir más formalmente el objetivo del estudio, determinando sus cinco parámetros:

- *Objeto de estudio*. Las técnicas de visualización 2D y su efectividad, en términos de la utilización de AK a la hora de realizar una serie de cambios arquitectónicos a la arquitectura del sistema EFT.
- *Propósito*. Evaluar las técnicas de visualización, presentadas en la sección 4.5.2.1.
- *Perspectiva*. Desde el punto de vista de los arquitectos software, es decir, el arquitecto software quiere saber si existe alguna diferencia entre las cinco técnicas de visualización cuando representan y capturan el AK.
- *Foco de calidad*. El principal efecto estudiado en este estudio es la efectividad de las técnicas de visualización, enfatizando el factor usabilidad.
- *Contexto*. Los casos de estudio se ejecutan en el contexto de la sucursal del Banco Popular de China en Guangzhou (PBC-GZ), y se llevan a cabo en al ámbito de una actividad de evolución de la arquitectura del sistema EFT del PBC-GZ. Así pues, un grupo de estudio, que hará las veces de un conjunto de arquitectos software, deberá realizar algunos cambios arquitectónicos. La caracterización del contexto experimental se corresponde con un "estudio sujeto-objeto bloqueado" (Wohlin et al. 2000), dado que hay más de un sujeto (arquitectos software, en principio) por objeto, y más de un objeto (cinco técnicas de visualización). Finalmente, se consideró un conjunto de estudiantes (no aleatorio), debido a la imposibilidad de conseguir un gran número de arquitectos software y el alto coste que ello implicaba.

De esta forma, a partir de esta definición, el estudio se puede considerar como múltiples casos de estudio exploratorios, donde cada sujeto es un caso de estudio individual, en lugar de un experimento controlado.

4.5.2.3.1.2 Resumen de la definición

A continuación, se presenta el resumen de la definición del objetivo del estudio. El propósito de este estudio es evaluar qué técnica de visualización es la más efectiva utilizando Conocimiento Arquitectónico a la hora de realizar una serie de cambios arquitectónicos, centrándose en la efectividad de las técnicas de visualización, desde el punto de vista de los arquitectos software, en el contexto de la sucursal de Guangzhou del Banco Popular de China.

4.5.2.3.2 Planificación

4.5.2.3.2.1 Selección del contexto

El contexto del estudio es una actividad de evolución de la SA del sistema EFT, donde dicho sistema está ubicado en el PBC-GZ. Obviamente, no podemos disponer de los arquitectos software originales del sistema EFT, por lo que simulamos esta actividad de evolución con estudiantes experimentados. Por tanto, estos casos de estudio se ejecutaron off-line (desarrollo de software no industrial), y se llevaron a cabo en la Universidad de Castilla-La Mancha (Albacete, España).

Los distintos casos de estudio se realizaron en el contexto de una sesión práctica, donde los sujetos fueron informados sobre cómo y qué debían hacer y, posteriormente, serían capaces de ejecutar distintas tareas para llevar a cabo el caso de estudio concreto.

4.5.2.3.2.2 Formulación de la hipótesis

Nuestra hipótesis nula, si estuviéramos haciendo un experimento controlado, sería que no existe diferencia en la efectividad de las herramientas de visualización, es decir, éstas serían igualmente efectivas. Nuestra intuición, sin embargo, es que la técnica Node-link and tree es la más efectiva a la hora de representar AK, dado que su vista gráfica proporciona más información que las otras técnicas. Así, lo que queremos es rechazar dicha hipótesis nula para hacer correcta nuestra intuición inicial. De esta manera, nuestra formulación formal de la hipótesis podría ser la siguiente:

- *Hipótesis nula,* H_0: no existe diferencia en cuanto a la efectividad (medida gracias a la puntuación media proporcionada por el cuestionario presentado en (Lund 2001)) entre las cinco técnicas de visualización.
- *Hipótesis alternativa,* H_1: existe diferencia en cuanto a la efectividad entre, al menos, un par de técnicas de visualización.

Aunque nuestros casos de estudio exploratorios soporten esta intuición de forma individual, debemos, no obstante, elaborar un análisis general de los datos, cuyos resultados soporten claramente esta

conclusión. Es por esto que son necesarias algunas medidas: *técnica de visualización* (Wiki, Node-link and tree, Indented list, Zoomable, Space-filling), y *puntuación media de la efectividad*. A la vista de estas suposiciones, se requiere la recogida de los siguientes datos:

- *Técnica de visualización*: medida por Wiki, Node-link and tree, Indented list, Zoomable o Space-filling (escala nominal).
- *Puntuación media de la efectividad*: medida por una escala numérica, utilizando números reales en el rango 1-7.

4.5.2.3.2.3 Selección de variables

La *variable independiente* (entrada) en todos los casos de estudio es *el AK de la arquitectura del sistema EFT*, representada por las distintas herramientas de visualización.

La *variable dependiente* (salida determinada por los distintos tratamientos – los cambios arquitectónicos) es la *efectividad* de las diversas herramientas de visualización, como respuesta a los distintos tratamientos. El cuestionario presentado en (Lund 2001), ha sido utilizado para determinar la efectividad de las herramientas de visualización cuando representaban el AK del sistema EFT. Para cada herramienta de visualización, se creó un cuestionario, utilizando la plataforma de e-learning llamada *Moodle*, la cual está disponible en la Universidad de Castilla-La Mancha para apoyo al profesorado. Esta plataforma nos permitió recolectar todos los resultados de los cuestionarios para su posterior análisis.

Nótese que la variable dependiente nos permite medir la puntuación media de la efectividad para cada herramienta de visualización, dado que podemos calcular el promedio de los datos recolectados a partir de las respuestas al cuestionario USE.

4.5.2.3.2.4 Selección de sujetos

Como se comentó antes, resultaba imposible disponer de arquitectos software en este contexto, dado que implicaba un gran esfuerzo, tanto económico como para reunir al personal. Por tanto, "reclutar" estudiantes era la mejor opción para llevar a cabo el estudio, convirtiéndose así en nuestro grupo de estudio.

Los estudiantes seleccionados pertenecían a cursos avanzados de carreras de ingeniería informática. Esto se debe a que eran necesarios sujetos con experiencia, dado que, de otro modo, no hubieran podido realizar el correspondiente caso de estudio.

Por eso, siguiendo el trabajo de (Wohlin et al. 2000), nuestro grupo de estudio consistía en una muestra no probabilística y de conveniencia, ya que estaba compuesta por los sujetos más cercanos y convenientes, es decir, estudiantes.

4.5.2.3.2.5 Diseño del experimento

Cada caso de estudio exploratorio consistió en 1) realizar varias tareas (cada una considerada como un tratamiento aplicado a la SA del sistema EFT) que, normalmente, se llevan a cabo por arquitectos software cuando evolucionan un sistema, y 2) evaluar la efectividad de las distintas técnicas, utilizando el cuestionario presentado en (Lund 2001). Estas tareas estaban relacionadas con la modificación del AK o de la SA del proyecto EFT descrito en la sección 4.5.2.2.

El grupo de estudio estaba formado por 15 estudiantes (tres mujeres y doce hombres) pertenecientes al último curso del Grado en Ingeniería Informática en la Universidad de Castilla-La Mancha, cuyas edades estaban comprendidas entre los 22 y los 25 años. Cada estudiante llevó a cabo el diseño del caso de estudio, realizando las tareas y la evaluación de la efectividad.

Los casos de estudio exploratorios se realizaron en el contexto de una sesión práctica de dos horas de la asignatura Interacción Persona-Ordenador II, por lo que los sujetos estaban familiarizados con términos como (usabilidad) efectividad y técnica de visualización. La información recogida durante los casos de estudio fue analizada utilizando técnicas estadísticas estándares, como se describe en la sección 4.5.2.3.4.

Nótese que, cuando realizamos este estudio, se crearon cinco grupos diferentes de estudiantes (de 2-3 miembros), de tal forma que cada grupo empezaba y terminaba con una herramienta diferente, aliviando el efecto aprendizaje. Así, el orden de aplicación de los tratamientos había sido controlado, ya que éste era diferente para los distintos grupos. Además, se redujo el efecto fatiga, ya que, como las tareas no eran muy complejas, podían completarse en poco tiempo, y el uso de diferentes técnicas de visualización ayudó a motivar a los estudiantes, así como también la introducción de diferentes tareas.

4.5.2.3.2.6 Instrumentación

Para preparar los casos de estudio, se presentan los antecedentes de la SA del sistema EFT y las cuestiones básicas relativas a la creación de dicha SA y su AK asociado:

- La selección del sistema y la plataforma de la arquitectura software es una de las cuestiones arquitectónicas más relevantes. El arquitecto debe tener en cuenta que el sistema EFT ofrece *soporte a la resistencia a fallos* (*fault-resilient support*), por lo que ha de incorporar un sistema para procesamiento continuo con pocas posibilidades de fallo. Hay dos opciones posibles: un *sistema resistente a fallos* que tenga siempre un nodo listo para asumir el control si otro nodo falla; o un *sistema tolerante a fallos* que tenga módulos de procesamiento de copia de seguridad incorporados. Para el sistema EFT, los arquitectos seleccionaron un sistema resistente a fallos porque cumplía con los requisitos de confiabilidad del cliente, evitando el alto coste de un sistema tolerante a fallos que lo que convertía en un candidato poco atractivo. Sin embargo, el sistema resistente a fallos conlleva una desventaja: otros productos asociados a la plataforma son requeridos para mantener el tiempo de actividad un 99.95%. De esta forma, el arquitecto tenía que tomar otra decisión: qué estrategias de recuperación era necesario implementar.
- Los arquitectos también tuvieron que prestar atención a la *confiabilidad de la red*, ya que las operaciones bancarias deben llevarse a cabo en un entorno seguro. Para ello, consideraron dos opciones: introducir un *enlace frame-relay* en el sistema; o introducir otra *línea frame-relay* como *copia de seguridad*. Los arquitectos seleccionaron la primera opción porque permitía la marcación de los bancos miembros a través de la Red Telefónica Conmutada Pública, mientras que la segunda opción era poco económica y más arriesgada, dado que la línea frame-relay de copia de seguridad también podía fallar.
- Otro aspecto importante era el *fallo de potencia* dado que el sistema EFT tenía que proporcionar un servicio continuo, por lo que se hacía necesario proporcionar un suministro de potencia secundario. Se evaluaron dos alternativas: un Suministro de Potencia Ininterrumpido (UPS); o un generador de potencia. Dado que la segunda alternativa requería un presupuesto más alto, los arquitectos seleccionaron la opción UPS.
- Para responder ante desastres naturales, como terremotos o incendios, que podrían dañar el centro de procesamiento, los arquitectos tuvieron que seleccionar un mecanismo adecuado: un *sitio remoto*, que pudiera asumir el control de procesamiento; o *procedimientos manuales*. Finalmente, los arquitectos seleccionaron los procedimientos manuales, porque no había suficiente presupuesto para la primera opción.

Dado este diseño del sistema, se crearon dos estructuras en cada herramienta, asociadas al AK del sistema EFT: una desde el punto de vista de los requisitos, y otra desde el punto de vista de los elementos arquitectónicos. Utilizando el AK del sistema EFT, representado con las distintas técnicas de visualización, los sujetos tuvieron que aplicar las siguientes tareas:

- Tarea 1: Se notificó a los sujetos que el cliente no presentaba ahora problemas monetarios, por lo que el sistema EFT tenía que ser evolucionado para implementar las mejores alternativas. Por esta razón, la primera modificación era cambiar el requisito de *Solución de Coste Efectivo* por *La Mejor Solución*.
- Tarea 2: Se añadió un nuevo requisito, *Monitorización 24h*, que permitía al sistema estar al tanto de cualquier problema en el suministro de potencia o fallos de computación o comunicación. La incorporación de este nuevo requisito afectó a todas las ADRs iniciales de la arquitectura definidas para el sistema.
- Tarea 3: El último cambio fue que la *base de datos ORACLE* tenía que ser reemplazada por *MySQL*, dado que el cliente estaba apostando por el software de código abierto.

Inicialmente, todos los miembros de cada grupo tuvieron que realizar las tres tareas propuestas. Como se puede observar, estas tres tareas permitieron que el grupo de estudio navegara a través de las estructuras de AK, y modificaran lo que consideraran más adecuado, teniendo en cuenta que los dos primeros cambios afectaban a la estructura de los requisitos, y el último cambio a la estructura de los elementos arquitectónicos. Además, se preparó una guía rápida, adjunta a este trabajo, para proporcionar ayuda sobre el funcionamiento de cada herramienta.

4.5.2.3.2.7 Evaluación de la validez

Existen varios niveles de validez a considerar (Wohlin et al. 2000): *validez interna*, *validez externa*, *validez de construcción* y *validez de conclusión*.

La *validez de construcción* es fuerte. La efectividad de la usabilidad y sus medidas constituyentes se comprenden bien por los distintos sujetos. Además, el uso de una arquitectura bien descrita junto a su AK representado con técnicas de visualización bien comprendidas, también tiene una fuerte validez de construcción.

La *validez interna* no es tan fuerte por el uso de estudiantes en los casos de estudio, en lugar de arquitectos software experimentados. Serán necesarios más estudios con arquitectos experimentados para ver si sus evaluaciones de la efectividad coinciden con las de los estudiantes. Sin embargo, las razones subyacentes para las evaluaciones actuales de efectividad sugieren que estos estudios tendrían resultados congruentes con nuestros estudios.

La fortaleza de la *validez externa* reside en el uso de una arquitectura software realista y su AK, y en realizar múltiples estudios. Su debilidad es análoga a la de la validez interna en el sentido de que los casos de estudio son realizados por estudiantes, pero con expectativas de resultados congruentes en otros estudios con arquitectos experimentados, consideramos que la validez externa, en general, es muy buena.

Finalmente, respecto a la *validez de conclusión*, podemos afirmar que nuestras conclusiones son estadísticamente válidas, ya que utilizamos el test de Tukey para llevar a cabo el análisis de nuestros datos, un test muy acertado y recomendado para realizar comparaciones múltiples, como es nuestro caso.

4.5.2.3.3 Operación

4.5.2.3.3.1 Preparación

Se preparó una máquina virtual con el sistema operativo Windows XP y todas las herramientas de visualización seleccionadas instaladas, de tal forma que permitía a cada sujeto realizar sus tareas. Seguidamente, esta máquina virtual se copió en quince PCs Dell™ Inspiron One 19.

4.5.2.3.3.2 Ejecución

Los casos de estudio exploratorios fueron ejecutados por el grupo de estudio en un laboratorio, utilizando el equipo descrito, y fueron llevados a cabo en tres fases diferentes:

- Primeramente, se proporcionó una introducción al experimento, describiendo su objetivo principal, el material experimental a utilizar, y las tareas a realizar.
- En segundo lugar, se realizó una breve introducción a cada una de las herramientas seleccionadas para que los sujetos adquirieran las habilidades necesarias para su manipulación.
- En tercer lugar, el grupo de estudio realizó cada una de las tareas, utilizando las distintas técnicas y, seguidamente, rellenaron el cuestionario sobre la efectividad.

Cabe destacar que, en cada fase, se notificó a los sujetos que el principal objetivo del experimento era evaluar la técnica de visualización y no la herramienta en sí, dado que las herramientas fueron seleccionadas para tener las misma funcionalidad, pero ofreciéndola de maneras diferentes, dependiendo de la técnica de visualización.

4.5.2.3.3.3 Validación de los datos

Cuando finalizaron las ejecuciones de los distintos casos de estudio, tuvimos que asegurarnos de que la información recogida a partir de los cuestionarios de efectividad era correcta. Así pues, revisamos dicha información para comprobar si todos los miembros del grupo de estudio habían respondido adecuadamente a todas las preguntas.

Finalmente, observamos que prácticamente todos los resultados eran correctos y, en todos los casos, eran muy similares entre sí. Sólo tuvimos que descartar algunos datos (outliers) que no seguían el "patrón" del resto de datos. Además, todos los sujetos respondieron, aproximadamente, en el mismo tiempo. Es por esto que los resultados (puntuaciones) no se muestran en este trabajo.

4.5.2.3.4 Análisis e Interpretación

Como ya se señaló, nuestra evaluación empírica fue diseñada como un conjunto de casos de estudio exploratorios; por tanto, nuestras conclusiones reales tienen que ser extraídas como resultado de cada caso de estudio individual, y cada uno de ellos presenta resultados para la efectividad de la usabilidad, gracias al cuestionario presentado en (Lund 2001). Como también se indicó, nuestra intuición es que una técnica de visualización, *Node-link and tree,* es mucho mejor que las otras, y nuestros múltiples casos de estudio soportan dicha intuición.

Para proporcionar una impresión inicial global sobre los resultados, se utilizan boxplots[6] (véase Figura 21) que muestran la relación *Puntuación-Técnica de Visualización* (*Score-Visualization Technique*), asociada a la efectividad de la usabilidad. Como se puede ver, la preferencia por *Node-link and tree* es claramente obvia, seguida de *Indented list*, *Zoomable*, *Space-filling* y *Wiki*, en ese orden.

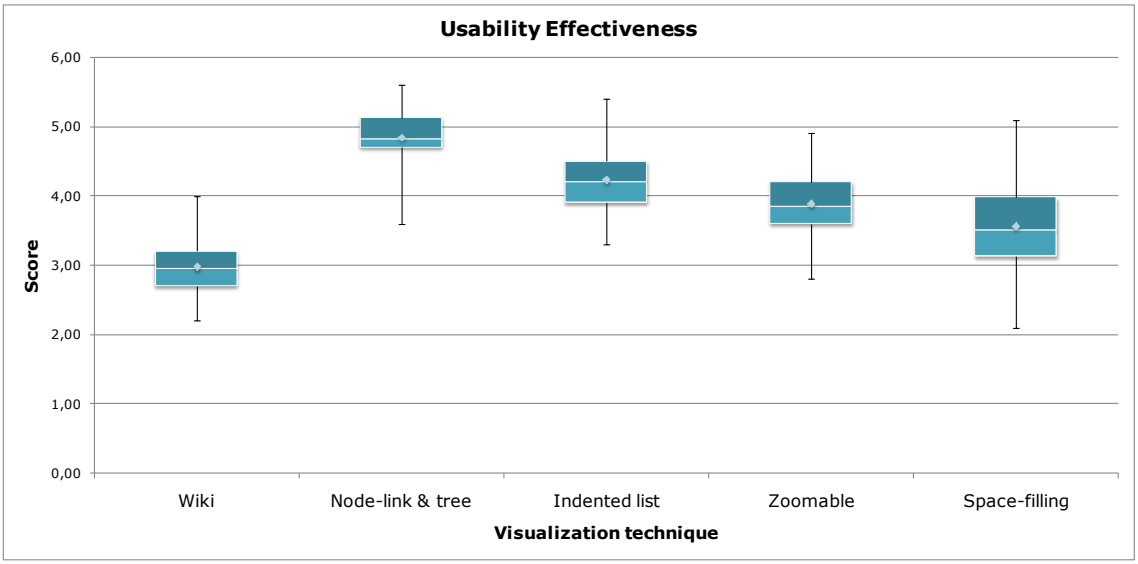


Figura 21. Boxplots para la *Efectividad de la Usabilidad*

4.5.2.3.4.1 Estadística descriptiva

Nuestro estudio empírico no fue diseñado como un experimento controlado, lo que significa que la validez y significancia estadística de cualquier resultado "promedio" tiene que ser cuidadosamente considerado. Entre muchos otros detalles, nuestro tamaño de la muestra es relativamente pequeño, y los individuos no estaban aleatoriamente seleccionados; y lo que es peor, las observaciones no pueden considerarse *independientes*, dado que se ha utilizado la misma población para evaluar todas las técnicas.

Por supuesto que esto no significa que tengamos que descartar completamente un análisis estadístico; más bien todo lo contrario, nuestras conclusiones sobre los valores promedio son más consistentes si están soportadas por un tipo de análisis que tenga en cuenta estos aspectos concretos. Por tanto, ideamos un contraste de hipótesis estadístico que difiere de los enfoques más usuales en diversos aspectos. Primero, no estamos considerando una única técnica, sino cinco; por consiguiente, el contraste de hipótesis es realizado por pares: puntuaciones de cada técnica son comparadas con aquellas de las otras cuatro restantes. Además, nuestras observaciones no eran independientes; por eso, tuvimos que realizar un análisis de subgrupos, dado que se presentó un problema de comparaciones múltiples. De esta forma, se seleccionó el *test de Tukey* para llevar a cabo este análisis de una manera más formal, ya que es utilizado normalmente cuando queremos realizar todas las comparaciones "a priori", y porque controla el error de tipo I, es decir, rechazar la hipótesis nula y aceptar la alternativa cuando la hipótesis nula es cierta.

Para automatizar el test de Tukey, se recurrió al programa R, utilizando el comando *TukeyHSD(aov(Averages ~ Technique, data=UsabilityData))*. La Tabla 7 presenta los resultados de esta ejecución, donde la columna *diff* se refiere a las diferencias entre las medias de cada técnica; las columnas *lwr* y *upr* indican los límites inferior y superior, respectivamente; y la columna *p adj* indica la probabilidad (p-valor) asociada a cada comparación. Nótese que, en la siguiente subsección, discutiremos detalladamente estos resultados estadísticos.

Tabla 7. Comparaciones múltiples de medias de Tukey (nivel de confianza del 95%)

Parejas de técnicas		diff	lwr	upr	p adj
Node-link & tree	Indented list	0.50	0.06343418	0.93656582	0.0160363
Space-filling	Indented list	-0.62	-1.05656582	-0.18343418	0.0012534
Wiki	Indented list	-1.24	-1.67656582	-0.80343418	0.0000000
Zoomable	Indented list	-0.35	-0.78656582	0.08656582	0.1801544
Space-filling	Node-link & tree	-1.12	-1.55656582	-0.68343418	0.0000000

[6] Un boxplot representa gráficamente datos numéricos mediante cinco estados (de abajo a arriba): la menor observación, cuartil inferior, mediana, cuartil superior, y la mayor observación. Además, la media ha sido representada también como un punto en medio del boxplot.

Wiki	Node-link & tree	-1.74	-2.17656582	-1.30343418	0.0000000
Zoomable	Node-link & tree	-0.85	-1.28656582	-0.41343418	0.0000029
Wiki	Space-filling	-0.62	-1.05656582	-0.18343418	0.0012534
Zoomable	Space-filling	0.27	-0.16656582	0.70656582	0.4320173
Zoomable	Wiki	0.89	0.45343418	1.32656582	0.0000009

4.5.2.3.4.2 Testeo de la hipótesis

Ahora, nuestra hipótesis nula puede ser (de nuevo) que "todas estas técnicas son igualmente efectivas", es decir, no existe diferencia entre cada par de técnicas, asociada con su efectividad de la usabilidad. Esto significa que, cada vez que la hipótesis nula es rechazada, la diferencia entre las dos técnicas de la pareja puede ser considerada como estadísticamente *significativa*.

La Tabla 8 sólo muestra los p-valores de las comparaciones del test de Tukey para así facilitar el análisis, presentando la información relevante de una mejor manera. Cada vez que el p-valor es menor que 0.05, se rechaza la hipótesis nula, y las técnicas del par correspondiente deben considerarse como no equivalentes. Echando un rápido vistazo a los resultados, se puede ver que la mayoría de las veces se rechaza la hipótesis nula, presentándose pocos conflictos (marcados en negrita).

Tabla 8. p-valores para la *Usabilidad*

p-value	Wiki	Node-link & Tree	Indented List	Zoomable	Space-filling
Wiki	-	0.0	0.0	0.0000009	0.0012534
Node-link & Tree	0.0	-	0.0160363	0.0000029	0.0
Indented List	0.0	0.0160363	-	**0.1801544**	0.0012534
Zoomable	0.0000009	0.0000029	**0.1801544**	-	**0.4320173**
Space-filling	0.0012534	0.0	0.0012534	**0.4320173**	-

Por tanto, a la vista de estos resultados, podemos afirmar que existen diferencias entre todos los pares de técnicas, excepto entre las parejas *Space-filling* – *Zoomable* y *Zoomable* – *Indented list*, es decir, se distinguen claramente tres grupos (véase Figura 22), donde vemos que las técnicas Wiki (peores resultados) y Node-link and tree (mejores resultados) se diferencian del resto de técnicas, por lo que podemos concluir que la técnica Wiki es la menos efectiva, mientras que la técnica Node-link and tree es la más efectiva de todas. Con respecto al resto de técnicas, no hay suficiente evidencia para alegar que existen diferencias entre ellas, es decir, son más o menos equivalentes en cuanto a su efectividad de la usabilidad.

Como se esperaba, *Node-link and tree* se considera mejor que los otros enfoques. Nuestra intuición fue correcta: esta técnica ha sido la más efectiva. El feedback de los usuarios refleja que dicha técnica presenta el AK de una manera simple y clara, de tal forma que los usuarios pueden navegar y explorar fácilmente la red de decisión del sistema EFT. Nótese que los estudiantes no fueron motivados para seleccionar una u otra herramienta, más bien todo lo contrario. Este resultado se predijo correctamente gracias a la experiencia, pero no por la manera en que se llevó a cabo el estudio.

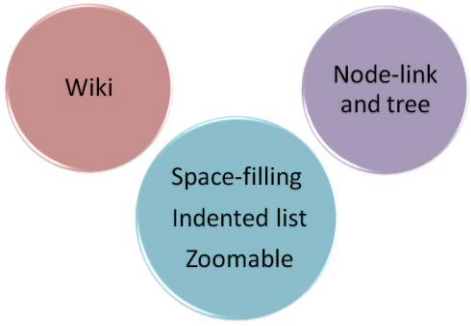


Figura 22. Grupos distinguidos en el test de Tukey

El cuestionario de Efectividad (Lund 2001) evalúa la efectividad de la usabilidad basándose en cuatro factores de la usabilidad: *Utilidad* (*Usefulness*), *Facilidad de uso* (*Ease of use*), *Facilidad de aprendizaje* (*Ease of learning*) y *Satisfacción* (*Satisfaction*). De esta manera, para enriquecer nuestra evaluación empírica, decidimos realizar un análisis adicional, teniendo en cuenta estos cuatro factores, es decir, cada caso de estudio presentaría resultados asociados a estos factores, utilizando el test de Tukey con un nivel de significación

de 0.05 (al igual que en el caso anterior). Así, podríamos ver si algún factor en particular tenía más influencia respecto a la efectividad de la usabilidad.

Como en el caso anterior, nuestra intuición es que una técnica de visualización (*Node-link and tree*) es mucho más efectiva que las demás. Observando las siguientes figuras: Figura 23, Figura 24, Figura 25 y Figura 26; vemos que nuestros múltiples casos de estudio soportan esta intuición, tanto individualmente como específicamente para cada uno de los factores de usabilidad.

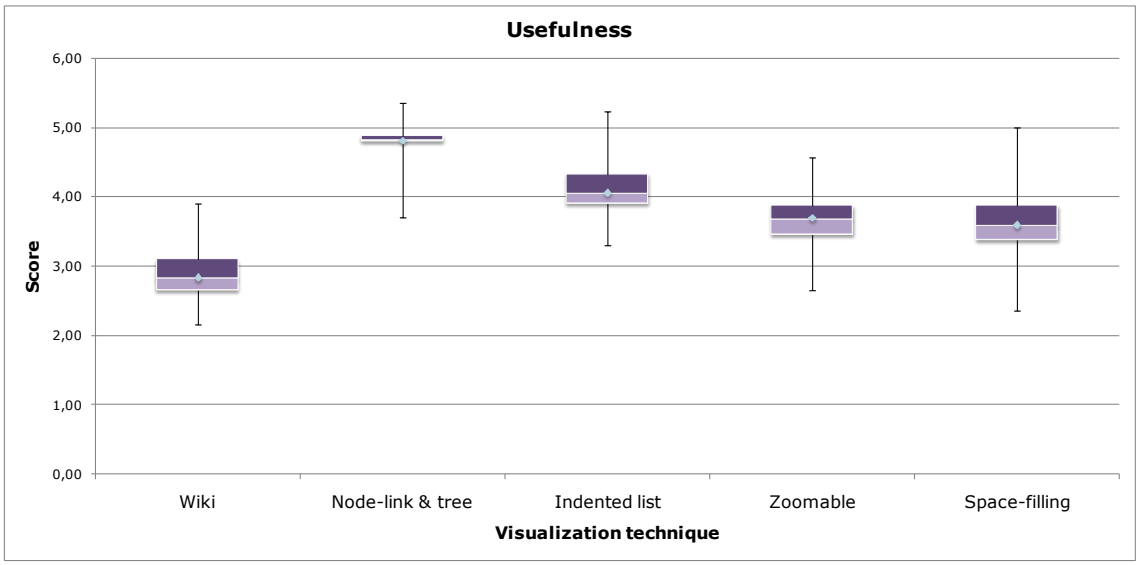


Figura 23. Boxplots para la *Utilidad*

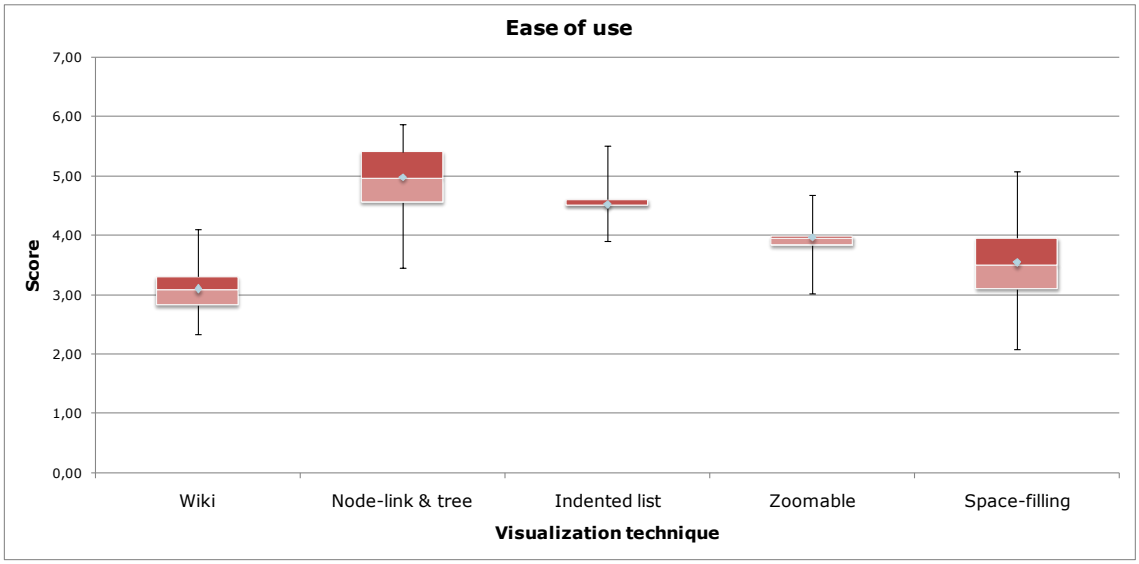


Figura 24. Boxplots para la *Facilidad de uso*

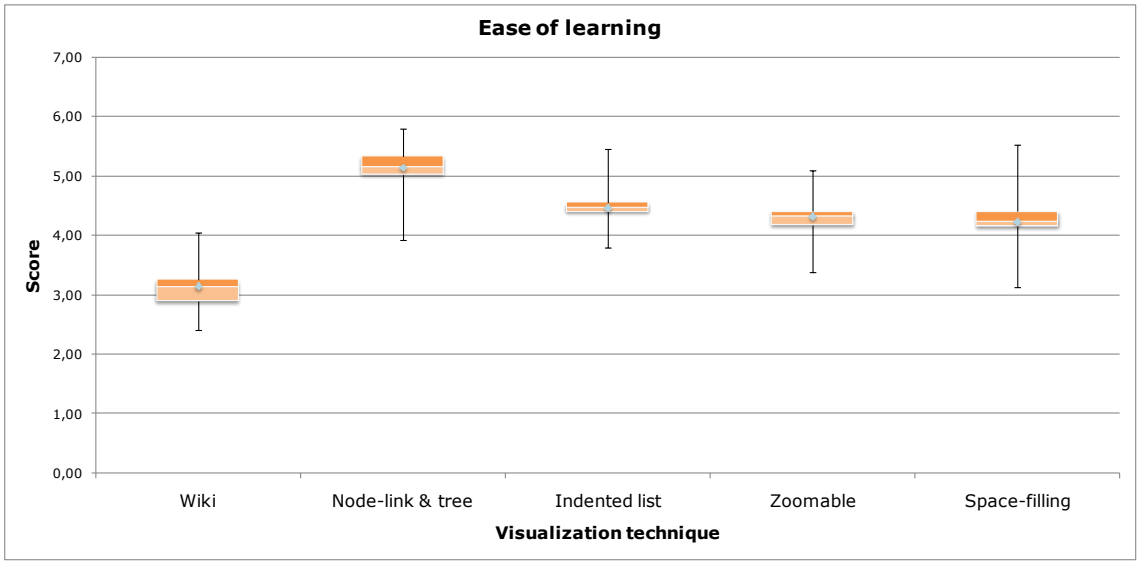


Figura 25. Boxplots para la *Facilidad de aprendizaje*

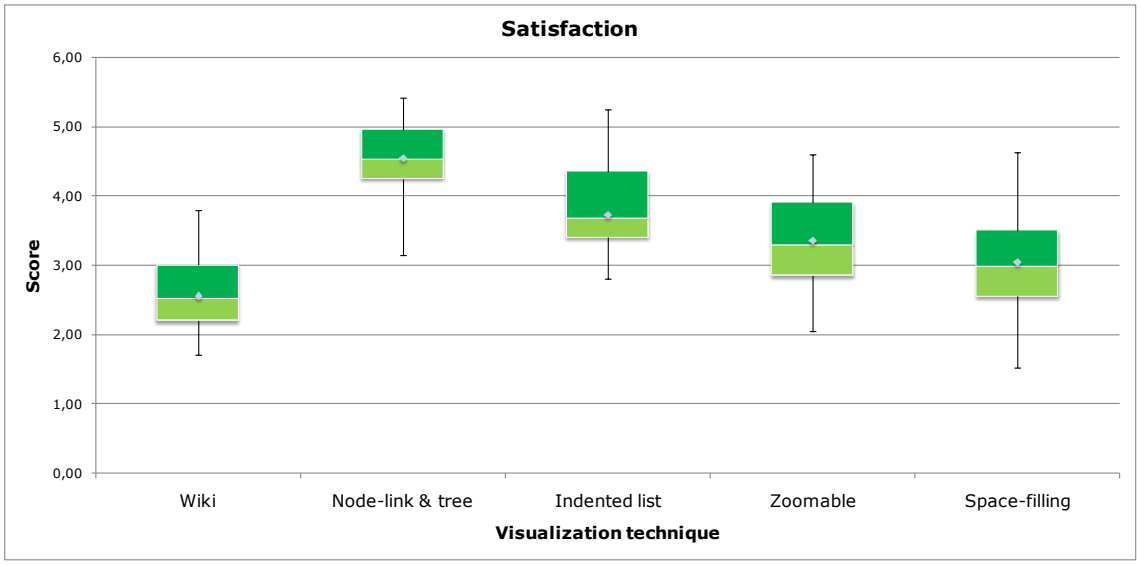


Figura 26. Boxplots para la *Satisfacción*

Para proporcionar un soporte formal, realizamos el test de Tukey para cada factor de usabilidad. Estos resultados se presentan en las siguientes tablas: Tabla 9, Tabla 10, Tabla 11 y Tabla 12.

Tabla 9. p-valores para la *Utilidad*

p-value	Wiki	Node-link & Tree	Indented List	Zoomable	Space-filling
Wiki	-	0.0000008	0.0001166	0.0086180	0.0250658
Node-link & Tree	0.0000008	-	**0.4713190**	0.0250658	0.0086180
Indented List	0.0001166	**0.4713190**	-	**0.5670302**	**0.3267639**
Zoomable	0.0086180	0.0250658	**0.5670302**	-	**0.9935101**
Space-filling	0.0250658	0.0086180	**0.3267639**	**0.9935101**	-

Tabla 10. p-valores para la *Facilidad de uso*

p-value	Wiki	Node-link & Tree	Indented List	Zoomable	Space-filling
Wiki	-	0.0	0.0000017	0.0004713	**0.3738494**
Node-link & Tree	0.0	-	0.0235088	0.0001596	0.0
Indented List	0.0000017	0.0235088	-	**0.4837848**	0.0008621
Zoomable	0.0004713	0.0001596	**0.4837848**	-	**0.0831812**
Space-filling	**0.3738494**	0.0	0.0008621	**0.0831812**	-

Tabla 11. p-valores para la *Facilidad de aprendizaje*

p-value	Wiki	Node-link & Tree	Indented List	Zoomable	Space-filling
Wiki	-	0.000023	0.000398	0.005210	0.002176
Node-link & Tree	0.000023	-	**0.500282**	**0.059478**	**0.133414**
Indented List	0.000398	**0.500282**	-	**0.659768**	**0.889668**
Zoomable	0.005210	**0.059478**	**0.659768**	-	**0.990349**
Space-filling	0.002176	**0.133414**	**0.889668**	**0.990349**	-

Tabla 12. p-valores para la *Satisfacción*

p-value	Wiki	Node-link & Tree	Indented List	Zoomable	Space-filling
Wiki	-	0.002208	0.044064	**0.281717**	**0.736672**
Node-link & Tree	0.002208	-	**0.757058**	**0.235324**	0.047826
Indented list	0.044064	**0.757058**	-	**0.880004**	**0.431397**
Zoomable	**0.281717**	**0.235324**	**0.880004**	-	**0.930594**
Space-filling	**0.736672**	0.047826	**0.431397**	**0.930594**	-

Como se observa, hay múltiples p-valores que igualan o superan el valor 0.05 (marcados con negrita). Así pues, para estos pares de técnicas, no podemos afirmar que existan diferencias entre ellas. A continuación, vamos a ver estos resultados en más detalle, asociados a cada factor de usabilidad.

Para el factor *Utilidad* (véase Tabla 9), podemos afirmar que existen diferencias entre las siguientes técnicas:

- Wiki - Indented list
- Wiki - Node-link and tree
- Wiki - Zoomable
- Wiki - Space-filling
- Node-link and tree - Zoomable
- Node-link and tree - Space-filling

En este caso, se distinguen claramente dos grupos: por un lado, la técnica Wiki (peor puntuación) y, por otro lado, el resto de técnicas. Por tanto, para este factor, no tenemos suficiente evidencia para asegurar que existe una técnica más efectiva que las demás.

Para el factor *Facilidad de uso* (véase Tabla 10), podemos afirmar que existen diferencias entre las siguientes técnicas:

- Space-filling - Indented list
- Node-link and tree - Space-filling
- Node-link and tree - Zoomable
- Node-link and tree - Indented list
- Wiki - Space-filling
- Wiki - Zoomable
- Wiki - Node-link and tree
- Wiki - Indented list

En este caso, se distinguen claramente tres grupos: por un lado, la técnica Wiki (peor puntuación); por otro lado, la técnica Node-link and tree (mejor puntuación); y, finalmente, el resto de técnicas. Por tanto, para este factor, tenemos suficiente evidencia para asegurar que existe una técnica (Node-link and tree) más efectiva que las demás.

Para el factor *Facilidad de aprendizaje* (véase Tabla 11), podemos afirmar que existen diferencias entre las siguientes técnicas:

- Wiki - Indented list
- Wiki - Node-link and tree
- Wiki - Space-filling
- Wiki - Zoomable

En este caso, sólo existen diferencias entre la técnica Wiki y el resto de técnicas. Así pues, se distinguen dos grupos: por un lado, la técnica Wiki (peor puntuación) y, por otro lado, el resto de técnicas. Por tanto, para este factor, tampoco tenemos suficiente evidencia para asegurar que existe una técnica más efectiva que las demás.

Finalmente, para el factor *Satisfacción* (véase Tabla 12), podemos afirmar que existen diferencias sólo entre las siguientes técnicas:

- Wiki - Indented list

- Space-filling - Node-link and tree
- Wiki - Node-link and tree

En este caso, no hay suficientes diferencias entre los pares de técnicas, por lo que sólo se tiene un grupo, es decir, no hay distinción entre ellas. Por tanto, para este factor, tampoco tenemos suficiente evidencia para asegurar que existe, tanto una técnica más efectiva que las demás, como otra menos efectiva que el resto.

Así pues, de este segundo análisis, podemos concluir que el factor *Facilidad de uso* es el único que permite distinguir la técnica más efectiva de entre todas las demás, ya que, en este caso, este factor es el que mejor refleja la efectividad de la usabilidad de las herramientas/técnicas de visualización aquí presentadas.

4.5.2.4 Conclusiones y trabajo futuro

Como se ha visto a lo largo de este trabajo, las decisiones de diseño y sus rationales han de estar bien documentadas para que el sistema bajo desarrollo/mantenimiento pueda evolucionar fácil y eficientemente. Sin embargo, algunas veces, este AK se presenta de manera inapropiada, lo que dificulta a los arquitectos la tarea de evolución/mantenimiento del sistema.

En este contexto, este trabajo describe cinco técnicas de visualización 2D para soportar la visualización del AK y las evalúa mediante una evaluación empírica del factor de calidad *efectividad de la usabilidad*. Este estudio empírico nos ha permitido ver qué técnica de visualización es la más efectiva con respecto a la representación y manipulación del AK, según los cuatro factores de calidad de la efectividad de la usabilidad: utilidad, facilidad de uso, facilidad de aprendizaje y satisfacción. Por tanto, la técnica *Node-link and tree* ha demostrado ser la más efectiva para este propósito, debido a su simplicidad y claridad a la hora de visualizar AK, utilizando un grafo comprensible, fácil de interpretar y navegar, utilizando nodos simples y entendibles.

Nuestro trabajo futuro estará centrado en las técnicas de visualización 3D para capturar AK e intentaremos determinar qué categoría es la más apropiada para este fin, al igual que se ha hecho en este trabajo con las técnicas bidimensionales. Además, confirmaremos nuestros resultados actuales con estudios adicionales, involucrando a arquitectos experimentados.

4.5.3 Utilizando Conocimiento Arquitectónico como operador de evolución

La Evolución del Software es una característica esencial de cualquier sistema desarrollado. Siempre hay argumentos convincentes que nos llevan a cambiar el sistema desarrollado para adaptarlo a las tendencias del mercado, a los avances tecnológicos, o, simplemente, a nuevos requisitos del cliente. Esta característica fue inicialmente advertida en los años 80, cuando la primera ley de la Ingeniería del Software fue establecida por (Bersoff et al. 1980): "No importa dónde estemos en el ciclo de vida del sistema, el sistema cambiará, y el deseo de cambiarlo persistirá durante todo el ciclo de vida" y fue confirmada por estudios posteriores (Lehman et al. 1997) (Lehman & Belady 1985). Por tanto, como la necesidad de cambiar se extenderá durante todo el ciclo de vida completo de los sistemas, la introducción de procesos apropiados, técnicas y herramientas que nos ayuden a hacerle frente se convierte en una parte esencial del proceso de desarrollo y mantenimiento del software.

Para llevar a cabo la evolución de software, la *Arquitectura Software* (*Software Architecture*, SA) emerge como uno de los pilares que deberían ser considerados desde dos puntos de vista diferentes: como un artefacto *para* la evolución y como un artefacto *de* la evolución. Esto es, la SA es un artefacto que puede ser utilizado *para* evolucionar el software ya que actúa como un modelo mental compartido de un sistema, expresado a un alto nivel de abstracción (Ric 2001), ayudando al arquitecto a planificar y reestructurar el sistema abstrayéndolo lejos de las cuestiones tecnológicas (D. Garlan et al. 2009). A esta alternativa se la conoce como *evolución del software basada en la arquitectura* (*architecture-based software evolution*). Pero, la SA es también uno de los resultados *de* la evolución, dado que también puede evolucionar con el fin de ser coherente con los cambios que van surgiendo. Por tanto, la introducción de mecanismos, técnicas, procesos, etc., que guíen al arquitecto durante la evolución de la SA, se convierte en una cuestión crítica a la hora de mantener su consistencia, calidad, etc.

Teniendo en cuenta la importancia de la rationale para entender porqué el sistema es como es, se convierte en una pregunta desafiante para evaluar qué implicaciones resultan durante la evolución del sistema. (Bratthall et al. 2000) trataron esta cuestión llevando a cabo un experimento con 17 sujetos de la industria y la docencia, y concluyeron que la mayoría de los arquitectos entrevistados señalaban que, utilizando las ADRs, podían reducir el tiempo necesario para llevar a cabo las tareas de cambio. Los sujetos entrevistados también concluyeron que la calidad de los resultados era mejor utilizando ADRs

cuando tenían que predecir cambios en sistemas de tiempo real desconocidos. Por tanto, hay argumentos convincentes para la explotación de la rationale mientras la SA está siendo evolucionada. Sin embargo, este uso de la rationale lo convierte en un *actor pasivo* del proceso de evolución, ya que se utiliza simplemente como artefacto de documentación. Es en este punto donde este trabajo centra su atención: ¿puede la rationale convertirse en un *actor activo* de la evolución que conduzca su aplicación? En este trabajo, se describe cómo se puede responder afirmativamente a esta cuestión, por medio de estilos de evolución (D. Garlan et al. 2009). Concretamente, se presenta una extensión a estos estilos que considera la introducción de ADDs y ADRs en su descripción para guiar al arquitecto durante el proceso de evolución de la SA. Este trabajo ha sido enviado para su evaluación a la siguiente revista:

- Carlos E. Cuesta, Elena Navarro, Dewayne E. Perry, Cristina Roda, *Using Architectural Knowledge as an Evolution Driver*, Software Evolution and Maintenance (enviado).

A continuación, la sección 4.5.3.1 presenta cómo los estilos de evolución han sido utilizados para evolucionar el software. La sección 4.5.3.2 describe la propuesta presentada en este trabajo: estilos de evolución considerando AK. La sección 4.5.3.3 demuestra, por medio de un caso de estudio, cómo esta propuesta puede ser puesta en práctica. Finalmente, la sección 4.5.3.4 describe las conclusiones y el trabajo futuro.

4.5.3.1 Estilos de evolución

La noción de *estilo arquitectónico* (*architectural style*), como introdujeron originalmente (D. E. Perry & A. L. Wolf 1992), es un concepto "con elementos abstractos y aspectos formales de varias arquitecturas específicas". Es un concepto *preceptivo*, en lugar de un concepto *descriptivo*: el mismo sistema o *configuración* puede cumplir simultáneamente con varias definiciones de estilo, es decir, la arquitectura resultante tendría varios estilos al mismo tiempo. La idea es que el estilo define una serie de restricciones – el rango de sistemas que cumplen estas restricciones reúne a los miembros de este estilo. Conforme el estilo es más específico, este rango es más pequeño, y la prescripción llega a ser casi una descripción.

Existe una relación obvia entre la evolución del software y los estilos arquitectónicos, aunque no siempre ha sido explícita. El estilo garantiza que la arquitectura mantenga una serie de propiedades – y por lo general estas propiedades son características de alto nivel que se mantienen a lo largo de la evolución del sistema. Por tanto, diferentes configuraciones de una arquitectura en evolución tienen el mismo estilo, es decir, en términos generales, la evolución arquitectónica respeta el estilo definido. Hacer caso omiso de la arquitectura y limitar estilos lleva a la deriva arquitectónica y a la erosión (D. E. Perry & A. L. Wolf 1992).

Recientemente, varios autores han intentado explotar esta intuición, desarrollando un concepto nuevo, aunque relacionado, denominado *estilo de evolución* (*evolution style*). En breve, el estilo arquitectónico original describe las restricciones a ser cumplidas por un conjunto de sistemas, es decir, el estilo de evolución, este conjunto de sistemas serían el conjunto de configuraciones potenciales del sistema evolutivo, es decir, el estilo define las restricciones a ser satisfechas por cualquier posible evolución del sistema. Donde la noción original prescribe la arquitectura, esta variante prescribe su evolución.

(Noppen & Tamzalit 2010) y otros autores como (Le Goaer et al. 2008) y (Tamzalit et al. 2006) han desarrollado una serie de trabajos sobre este tema. Su noción de estilo de *evolución* utiliza el concepto original de estilo *arquitectónico* para delimitar el rango de cambios a un sistema dado. Intentan proporcionar transformaciones genéricas (*patrones de evolución*) que describen estos cambios como una serie de pasos (*operaciones de evolución*). Para ser genérico, estas transformaciones han sido definidas con respecto a las restricciones de un estilo arquitectónico, por tanto, la evolución está dentro de (o hacia) un estilo.

Inspirado por estos trabajos, (D. Garlan et al. 2009) han realizado también algunos desarrollos en este contexto. Sus *estilos de evolución* son definidos directamente como transformaciones sobre estilos: proporcionan un estilo *inicial*, un estilo *objetivo*, y posiblemente varias secuencias de pasos evolutivos intermedios, conocidos como *caminos de evolución* (*evolution paths*). Estos caminos definen distintas maneras de evolucionar desde un estilo inicial hasta otro objetivo, utilizando *operaciones* arquitectónicas para generar estilos intermedios "evolutivos". También definen un conjunto de *restricciones de camino* (*path constraints*), las cuales están especificadas utilizando lógica temporal y funciones de evaluación predefinidas. Esto hace posible que el arquitecto lleve a cabo ciertos análisis.

Aunque el trabajo de Tamzalit se refiere explícitamente al impacto de los estilos de evolución sobre el conocimiento arquitectónico (Noppen & Tamzalit 2010) (Le Goaer et al. 2008), ninguna de estas propuestas hace uso del AK existente para definir o incluso influenciar dichos estilos. Mucha de esta información es específica del sistema, y eso la hace más relevante para nuestro trabajo. Nuestra propuesta

intenta cubrir este hueco exactamente, es decir, definir estilos de evolución que hagan uso explícito de AK.

Nuestro enfoque es, hasta cierto punto, similar a los citados arriba – el estilo será descrito como una secuencia de pasos. Pero la característica que lo define es que nuestras *decisiones de evolución* están influenciadas por el AK existente. En el enfoque de Garlan et al., estamos obligados a tomar decisiones, utilizando sólo información estructural, como el número de nodos – de ahí la necesidad de funciones predefinidas. Sin embargo, en nuestra aproximación, *cualquier decisión tomada* está siempre disponible, por lo que nuestras decisiones de evolución pueden tener en cuenta *incluso decisiones anteriores*. En concreto, nuestro enfoque es capaz de detectar cuándo las condiciones que dieron lugar a una decisión en el pasado ya no la soportan, y decidir si este cambio en las condiciones es suficiente para causar, o al menos permitir, un paso evolutivo. Nuestra propuesta está diseñada para no depender de ningún lenguaje o plataforma de descripción existente, aunque tenga que ser adaptada a cualquier aproximación específica. Nuestra única restricción para su aplicación es que utilizamos *Conocimiento Arquitectónico Estructurado* (SAK), esto es, que el sistema incluye una representación explícita de AK.

La Figura 27 proporciona una representación simple de una definición de *estilo de evolución* (un *paso evolutivo*, de hecho) en el contexto de un sistema basado en SAK. Al igual que en la propuesta de (D. Garlan et al. 2009), nuestro estilo está definido como secuencias de dichos pasos, evolucionando el sistema a partir de la configuración en el instante **t** hasta la situación en **t+1**.

Cada situación en el tiempo se describe como un "fotograma" de una secuencia temporal. Inicialmente (instante **t**), la SA tiene una configuración y algún conocimiento asociado. Por ejemplo, el sistema actual está compuesto por tres componentes (véase óvalo izquierdo), y el *árbol de decisión* (en la parte superior) describe cómo una determinada decisión (A) ha influido en nuestra elección final, lo que implica la selección (limita, *constrains*) de B, en lugar de la alternativa C, la cual es rechazada (inhibe, *inhibits*), y habría dado lugar a una arquitectura de dos componentes (véase óvalo derecho). Si ocurre algún otro suceso, esta decisión A es modificada (**Δ**). Finalmente (instante **t+1**), se tiene la situación en la que el sistema sólo tiene dos componentes, y el árbol de decisiones ha sido modificado: ahora una nueva decisión (A') impide la antigua elección (todavía B) y limita nuestra elección a la configuración actual (todavía C).

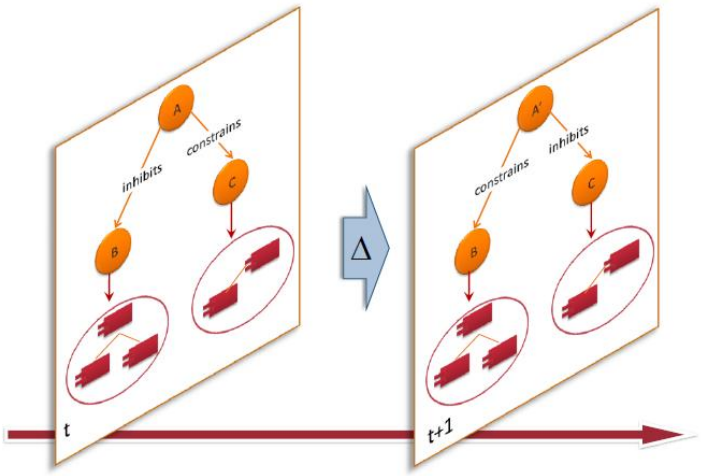


Figura 27. Conocimiento Arquitectónico para evolucionar la Arquitectura Software

Finalmente, nótese que, si bien nuestra intuición inicial (cuando relacionamos evolución con estilo) era que, *por lo general*, la evolución *respeta* el estilo arquitectónico, esto no es un requisito en la definición de estilos de evolución. Todo lo contrario, los enfoques existentes los utilizan para superar sus limitaciones. Así, por ejemplo, (Tamzalit et al. 2006) son capaces de introducir un *nuevo* estilo arquitectónico en un sistema existente. El trabajo de (D. Garlan et al. 2009) está directamente concebido para evolucionar desde un estilo inicial a uno objetivo – por tanto, el estilo siempre cambia. Nuestro enfoque no está directamente vinculado a estas limitaciones, ya que depende del *conocimiento disponible*, en lugar del estilo actual.

4.5.3.2 AK como un operador para la evolución

La evolución del software (Madhavji et al. 2006) es un campo muy complejo; incluso cuando el clásico esquema SPE (*Software Performance Engineering*) (Lehman et al. 1997)(Bass et al. 2003) hace posible limitar nuestro campo de acción a los sistemas de tipo E (es decir, aquellos que tratan una actividad del mundo real). La Primera Ley de la evolución (Lehman et al. 1997) añade también la suposición de que la

evolución en estos sistemas es *continua*, por lo que existen diversos aspectos a considerar. Concretamente, nuestra propuesta debe tener en cuenta diferentes *dimensiones* (que son los ingredientes requeridos para desarrollar buenos sistemas software) (D. E. Perry 2006), y *facetas* (que son las distintas interpretaciones asociadas al término *evolución del software*) (Mittermeir 2006), incluyendo, tanto los *operadores* de evolución, como los *estratos* software.

(D. E. Perry 2006) afirma que, realmente, las tres *dimensiones* (o fuentes) fundamentales que permiten la evolución del software son los *dominios*, la *experiencia*, y el *proceso*, en lugar de la clásica perspectiva de mantenimiento del software *corregir-perfeccionar-mejorar*. En el ámbito de la SA, la principal preocupación parece haber sido siempre el proceso – en lugar de esto, nuestro enfoque de *estilos de evolución* se centra en el *dominio* de los modelos y las especificaciones, y deriva de la *experiencia*, la cual aparece explícitamente en la *rationale arquitectónica*, y será soportada por un *proceso* técnico concreto.

Las *facetas* de evolución (Mittermeir 2006) consideran las características principales que se deben tener en cuenta cuando se evoluciona un sistema. Concretamente, características como la naturaleza de los *operadores para la evolución* – por lo general, la principal presión es el mercado, junto con cuestiones técnicas relevantes, pero en general, podemos deducirlas como consideraciones *basadas en personas* (*human-based*). Esto está directamente relacionado con la segunda hipótesis de Mittermeir: *"las limitaciones humanas en el procesamiento de la información, limitan el alcance de la evolución del software"*. Por tanto, la fuente de información se considera fuente para la evolución – y, por tanto, cuando nuestro AK se hace explícito, éste se puede considerar como un operador de evolución. De hecho, si nuestras descripciones arquitectónicas son limitadas (ya que carecen de información relevante), nuestras capacidades para la evolución también lo son.

Otra faceta importante propuesta por (Mittermeir 2006) son los *estratos de la evolución*, es decir, categorizar el software en diferentes estratos, dependiendo del patrón evolutivo seguido – el mismo software es capaz de *"seguir diferentes patrones evolutivos"* en diferentes niveles. De nuevo, en este caso, se hace hincapié en los estratos más bajos, siendo la *arquitectura* (como la estructura esquelética del sistema) el primero de los estratos superiores. Por tanto, el *estrato arquitectónico* debería definir, no sólo sus propios operadores de evolución (conocimiento arquitectónico), sino también sus propios patrones de evolución.

Aunque este artículo se refiere, principalmente, a este estrato arquitectónico, nuestra propuesta puede aplicarse también a los estratos superiores, como sistemas y sistemas de sistemas, ya que son, generalmente, gestionados como arquitecturas a gran escala (Watson et al. 2010)(Northrop et al. 2006).

4.5.3.2.1 Detectando la necesidad de evolución

Como se comentó anteriormente, el papel obvio de nuestro AK, en el contexto de evolución del software, es actuar como un *operador de evolución*, es decir, ayudar al arquitecto a detectar *cuándo* y *cómo* debería evolucionar la arquitectura. Brevemente, el AK puede ser utilizado para la evolución de tres maneras:

- El AK puede ayudar a *detectar* cuándo un *cambio* es permitido.
- El AK puede ayudar a *detectar* cuándo un *cambio* es requerido.
- El AK puede ayudar a determinar qué *estilo de evolución* debería seleccionarse para evolucionar el sistema, entre varios candidatos potenciales.

La mayoría de la investigación en evolución arquitectónica se ha centrado en *cómo* podemos realizar una reconfiguración arquitectónica, pero se ha hecho poco hincapié en decidir *cuándo* hacerlo. La razón es, probablemente, que la mayoría de la información que podría ayudar no era explícita – ahora, esta información está incluida en el AK.

La *decisión* de evolucionar es, probablemente, semiautomática, en el sentido de que un humano tiene, por lo general, la última palabra. Pero nuestra razón de disparar el paso evolutivo a menudo puede ser encapsulada en una fórmula lógica, mediante una *condición de evolución*. Cuando esta condición es verdadera, entonces, por lo general, debería tener lugar la evolución.

Las condiciones de evolución se describen como una ecuación lógica – o, alternativamente, como texto plano, que puede unir y combinar cualquier cantidad de los siguientes factores:

- Situaciones en la arquitectura del sistema actual, es decir, sobre todo características estructurales (número de conexiones, etc.);
- Situaciones en el contexto externo, es decir, algo que ocurre en el exterior, incluyendo la intervención humana;
- *Decisiones activas*, es decir, una decisión arquitectónica hecha en el pasado e incluida en el AK, que todavía afecta directamente a la SA actual. Cuestionar si dicha decisión está todavía justificada,

considerando el contexto actual, e, incluso, la posibilidad de revocarla, es un método estándar para disparar un cambio en el sistema;

- *Decisiones pasadas*, es decir, cualquier decisión tomada en el pasado e incluida en el AK, independientemente de que fuera *tomada* o *rechazada*. Esta decisión pasada puede afectar todavía a las decisiones activas por medio de las relaciones de AK (como se observa en la Figura 27), o puede ser utilizada como "registros de memoria" para decidir sobre la decisión actual, utilizada por medio de relaciones de trazabilidad.

Las *condiciones de evolución*, descritas de esta manera, son capaces de proporcionar más información sobre el sistema existente que cualquier otra aproximación – de hecho, son potencialmente capaces de saber más sobre el sistema que el propio arquitecto. Por tanto, proporcionan los medios para definir cada paso en nuestro estilo de evolución.

4.5.3.2.2 Definiendo estilos de evolución

Como ya se indicó antes, nuestro enfoque para la evolución del software se basa en la suposición de que el sistema a evolucionar tiene una *rationale arquitectónica explícita*, es decir, incluye Conocimiento Arquitectónico Estructurado (SAK).

Esto significa que, en términos SPE, no sólo el sistema de tipo E es también un sistema de tipo S, sino que también su especificación debe ser suficientemente detallada para incluir la *rationale* junto al diseño elegido y las decisiones específicas, es decir, la definición del estilo arquitectónico elegido.

En el caso ideal, esta rationale arquitectónica no es sólo una descripción en texto plano (útil a pesar de ello), sino una representación estructurada que somos capaces de manipular (lo que hemos llamado SAK). De nuevo, ni el lenguaje ni la representación concreta son importantes: el único requisito consistente es ser capaz de acceder y utilizar dicha información.

El Conocimiento Arquitectónico Estructurado asume una representación explícita del AK, que incluye un conjunto de relaciones internas, el cual puede variar según el lenguaje arquitectónico o plataforma. Hay diversas propuestas en la literatura, pero, para el resto de la discusión, utilizaremos el conjunto que describimos en (Navarro, Cuesta, et al. 2009), y que se utiliza también en la Figura 27.

Nuestra definición de un *estilo de evolución* es implícita: en lugar de proporcionar una definición directa, comenzamos describiendo los elementos que lo compondrán hasta llegar a una forma final. Quizá, de modo diferente a otras propuestas, nuestros estilos de evolución no se conciben para ser definidos "desde cero", o tratando de llegar a alguna "gran idea". Nuestro principal propósito, en lugar de esto, es *reutilizar conocimiento* – que, de hecho, es el principal objetivo de los estilos arquitectónicos, e incluso del conocimiento mismo.

Por tanto, nuestros estilos no comienzan transformando otros estilos, como hacen algunas propuestas, sino que son creados *abstrayendo fragmentos concretos del conocimiento arquitectónico* y sus influencias sobre sistemas específicos. Cuando se detecta algún potencial para la evolución, se extraen algunos patrones, y éstos se recogen en un estilo genérico. En ese momento, este estilo puede ser aplicado a diferentes sistemas, no relacionados con los fragmentos originales.

Por tanto, un *estilo de evolución* está compuesto por:

- *Condiciones de evolución*, que ya fueron descritas en la sección anterior.
- *Decisiones de evolución.* La elección de una de estas condiciones. Esta elección debe ser almacenada como parte del AK, utilizando una categoría diferente. Nótese que esta decisión estará relacionada con otras decisiones arquitectónicas, usando las relaciones antes mencionadas.
- *Pasos evolutivos.* Lo que ocurre cuando se toma una decisión, es decir, cuando la evolución tiene lugar. Nótese que esta definición depende de la *decisión*, no de la arquitectura específica. Por tanto, un único *paso evolutivo* puede implicar varios pasos tecnológicos a nivel arquitectónico – incluso una reconfiguración completa, aunque sólo utilizando una única decisión (véase de nuevo la Figura 27).
- *Patrones de evolución.* Los pasos están relacionados con otros pasos por secuencias de decisiones, posiblemente encadenados en un razonamiento lógico (una *rationale de evolución*). Esta cadena crea una secuencia de pasos, posiblemente ramificada en cada decisión. Cuando dicha secuencia puede ser *abstraída* de una configuración específica, para ser descrita en términos de algunas decisiones genéricas (es decir, como parte de algún AK genérico), entonces toma la forma de un *patrón de evolución*. Estos patrones incluyen caminos de evolución como un caso específico.
- *Estilos de evolución.* Un conjunto de patrones de evolución pueden estar conceptualmente relacionados, es decir, o bien afectan al mismo conjunto de características, o a un conjunto de características relacionadas, las cuales, conjuntamente, pueden lograr algún efecto combinado.

Nótese que este conjunto de patrones no necesita estar conectado y, por supuesto, incluye el conjunto unitario como un caso especial.

En resumen, el estilo está construido de abajo a arriba: el arquitecto, mientras trabaja con un sistema concreto, identifica un paso significativo que puede ser abstraído desde el AK específico a una situación genérica. A continuación, reúne secuencias de dichos pasos que tengan sentido en su conjunto, y este conjunto define patrones de evolución. Finalmente, un conjunto de patrones relacionados define un *estilo de evolución*. Como en muchos otros casos, estos estilos se crean a partir de fragmentos dispersos de conocimiento – aunque siguen siendo construidos de manera que garantice la coherencia conceptual. Los estilos de evolución deben ser aplicados a un sistema – o, más concretamente, al AK de un sistema. Esto se conoce como el *sistema base* (*base system*) para el estilo. Como los estilos están concebidos para ser genéricos, cualquier estilo puede ser aplicado a una gran variedad de sistemas base. Además, como ya se señaló, esta definición está diseñada para ser genérica, no dependiendo de ningún lenguaje específico. Un ejemplo más concreto será proporcionado en la sección 4.5.3.3.4.

4.5.3.2.3 Aplicando un estilo de evolución

El estilo de evolución se ha concebido para ser aplicado como parte de un proceso semiautomático, donde cada paso de cada patrón debe ser precedido por una *decisión*, tanto arquitectónica, como de evolución. Estas decisiones son tomadas, generalmente, por humanos (Watson et al. 2010). En este sentido, cualquier evolución, que es un cambio en el AK y, por tanto, en la arquitectura asociada, tiene que ser explícitamente aprobada por el arquitecto.

Los estilos de evolución serían útiles incluso si fueran puramente documentales, aunque, asumamos algún tipo de soporte automático. Por ejemplo, el soporte existente dirigido por modelos para las decisiones de diseño y estilos arquitectónicos, soportado por MORPHEUS (Navarro & Cuesta 2008), puede ser fácilmente extendido para describir este tipo de estructuras, y para proporcionar asistencia al arquitecto humano. El sistema resultante será capaz de *sugerir* al arquitecto una evolución potencial, y éste sólo necesitaría *aceptarla* para que la evolución tenga lugar.

Por supuesto, en los pocos casos en los que la intervención humana no se requiere, los estilos de evolución tienen todo lo que necesitan para tomar una decisión. Por tanto, se podría llevar a cabo *automáticamente* el proceso completo de evolución sin más ayuda, utilizando las mismas técnicas dirigidas por modelos antes mencionadas. Sin embargo, este tipo de situación es poco común, y no describe el caso general, de tal forma que no será considerada de aquí en adelante.

4.5.3.3 Caso de estudio

Para ilustrar los conceptos introducidos en la sección anterior, en esta sección se presenta un caso práctico de una arquitectura de evolución y cómo la gestión del conocimiento arquitectónico conduce a esta evolución, y a la definición de los estilos de evolución.

En lugar de un ejemplo trivial, se presenta un caso de estudio real, incluyendo una compleja arquitectura con un conjunto de características, y que se enfrenta a un problema complejo. El propósito es doble: primero, describir un problema en un contexto interesante, mostrando que nuestro enfoque no es sólo una "construcción de laboratorio" que puede ser aplicado en un entorno no controlado; y segundo, mostrar la potencia actual de estos conceptos, que no se percibe hasta que no se aplica a un problema complejo.

Otra característica interesante de este ejemplo es la razón de evolución: el *incremento de los costes*. Por tanto, el operador de evolución es el *coste*, en lugar de alguna razón técnica escondida (y el sistema es lo suficientemente complejo como para tener gran cantidad de estas razones). Esto es importante dado que es el tipo de situaciones que sólo pueden ser adecuadamente descritas cuando el AK es explícito.

4.5.3.3.1 Ejemplo de motivación: evolucionando una arquitectura en la nube

El ejemplo que se presenta en este trabajo ha sido desarrollado en el contexto del *cloud computing* (Armbrust et al. 2009). Las razones detrás de esta elección son el énfasis actual sobre esta aproximación, que ayuda a situar la viabilidad de nuestra propuesta; y también el hecho de que las *arquitecturas de nube* tienen una gran importancia para las aplicaciones basadas en la nube – esto es, el nivel arquitectónico es particularmente significativo para su función y, por tanto, su evolución es relevante para el sistema en su conjunto.

Quizá la característica más importante del cloud computing, e indudablemente lo que la distingue de otras propuestas relacionadas, tales como el *software como servicio* (*Software as a Service*, SaaS), es la *escalabilidad*, también conocida como *elasticidad*. Una aplicación habilitada para la nube se ejecuta en un entorno *elástico*, lo que significa que la aplicación es capaz de reaccionar automáticamente ante una demanda creciente de recursos, dado que, cuando éstos son necesarios, simplemente hay más disponibles.

Sin perder generalidad, vamos a concebir una aplicación habilitada para la nube como un conjunto de *servicios* independientes, relacionados entre sí por medio de *colas* y gestionados por *controladores* específicos. Esta configuración se conoce algunas veces como *arquitectura de nube canónica* (Varia 2008). La mayoría de las políticas de gestión tienen que ser, o bien distribuidas en la arquitectura o gestionadas por dichos controladores.

Nuestro ejemplo describe una "estación de radio" bajo demanda, basada en la nube, que emite en Internet por medio de técnicas de *streaming*. Esta estación ha almacenado un gran conjunto de programas, los cuales son identificados unívocamente. Cuando un oyente (un usuario) sintoniza la estación, solicita algún programa específico; el sistema contesta proporcionando la URI de un servidor de streaming, donde el usuario puede escuchar el programa solicitado. El proceso finaliza cuando la radiodifusión termina y cada recurso utilizado se libera.

Como se representa en la Figura 28, el sistema está compuesto por tres tipos de servicios: un *servicio de base de datos* (DBS), un *servicio de streaming* (SS) y un *servicio de almacenamiento de datos* (DSS). Estos servicios se gestionan por medio de tres controladores, conocidos, respectivamente, como *sintonizador*, *monitor* y *terminador*. Cada uno de ellos tiene su propia *cola* para recibir y almacenar peticiones.

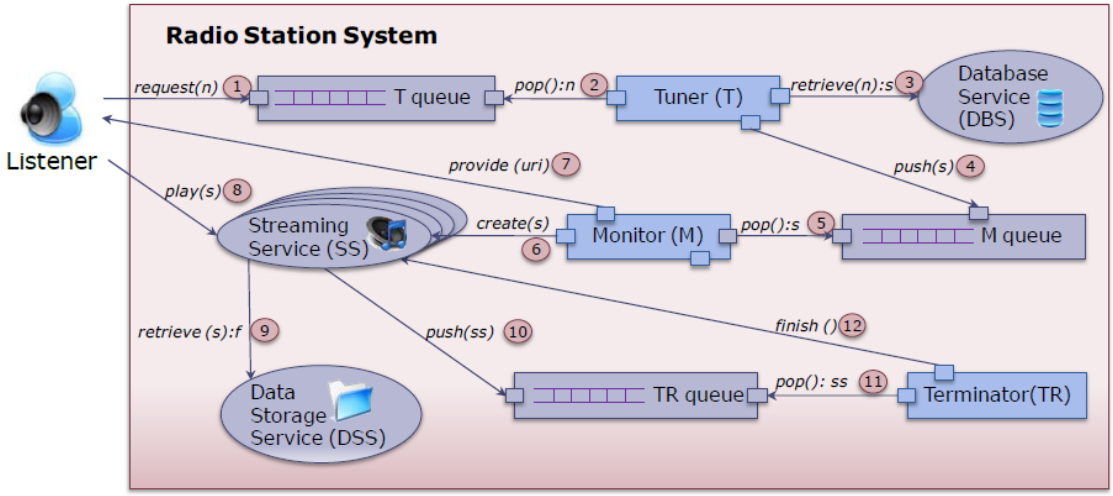


Figura 28. Sistema de Estación de Radio en tiempo de ejecución: versión 1.0

Cada vez que un oyente entra en la estación, busca un determinado programa, es decir, lanza una petición a la cola del sintonizador. El sintonizador ubica el programa solicitado en la base de datos (DBS) y, seguidamente, el monitor crea una nueva instancia del servicio de streaming (SS). Este SS obtiene la grabación del programa del depósito de almacenamiento (DSS), y comienza a transmitir su contenido sobre una nueva URI creada. Una vez que el programa ha finalizado, el monitor notifica al terminador que destruya el viejo SS, liberando los recursos asociados.

Por supuesto que el sistema real es más complicado; este trabajo simplifica la presentación concentrándose en los aspectos relacionados con la evolución, los cuales son nuestra principal preocupación aquí.

4.5.3.3.2 Breve discusión sobre el ejemplo

Estos servicios son concebidos como servicios software "en la nube" dentro de una plataforma de nube arquetípica. Esto significa que todos estos servicios (nuestros DBS, SS y DSS) son implementados y exportados, utilizando el enfoque SaaS, donde los usuarios acceden a ellos como clientes de un servicio. Sin embargo, como ya se comentó antes, utilizar un enfoque orientado a servicios no es suficiente. De hecho, estos servicios están generalmente diseñados casi de la misma manera que las aplicaciones convencionales: son aplicaciones en la *nube* debido, sobre todo, a *dónde* se despliegan – son escalables y resistentes porque esto se soporta por la plataforma subyacente. Pero, esta plataforma es también orientada a servicios, y se presenta como la *infraestructura como servicio* (IaaS).

En resumen, nuestra aplicación está definida como un conjunto de servicios SaaS, que a su vez están soportados por un conjunto de servicios IaaS.

Tanto la *arquitectura de nube canónica* como nuestro ejemplo concreto, presentados antes en la Figura 28, mezclan ambos tipos de servicios, por lo que la arquitectura puede *parecer* un poco compleja. Por supuesto que son *sólo* servicios, puesto que nuestra arquitectura es sólo un conjunto de servicios que interactúan. Además, tener un flujo de trabajo bien definido, como es el caso, hace que sea muy simple.

A modo de aclaración, a continuación se introduce una breve explicación. Nuestros *servicios a nivel de usuario* (DBS, SS y DSS) son los servicios SaaS; cada uno de ellos se ejecuta a un nivel inferior del servicio IaaS. Nuestros *controladores* (sintonizador, monitor y terminador) están controlando eficazmente estos servicios subyacentes, y cómo proporcionar recursos a los niveles superiores: por tanto, tienen que ser considerados IaaS por sí mismos, pero su función define nuestra escalabilidad de aplicación. Finalmente, las diferentes *colas* (T, M, TR) son servicios en sí mismos. De hecho, son instancias del mismo servicio. Su existencia es casi obligatoria en las arquitecturas de nube, dado que la *elasticidad* implica que el número de clientes de un servicio, en un momento dado, puede exceder su capacidad. Las colas, por tanto, se proporcionan para asegurar que ninguna petición (o respuesta) se pierde. Los servicios de colas trabajan eficazmente como los "conectores" en esta arquitectura y, obviamente, es seguro clasificarlos como IaaS, ya que sirven como infraestructura básica.

Por tanto, en el nivel de infraestructura (IaaS), sólo necesitamos un servicio de almacenamiento, un servicio de computación, un servicio genérico de base de datos y un servicio de colas, con sus correspondientes controladores, para ubicar respectivamente nuestro DSS, múltiples instancias SS, el DBS y todas las colas a nivel de usuario. Utilizando la popular plataforma Amazon AWS, por ejemplo, todo ello habría sido gestionado por el depósito de almacenamiento S3, la computación EC2, el servicio SimpleDB y las colas SQS (Varia 2008). Nuestros servicios SaaS se ejecutarían por encima de éstos.

Por supuesto que podría haber más elementos. Por ejemplo, el subsistema de *facturación* ha sido omitido deliberadamente. Sin embargo, debe ser presentado en cualquier aplicación basada en la nube, dado que cada servicio en la nube cuesta dinero. Nosotros asumimos su función, es decir, el cliente todavía paga, pero no ha sido incluido en la arquitectura para simplificar la presentación.

4.5.3.3.3 Detectando la necesidad: Asignación Dinámica Excesiva

Vamos a suponer que el sistema presentado es satisfactorio en términos de eficiencia y funcionalidad: la estación de radio trabaja como es de esperar, y la experiencia de usuario es positiva. El programa seleccionado se envía, y el rendimiento es correcto, incluso durante picos ocasionales (y bruscos) de audiencia.

Sin embargo, después de algún tiempo, se detecta claramente que el sistema es demasiado caro: la elasticidad *cuesta* dinero. Un nuevo oyente implica un acceso al DBS, una nueva instancia computacional del SS, y una o varias cargas desde el DSS. Cada uno de estos pasos está sujeto a una contribución y se incluye en la factura. Pero esto también significa que, si la estación logra el éxito, tendrá múltiples oyentes, quienes causarán también multitud de gastos. Normalmente, se espera que nuestros ingresos cubran estos gastos, pero, algunas veces, este no es el caso – por muchas razones. Por tanto, estamos en una situación curiosa. Desde un punto de vista técnico, el sistema puede crecer tanto como se desee: el entorno elástico garantiza que no existen problemas de escalabilidad. Pero la tasa de crecimiento puede seguir siendo un problema, en este caso, desde el punto de vista del *negocio*. De nuevo, como se comentó en la sección 4.5.3.2, el operador real de evolución está basado en humanos.

Esta forma de trabajar, sin embargo, está considerada estándar en arquitecturas de nube actuales, probablemente por los bajos precios de los proveedores de nube actuales. Sin embargo, es obvio que no es muy eficiente: *cada vez* que un nuevo oyente accede a la estación, se debe crear una nueva instancia de SS. En términos de funcionalidad, esta es la solución perfecta, pero esto es obviamente un malgasto de recursos. De hecho, este argumento también se encuentra en un antipatrón de rendimiento bien conocido: *asignación dinámica excesiva* (Smith & Williams 2001). Este antipatrón critica la práctica de crear una nueva instancia completa de un objeto o servidor para proporcionar un servicio a un nuevo cliente.

Esta necesidad de que el sistema evolucione puede ser detectada por una sola persona – pero examinar la estructura del AK también es de gran ayuda. Algunas veces, este proceso puede incluso ser soportado parcialmente por herramientas automáticas. Por ejemplo, parte de nuestro trabajo previo en la herramienta Morpheus (Navarro et al. 2010) añade la capacidad de ayudar en la detección de antipatrones (como el mencionado antes) mediante chequeo de la red de relaciones en el AK. Una vez que se ha detectado la necesidad de evolución, se planifica el paso evolutivo – la primera vez, será realizado a mano; pero en el futuro, situaciones similares serán manejadas por el estilo de evolución creado a partir de dicha experiencia.

4.5.3.3.4 Describiendo la solución: un estilo de evolución

En esta sección, se describe el *razonamiento* que utilizamos para describir la evolución de la arquitectura. Esta rationale será explícitamente capturada como parte del AK, incluyendo los pasos evolutivos – y, por tanto, será utilizada para *construir* un *patrón de evolución* como el definido en la sección 4.5.3.2.2.

Primero, se detecta una *condición de evolución*, requerida por la arquitectura para evolucionar (véase la sección previa). Esta condición tiene que ser evaluada dentro de una *decisión de evolución* (ED.1, véase Figura 29). ED.1 evalúa una primera decisión asociada a la condición. El arquitecto considera, como una primera alternativa, utilizar la cola y servir las peticiones con una política FCFS (*First-Come, First-Serve*) – de tal forma que los oyentes tienen que esperar para ser servidos. Esta rama alternativa (AB.1) no se considera aceptable – por tanto, es inhibida (rechazada) por la decisión ED.1.

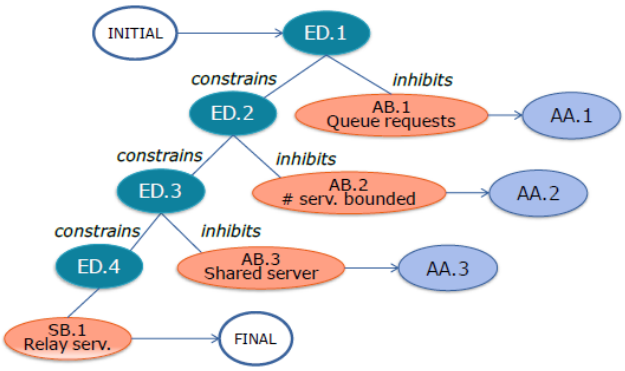


Figura 29. Proceso de decisión para el caso de estudio

Esto permite que el arquitecto considere una alternativa diferente, la nueva decisión se etiqueta como ED.2. Ahora, la arquitectura a considerar utiliza un número limitado de instancias del SS. Esto proporciona un coste fijo, fijando nuestro problema, pero tiene la consecuencia negativa de limitar el número de oyentes simultáneos, al igual que en AB.1. Por tanto, esta alternativa (AB.2) también es rechazada por ED.2.

El arquitecto considera ahora (ED.3) un enfoque diferente. Muchos usuarios estarían interesados exactamente en el mismo programa, e incluso al mismo tiempo: por tanto, en lugar de crear un servidor específico cada vez, se considera *compartir la misma instancia del servidor* para servir a todos estos oyentes. Pero esta instancia tendría una capacidad limitada, ya que no estamos usando toda la potencia del entorno elástico. Entonces, sólo seríamos capaces de servir a un número limitado de oyentes, y la situación es similar a lo que pasaba en AB.2. Por tanto, esta alternativa (AB.3) es rechazada, por las mismas razones.

Finalmente, el arquitecto considera (ED.4) una solución relacionada: utilizar un *sistema de retransmisión*. Sigue siendo cierto, como se consideraba antes, que a muchos oyentes les gustaría tener acceso al mismo programa. Por tanto, en lugar de conectarlos al mismo servidor, se proporcionan servidores *adicionales*. Para ello, es necesario un cuarto tipo de servicio que no estaba presente en la arquitectura original, el *servicio de retransmisión* (RS). Este servicio simplemente se conecta a la transmisión en vivo de algunos SS existentes, y comienza a emitir exactamente los mismos contenidos que está recibiendo, utilizando su propia URI. Un RS puede servir a un cierto número de clientes de una forma segura y, por tanto, la carga de trabajo se distribuye entre varias instancias de estos servicios. Por tanto, la rama seleccionada (SB.1) se basa en el enfoque de retransmisión para solucionar el problema. Como se verá más adelante, esto definirá el paso *final* en esta evolución específica.

La Figura 29 proporciona una representación gráfica simplificada de este proceso de decisión. Esta figura también es una representación de la información que será incluida como parte del AK. Además, puede ser considerada como una representación del *estilo de evolución* que acabamos de definir. De hecho, es sólo un patrón de evolución, aunque esto es sólo una diferencia de escala.

Como hemos visto, el proceso comienza con una única decisión de evolución (ED.1), la cual conduce a otras decisiones (ED.2-4) y cada vez sugiere una solución alternativa (AB.1-3). Esto define una cadena lógica de decisiones que va desde la cuestión inicial (ED.1) hasta la elección final (ED.4). Cada decisión define un paso evolutivo, donde cada rama alternativa propondría una arquitectura diferente (AA.1-3), por lo que, si se hubiera tomado una decisión diferente, el resultado habría sido muy diferente. Pero siguiendo nuestra cadena lógica de decisiones (todas ellas relacionadas), terminamos aceptando la solución final (SB.1), la cual es el resultado de una secuencia de pasos evolutivos. Ésta es exactamente nuestra definición de *patrón de evolución*.

Un *estilo de evolución*, como ya se ha definido (4.5.3.2.2), será un conjunto de dichos patrones de evolución, los cuales tendrían un objetivo común. El caso particular de un estilo con un único patrón también está permitido, por lo que podemos considerar este ejemplo como un *estilo de evolución* completo, definiendo el cambio desde la Figura 28 a la Figura 30.

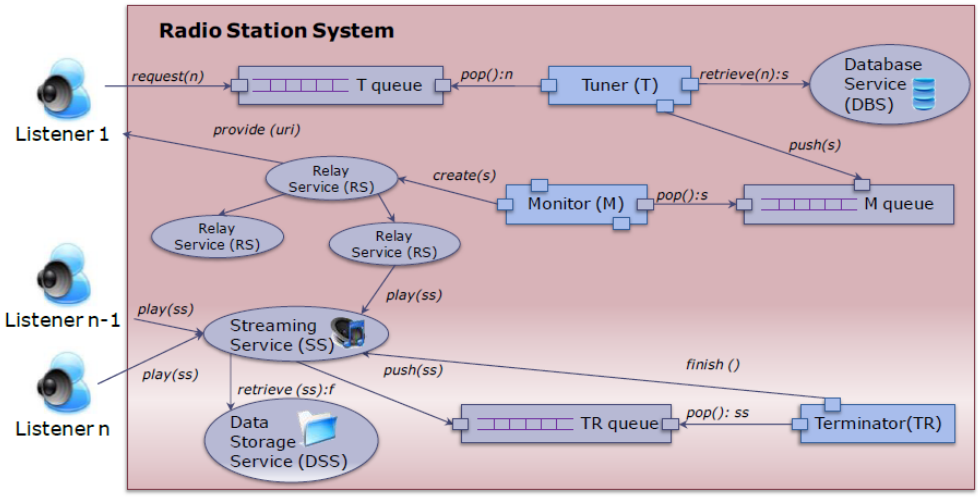


Figura 30. Sistema de Estación de Radio en tiempo de ejecución: versión 2.0

Se deben hacer dos comentarios importantes sobre la solución escogida. Primero, sobre la manera en que esta única opción (*"utilizar un servicio de retransmisión"*) puede afectar a toda la arquitectura; y segundo, explicar por qué es una solución acertada.

Primero, nótese que sólo se necesita una instancia SS, la original; el primer RS sólo necesita acceder a la transmisión proporcionada por esta instancia, y comienza en sí mismo la transmisión. A medida que se solicitan más instancias del mismo programa, otra instancia del RS podría ser necesaria. Este segundo RS no necesita conectarse al SS original, sino al primer RS (véase Figura 30). Por tanto, el sistema de retransmisión crea una *cadena de retransmisiones*, cada una capaz de servir a varios clientes, y también a la siguiente retransmisión.

Así tenemos, primero, la arquitectura original (Figura 28) que ha evolucionado desde un estilo cliente-servidor a una arquitectura híbrida que incluye una cadena P2P de retransmisiones (Figura 30); y segundo, el flujo de trabajo tiene que adaptarse para que el monitor cree una nueva instancia RS cuando un nuevo oyente la necesite, y que le proporcione al mismo la correspondiente URI.

Esta solución no es una solución obvia: el sistema todavía crea múltiples instancias de servidor. Y el RS también consume recursos, por lo que también cuesta dinero. Sin embargo, esta solución es más barata: primero, se crean menos instancias; y segundo, cada instancia es además más barata. El RS es, probablemente, menos complejo que el SS; y lo que es más importante, el RS *no necesita* utilizar toda la información almacenada, ya que la obtiene del otro servicio. Por tanto, es puramente *computacional*.

En resumen, el ejemplo muestra cómo nuestro uso del AK hace posible decidir cómo evolucionar desde la arquitectura original al enfoque final – y en el proceso, cómo definir un estilo de evolución para reutilizar este conocimiento.

4.5.3.4 Conclusiones y trabajo futuro

La arquitectura software es un operador crítico en el desarrollo y evolución de sistemas software. Además, demandamos que el conocimiento de la arquitectura es igualmente un operador crítico. Brindamos dos conclusiones principales: primero, que el AK puede considerarse en sí mismo como un *operador de evolución*, en el sentido de que proporciona mucha información de especial relevancia al proceso de evolución; y segundo, que gran parte del proceso de evolución en sí mismo puede ser capturado como parte del AK, utilizando conceptos de evolución a nivel arquitectónico.

Para simplificar la reutilización de este conocimiento, también proponemos una estructura específica que captura la información y las decisiones relacionadas con la evolución arquitectónica: el *estilo de evolución*. Describimos la estructura de este estilo, y el proceso para definirlo y reutilizarlo. Para ilustrar la viabilidad y utilidad de este enfoque, se describe con cierto detalle el uso de este estilo de evolución en el contexto de un ejemplo de aplicación basada en la nube, proporcionando una perspectiva clara de los conceptos involucrados.

El trabajo futuro en este contexto incluye la completa integración de esta propuesta (genérica) en un enfoque específico. Actualmente, ya estamos incluyendo estas nociones en ATRIUM (Navarro & Cuesta 2008), un proceso que define y gestiona las arquitecturas software desde la fase de requisitos. La incorporación de estilos de evolución en ATRIUM (y su conjunto de herramientas) será sólo cuestión de extender la red AK para incluir los nuevos tipos de relaciones (aquellas relacionadas con la evolución), y

proporcionar un ciclo adicional en su proceso de desarrollo dirigido por modelos que aplicaría estilos de evolución en la parte superior de la arquitectura actual y su conocimiento asociado.

Capítulo 5

Currículum Vitae

5.1 Información Personal

Apellidos, Nombre: Roda Sánchez, Cristina

Nacionalidad: Española

Fecha de Nacimiento: 01/10/1987

Lugar de Nacimiento: Albacete

DNI: 47088956-K

Domicilio: C/Santander, 3, 6ºG, 02002, Albacete, España

Teléfono: +34 617 74 13 00

e-mail: cristinarodasanchez@gmail.com

5.2 Títulos

- Ingeniería Informática Superior (2005-2010), Escuela Superior de Ingeniería Informática de Albacete, Universidad de Castilla-La Mancha
  - Nota Media: 79/100
  - Proyecto Final de Carrera realizado con la compañía *Hewlett-Packard* sobre Custodia Electrónica y Firma Electrónica Avanzada (Matrícula de Honor).
- Grado en Ingeniería Informática (2011), Escuela Superior de Ingeniería Informática de Albacete, Universidad de Castilla-La Mancha.
- Grado Intermedio de Inglés (2005), Escuela Oficial de Idiomas de Albacete.

5.3 Publicaciones

Los trabajos desarrollados hasta la fecha, relacionados con la futura tesis doctoral, han sido publicados o enviados a los siguientes foros:

- Elena Navarro, Carlos E. Cuesta, Dewayne E. Perry, Cristina Roda: *Using Model Transformation Techniques for the Superimposition of Architectural Styles*. Emerging paper in the 5th European Conference on Software Architecture (ECSA 2011), Essen, Germany, 13-16 September 2011 (aceptado).
- Cristina Roda, Elena Navarro, Carlos E. Cuesta y Dewayne E. Perry, *Técnicas de Visualización para Conocimiento Arquitectónico: una Evaluación Empírica*, XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2011), A Coruña, 5-7 Septiembre 2011 (aceptado).
- Cristina Roda, Elena Navarro, Carlos E. Cuesta y Dewayne E. Perry, *An Empirical Evaluation of Visualization Techniques for Architectural Knowledge*, IET Software (enviado).
- Carlos E. Cuesta, Elena Navarro, Dewayne E. Perry, Cristina Roda, *Using Architectural Knowledge as an Evolution Driver*, Software Evolution and Maintenance (enviado).

5.4 Experiencia Laboral

- Clases particulares a alumnos de E.S.O. y Bachillerato.
- Cuidado de niños pequeños.

5.5 Trabajos Voluntarios

- Participación en proyectos, eventos, recogidas de alimentos, etc., organizados por el Centro de Juventud de Albacete y Manos Unidas.
- Organización de la Semana Cultural de la Escuela Superior de Ingeniería Informática de Albacete.
- Organización de convivencias al aire libre para niños y jóvenes.

5.6 Cursos y Actividades

- Título: "Computación Grid: del Middleware a las aplicaciones"
  - Lugar: Albacete
  - Organizador: Vicerrectorado de Extensión Universitaria, UCLM
  - Año: 2007
- Título: "Torneo First Lego League y taller de robótica"
  - Lugar: Albacete
  - Organizador: Parque Científico y Tecnológico de Albacete
  - Año: 2009
- Título: "Eclipse en el Soporte al Desarrollo de Software Profesional. III Edición"
  - Lugar: Albacete
  - Organizador: Escuela Superior de Ingeniería Informática de Albacete
  - Año: 2010

Referencias

Akerman, A. & Tyree, J., 2006. Using ontology to support development of software architectures. *IBM Systems Journal*, 45(4), p.813–825. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5386636 [Accessed May 4, 2011].

Andrews, K. & Heidegger, H., 1998. Information Slices : Visualising and Exploring Large Hierarchies using Cascading , Semi-Circular Discs. In *Proceedings of the IEEE Information Visualization Symposium*. Carolina, pp. 9-12.

Anon, 2000. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*,

Armbrust, M. et al., 2009. *Above the Clouds: A Berkeley View of Cloud Computing*, TechRep No. UCB/EECS-2009-28.

Baader, F. et al., 2003. *The Description Logic Handbook* Franz Baader et al., eds., Cambridge University Press. Available at: http://portal.acm.org/citation.cfm?id=1215128.

Babar, M.A., Gorton, I. & Kitchenham, B., 2006. A framework for supporting architecture knowledge and rationale management. In A. H. Dutoit et al., eds. *Rationale Management in Software Engineering*. Springer, pp. 237-254. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+Framework+for+Supporting+Architecture+Knowledge+and+Rationale+Management#0.

Babu T, L. et al., 2007. ArchVoc--Towards an Ontology for Software Architecture. In *Second Workshop on Sharing and Reusing Architectural Knowledge*. IEEE Comput. Soc, pp. 5-5. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4273345.

Barabási, A.-L., 2002. *Linked: The New Science of Networks*, Perseus.

Barabási, A.-L., Newman, M. & Watts, D.J., 2006. *The Structure and Dynamics of Networks*,

Bass, L., Clements, P. & Kazman, Rick, 2003. *Software Architecture in Practice* Second Edi., Boston: Addison-Wesley Professional.

Bersoff, E.H., Henderson, V.D. & Siegel, S.G., 1980. *Software configuration management. An investment in product integrity.*, New York, New York, USA: Addison-Wesley.

Biehl, M. & Torngren, M., 2010. An Executable Design Decision Representation Using Model Transformations. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, pp. 131-134. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5598089 [Accessed April 1, 2011].

Borgatti, S.P., 2002. NetDraw: Graph Visualization Software. *Harvard Analytic Technologies*, 2006.

Borgatti, S.P., Everett, M.G. & Freeman, L.C., 2002. Ucinet for Windows: Software for Social Network Analysis. *Harvard Analytic Technologies*, 2006. Available at: http://www.analytictech.com/downloaduc6.htm.

Bosch, J., 2004. Software Architecture: The Next Step. In *1st European Workshop in Software Architecture (EWSA'04)*. Heidelberg: Springer, pp. 194-199.

Brandozzi, M. & Perry, D E, 2001. Transforming Goal Oriented Requirement Specifications into Architecture Prescriptions. In *Workshop from Soft. Req. to Arch.* pp. 54-61. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.7510&rep=rep1&type=pdf.

Bratthall, L., Johansson, E. & Regnell, Björn, 2000. Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software. In *2nd International Conference on Product Focused Software Process Improvement (PROFES 2000)*. Springer, pp. 126-139.

Bruegge, B., Dutoit, A.H. & Wolf, T., 2006. Sysiphus : Enabling informal collaboration in global software development. In *1st International Conference on Global Software Engineering*. Costao do Santinho, Florianópolis, Brazil.

Burge, J.E. & Brown, D.C., 2004. An Integrated Approach for Software Design Checking Using Design Rationale. In *Proceedings of the Design, Computing and Cognition Conference*. Cambridge.

Burge, J.E. et al., 2008. *Rationale-Based Software Engineering*, Springer.

Capilla, R. et al., 2006. A Web-based Tool for Managing Architectural Design Decisions. *Design*, 31(5), p.4. Available at: http://portal.acm.org/citation.cfm?id=1178644.

Caseau, Y., Krob, D. & Peyronnet, S., 2007. Complexité des Systèmes d'Information: une Famille des Mesures de la Complexité Scalaire d'un Schéma d'Architecture. *Génie logiciel*, 82, pp.23-30. Available at: http://www.usine-logicielle.org/downloads/Art-Public-Mesures_Complexite_FR.pdf [Accessed May 7, 2011].

Castro, J., Kolp, M. & Mylopoulos, J., 2002. Towards Requirements-Driven Software Development Methodology: The Tropos Project. *Information Systems*, 27(6), pp.365-389.

Clements, P. et al., 2002. *Documenting Software Architectures: Views and Beyond*, Addison-Wesley Professional. Available at: http://portal.acm.org/citation.cfm?id=599933.

Clements, P., Kazman, Rick & Klein, Mark, 2001. *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley Professional. Available at: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/020170482X.

Conklin, J. & Begeman, M.L., 1988. gIBIS : A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6(4), pp.303-331.

Czarnecki, K. & Helsen, S., 2006. Classification of Model Transformation Approaches J. Bettin et al., eds. *IBM Systems Journal*, 45(3), pp.621-645. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.6773&rep=rep1&type=pdf.

De Boer, R.C. et al., 2009. Ontology-driven visualization of architectural design decisions. *2009 Joint Working IEEEIFIP Conference on Software Architecture European Conference on Software Architecture*, 82(8), pp.51-60. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5290791.

de Boer, R.C. et al., 2007. Architectural Knowledge: Getting to the Core. In S. Overhage et al., eds. *3rd International Conference on Quality of Software Architectures (QoSA'07)*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 197-214. Available at: http://dblp.uni-trier.de/db/conf/qosa/qosa2007.html#BoerFLVCJ07 [Accessed March 25, 2011].

De Bruin, H. & Van Vliet, H., 2003. Quality-driven software architecture composition. *Journal of Systems and Software*, 66(3), pp.269-284. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0164121202000791.

Delisle, N. & Garlan, D, 1989. Formally Specifying Electronic Instruments. In *Proceedings of the 5th international workshop on Software specification and design*. New York, USA: ACM, pp. 242-248. Available at: http://portal.acm.org/citation.cfm?id=75200.75236.

Dutoit, A.H. et al., 2006. *Rationale Management in Software Engineering*, Springer.

Eindhoven, T.C. science department T.U., 2002. SequoiaView. Available at: http://www.win.tue.nl/sequoiaview/.

Eklund, P., Roberts, N. & Green, S., 2002. OntoRama: Browsing RDF ontologies using a hyperbolic-style browser. In *First International Symposium on Cyber Worlds, 2002. Proceedings.* IEEE Comput. Soc, pp. 405-411. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1180907.

Elrad, T., Filman, R.E. & Bader, A., 2001. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10), pp.29-32. Available at: http://portal.acm.org/ft_gateway.cfm?id=383853&type=html.

Erfanian, A. & Shams Aliee, F., 2008. An ontology-driven software architecture evaluation method. *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge SHARK 08*, p.79. Available at: http://portal.acm.org/citation.cfm?doid=1370062.1370081.

Falessi, D., Becker, M. & Cantone, G., 2006. Design decision rationale: experiences and steps ahead towards systematic use. *ACM SIGSOFT Software Engineering Notes*, 3(5).

Falessi, D., Cantone, G. & Kruchten, P., 2008. Value-Based Design Decision Rationale Documentation: Principles and Empirical Feasibility Study. In *Seventh Working IEEE/IFIP Conference on Software Architecture WICSA 2008*. New York, USA: IEEE CS Press, pp. 189-198. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4459157.

Farenhorst, R. & de Boer, R.C., 2006. *Core Concepts of an Ontology of Architectural Design Decisions*, Available at: http://www.cs.vu.nl/~se/griffin/pubs/IR-IMSE-002.pdf [Accessed May 4, 2011].

Farenhorst, R., Lago, P. & Van Vliet, H., 2007. EAGLE: Effective Tool Support for Sharing Architectural Knowledge. *International Journal of Cooperative Information Systems*, 16(3-4), pp.413-437. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-36749031758&partnerID=40.

Farenhorst, R., Lago, P. & Vliet, H.V., 2007. Effective Tool Support for Architectural Knowledge Sharing. In *1st European Conference on Software Architecture*. Madrid, pp. 123-138.

Fenton, N.E. & Pfleeger, S.L., 1997. *Software Metrics: A Rigorous and Practical Approach*, Thomson Computer Press.

Fischer, G. et al., 1990. JANUS : Integrating Hypertext With a Knowledge-Based Design Environment. *Source*, (July).

FZI, 2011. KAON. *http://kaon.semanticweb.org/*. Available at: http://kaon.semanticweb.org/ [Accessed April 14, 2007].

G3RD-CT-00794, G., 2003. EFTCOR: Environmental Friendly and cost-effective Technology for Coating Removal.

Gamma, E. et al., 1995. *Design patterns: elements of reusable object-oriented software* B. Kernighan, ed., Addison-Wesley. Available at: http://www.cs.up.ac.za/cs/aboake/sws780/references/patternstoarchitecture/Gamma-DesignPatternsIntro.pdf.

Garcia, A. et al., 2006. Driving and managing architectural decisions with aspects. In *SHARK 2006*. New York, USA: ACM DL.

Garlan, D. et al., 2009. Evolution Styles: Foundations and Tool Support for Software Architecture Evolution. In *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA 2009)*. pp. 131-140.

Garlan, D. & Perry, D E, 1995. Introduction to the Special Issue on Software Architecture. *IEEE Transactions on Software Engineering*, 21(4), pp.269-274. Available at: http://portal.acm.org/citation.cfm?id=205314.

Garlan, D. & Shaw, M., 1993. An Introduction to Software Architecture. In V. Ambriola & G. Tortora, eds. *Advances in Software Engineering and Knowledge Engineering*. Singapore, p. 1–40. Available at: http://portal.acm.org/citation.cfm?id=865128.

Gruber, T.R., 1993. Technical Report KSL 92-71 Revised April 1993 A Translation Approach to Portable Ontology Specifications by A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, (April).

Hanneman, R.A. & Riddle, M., 2005. *Introduction to social network methods*, University of California, Riverside. Available at: http://faculty.ucr.edu/~hanneman/.

Harrison, N.B., Avgeriou, Paris & Zdun, Uwe, 2007. Using Patterns to Capture Architectural Decisions. *IEEE Software*, 24(4), pp.38-45.

Hilliard, R. et al., 2010. Realizing architecture frameworks through megamodelling techniques. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, p. 305–308. Available at: http://portal.acm.org/citation.cfm?id=1859057 [Accessed May 4, 2011].

Hofmeister, C., Nord, R & Soni, D., 1999. *Applied software architecture*, Addison-Wesley. Available at: http://books.google.com/books?id=3klAPCIB3hQC&pgis=1.

Jansen, A., Avgeriou, Paris & van der Ven, J.S., 2009. Enriching software architecture documentation. *Journal of Systems and Software*, 82(8), pp.1232-1248. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0164121209001046 [Accessed April 7, 2011].

Jansen, A. & Bosch, J., 2005. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE / IFIP Conference on Software Architecture (WICSA 2005)*. pp. 109-120. Available at: http://dblp.uni-trier.de/db/conf/wicsa/wicsa2005.html#JansenB05 [Accessed March 25, 2011].

Jansen, A., Bosch, J. & Avgeriou, Paris, 2008. Documenting after the fact: Recovering architectural design decisions. *Journal of Systems and Software*, 81(4), pp.536-557. Available at: http://dblp.uni-trier.de/db/journals/jss/jss81.html#JansenBA08 [Accessed March 18, 2011].

Jansen, A. et al., 2008. Sharing the Architectural Knowledge of Quantitative Analysis. In *Proceedings of the Quality of Software- Architectures (QoSA 2008)*. Springer, pp. 220-234. Available at: http://www.springerlink.com/index/2416705473611043.pdf.

Kajsa, P., Majtas, L. & Navrat, P., 2011. Design pattern instantiation directed by concretization and specialization. *Computer Science and Information Systems*, 8(1), pp.41-72. Available at: http://www.doiserbia.nb.rs/Article.aspx?ID=1820-02141000032K [Accessed May 4, 2011].

Katifori, A. et al., 2007. Ontology visualization methods: a survey. *ACM Computing Surveys*, 39(4), p.10-es. Available at: http://portal.acm.org/citation.cfm?doid=1287620.1287621.

Kazman, Rick et al., 1999. Experience with performing architecture tradeoff analysis. *Proceedings of the 21st international conference on Software engineering ICSE 99*, pp.54-63. Available at: http://portal.acm.org/citation.cfm?doid=302405.302452.

Kazman, Rick et al., 1994. SAAM: a method for analyzing the properties of software architectures. In *Proceedings of 16th International Conference on Software Engineering*. IEEE Comput. Soc. Press, pp. 81-90. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=296768.

Kitchenham, B.A. et al., 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8), pp.721-734. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1027796.

Klein, M., 1997. An Exception Handling Approach to Enhancing Consistency, Completeness, and Correctness in Collaborative Requirements Capture. *Journal of Concurrent Engineering Research and Applications*, 5(1), pp.73-80. Available at: http://cer.sagepub.com/cgi/doi/10.1177/1063293X9700500105.

Kruchten, P., 2004. An Ontology of Architectural Design Decisions. In *Proceedings of 2nd Groningen Workshop on Software Variability Management Groningen NL 2004*. pp. 54-61.

Kruchten, P., 1995. The 4+1 View Model of Architecture. *IEEE Software*, 12(6), pp.42-50. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=469759.

Kruchten, P., 2000. *The Rational Unified Process: An Introduction*, Addison-Wesley Professional. Available at: http://www.amazon.com/Rational-Unified-Process-Introduction-2nd/dp/0201707101.

Kruchten, P., Capilla, R. & Dueñas, J.C., 2009. The Decision View's Role in Software Architecture Practice. *IEEE Software*, 26(2), pp.36-42.

Kruchten, P., Lago, P. & Van Vliet, H., 2006. Building Up and Reasoning about Architectural Knowledge. In Christine Hofmeister, I. Crnkovic, & R. Reussner, eds. *2nd International Conference on Quality of Software Architectures (QoSA'06)*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 43-58. Available at: http://dblp.uni-trier.de/db/conf/qosa/qosa2006.html#KruchtenLV06 [Accessed March 9, 2011].

Kunz, W. & Rittel, H., 1970. Issues as elements of information systems. *Journal of the Electrochemical Society*, (131). Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.134.1741&rep=rep1&type=pdf [Accessed June 23, 2011].

Könemann, P. & Zimmermann, O., 2010. Linking Design Decisions to Design Models in Model-Based Software Development (Draft). In *European Conference on Software Architecture*. Springer, pp. 246-262. Available at: http://www.springerlink.com/index/93W138L486561074.pdf.

Lago, P. & Van Vliet, H., 2005. Explicit Assumptions Enrich Architectural Models. In *27th International Conference on Software Engineering (ICSE'05)*. New York, USA: IEEE CS Press, pp. 206-214.

Le Goaer, O. et al., 2008. Evolution Styles to the Rescue of Architectural Evolution Knowledge. In *Proceedings of the 3rd International workshop on Sharing and reusing architectural knowledge (SHARK'08)*. pp. 31-36.

Lee, J., 1990. Sibyl: A Qualitative Decision Management System. *Cambridge, Mass. : Center for Coordination Science, Massachusetts Institute of Technology, Sloan School of Management*, pp.106-133.

Lee, L. & Kruchten, P., 2008. A Tool to Visualize Architectural Design Decisions. In *QoSA 2008*. pp. 43-54.

Lehman, M.M. & Belady, L.A., 1985. *Program evolution*, Academic Press.

Lehman, M.M. et al., 1997. Metrics and Laws of Software Evolution - The Nineties View. In *Fourth International Software Metrics Symposium (METRICS'97)*. p. 20.

Lund, A.M., 2001. Questionnaire for User Interface Satisfaction. *http://oldwww.acm.org/perlman/question.cgi?form=USE*. Available at: http://oldwww.acm.org/perlman/question.cgi?form=USE [Accessed April 2, 2011].

Maclean, A. et al., 1991. Questions, options, and criteria: Elements of design space analysis T. P. Moran & J M Carroll, eds. *Human-Computer Interaction*, 6(3), pp.201-250. Available at: http://www.informaworld.com/index/784767489.pdf.

Madhavji, N.H., Fernández-Ramil, J. & Perry, Dewayne E. eds., 2006. *Software Evolution and Feedback: Theory and Practice*, John Wiley.

Mattsson, A. et al., 2007. Experiences from Representing Software Architecture in a Large Industrial Project Using Model Driven Development. In *Second Workshop on Sharing and Reusing Architectural Knowledge Architecture Rationale and Design Intent SHARKADI07 ICSE Workshops 2007*. Ieee, pp. 6-6. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4273346.

Mattsson, A. et al., 2008. Linking model-driven development and software architecture: A case study. *IEEE Transactions on Software engineering*, 35(1), p.83–93. Available at: http://www.computer.org/portal/web/csdl/doi/10.1109/TSE.2008.87 [Accessed May 4, 2011].

McCall, R., 1991. PHI: a conceptual foundation for design hypermedia. *Design Studies*, 12(1), pp.30-41. Available at: http://linkinghub.elsevier.com/retrieve/pii/0142694X9190006I.

Medvidovic, Nenad & Taylor, R.N., 2000. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Software Eng.*, 26(1), pp.70-93.

Mehta, N.R. & Medvidovic, N, 2003. Composing architectural styles from architectural primitives. *ACM SIGSOFT Software Engineering Note*, 28(5), pp.347-350. Available at: http://portal.acm.org/citation.cfm?doid=949952.940118.

Merelo, J.J., 2006. Redes Sociales: una introducción. *Revista Redes*.

Mettala, E. & Graham, M.H., 1992. *The domain-specific software architecture program* Special Re., Citeseer. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.2572&rep=rep1&type=pdf [Accessed June 14, 2011].

Mittermeir, R.T., 2006. Facets of Software Evolution. In N. H. Madhavji, J. Fernández-Ramil, & Dewayne E. Perry, eds. *Software Evolution and Feedback: Theory and Practice*. Wiley, p. 71--94.

Montero, F. & Navarro, E., 2009. ATRIUM: Software Architecture Driven by Requirements. In *2009 14th IEEE International Conference on Engineering of Complex Computer Systems*. Potsdam, Germany: IEEE, pp. 230-239. Available at: http://dblp.uni-trier.de/db/conf/iceccs/iceccs2009.html#MonteroN09 [Accessed March 11, 2011].

Nakamura, T. & Basili, V.R., 2005. Metrics of Software Architecture Changes Based on Structural Distance. In F. Lanubile & C. Seaman, eds. *Proceedings of the 11th International Software Metrics Symposium*. IEEE Computer Society, pp. 8-18.

Navarro, E., 2007. *ATRIUM: Architecture Traced from Requirements by Applying a Unified Methodology*. PhD thesis, University of Castilla-La Mancha. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:ATRIUM:+Architecture+Traced+from+RequIrements+by+applying+a+Unified#0 [Accessed May 5, 2011].

Navarro, E. & Cuesta, C.E., 2008. Automating the Trace of Architectural Design Decisions and Rationales Using a MDD Approach. In R. Morrison, D. Balasubramaniam, & K. Falkner, eds. *Second European Conference Software Architecture (ECSA 2008)*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 114-130. Available at: http://dblp.uni-trier.de/db/conf/ecsa/ecsa2008.html#NavarroC08 [Accessed March 11, 2011].

Navarro, E., Cuesta, C.E. & Perry, D E, 2009. Weaving a network of architectural knowledge. In *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*. IEEE, pp. 241-244. Available at: http://dblp.uni-trier.de/db/conf/wicsa/wicsa2009.html#NavarroCP09 [Accessed March 11, 2011].

Navarro, E., Cuesta, C.E. & Perry, Dewayne E., 2010. Antipatterns for architectural knowledge management. *(under review)*.

Navarro, E. et al., 2009. MORPHEUS: a supporting tool for MDD. In *18th Int. Conf. on Information Systems Development (ISD2009)*. Nanchang, China: Springer, p. 255–267. Available at: http://www.springerlink.com/index/L257652626158NN4.pdf [Accessed June 9, 2011].

Newman, M., Barabasi, A.-L. & Watts, D.J., 2006. *The structure and dynamics of networks* M. E. J. Newman, A. L. Barabási, & D. J. Watts, eds., Princeton University Press. Available at: http://www.amazon.com/dp/0691113572.

Noppen, J. & Tamzalit, D., 2010. ETAK: Tailoring Architectural Evolution by (re-)using Architectural Knowledge. In *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge - SHARK '10*. New York, New York, USA: ACM Press, pp. 21-28. Available at: http://portal.acm.org/citation.cfm?doid=1833335.1833339 [Accessed March 29, 2011].

Northrop, L. et al., 2006. *Ultra-Large-Scale Systems - The Software Challenge of the Future* W. Pollak, ed., Software Engineering Institute, Carnegie Mellon University. Available at: http://www.sei.cmu.edu/uls/downloads.html.

Noy, N.F., Fergerson, R.W. & Musen, M.A., 2000. The knowledge model of Protégé-2000 : combining interoperability and flexibility. , (1), pp.1-20.

Object Management Group, 2003. MDA Guide V1.0.1.

OMG doc. ptc/05-11-01, 2005. QVT. *MOF Query/Views/Transformations final adopted specification*.

Ortiz, F. et al., 2005. A reference control architecture for service robots implemented on a climbing vehicle. In *10th Ada-Europe Int. Conf. on Reliable Software Technologies*. York, UK: Springer-Verlag, p. 13–24. Available at: http://www.springerlink.com/index/JF0PL7YGVKKYGUHK.pdf [Accessed June 29, 2011].

Padak, N. & Padak, G., 1994. Guidelines for Planning Action Research Projects. Research to Practice. Available at: http://eric.ed.gov/ERICWebPortal/recordDetail?accno=ED380699.

Parsia, B., Wang, T. & Golbeck, J., 2005. Visualizing Web Ontologies with CropCircles. In *Proceedings of the 4th International Semantic Web Conference*. pp. 6-10.

Perry, D. E., 2006. A Nontraditional View of the Dimensions of Software Evolution. In N. H. Madhavji, J. Fernández-Rami, & Dewayne E. Perry, eds. *Software Evolution and Feedback: Theory and Practice*. Wiley, p. 41--52.

Perry, D. E., 1998. Generic Architecture Descriptions for Product Lines. In *Development and Evolution of Software Architectures for Product Families*. LNCS 1429, pp. 51-56.

Perry, D. E. & Wolf, A.L., 1992. Foundations for the Study of Software Architecture. *ACM Software Engineering Notes*, 17(4), pp.40-52.

Peña-Mora, F. & Vadhavkar, S., 1997. Augmenting Design Patterns with Design Rationale. *Artif. Intell. For Eng. Design, Analysis and Manuf.*, 11(2), pp.93-108.

Putman, J., 2001. *Architecting With RM-ODP*, Prentice Hall.

Pérez, J. et al., 2006. Designing software architectures with an aspect-oriented architecture description language. In *9th Int. Symp. on Component-Based Software Engineering*. Springer, pp. 123 - 138. Available at: http://www.springerlink.com/index/l6p247v53864w7t8.pdf.

Regli, W.C. et al., 2000. A Survey of Design Rationale Systems: Approaches, Representation, Capture and Retrieval. *Engineering With Computers*, 16(3-4), pp.209-235. Available at: http://www.springerlink.com/index/10.1007/PL00013715.

Ric, H., 2001. Sofware Architecture as a Shared Mental Model. In *ASERC Workshop Software Architecture*.

Rivadeneira, W. & Bederson, B.B., 2003. *A Study of Search Result Clustering Interfaces : Comparing Textual and Zoomable User Interfaces*,

Rozanski, N. & Woods, E., 2005. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*, Addison-Wesley Professional. Available at: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0321112296.

Selic, B., 2003. The pragmatics of model-driven development. *IEEE Software*, 20(5), pp.19-25. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1231146.

Shahin, M., Liang, P. & Khayyambashi, M.R., 2010a. Improving Understandability of Architecture Design through Visualization of Architectural Design Decision. In *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. ACM Press, pp. 88-95.

Shahin, M., Liang, P. & Khayyambashi, M.R., 2010b. Rationale visualization of software architectural design decision using Compendium. In *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*. New York, New York, USA: ACM Press, p. 2. Available at: http://portal.acm.org/citation.cfm?doid=1774088.1774577.

Shaw, M., 1994. Procedure Calls Are the Assembly Language of Software Interconnection: Connectors Deserve First-Class Status. In D. Lamb, ed. *Studies of Software Design*. Springer, pp. 17-32. Available at: http://portal.acm.org/citation.cfm?id=864998.

Shaw, M. & Garlan, David, 1994. *Characteristics of Higher-level Languages for Software Architecture, Technical Report CMU-CS-94-210*, Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.8626.

Shipman, F.M. & McCall, R., 1997. Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication. *Artif. Intell. for Eng. Design, Analysis and Manuf.*, 11(2), pp.141-154. Available at: http://www.journals.cambridge.org/abstract_S089006040000192X.

Shneiderman, B., 1992. Tree visualization with Tree-maps : A 2-d space-filling approach. *ACM Trans. Graph.*, 11(1), pp.92-99.

Sinnema, M. & Deelstra, S., 2007. Classifying variability modeling techniques. *Information and Software Technology*, 49(7), pp.717-739. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0950584906001042.

Sinnema, M., Van der Ven, J. & Deelstra, S., 2006. Using Variability Modeling Principles to Capture Architectural Knowledge. In *Sharing and Reusing Architectural Knowledge (SHARK 2006)*. ACM DL.

Smith, C.U. & Williams, L.G., 2001. Software Performance AntiPatterns; Common Performance Problems and their Solutions. In *27th International Computer Measurement Group Conference*. pp. 797-806.

Storey, M.-A. et al., 2001. Jambalaya : Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé. In *Workshop on Interactive Tools for Knowledge Capture (K-CAP 2001)*. Victoria, BC, Canada.

Sure, Y., Angele, J. & Staab, S., 2002. OntoEdit : Guiding Ontology Development by Methodology and Inferencing. *Management*, pp.1205-1222.

Sutcliffe, A.G. & Ryan, M., 1998. Experience with SCRAM , a SCenario Requirements Analysis Method. *3rd International Conference on Requirements Engineering, CA: IEEE Computer Society Press*, pp.164-171.

Szyperski, C., 1997. *Component Software: Beyond Object-Oriented Programming (ACM Press)*, Addison-Wesley Professional. Available at: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0201178885.

Sánchez, P. et al., 2010. Model-driven development for early aspects. *Information and Software Technology*, 52(3), pp.249-273. Available at: http://www.sciencedirect.com/science/article/B6V0B-4XMD5H4-1/2/6ea38b8dfd8b3a555a3eacba8e25f226.

Tamzalit, D. et al., 2006. Updating software architectures : A style-based approach. In *International Conference on Software Engineering Research and Practice*. pp. 313-318.

Tang, A., 2007. *A Rationale-based Model for Architecture Design Reasoning*. Swinburne University of Technology.

Tang, A. et al., 2010. A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, 83(3), pp.352-370. Available at: http://dblp.uni-trier.de/db/journals/jss/jss83.html#TangAJCB10 [Accessed February 16, 2011].

Tang, A. et al., 2006. A survey of architecture design rationale. *Journal of Systems and Software*, 79(12), pp.1792-1804. Available at: http://dblp.uni-trier.de/db/journals/jss/jss79.html#TangBGH06 [Accessed March 25, 2011].

Tang, A. & Han, J., 2005. Architecture rationalization: A methodology for architecture verifiability, traceability and completeness. In P. J. Rozenblit J O'Neill T, ed. *Proceedings 12th IEEE International Conference and Workshops on the Engineering of ComputerBased Systems ECS 2005*. IEEE Comput. Soc, pp. 135-144. Available at: http://www.computer.org/portal/web/csdl/doi/10.1109/ECBS.2005.17.

Tang, A., Jin, Y. & Han, J., 2007. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6), pp.918-934. Available at: http://dblp.uni-trier.de/db/journals/jss/jss80.html#TangJH07 [Accessed March 25, 2011].

Tang, A. et al., 2005. Predicting Change Impact in Architecture Design with Bayesian Belief Networks. In *WICSA 2005 5th Working IEEE/IFIP Conference on Software Architecture*. IEEE Comput. Soc, pp. 67-76. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1620092.

Tyree, J. & Akerman, A., 2005. Architecture Decisions: Demystifying Architecture. *IEEE Software*, 22(2), pp.19-27.

van der Ven, J.S. et al., 2006. Design Decisions: the Bridge between Rationale and Architecture. In A. Dutoit, ed. *Rationale Management in Software Engineering*. Berlin, Heidelberg: Springer, pp. 329-346.

Varia, J., 2008. *Building GrepTheWeb in the Cloud, Part 1: Cloud Architectures*, Available at: http://aws.amazon.com/articles/1632?_encoding=UTF8&jiveRedirect=1.

Velázquez, O.A. & Aguilar, N., 2005. Manual Introductorio al Análisis de Redes Sociales. *Revista Redes*.

Wang, T. & Parsia, B., 2006. CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies. In *Proceedings of the International Semantic Web Conference (ISWC 06)*. Storming Media, pp. 695-708.

Wasserman, S. & Faust, K., 1994. *Social Network Analysis: Methods and Applications*, Cambridge University Press.

Watson, R., Bhattacharya, S. & Perry, Dewayne E., 2010. Statically Defined Dynamic Architecture Evolution. In *1st International Workshop on Automated Configuration and Tailoring of Applications (ACOTA 2010) collocated with ASE 2010*. Antwerp, BE.

Wils, A. et al., 2006. *Agility in the avionics software world* 2006th ed. P. Abrahamsson, M. Marchesi, & G. Succi, eds., Berlin Heidelberg: Springer-Verlag. Available at: http://www.springerlink.com/index/f21058t8m7414650.pdf [Accessed May 7, 2011].

Wohlin, C. et al., 2000. *Experimentation in Software Engineering: An Introduction*,

Wu, J. & Storey, M.-A., 2000. A Multi-Perspective Software Visualization Environment. In *Proceedings of the 2000 Conference of the Centre for Advanced Studies on Collaborative Research*.

Yin, R.K., 2002. *Case Study Research: Design and Methods, Third Edition, Applied Social Research Methods Series, Vol 5*, Sage Publications, Inc. Available at: http://www.amazon.com/Case-Study-Research-Methods-Applied/dp/0761925538 [Accessed March 18, 2011].

Zdun, U & Avgeriou, P, 2008. A catalog of architectural primitives for modeling architectural patterns. *Information and Software Technology*, 50(9-10), pp.1003-1034. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0950584907001073.

Zhu, L. & Gorton, I., 2007. UML Profiles for Design Decisions and Non-Functional Requirements. In *Second Workshop on Sharing and Reusing Architectural Knowledge Architecture Rationale and Design Intent SHARK/ADI 2007 ICSE Workshops 2007*. IEEE DL, pp. 8-8. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4273348.

Zimmermann, O. et al., 2007. Reusable Architectural Decision Models for Enterprise Application Development. In S. Overhage et al., eds. *Quality of Software Architecture (QoSA) 2007*. Springer, p. 15. Available at: http://www.springerlink.com/index/e4j1473502644438.pdf.

Zimmermann, O. et al., 2009. Managing architectural decision models with dependency relations, integrity constraints, and production rules. *Journal of Systems and Software*, 82(8), pp.1249-1267. Available at: http://www.sciencedirect.com/science/article/B6V0N-4VJ4WJD-1/2/dbde5f1204552d76cf45454faa85c0f3.