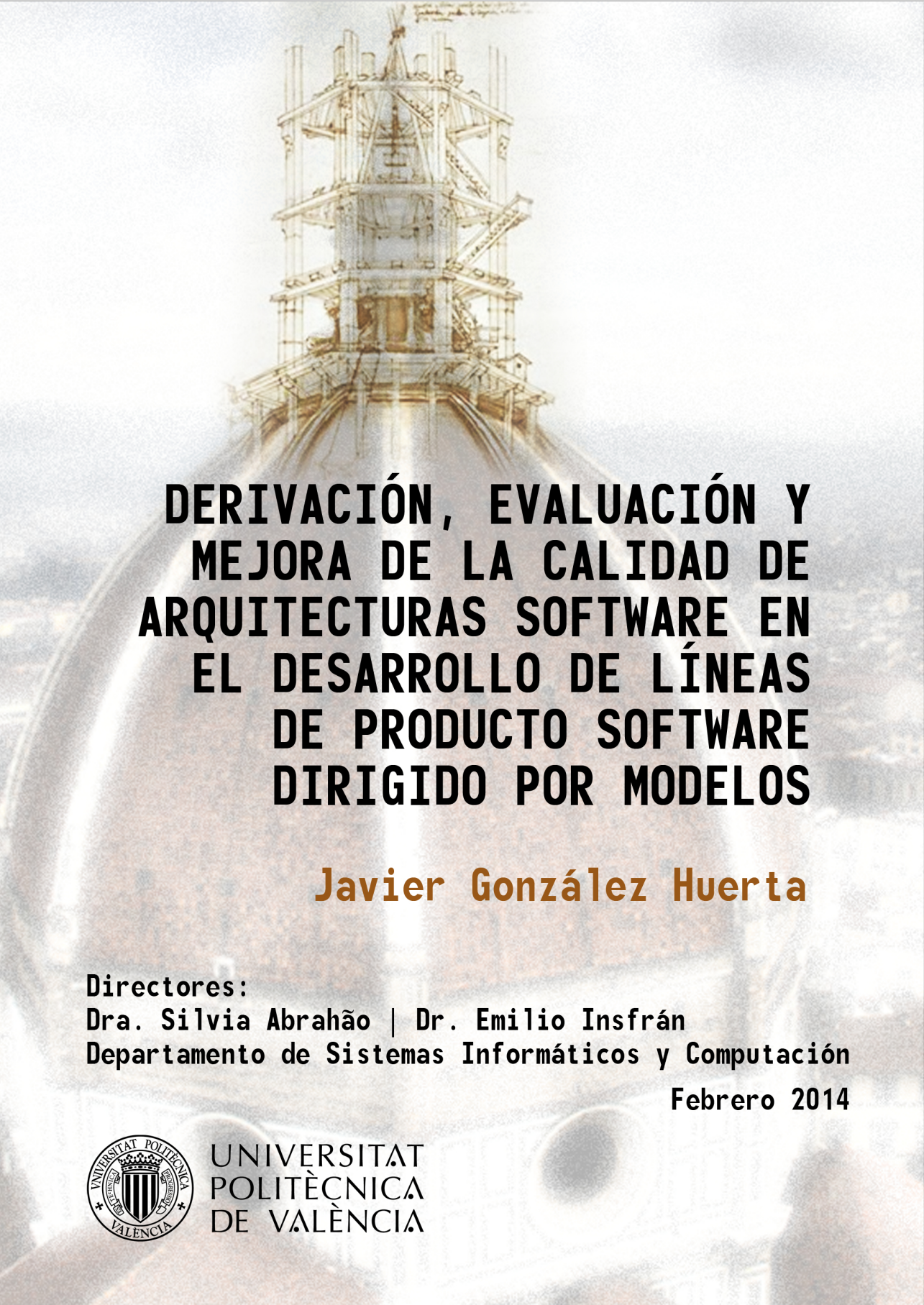


DERIVACIÓN, EVALUACIÓN Y MEJORA DE LA CALIDAD DE ARQUITECTURAS SOFTWARE EN EL DESARROLLO DE LÍNEAS DE PRODUCTO SOFTWARE DIRIGIDO POR MODELOS

Javier González Huerta

Directores:
Dra. Silvia Abrahão | Dr. Emilio Insfrán
Departamento de Sistemas Informáticos y Computación

Febrero 2014

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación

Tesis Doctoral depositada en cumplimiento parcial de los requisitos para el grado de Doctor en Informática

Derivación, Evaluación y Mejora de la Calidad de Arquitecturas Software en el Desarrollo de Líneas de Producto Software Dirigido por Modelos

Javier González Huerta

Directores:

Dra. Silvia Abrahão

Dr. Emilio Insfrán

Valencia, 24 de Febrero de 2014

Tesis Doctoral

Diseño de la Portada: Amparo Desfilis Barceló

Título:	Derivación, Evaluación y Mejora de la Calidad de Arquitecturas Software en el Desarrollo de Líneas de Producto Software Dirigido por Modelos
Presentado por:	Javier González Huerta (jagonzalez@dsic.upv.es)
Directores:	Dra. Silvia Abrahão (sabrahao@dsic.upv.es) Dr. Emilio Insfrán (einsfran@dsic.upv.es)
Institución:	Universitat Politècnica de València (UPV)
Departamento:	Sistemas Informáticos y Computación (DSIC)
Programa de doctorado:	Doctor en Informática
Financiado por:	VALi+d, Conselleria d'Educació, Generalitat Valencia, (Ref. ACIF/2011/235).
Fecha de envío:	Valencia, 16 de Enero de 2014
Defensa:	Valencia, 24 de Febrero de 2014
Revisores externos:	Dra. Ana Moreira, Universidade Nova de Lisboa, Portugal Dr. Jordi Cabot Sagrera, École des Mines de Nantes, Francia Dra. Marcela Genero Bocco, Universidad de Castilla-La Mancha, España
Tribunal:	Dr. Isidro Ramos Salavert, Universitat Politècnica de Valenica, España Dra. Ana Maria Diniz Moreira, Universidade Nova de Lisboa, Portugal Dr. John McGrergor, Clemson University, EEUU

With a little help from my friends.

Agradecimientos

Este es el final de un largo camino lleno de obstáculos, que uno no puede recorrer solo. Por ello quisiera agradecer a todos aquellos que me han ayudado en este tiempo:

A Rosa, por acompañarme y estar siempre a mi lado en todos los caminos que hemos decidido andar en esta media vida juntos. Sin ti no habría sido posible.

A Silvia y Emilio, por su esfuerzo, confianza y apoyo durante todos estos años.

A Isidro por sus siempre sabios consejos y críticas

A Michel por su acogida, sus consejos, su actitud crítica y por haberme enseñado a ver las cosas de otro modo.

A David, por el trabajo llevado a cabo en la implementación de la herramienta que soporta la metodología propuesta en esta tesis doctoral.

A los compañeros del laboratorio, a los que fueron y a los que son (Abel, Adri, Alex, Fernando, José Antonio, Kamil, Patricia, Priscila, Sonia y Vicente) por el ambiente que hace que los momentos duros lo sean menos.

A Giuseppe y Raphael por llevar a cabo las réplicas de los experimentos y estudios del caso que han permitido validar empíricamente la propuesta.

A todos los alumnos que han participado en las sesiones experimentales.

A Claudio, Mari Carmen y Rober por enseñarme a discutir, ser crítico y a perseverar.

A mi padre (en su memoria) y a mi madre por enseñarme que aquello que quieras has de ganártelo trabajando duro.

Y por último a todos aquellos que de un modo u otro han contribuido a hacer posible este proyecto.

Abstract

The production of quality software, on time, and within budget, remains an open problem of Software Engineering that has been addressed from different approaches. An industrial approach to tackle this problem is the adoption of Software Product Lines (SPL). Software product line engineering involves domain engineering (building the product line) and application engineering (deriving products from the product line). One of the most critical tasks during product derivation is meeting the required quality attributes.

Quality assurance is a crucial activity for the success of any software development effort, but is even more important in software product line engineering since a defect in a software asset may impact negatively on the quality of the whole set of products within the product line. This fact is especially relevant when dealing with the software architecture. Software architectures are the means for the attainment of the non-functional requirements of the products that will be derived from the product line, and thus assuring the achievement of those non-functional requirements during the architecture derivation process is a critical activity in the development process.

Despite the huge number of research work dealing with architecture derivation in software product line development, the introduction of quality concerns in this process has not received a proper coverage. The majority of the proposed approaches are only applicable to a specific set of attributes (e.g., performance, reliability, modifiability) or kind of architectures specified using a particular architectural description language (e.g., SysML, AADL). With regard to the architecture evaluation, there is a lack of metric-based architecture evaluation methods defined for software product line that allow the evaluation of product architectures at derivation time.

The objective of this PhD thesis is to define and empirically validate a method, QuaDAI, for the derivation, evaluation, and improvement of product architectures in model-driven software product line development environments. The method comprises a multimodel, which represents the different viewpoints of the software product line, and a process conducted by model transformations that automate the derivation, evaluation, and improvement of product architectures.

This method allows, on the one hand, the introduction of quality concerns in the decision-making processes that occur during the product architecture derivation, whereas makes possible its automation. On the other hand, may help novice architects to be able i) to carry out the evaluation of software architectures and, when required, ii) to apply architectural transformations to improve some product architecture quality attributes.

The QuaDAI method has been empirically validated through a case study conducted in Spain and Brazil as well as through a family of five experiments, with internal and external replications that took place on Spain, Italy and Paraguay. The objective of the case study was to analyze the perceived ease of use, perceived usefulness and intention to use with regard to professionals and software engineering researchers applying the QuaDAI derivation process. The statistical analysis indicated there is a significant positive effect on all the variables under study associated with the use of QuaDAI as a product architecture derivation method. The family of experiments involved 108 subjects including Computer Science and Software Engineering Master students and PhD students. The objective of this study was to compare the effectiveness, efficiency, perceived ease of use, perceived usefulness and intention to use with regard to participants using QuaDAI as opposed to the Architecture Tradeoff Analysis Method (ATAM), a widely applied evaluation method. The statistical analysis and meta-analysis of the data obtained from each experiment indicate that the participants produced their best results when applying QuaDAI, signifying that they obtained architectures with better quality attribute levels faster, and that they found the method easier to use, more useful and more likely to be used in practice. These results make us to consider QuaDAI as a promising approach for the derivation, evaluation and improvement of product architectures in model-driven software product line development.

This PhD thesis contributes to the software product line engineering field by providing an automated, integrated, and generic method for deriving, evaluating and improving software architectures based on quality attributes.

Resumen

La producción de software de calidad, en el tiempo adecuado y con unos costes razonables sigue siendo un problema abierto de la Ingeniería del Software que ha sido abordado desde distintas aproximaciones. Una aproximación industrial al problema, consiste en usar Líneas de Producto Software (LPS). El desarrollo de líneas de producto comprende la ingeniería de dominio (la construcción de la línea de productos) y la ingeniería de aplicación (la derivación de los productos a partir de la línea de productos). Una de las tareas más críticas en la derivación de productos es el cumplimiento de los atributos de calidad requeridos.

En general, el aseguramiento de la calidad del producto es una actividad crucial para el éxito de la industria del software, pero es, si cabe, más importante cuando se trata del desarrollo de líneas de producto software, donde un defecto en un activo software puede impactar negativamente en la calidad de todos los productos de una línea de producto. Este hecho es de especial relevancia cuando tratamos con la arquitectura software, ya que es el artefacto software más relevante para asegurar el cumplimiento de los requisitos no-funcionales de los productos que serán derivados de una línea de productos, por lo que asegurar que estos requisitos se cumplen durante el proceso de derivación de la arquitectura, es una actividad crítica en el proceso de desarrollo.

A pesar de que existen muchos trabajos que tratan el proceso de derivación de arquitecturas en el desarrollo de líneas de producto, el tratamiento de los atributos de calidad de este proceso en la literatura no ha recibido un tratamiento adecuado, y las propuestas presentadas en la mayoría de los casos son aplicables únicamente para un conjunto limitado de atributos de calidad (por ejemplo el rendimiento, la fiabilidad o la modificabilidad) o para un tipo de arquitecturas especificadas, utilizando un lenguaje de descripción de arquitecturas concreto (como SysML o AADL).

Con respecto a los métodos de evaluación, los métodos específicos para líneas de productos que soportan la evaluación de arquitectura de producto, no soportan la evaluación basada en métricas que permita la evaluación de la arquitectura en tiempo de derivación.

El principal objetivo de esta tesis doctoral es la definición y validación empírica del método QuaDAI, un método integrado para la derivación, evaluación y

mejora de arquitecturas de producto software en un entorno de desarrollo de líneas de producto software haciendo uso del paradigma de desarrollo dirigido por modelos. El método se compone de un multimodelo, que representa los diferentes puntos de vista de la línea de productos, y de un proceso dirigido por transformaciones que permite automatizar el proceso de derivación, evaluación y mejora de arquitecturas de producto.

El método permite, por un lado, la introducción de criterios de calidad en los procesos de toma de decisión que tienen lugar durante la derivación de la arquitectura, al tiempo que permite su automatización. Por otro lado, puede ayudar a los arquitectos noveles a ser capaces de: i) llevar a cabo procesos de evaluación de arquitecturas software y, en caso necesario, ii) aplicar transformaciones arquitectónicas para mejorar los atributos de calidad de la arquitectura de producto.

El método propuesto ha sido validado empíricamente mediante un estudio de caso llevado a cabo en España y Brasil y mediante una familia de cinco experimentos, con replicaciones tanto internas como externas en España, Italia y Paraguay. El objetivo del estudio de caso fue el análisis de la facilidad de uso percibida, la utilidad percibida y la intención de uso de los profesionales e investigadores en ingeniería del software que aplicaban QuaDAI como método de derivación de arquitecturas. El análisis estadístico de los resultados indicó que hay un efecto positivo significativo sobre todas las variables estudiadas, asociado al hecho de emplear QuaDAI como método de derivación de arquitecturas de producto. La familia de experimentos ha involucrado a 108 sujetos, estudiantes de grado e ingeniería en informática, másteres en ingeniería del software y de doctorado. El objetivo del estudio fue comparar la efectividad, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso de los participantes aplicando QuaDAI en oposición al *Architecture Tradeoff Analysis Method* (ATAM), un método de evaluación ampliamente difundido. El análisis estadístico y el posterior meta-análisis de los datos obtenidos de cada experimentos indica que los participantes produjeron mejores resultados cuando aplicaban QuaDAI, en el sentido que obtenían arquitecturas con mejores niveles de atributos de calidad de forma más rápida, y percibían el método como más fácil de usar, más útil y veían más probable utilizarlo en el futuro. Estos resultados nos permiten considerar QuaDAI como una aproximación prometedora para la evaluación y mejora de arquitecturas de producto para el desarrollo de líneas de producto, aplicando un enfoque dirigido por modelos.

Esta tesis doctoral contribuye al campo del desarrollo de líneas de producto software proveyendo un método integrado, automatizado y genérico para la derivación, evaluación y mejora de arquitecturas basado en atributos de calidad.

Resum

La producció de programari de qualitat, en el moment oportú i amb costos raonables és encara un problema obert de la Enginyeria del Programari, i que ha estat tractat mitjançant aproximacions diverses. Una aproximació industrial al problema consisteix en emprar Línies de Productes de Programari (LPP). El desenvolupament de línies de producte del programari compren la enginyeria del domini (la construcció de la línia de productes) i la enginyeria de aplicació (la derivació de productes des de la línia de productes). Una de les tasques mes critiques en la derivació de productes es el compliment dels atributs de qualitat.

En general, l'assegurament de la qualitat del producte és una activitat crucial per a l'èxit de la indústria del programari, però és, si cap, més important quan es tracta del desenvolupament de línies de producte, atès que un defecte en un actiu pot impactar negativament en la qualitat de tots els productes de la línia de productes. Aquest fet és d'especial rellevància quan tractem amb l'arquitectura de programari. L'arquitectura de programari és el artefacte mes rellevant per a assegurar el compliment dels requisits no-funcionals dels productes derivats de la línia de productes, per la qual cosa assegurar que estos requisits es compleixen durant el procés de derivació de l'arquitectura és una activitat crítica en el procés de desenvolupament.

A pesar de que existisquen molts treballs que tracten el procés de derivació d'arquitectures en entorns de desenvolupament de línies de producte, el tractament dels atributs de qualitat en aquest procés en la literatura no ha rebut un tractament adequat, i les propostes presentades en la majoria dels casos són específiques per a un conjunt limitat d'atributs de qualitat (per eixample el rendiment, la fiabilitat o la modificabilitat) o per a un tipus d'arquitectures especificades mitjançant un llenguatge de descripció d'arquitectures concret (como pot ser SysML o AADL). Respecte als mètodes d'avaluació, els mètodes específics per a línies de productes que suporten l'avaluació d'arquitectura de producte, no suporten l'avaluació basada en mètriques que permeta l'avaluació de l'arquitectura en temps de derivació.

El principal objectiu d'aquesta tesi doctoral és la definició i validació empírica del mètode QuaDAI, un mètode integrat per a la derivació, avaluació i millora d'arquitectures de producte programari en un entorn de línies de producte

programari, fent ús del paradigma de desenvolupament dirigit per models. El mètode es compon d'un multi-model, que representa els diferents punts de vista de la línia de productes, i d'un procés dirigit per transformacions que permet automatitzar el procés de derivació, avaluació i millora d'arquitectures de producte.

Aquest mètode permet, d'una banda, la introducció de criteris de qualitat en els processos de toma de decisió que tenen lloc al llarg de la derivació de l'arquitectura, al temps que permeten la seua automatització. D'altra banda, pot ajudar als arquitectes novells a ser capaços de i) portar a terme processos d'avaluació d'arquitectures i, en cas necessari, ii) aplicar transformacions arquitectòniques per tal de millorar els atributes de qualitat de l'arquitectura de producte.

El mètode proposat, ha sigut validat empíricament mitjançant d'un cas d'estudi executat a España i a Brasil y mitjançant d'una família de cinc experiments amb replicacions, tant internes com externes a Espanya, Itàlia i al Paraguai. L'objectiu del cas d'estudi va ser l'anàlisi de la facilitat d'us percebuda, la utilitat percebuda i la intenció d'us dels professionals e investigadors de l'enginyeria del programari que aplicaven QuaDAI com a mètode de derivació d'arquitectures de producte. L'anàlisi estadístic dels resultats va indicar que hi ha un efecte positiu i significatiu sobre totes les variables estudiades associat a l'ús de QuaDAI com a mètode de derivació. La família de experiments va involucrar a 108 subjectes, estudiants de grau i enginyeria en informàtica, màsters en enginyeria del programari i de doctorat. L'objectiu de l'estudi va ser la comparació de la efectivitat, la eficiència, la facilitat d'us percebuda, la utilitat percebuda i la intenció d'us dels participants aplicant QuaDAI en oposició al *Architecture Tradeoff Analysis Method* (ATAM), un mètode d'avaluació àmpliament difós. L'anàlisi estadístic i el posterior meta-anàlisi de les dades obtingudes a cada experiment indica que els participants produïren els seus millors resultats quan aplicaven QuaDAI, en el sentit que obtenien arquitectures amb millors nivells d'atributs de qualitat mes ràpidament, y que percebien el mètode com mes fàcil d'usar, mes útil i veien com a mes probable el seu us en el futur.. Aquest fet fa que es puga considerar QuaDAI com una aproximació prometedora per a l'avaluació i millora d'arquitectures de producte aplicant un l'aproximació de desenvolupament dirigit per models.

Aquesta tesí contribueix al camp del desenvolupament de les línies de producte del programari proveint un mètode integrat, automatitzat i genèric per a la derivació, avaluació i millora d'arquitectures basat en atributs de qualitat.

Palabras Clave

Key Words: Software Architectures, Software Product Lines, Model-Driven Software Development, Software Quality, Empirical Evaluation.

Palabras Clave: Arquitecturas Software, Líneas de Producto, Desarrollo de Software Dirigido por Modelos, Calidad del Software, Validación Empírica.

Paraules Clau: Arquitectura de Programari, Línies de Productes de Programari, Desenvolupament del Programari Dirigit pels Models, Qualitat del Programari, Validació Empírica.

Contenido

Índice de Figuras

Índice de Tablas

Índice de listados

Acrónimos

AADL	Architecture Analysis and Design Language
AUTOSAR	AUTomotive Open System ARchitecture
CVL	Common Variability Languaje
DSDM	Desarrollo de Software Dirigido por Modelos
MDA	Model Driven Architecture
MOF	Meta Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Independent Model
QVT	Query View Transformation
RNF	Requisito No-funcional
SysML	Systems Modeling Language
TOGAF	The Open Group Architecture Framework
VSpec	Variability Especification
SPEM	Software & Systems Process Engineering Metamodel
SUM	Single-Underlying Model
UML	Unified Modeling Languaje
XMI	XML Metadata Intechange

Capítulo 1

Introducción

En este capítulo, en primer lugar, se contextualiza el trabajo de investigación desarrollado en esta tesis. En segundo lugar, se plantea el problema a resolver, se establecen los objetivos, se recogen las contribuciones de la misma y por último, se describe la metodología de investigación.

La sección 1.1 introduce el papel de las arquitecturas software en el desarrollo de líneas de producto software.

La sección 1.2 introduce los aspectos relacionados con la derivación, evaluación y mejora de la calidad de arquitecturas software en el desarrollo de líneas de producto software.

La sección 1.3 plantea el problema que se aborda en la presente tesis doctoral: la definición y evaluación empírica de un método integral y genérico para la derivación, evaluación y mejora de arquitecturas de producto.

La sección 1.4 describe los objetivos a desarrollar, así como las hipótesis que se pretenden contrastar.

La sección 1.5 describe las distintas contribuciones como resultado de la consecución de los distintos objetivos de investigación.

La sección 1.6 describe el contexto en el que se ha desarrollado los trabajos de investigación que constituyen esta tesis doctoral.

La sección 1.7 describe la metodología de investigación aplicada y las técnicas previstas para validar empíricamente la aplicabilidad del método propuesto.

Por último, la sección 1.8 describe la estructura del documento.

1.1 Las arquitecturas software en el desarrollo de líneas de producto software

El desarrollo de líneas de producto software es una aproximación industrial para el desarrollo de sistemas software altamente complejos que está extendiéndose de manera notable en los últimos años como medio para mejorar la productividad y la calidad del producto software desarrollado. Esta mejora se basa en sacar provecho de la reutilización masiva y sistemática de *activos software*[1] para producir sistemas software que comparten un conjunto común de características.

El desarrollo de líneas de producto software, se divide en dos fases o procesos de desarrollo. En una primera fase se establecen las partes comunes y variables de los productos que la conforman y se desarrollan los activos software como partes de los productos software a desarrollar. En una segunda fase, los activos software son reutilizados sistemáticamente en la derivación de productos específicos. En este escenario la *arquitectura software* juega un papel diferencial.

Las arquitecturas software son el precursor y el medio para alcanzar los requisitos de calidad del producto software en desarrollo. Una de las definiciones más comúnmente aceptadas de arquitectura software es la propuesta por Bass et al. (1998):

> *"La arquitectura de un programa o sistema computacional es la estructura o estructuras del sistema, que comprende los componentes software, las propiedades de estos componentes visibles desde el exterior y las relaciones entre ellos".*

De esta definición se desprende que la arquitectura software debe abstraer la información acerca del sistema que nos permita razonar sobre el mismo, ser base para el análisis y la toma de decisiones acerca de dicho sistema.

[1] Activo software son aquellos activos que forman la base de la línea de productos. Los activos software incluyen, pero no están limitados a, la arquitectura, los componentes reusables, los modelos de dominio, las especificaciones de requisitos, la documentación, las planificaciones, presupuestos, planes de prueba, etc. (Clements y Northrop 2001).

En el desarrollo de líneas de producto software se presentan dos arquitecturas software que juegan dos roles diferenciados:

1. La *arquitectura de la línea de productos* que da soporte a todos los posibles productos que pueden ser obtenidos a partir de la línea de productos y para ello cuenta con una serie de mecanismos de variabilidad que le permite adaptarse a los requisitos de dichos productos.
2. La *arquitectura de producto* que se deriva a partir de la arquitectura de la línea de productos resolviendo la variabilidad existente en la arquitectura de la línea de productos, mediante los mecanismos de variabilidad previstos, con el fin de cumplir los requisitos del producto en desarrollo.

1.2 Derivación, evaluación y mejora de la calidad de arquitecturas software en el desarrollo de líneas de producto

En general, el aseguramiento de la calidad del producto es una actividad crucial para el éxito de la industria del software, pero lo es más, si cabe, cuando se trata del desarrollo de líneas de producto software, dado que la reutilización masiva de activos software hace que los *atributos de calidad*[2] de los activos software impacten en la calidad de todos los productos en la línea de producto. Este hecho es de especial relevancia cuando tratamos con la arquitectura software, que es el activo software más crítico en el desarrollo de líneas de producto. Asegurar el cumplimiento de los requisitos no-funcionales durante la derivación de la misma a partir de la arquitectura de la línea de producto, va a ser una actividad crucial en el proceso de desarrollo.

Podemos encontrar distintas aproximaciones que de una manera automatizada nos permitan, a partir de los requisitos del producto a desarrollar, derivar la arquitectura de dicho producto a partir de la arquitectura de la línea de productos (Perovich y Rossel y et al. 2009), (Cabello et al. 2009), (Duran-Limon et al. 2011). En estas aproximaciones el interés se centra en obtener una arquitectura a partir de diferentes artefactos que expresan los requisitos, funcionales y no-funcionales, que el producto ha de cumplir.

[2] *Atributo de Calidad:* propiedades físicas o abstractas de un artefacto software (ISO 2005).

La evaluación de la arquitectura software se define como (Dolan 2001):

> *"El examen sistemático de la extensión con la que una arquitectura software cumple con los requisitos".*

La evaluación de arquitecturas software es un proceso que pretende, por lo tanto, detectar si la arquitectura nos permitirá alcanzar los requisitos de calidad además de conocer los riesgos y puntos sensibles. La salida es una lista priorizada de los requisitos de calidad en relación a los que la arquitectura ha sido evaluada, una correspondencia entre las distintas aproximaciones arquitectónicas y los requisitos de calidad, y por último, los riesgos y no-riesgos de la arquitectura (Clements et al. 2002).

Las evaluaciones de la arquitectura software pueden ser llevadas a cabo en distintas fases del desarrollo de software. Estas evaluaciones servirán para comparar e identificar puntos fuertes y debilidades de las alternativas de diseño en fases tempranas del ciclo de desarrollo (evaluación temprana), o para identificar problemas una vez la arquitectura y el sistema han sido implementados por completo (evaluación tardía).

Desde que se acuñó el término arquitectura software a mediados de los años 90, se han definido multitud de métodos ampliamente difundidos para la evaluación de arquitecturas para el desarrollo convencional de sistemas tales como SAAM (Kazman et al. 1994), ATAM (Kazman et al. 2000), ALMA (Bengtsson et al. 2004) o PASA (Williams y Smith 2002).

SAAM (Software Architecture Analysis Method) es un método cuya finalidad es la evaluación de la arquitectura software en relación a una serie de atributos de calidad de interés. Este método también es capaz de comparar distintas arquitecturas software con respecto a un conjunto dado de propiedades. Fue inicialmente definido para analizar la modificabilidad, aunque posteriormente ha sido empleado con múltiples propósitos.

ATAM (Architecture TradeOff Analysis Method) es el método de referencia cuando se habla de evaluación de arquitecturas software. Es un método que tiene como propósito evaluar las consecuencias de las decisiones arquitectónicas en relación a determinados atributos de calidad. La principal meta de ATAM es promover un método estructurado de análisis de las capacidades de la arquitectura software en relación a múltiples atributos de calidad.

Hay otros métodos enfocados a la evaluación de atributos concretos como ALMA (Architecture Level Modificability Analysis)(Bengtsson et al. 2004), que es un método de evaluación de arquitecturas dirigido por metas o PASA (Performance Assessment of Software Architecture)(Williams y Smith 2002), cuyo propósito es la evaluación de la arquitectura con respecto a una serie de objetivos en cuanto a prestaciones, mediante el análisis de estilos arquitectónicos orientados a la mejora de prestaciones.

Con respecto a los métodos de evaluación definidos para líneas de producto y de acuerdo con la clasificación definida en (Etxeberria y Sagardui 2005), podemos distinguir varios tipos de evaluaciones arquitectónicas, dependiendo de: el objeto de la evaluación, su propósito y el instante del ciclo de vida en que tienen lugar. Así, podemos distinguir la evaluación de la arquitectura de la línea de productos o la evaluación de la arquitectura de producto si atendemos al objeto de le evaluación. Dentro de la evaluación de la arquitectura de la línea de producto, podemos distinguir la evaluación en tiempo de diseño con propuestas como FAAM (Familiy Architecture Assessment Method) (Dolan 2001) o AQA (Architecture Quality Analysis) (Matinlassi et al. 2002), evaluación de arquitecturas existentes (Gannod y Lutz 2000) y dentro de estas últimas la evaluación de la arquitectura de la línea de productos en relación a la evolución de la misma para albergar nuevos requisitos como las propuestas de Maccari (2002) o (Riva y Rosso 2003). Si atendemos a la evaluación de arquitecturas de producto, de nuevo, podemos hablar de evaluación de arquitecturas de producto en relación con la evolución o de métodos de evaluación de la arquitectura de producto en tiempo de derivación como el trabajo de Alonso et al. (1998) en el que se analizan las propiedades temporales de la arquitectura.

Tras la evaluación de la arquitectura, si uno o más requisitos no se han satisfecho tras la resolución de la variabilidad arquitectónica, la arquitectura deberá ser modificada para cumplir con dichos requisitos mediante la aplicación de transformaciones arquitectónicas (Bosch 2000). Estas transformaciones arquitectónicas tienen como objetivo mejorar ciertas propiedades de la arquitectura a través de la aplicación de estilos arquitectónicos (Shaw y Garlan 1996), patrones arquitectónicos (Buschmann et al. 1996), o patrones de diseño (Gamma et al. 1995).

1.3 Planteamiento del problema

A pesar de la importancia de los requisitos no-funcionales en el éxito de los sistemas software, y de la complejidad del desarrollo de líneas de producto software, éstos no son tratados adecuadamente durante todo el ciclo de vida de las líneas de producto software. La calidad software debe ser soportada desde las fases tempranas e incorporada al diseño, ya que de otro modo no podrá ser incorporada posteriormente (Bass et al. 1998). Muchos de los esfuerzos se centran en obtener una configuración óptima del producto, pero muy pocas propuestas verifican *a posteriori* que los requisitos no-funcionales se satisfacen. Uno de los desafíos a afrontar en la ingeniería de líneas de producto software es la falta de técnicas y métodos para asegurar la calidad los productos que se derivan de una línea de productos software (Etxeberria 2008; Montagud y Abrahão 2009; Montagud et al. 2011).

En las propuestas para el desarrollo de líneas de producto, en algunos casos, va ligada a la aplicación del paradigma de desarrollo dirigido por modelos. El Desarrollo de Software Dirigido por Modelos (DSDM) promueve el uso de modelos a lo largo de todo el proceso de desarrollo de software, elevando el nivel de abstracción y permitiendo que estos modelos puedan ser transformados sucesivamente hasta la obtención del producto final. En el ámbito de las líneas de producto, estos modelos se centran, en la mayoría de los casos, en la representación de la funcionalidad y de la variabilidad de la línea de productos, que por sí solos no definen la extensión de una línea de productos. La premisa de que es posible definir una línea de productos con un número limitado de variaciones perfectamente definidas, que podemos plasmar en un modelo de características y que además, la influencia de la selección de una u otra variante no va a afectar a los requisitos no-funcionales, es cuanto menos limitada. La realidad no siempre se va a ajustar a esa premisa. La selección de una u otra variante va a afectar a la estructura, al comportamiento y a las propiedades no-funcionales del producto. Si la configuración del producto se lleva a cabo únicamente teniendo en cuenta criterios funcionales, el producto obtenido no tiene por qué ajustarse a los requisitos no-funcionales. Además la resolución de la variabilidad, en la mayoría de los casos, no se va a traducir en añadir o quitar monotónicamente gránulos de funcionalidad de la arquitectura del producto, sino que, en la mayoría de casos, implica cambios más profundos en la arquitectura que además de afectar a la estructura y al comportamiento, van a tener un impacto en la calidad final del producto e incluso pueden hacerlo inviable.

Esto nos lleva a la parametrización del proceso de derivación evaluación y mejora de la arquitectura software del producto mediante un Multimodelo que siendo capaz de captar las diferentes vistas de la línea productos y las relaciones entre elementos de dichas vistas, nos permite introducir criterios-no-funcionales en dicho proceso.

La inclusión de requisitos no-funcionales en la derivación de la arquitectura, por todo lo mencionado anteriormente, va a permitirnos mejorar el proceso de obtención de arquitecturas software que cumplan con los requisitos tanto funcionales como no-funcionales. Ahora bien, una vez obtenidos los artefactos que representan la arquitectura del producto, será necesaria la evaluación de los mismos para comprobar el cumplimiento de dichos requisitos no-funcionales. Cuando hablamos de requisitos no-funcionales, la adición de variantes que tienen efecto en un determinado atributo de calidad no responde a una función monotónica y no siempre las propiedades no-funcionales del producto se pueden expresar como la suma de las propiedades no-funcionales de sus componentes (Crnkovic et al. 2004). Es por todo ello que, en ocasiones, la variabilidad prevista en la arquitectura de la línea de producto, puede no permitirnos alcanzar los requisitos no-funcionales del producto en desarrollo, en cuyo caso deberemos proveer mecanismos de transformación de la arquitectura para tratar de alcanzar dichos requisitos no-funcionales.

1.4 Objetivos e hipótesis

El principal objetivo de esta tesis doctoral es la definición y validación empírica de un método para la derivación, evaluación y mejora de arquitecturas de producto software en un entorno de líneas de producto software haciendo uso del paradigma de desarrollo dirigido por modelos.

Dicho objetivo general se descompone en las siguientes metas:

1. Analizar en profundidad los métodos de derivación de arquitecturas de producto para líneas de producto software, haciendo especial énfasis en aquellos que siguen el paradigma de desarrollo dirigido por modelos.
2. Analizar los métodos de evaluación y mejora de arquitecturas software específicamente definidos para ser aplicados en entornos de desarrollo de líneas de producto software.
3. Definir una estructura de modelos y relaciones entre elementos de dichos modelos (multimodelo) que permita capturar las distintas vistas

de interés del sistema (variabilidad, arquitectónica, calidad, etc.) de una forma integral e interrelacionada.

4. Definir un proceso, conducido por transformaciones de modelos, que haciendo uso del multimodelo dé cobertura a la derivación de la arquitectura de producto a partir de la arquitectura de la línea de producto, ejerciendo sus mecanismos de variabilidad arquitectónica y a la evaluación y mejora de la arquitectura obtenida.
5. Validar empíricamente el método propuesto mediante estudios de caso que permitan validar la propuesta en casos complejos y mediante experimentos controlados que comparen la propuesta con métodos análogos empleados en la industria. Esta validación nos va a permitir, por un lado, analizar la aplicabilidad del método y por otro recibir retroalimentación para mejorar la propuesta.

En lo referente a las hipótesis de investigación, éstas son las siguientes:

- La aplicación del método permite reducir el esfuerzo de derivación y evaluación de la arquitectura y, en aquellos casos en que los mecanismos de variabilidad arquitectónica no nos permitan alcanzar alguno de los requisitos de calidad, facilita la mejora de la arquitectura mediante la aplicación de transformaciones arquitectónicas.

- La representación explícita de las relaciones entre las vistas de variabilidad, arquitectónica y de transformaciones con la vista de calidad, y el seguimiento del proceso definido en el método permite mejorar la calidad de las arquitecturas de producto obtenidas.

- Los métodos de evaluación de arquitecturas existentes (SAAM, ATAM, etc.) son en su mayoría subjetivos y requieren de un tiempo razonable para su correcta aplicación. Por este motivo, la representación explícita de las relaciones de impacto entre las vistas de variabilidad, arquitectónica, de transformaciones y de calidad permite por un lado representar los criterios utilizados a la hora de tomar decisiones de diseño y por otro ayudar a los arquitectos software noveles a tomar decisiones durante la evaluación de una arquitectura, ya que hace explícito el conocimiento de los arquitectos software sobre cómo los atributos de calidad impactan en el diseño de la arquitectura.

1.5 Contribuciones

En el contexto de lo expuesto en la sección anterior, la principal contribución de esta tesis va enfocada al siguiente problema:

- Obtención de arquitecturas de producto a partir de una arquitectura de la línea de productos que cumplen con los requisitos no-funcionales del producto en desarrollo.

  Problema: Ausencia de métodos que soporten de forma completa, sistemática y automatizada, la obtención de arquitecturas de producto software que cumplan una serie de requisitos no-funcionales.

  Contribución: Se propone QuaDAI, un método de derivación, evaluación y mejora de la calidad de arquitecturas software en el desarrollo de líneas de producto dirigido por modelos que permite mejorar la eficiencia y efectividad en la obtención de arquitecturas que cumplen los requisitos no-funcionales.

A lo largo de la definición del método también se han aportado las siguientes contribuciones:

- Estudio de las técnicas de representación de la variabilidad arquitectónica.

  Problema: La variabilidad de una línea de productos representa la variabilidad en el espacio del problema, pero la arquitectura requiere de mecanismos y notaciones más potentes que permitan representar la variabilidad arquitectónica en el espacio de la solución.

  Contribución: Se ha realizado un análisis de las diferentes técnicas de representación de variabilidad arquitectónica, y se ha adoptado una representación de la variabilidad arquitectónica basada en el estándar *Common Variability Language* (CVL) (Object Management Group 2012a) que nos permite expresar la variabilidad arquitectónica en las diferentes vistas arquitectónicas y relacionar esta variabilidad en el espacio de la solución con la variabilidad de la línea de productos en el espacio del problema.

- Definición de un artefacto que permite el modelado de líneas de producto software desde distintos puntos de vista complementarios e interrelacionados incluyendo de forma explícita la calidad.

  Problema: La descripción de una línea de productos software no puede verse limitada a la mera descripción de la funcionalidad y su variabilidad.

  Contribución: Un multimodelo que permite la definición de distintas vistas (en el caso de las líneas de producto la vista de variabilidad, arquitectónica, de transformaciones y de calidad) que permite representar los distintos tipos de relaciones entre elementos de las distintas vistas y el impacto entre ellos (por ejemplo cómo la aplicación de una determinada transformación mejora un determinado requisito no-funcional o cómo la selección de una característica impacta en un determinado requisito no-funcional).

- Definición de una infraestructura software que soporte el modelado de líneas de producto empleando múltiples vistas.

  Problema: En general el modelado de líneas de producto es complejo debido a la cantidad de lenguajes de modelado necesarios para capturar las distintas vistas y los aspectos de variabilidad en dichas vistas.

  Contribución: Definición de una infraestructura software que permite extender lenguajes de modelado existentes, para establecer relaciones entre ellos sin que la definición de los distintos lenguajes de modelado se vea afectada. De esta forma se proporciona una visión uniforme de cada una de las partes y una visión del todo a través de las relaciones entre ellas.

- Validación empírica del método propuesto.

  Problema: En el campo de la Ingeniería del Software en general y, en el campo de las arquitecturas software en particular, hay una gran carencia de estudios empíricos que demuestren la efectividad de los distintos métodos para la derivación y/o evaluación de arquitecturas. Los investigadores en el campo de las arquitecturas software deben por un lado desarrollar nuevos métodos, técnicas o herramientas que mejoren las prácticas actuales y por otro, llevar a cabo validaciones rigurosas y sistemáticas de los métodos

propuestos y de los existentes, siguiendo el paradigma empírico (Falessi et al. 2011).

Contribución: Validación empírica de la fase de evaluación y mejora de arquitecturas software de QuaDAI, a través de una familia de experimentos y de la fase de derivación de QuaDAI a través de un estudio de caso.

1.6 Contexto de la investigación

Esta tesis doctoral se ha desarrollado en el contexto del grupo de investigación de Ingeniería del Software y Sistemas de Información (ISSI) del Departamento de Sistemas Informáticos y Computación de la Universitat Politècnica de València (UPV).

Los trabajos que han hecho posible el desarrollo de esta tesis, se engloban en proyectos de I+D financiados con fondos públicos. Los proyectos son los siguientes:

- Proyecto CALIMO: Integración de la Calidad en el Desarrollo de Software Dirigido por Modelos. Financiado por la Generalitat Valenciana, Conselleria D'Educació – GV/2009/103. De Enero de 2009 a Enero de 2010.
- Transformación de Modelos Dirigida por Atributos de Calidad. Financiado por la Universitat Politècnica de València – PAID-06-07-3286. De Diciembre de 2007 a Diciembre de 2009.
- Proyecto MULTIPLE: Multimodeling Approach for Quality-Aware Software Product Lines. Financiado por el Ministerio de Ciencia e Innovación, Gobierno de España – TIN2009-13838. De Octubre de 2009 a Septiembre de 2013.
- Proyecto TwinTIDE: Towards the Integration of Transectorial IT Design and Evaluation. Financiado por la acción COST de la Unión Europea IC0904. De Noviembre de 2009 a Noviembre de 2013.

1.7 Método de investigación

Las actividades de investigación que han dado lugar a esta tesis doctoral, están estructuradas siguiendo una extensión del modelo para la transferencia de

tecnología propuesto por Gorschek et al. (2006) basado en las necesidades de la industria incluyendo actividades de evaluación y de observación.

Este modelo de investigación y transferencia de tecnología se basa en ocho actividades relacionadas (ver Figura 1.1) donde la búsqueda de soluciones adecuadas se realiza de forma iterativa por medio de la formulación de soluciones candidatas y la correspondiente validación empírica que permite dirigir los esfuerzos hacia una solución realista. Las actividades a realizar se describen a continuación:

1. Identificación del problema: Se pretende entender el problema que el socio industrial pretende resolver. Este paso es llevado a cabo a través de reuniones y workshops donde los expertos de la compañía hacen presentaciones acerca de los retos que ellos perciben.
2. Formulación del problema: Una vez que el problema está identificado, éste es formulado de manera más precisa y los factores contextuales y las asunciones de trabajo, son especificadas claramente.
3. Revisión del estado del arte: Se realiza una revisión crítica de la literatura así como de las soluciones tecnológicas comerciales y *open source* disponibles para identificar hasta qué punto las metas han sido abordadas y cuáles son los problemas abiertos a resolver con la investigación a desarrollar.
4. Solución candidata: Se idean una o más soluciones potenciales. Dichas soluciones serán más tarde depuradas a través de las distintas etapas de refinamiento (etapas 6 y 7).
5. Entrenamiento: Se trata de una etapa incremental. En las primeras fases, el entrenamiento se enfoca a crear el conocimiento necesario para que los profesionales del área puedan formarse una opinión de la aplicabilidad de la propuesta para determinar si los constructos son naturales y sencillos de construir. En las fases tardías, el entrenamiento sirve para crear guías y pasos metodológicos detallados para aplicar las soluciones.
6. Validación inicial: Se lleva a cabo una evaluación preliminar de las soluciones, en un entorno de laboratorio o en una configuración industrial limitada. En el caso de entornos de laboratorio se llevarán a cabo estudios controlados (por ejemplo experimentos controlados y estudios de caso con alumnos o profesionales). Para las

configuraciones industriales se llevarán a cabo una mezcla de seminarios, talleres prácticos y encuestas.

7. Validación realista: Se llevan a cabo estudios de caso en configuraciones industriales, empezando por estudios piloto para después extenderse en usos más amplios. En esta fase se definirán guías prácticas y se desarrollarán las herramientas que soporten la propuesta.
8. Lanzamiento de la solución: En este último paso se valoran los resultados obtenidos y las herramientas y el material de entrenamiento son preparados para su uso en contextos industriales.

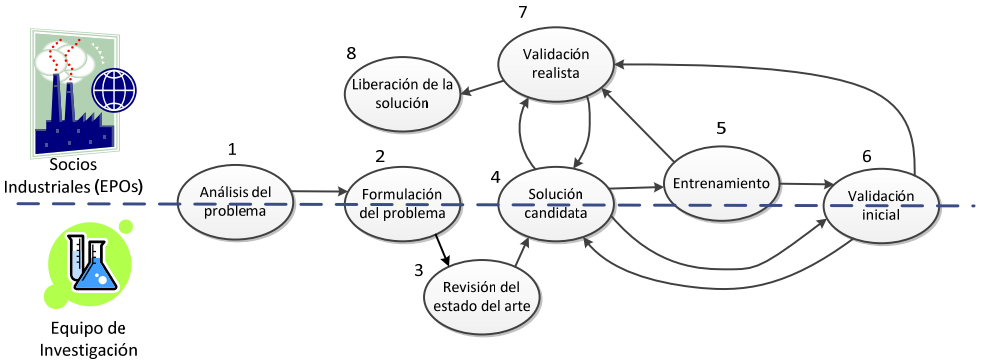


Figura 1.1 Modelo de transferencia tecnológica

En las primeras fases de la tesis doctoral, y fruto de la colaboración de Rolls Royce en el proyecto MULTIPLE, se identificó el problema de la derivación de las arquitecturas software en entornos de desarrollo de líneas de producto software. Rolls Royce, en su división de motores para la industria aeronáutica, lleva desde 2008 aplicando la aproximación de líneas de producto software en el desarrollo de controladores electrónicos de motores para una gama de turbinas de medio y gran tamaño. La variabilidad de su línea de productos consta de más de 1.200 características. La resolución de esa variabilidad y su correspondencia en la arquitectura, fue el inicio de este trabajo de investigación. Las fases iniciales se desarrollaron en dos talleres conjuntos con profesionales de la firma. En este trabajo de investigación nos hemos centrado en cubrir las seis primeras fases del proceso, incluyendo la validación inicial llevada a cabo mediante una familia de experimentos controlados para la fase de evaluación y mejora, y un estudio de caso sobre el dominio automovilístico para la fase de derivación del método. Es cierto que algunos de los sujetos de

dichos estudios eran profesionales, pero al haberse realizado en contextos académicos no pueden ser considerados como validación realista.

Por motivos de confidencialidad, los ejemplos de esta tesis, así como los estudios de caso, hacen referencia a la industria automovilística, aunque el problema inicial y la solución aportada son de aplicación tanto para la industria automovilística como para la aeronáutica.

La validación empírica es un pilar necesario en la construcción de cualquier cuerpo de conocimiento en el campo de las ingenierías. Hay una gran carencia de estudios validados empíricamente en el campo de la ingeniería del software (Basili et al. 1986; Zelkowitz y Wallace 1998; Kitchenham et al. 2002), y especialmente en el campo de las arquitecturas software como demuestran recientes estudios (Falessi et al. 2009a; Ali Babar et al. 2011). En los últimos años se han presentado diferentes guías de diseño y frameworks para la elaboración de estudios empíricos en el ámbito de la ingeniería del software, con el fin de ayudar a los investigadores a planificar, llevar a cabo y analizar estudios empíricos (Wohlin et al. 2000; Kitchenham et al. 2002; Runeson y Höst 2008; Wohlin et al. 2012). Las siguientes subsecciones ilustran tanto el proceso experimental como los estudios de caso que sirven como validación de las distintas fases del método propuesto.

1.7.1 Experimentos

La experimentación es una fase crucial de la validación y puede ayudar a determinar si los métodos utilizados se ajustan a una teoría particular. Los experimentos controlados son apropiados para investigar diferentes aspectos, como la confirmación o prueba de teorías existentes, la evaluación de la precisión de modelos, validación de medidas, etc. Los experimentos que aparecen en esta tesis han sido diseñados siguiendo el framework para la experimentación en Ingeniería del Software (Wohlin et al. 2000; Wohlin et al. 2012).

La Figura 1.2 muestra las principales actividades del proceso experimental sugerido por Wohlin et al. (2012) y los artefactos generados en cada una de las actividades.

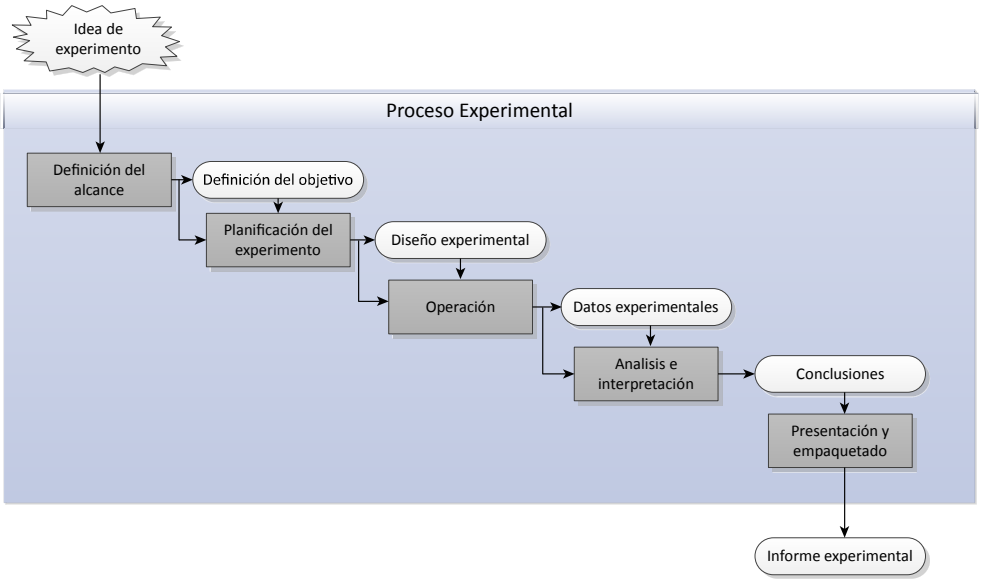


Figura 1.2 Proceso experimental con los artefactos generados es las distintas actividades

La primera actividad es la *definición del alcance*, aunque todavía no estén definidas formalmente, si deberán estar claras las hipótesis del experimento, así como el objetivo y las metas. La meta del experimento es formulada a partir del problema a resolver siguiendo el framework *Goal/Question/Metric* (GQM) (Basili et al. 1988) con el esquema:

Analizar	<Objeto de Estudio: ¿qué se analiza?>
Con el propósito	<Propósito: ¿cuál es la intención?>
Con respecto a su	<Enfoque de Calidad ¿cuál es el efecto estudiado?>
Desde el punto de vista	<Perspectiva: ¿qué vista?>
En el contexto de	<Contexto: ¿dónde tiene lugar el estudio, sobre qué artefactos y con qué tipo de sujetos?>

La siguiente actividad, es la *planificación del experimento*, donde se decide el diseño. Es en esta actividad, donde se define en profundidad el contexto, que incluye tanto la componente personal (que perfiles tendrán los sujetos) como el entorno en el que tendrá lugar. También se especifican formalmente las hipótesis experimentales, incluyendo las hipótesis nulas e hipótesis alternativas, así como las variables, tanto las independientes (entradas) como las variables dependientes (salidas). Además, se seleccionará un diseño experimental y se identificara y preparara la instrumentación a utilizar. El diseño experimental

describe, por ejemplo, como se llevaran a cabo los ensayos (online vs offline) o la aleatorización de los sujetos. Por último, en la fase de planificación, es donde se ha de considerar la cuestión de la validez de los resultados que cabe esperar.

Durante la *operación* del experimento es donde se van a preparar, ejecutar y validar los datos recogidos. Durante la ejecución se debe asegurar que el experimento se lleva a cabo siguiendo el diseño definido en la fase de planificación y se recogen los datos experimentales. Por último, se validan los datos recogidos para asegurar que son correctos y proveen una imagen real del experimento.

En el *análisis e interpretación* se emplean los datos recogidos y validados en la fase de operación. Se emplean estadísticos descriptivos con el fin de entender los datos de manera informal. El siguiente paso es analizar si el conjunto de datos a considerar debe ser reducido, bien eliminando puntos o bien eliminando variables, tras analizar si hay variables redundantes que nos ofrecen la misma información. Una vez reducido el conjunto de datos, se llevaran a cabo las pruebas de las hipótesis. Se seleccionarán las pruebas en función del tipo de escala, variables de entrada y tipo de resultados que se buscan. La interpretación se centra en determinar si, en base a las pruebas, se pueden aceptar o rechazar las distintas hipótesis, esto es, determinar la influencia de las variables independientes sobre las variables dependientes en el caso en que se rechacen las hipótesis nulas.

Por último, la actividad de *presentación y empaquetado* está relacionada con la preparación de la documentación, ya sea un artículo de investigación para difundir los resultados o un paquete de laboratorio con el fin de llevar a cabo replicaciones del experimento.

1.7.2 Estudios de caso

A diferencia de los experimentos controlados, el estudio de caso es una indagación empírica que investiga un fenómeno contemporáneo en su contexto, especialmente cuando la frontera entre el fenómeno y su contexto es difusa (Benbasat et al. 1987; Robson 2002; Yin 2009). La metodología de estudio de caso está especialmente recomendada para la investigación en ingeniería del software, puesto que los objetos de investigación son fenómenos contemporáneos que, en general, son difíciles de estudiar de manera aislada (Runeson y Höst 2008; Wohlin et al. 2012). La experimentación en el campo de la ingeniería del software ha mostrado a lo largo de los años que hay múltiples factores que impactan en el resultado de una actividad de ingeniería

del software cuando se trata, por ejemplo, de replicar un estudio. Los estudios de caso no proveen el mismo tipo de resultados, por ejemplo, acerca de relaciones causales tal como hacen los experimentos controlados, pero pueden ser muy útiles para entender un determinado fenómeno en su contexto real. En el ámbito del modelo de transferencia de tecnología, el estudio de caso puede verse como una actividad puramente observacional, enfocada a la validación del objeto de la investigación que conlleva un cambio en el proceso (Wohlin et al. 2012).

Cuando se lleva a cabo un estudio de caso, hay cinco pasos principales (Runeson y Höst 2008; Wohlin et al. 2012):

1. Diseño del estudio de caso: al llevar a cabo un estudio de caso hay diferentes problemas que deben ser planificados, como el método usado para la recolección de datos, que departamentos de la organización van a ser analizados, documentos a analizar, personas a entrevistar etc.

   El plan para un estudio de caso, debería al menos contener los siguientes elementos:

   - Objetivo: qué meta se pretende alcanzar.
   - El caso: qué se pretende estudiar.
   - Teoría: marco de referencia.
   - Métodos: cómo recopilar la información.
   - Estrategia de selección: donde buscar dicha información.

2. Preparación para la recogida de datos: en un estudio de caso se puede procesar información proveniente de distintas fuentes.

   Las técnicas de recogida de los datos pueden ser clasificadas en tres niveles:

   - Primer nivel: el investigador está en contacto con los sujetos y recoge la información en tiempo real.
   - Segundo nivel: el investigador recoge la información en bruto sin interactuar con los sujetos durante la recolección.
   - Tercer nivel: el investigador analiza de manera independiente los artefactos de trabajo donde en ocasiones se emplea información agregada.

3. Recogida de los datos: la recolección de los datos puede llevarse a cabo a través de entrevistas, observación, análisis de datos históricos o métricas.
4. Análisis de los datos: en el análisis de datos hay que distinguir entre el análisis cuantitativo y análisis cualitativo. Para el análisis cuantitativo se aplican estadísticos descriptivos para entender los datos de manera informal y se emplean análisis de correlación o modelos predictivos para describir como una medición de las últimas actividades se relacionan con las mediciones anteriores del proceso. Para el análisis cualitativo hay dos partes diferenciadas: las técnicas de generación de hipótesis a partir de los datos como la de *comparación constante* y *análisis de casos cruzados* (Seaman 1999), y las técnicas de confirmación de hipótesis como las técnicas de *triangulación* y de *replicación* (Seaman 1999).
5. Presentación de informes: dado que, en ocasiones, los estudios de caso están basados en datos cualitativos, la forma de presentar los informes es más abierta que en otros tipos de estudios empíricos. A pesar de ello se pueden encontrar ejemplos de guías para la elaboración de informes de estudio de caso (Robson 2002).

1.8 Estructura de la tesis

En este capítulo hemos presentado la motivación, el planteamiento del problema, los objetivos e hipótesis de la investigación, las contribuciones, el contexto de la investigación y el método de investigación aplicado. El resto de la tesis se organiza en los siguientes capítulos:

- Capítulo 2: Marco tecnológico

  En este capítulo se introducen los pilares tecnológicos sobre los que se asienta esta tesis: líneas de producto software, arquitecturas software, desarrollo dirigido por modelos y aseguramiento de calidad en líneas de producto software.

- Capítulo 3: Estado del arte

  En este capítulo se analizan las propuestas existentes en el ámbito de la derivación, evaluación de arquitecturas software y el aseguramiento de la calidad en líneas de producto software.

- Capítulo 4: QuaDAI un método para la derivación, evaluación y mejora de arquitecturas de producto software

En este capítulo se presenta la contribución metodológica de esta tesis: el método QuaDAI, integrado por un multimodelo empleado para la especificación de las diferentes vistas de la línea de productos y un proceso dirigido por transformaciones para la derivación, evaluación y mejora de arquitecturas de producto.

- Capítulo 5: Derivación de arquitecturas de producto

  En este capítulo se detalla el proceso de derivación de la arquitectura, teniendo en cuenta tanto requisitos funcionales como no-funcionales. Este proceso incluye la configuración del producto y la obtención, mediante un proceso de transformación, de los modelos arquitectónicos que representan la arquitectura software del producto en desarrollo.

- Capítulo 6: Evaluación y mejora de la arquitectura obtenida

  En este capítulo se detalla el proceso de evaluación y de mejora de la arquitectura mediante la aplicación de transformaciones arquitectónicas.

- Capítulo 7: Herramienta de soporte al método QuaDAI

  En este capítulo se presenta la herramienta que da soporte al método, las funcionalidades incluidas (analizando las distintas decisiones de diseño que se presentaron durante su desarrollo) y los detalles de implementación y funcionamiento.

- Capítulo 8: Estudio de caso para la validación del proceso de derivación de arquitecturas

  En este capítulo se presenta la validación empírica del proceso de derivación mediante un estudio de caso en el dominio de los sistemas software de control de la industria automovilística.

- Capítulo 9: Familia de experimentos para la validación del proceso de evaluación y mejora de arquitecturas software

  En este capítulo se presenta la validación empírica del proceso de evaluación y mejora de arquitecturas mediante una familia de experimentos controlados, cuyo objetivo es comparar la efectividad y eficiencia, facilidad de uso, intención de uso y utilidad de la propuesta en comparación con ATAM, un método de evaluación de arquitecturas software con amplia difusión en la industria.

- Capítulo 10: Conclusiones y trabajos futuros

  En este capítulo se presentan las contribuciones más importantes de esta tesis, así como las líneas de investigación presentes y futuras, junto con las publicaciones que se originaron a partir de este trabajo de investigación.

- Capítulo 11: Conclusions and futher Work

  Este capítulo es la traducción a la lengua inglesa del capítulo 10, incluido en esta memoria con el fin de cumplir con los requisitos de la Universitat Politècnica de València para la obtención del Doctorado Internacional.

- Apéndice A: Case Study Materials

  Este apéndice (en inglés) resume los materiales definidos para detallar las tareas del estudio de caso descrito en el Capítulo 8: boletín del caso, cuestionario de evaluación de las variables subjetivas, formularios de captura de datos, artefactos empleados en las tareas del caso y el cuestionario de nivelación llevado a cabo antes de la ejecución del caso de estudio para detectar el perfil de los participantes.

- Apéndice B: Material experimental

  Este apéndice resume los materiales empleados para llevar a cabo los experimentos descritos en el Capítulo 9: los documentos de captura de datos, cuestionarios, artefactos empleados en las tareas experimentales y cuestionarios de evaluación de las variables subjetivas.

Capítulo 2

Marco Tecnológico

Este capítulo presenta los cuatro pilares sobre los que se asienta este trabajo de investigación: las líneas de producto software, las arquitecturas software, el desarrollo de software dirigido por modelos y el aseguramiento de calidad en líneas de producto software. El objetivo del capítulo es introducir el marco conceptual que permite entender el trabajo de investigación en su totalidad. En las siguientes secciones se van a introducir y discutir cada uno de estos campos, así como las interacciones que aparecen entre ellos. La estructura del capítulo es la siguiente:

La sección 2.1 introduce la aproximación de líneas de producto, la ingeniería de líneas de producto y los conceptos relacionados con la variabilidad.

La sección 2.2 introduce el concepto de arquitectura software, su impacto en la calidad del software, su papel en el desarrollo de líneas de producto y la variabilidad arquitectónica.

La sección 2.3 presenta el enfoque de desarrollo dirigido por modelos, el estándar MDA, la transformación de modelos, el lenguaje de transformaciones QVT y el estándar CVL para la representación de la variabilidad.

La sección 2.4 introduce el aseguramiento de calidad en el desarrollo de líneas de producto software.

2.1 Líneas de producto software

La producción de software de calidad, en el tiempo adecuado y con unos costes razonables, continúa siendo un problema abierto de la ingeniería del software que ha sido abordado desde distintas aproximaciones. Una aproximación industrial consiste en aplicar el desarrollo de líneas de producto software.

> *"Una línea de productos software es un conjunto de sistemas software que comparten un conjunto gestionado de características comunes que satisfacen las necesidades específicas de un segmento de mercado particular o misión y que se desarrollan a partir de un conjunto de activos software de un modo preestablecido"* (Clements y Northrop 2001).

Los beneficios de la aplicación de esta aproximación se alinean con los objetivos de negocio (Clements y Northrop 2001): mejora de la productividad, reducción del tiempo de desarrollo de nuevos productos, aumento de calidad de los productos desarrollados, disminución de los riesgos, incremento de la satisfacción del cliente, habilidad de adaptación a la personalización del producto, etc. Todos estos beneficios provienen del propio proceso de desarrollo de líneas de producto software. Se van a obtener sustanciales mejoras de la productividad y reducciones en los costes cuando los sistemas sean desarrollados a partir de un conjunto de activos existentes de un modo predefinido, en vez de desarrollarlos por separados desde cero o de un modo arbitrario. Cada producto va a estar formado por un conjunto de componentes seleccionados de la base común de activos software, que serán adaptados gracias a una serie de mecanismos de variación, añadiendo los nuevos componentes cuando estos sean necesarios.

Para la adopción del enfoque de líneas de productos software se pueden considerar tres enfoques principales: enfoque proactivo, extractivo, y reactivo (Krueger 2002):

- El enfoque proactivo es análogo al ciclo de desarrollo en cascada para sistemas convencionales y en él se analizan, y diseñan todas las variantes en la línea de productos con toda la variabilidad conocida por adelantado.
- El enfoque extractivo consiste en reutilizar uno o más productos software monolíticos para generar la base común inicial de la línea de productos.

- El enfoque reactivo es análogo al sistema de desarrollo en espiral, y en él se añade incrementalmente variabilidad a un producto o conjunto de productos software en desarrollo.

2.1.1 Ingeniería de líneas de producto software

La ingeniería de líneas de producto software es el paradigma para el desarrollo de sistemas software a partir de un conjunto de subsistemas software e interfaces que conforman una estructura común de la que se pueden desarrollar una serie de productos derivados en los que se permite la personalización masiva (Pohl et al. 2005). Esta definición se complementa con la de (Clements y Northrop 2001), que define prácticas de líneas de producto como el uso sistemático de activos software para ensamblar, instanciar o generar los múltiples productos que integran la línea de productos (Clements y Northrop 2001).

En el ámbito del desarrollo de líneas de producto software, los activos software son la base para la producción de un determinado producto de la línea de productos. Se considera activos software cualquier forma de documentación de diseño, planes de prueba, casos de prueba, especificaciones de requerimientos así como modelos de dominio. Los activos software van a ir acompañados de un *proceso adjunto* que especifica el modo en que deberá ser reutilizado en el desarrollo de productos. El *plan de producción* va a definir como un producto se deriva de los activos software. El plan de producción orquesta los procesos adjuntos de los activos software que intervienen en el desarrollo de un determinado producto definiendo cómo dichos procesos adjuntos pueden conjugarse para construir el producto final.

En la ingeniería de líneas de producto software se distinguen dos procesos de desarrollo diferenciados: la *Ingeniería de Dominio* y la *Ingeniería de Aplicación* (Pohl et al. 2005), que convivirán con las *actividades de gestión* de la línea de productos(Bosch 2000; Clements y Northrop 2001).

La ingeniería de dominio tiene como cometido el *desarrollo de activos software,* que serán combinados para obtener un determinado producto en la fase de ingeniería de aplicación. Para ello, se identifican las similitudes y la variabilidad entre los distintos productos que integran la línea de productos y se establece el *Alcance de la línea de productos.*

> *Alcance de la línea de productos: descripción de los productos que integrarán la línea de productos o que la línea de productos será capaz de incluir* (Clements y Northrop 2001).

La ingeniería de aplicación tiene como cometido el *desarrollo de productos* mediante la reutilización de los activos software desarrollados en la fase de ingeniería de dominio.

Durante el desarrollo del producto, las principales tareas son las de *configuración* y las de *composición*. Las de configuración "instancian" los elementos comunes, resolviendo los puntos de variación. Aquí se pueden utilizar mecanismos como los #ifdefs, o bien, técnicas manuales donde se rellena con código plantillas de funciones ("stubs"). Por otro lado, la composición implica la escritura de código alrededor de estos elementos comunes ("glue code") para facilitar su interconexión.

Los dos procesos de desarrollo (ingeniería de dominio e ingeniería de aplicación) junto con las actividades de gestión, se ejecutan de manera cíclica y concurrente tal como se muestra en la Figura 2.1. Por un lado, el desarrollo de nuevos activos o su evolución, puede dar lugar a la revisión o evolución de productos previamente desarrollados o a la extensión del alcance de la línea de productos con la especificación de nuevos productos. Por otro lado, el desarrollo de un determinado producto en el que se integren características hasta ese momento no soportadas por la base de activos, puede dar lugar a la evolución de la base de activos y la extensión del alcance de la línea de productos.

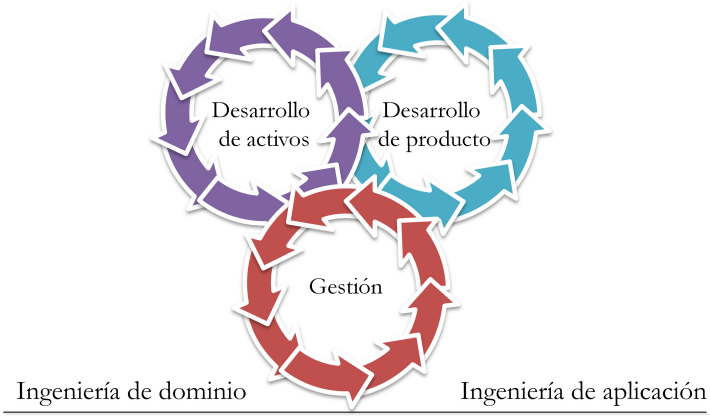


Figura 2.1 Actividades principales de la ingeniería de líneas de producto software

2.1.2 Variabilidad en las líneas de producto software

Uno de los aspectos clave del desarrollo de líneas de producto es la gestión de la variabilidad. La variabilidad en el ámbito de la ingeniería del software se define como:

> *"Habilidad de un sistema software de ser cambiado, personalizado o configurado para su uso en un determinado contexto"* (van Gurp y Bosch 2002).

Durante el proceso de ingeniería de dominio se lleva a cabo un análisis que permite identificar las diferencias entre los distintos productos que integran la línea de productos (Clauß 2001). Estas diferencias se expresan en términos de *características (features)*.

> *Característica: "aspectos o características del dominio visibles desde el punto de vista del usuario"* (Kang et al. 1990).

Estas características identificables por el usuario definen la *variabilidad externa* (Pohl et al. 2005) en el espacio del problema, en oposición a la *variabilidad interna* (Pohl et al. 2005) en el espacio de la solución. Variabilidad interna es la variabilidad existente en los propios activos software que permiten su configuración para dar soporte a la variabilidad externa de la línea de productos.

En una línea de productos debemos distinguir tres tipos principales de variabilidad (van der Linden et al. 2007):

1. *Características comunes:* aquellas que son comunes a todos los productos dentro del alcance de la línea de productos.
2. *Variabilidad:* característica/s que puede/n ser común/es a algunos productos, pero no a todos, y que deberán ser modelada de forma explícita como una posible variabilidad e implementada de manera que solo esté presente en los productos en la que éstas se seleccionen.
3. *Especifica de producto:* aquellas características que están presente únicamente en un producto (al menos en un futuro cercano). Determinadas características pueden no ser requerida por el mercado, pero puede formar parte de los requisitos de un determinado cliente.

La mayoría de enfoques para el modelado de variabilidad se basan en la propuesta inicial de *Feature Oriented Domain Analysis* (FODA) (Kang et al. 1990).

Las características se organizan en árboles o nodos *and* / *or* dando lugar a modelos de variabilidad que pueden ser también extendidos con reglas que restringen las posibles configuraciones de productos validos (Chen et al. 2009). El propósito de los modelos de variabilidad, es describir los requisitos de las soluciones. La configuración de un producto se lleva a cabo mediante la selección de características sobre este modelo de variabilidad.

En los modelos de variabilidad clásicos, se definen únicamente tres tipos básicos de relaciones entre una característica padre y sus características hijas y dos relaciones o reglas de composición entre características:

- *Obligatorias:* Una característica obligatoria es aquella que ha de estar de forma preceptiva en una configuración valida, representando características comunes de la línea de producto. Una característica hija que está conectada con una característica padre mediante la relación de obligatoriedad implica que la característica hija requiere que la característica padre esté presente en el producto final. Esta relación en la notación FODA original se expresa mediante dos características unidas por una línea simple. En la Figura 2.2.(a) se muestran dos características obligatorias (A y B).

- *Opcional:* Una característica opcional es aquella que puede estar o no presente en una configuración valida. En la notación FODA original las características opcionales se denotan mediante un círculo. En la Figura 2.2.(b) la característica A es obligatoria mientras que la B es opcional.

- *Alternativas:* Esta relación ha de definirse entre una característica padre y un conjunto de características hijas. En este caso, la relación implica que la característica hija pueda considerarse como una especialización o refinamiento de la característica padre y que, únicamente una característica hija, puede estar presente en una configuración valida. En la Figura 2.2.(c) se muestra un modelo de características con dos posibles configuraciones validad, la combinación A y B o la combinación A y C.

- *Dependencia Mutua:* Esta relación de composición explicita que la presencia de una característica requiere la presencia de otra y viceversa (interdependencia). En la Figura 2.2.(d) se muestra un modelo de características en la que la presencia de la característica opcional B, implica la presencia de la característica opcional C y viceversa, así pues la única configuración validad es A, B y C.

- *Exclusión Mutua (excludes):* Esta relación de composición explicita que la presencia de una característica excluye la presencia de otra. En la Figura 2.2.(e) se muestra un modelo de características en la que la presencia de la característica opcional B excluye la presencia de la característica opcional C y viceversa, así pues la única configuración validad es A.

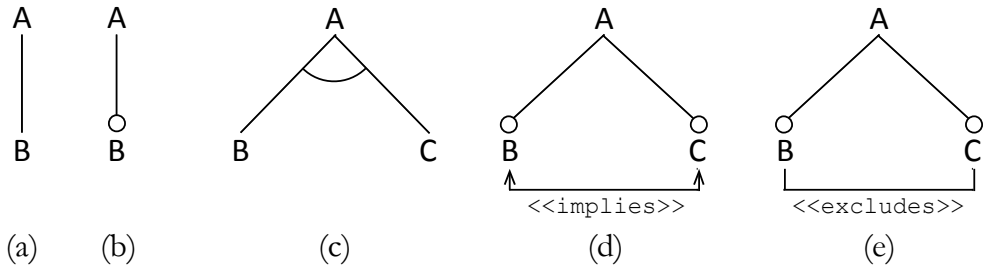


Figura 2.2 Relaciones FODA

A partir de la propuesta de FODA se han definido diferentes extensiones, como los modelos de características con cardinalidades (Riebisch et al. 2002; Czarnecki y Kim 2005; Gómez y Ramos 2010) o los modelos de características extendidos (Kang et al. 1998; Benavides et al. 2005; Roos-Frantz et al. 2011). En los modelos de características con cardinalidades, además de la jerarquía de características, se pueden definir cuantos clones de una determinada característica pueden estar presentes en una configuración. La clonación de características es útil cuando se pretenden definir múltiples copias de una parte del sistema que pueden ser configuradas de forma independiente(Gómez y Ramos 2010). Los modelos de características extendidos, permiten la inclusión de información adicional a las características en forma de atributos, para poder expresar información acerca de propiedades no-funcionales.

Una determinada configuración podrá estar compuesta por características obligatorias, características opcionales seleccionadas y características específicas del producto en desarrollo (características que no están soportadas por la variabilidad de la línea de productos pero que estarán presentes en ese producto en particular).

2.2 Arquitecturas software

A medida que los sistemas software han ido creciendo en complejidad, se ha hecho necesario aislar los detalles de implementación para poder prestar

atención al comportamiento e interacción de los elementos que integran el sistema como "*cajas negras*" y poder diseñar sistemas que posean las propiedades deseadas. Ésta es la idea subyacente de las arquitecturas software.

De manera abstracta, la arquitectura software implica la descripción de los elementos de los que se compone el sistema, las interacciones entre esos elementos, los patrones que guían su composición y las restricciones en esos patrones (Shaw y Garlan 1996).

Desde la aparición del término arquitectura software en los años noventa, se han propuesto más de 150 definiciones (Clements et al. 2011). La más aceptada es la propuesta por Bass et al. (1998):

> *"La arquitectura de un programa o sistema computacional es la estructura o estructuras del sistema, que comprende los componentes software, sus propiedades visibles desde el exterior y las relaciones entre ellos".*

Esta definición va en línea con la adoptada en el estándar ISO/IEC/IEEE para la descripción de arquitecturas ISO/IEC/IEEE 42010:2011 *Systems and Software Engineering – Architecture description* (ISO 2011) que sustituye a la ISO1471:2000 y que define el término como:

> *"La arquitectura son los conceptos fundamentales o propiedades del sistema en su ambiente expresada a través de sus elementos, relaciones, los principios de su diseño y evolución".*

Lo que aparece como denominador común en las distintas definiciones, es que la arquitectura software se compone de elementos, conexiones o relaciones entre ellos y propiedades relevantes.

El diseño de arquitecturas software puede ser dividido en *diseño de alto nivel* y *diseño de bajo nivel*. El diseño de alto nivel se centra en la división del sistema en partes, de las decisiones de diseño y de la estructura general del sistema. El diseño de bajo nivel se centra en el diseño detallado de los artefactos que guiarán los detalles de implementación (Loughran et al. 2007).

La arquitectura software de sistema es una entidad compleja que no puede ser descrita de una forma simple utilizando una representación mono-dimensional (Clements et al. 2011). Documentar una arquitectura consiste en representar las vistas relevantes y añadir documentación que involucre a más de una vista. El número y nombre de las vistas necesarias para la representación de una

arquitectura software es un tema muy controvertido y en el que no se ha logrado un consenso. Una de las representaciones más conocidas es la representación 4 + 1 (Kruchten 1995), que aboga por describir la arquitectura mediante 5 vistas: la vista lógica, la vista de desarrollo, la vista de proceso, la vista física y la vista de escenarios. El estándar ISO/IEC/IEEE 42010:2011 (ISO 2011), propone que el número de vistas y su naturaleza se definan en función de las características del sistema a desarrollar. No obstante, el conjunto de vistas más comúnmente aceptado es el propuesto por Clements et al. (2011), en el que se propone que la descripción de la arquitectura software se lleve a cabo mediante la vista modular, la vista de componente-conector y la vista de despliegue.

2.2.1 Descripción de arquitecturas software

Para la descripción de las vistas arquitectónicas existen varias opciones en función del grado de formalidad requerida:

- **Notaciones informales:** las distintas vistas arquitectónicas se describen mediante diagramas de propósito general y convenciones visuales ad-hoc para la representación tanto de los elementos de la arquitectura como de las relaciones entre ellas.
- **Notaciones semi-formales:** las vistas arquitectónicas se describen mediante notaciones estandarizadas en las que se prescriben los elementos gráficos y las reglas de construcción, pero en las que no hay una semántica definida para los distintos elementos que constituyen la arquitectura.
- **Notaciones formales:** las distintas vistas arquitectónicas se describen mediante notaciones en las que los elementos y las relaciones cuentan con una semántica definida. Las notaciones formales definidas explícitamente para la descripción de arquitecturas software se denominan *Lenguajes de Descripción Arquitectónica.*

La descripción de las vistas arquitectónicas mediante lenguajes de descripción arquitectónica permite validar la descripción tanto a nivel sintáctico (la arquitectura está correctamente definida en términos de ese lenguaje), como a nivel semántico, y puede permitirnos además, analizar de manera automatizada propiedades no-funcionales de la arquitectura como puedan ser los análisis de rendimiento o de fiabilidad.

En los últimos años se han propuesto multitud de lenguajes de descripción arquitectónica como AADL (Feiler et al. 2006), SysML (Object Management Group 2012b) o EAST-ADL2 (Berntsson et al. 2008) que junto con UML, son los lenguajes que más atención han recibido por parte de investigadores y profesionales, como se desprende de recientes revisiones sistemáticas que analizan el uso de los lenguajes de descripción arquitectónica en el ámbito de los sistemas empotrados (Dajsuren et al. 2012; Guessi et al. 2012; Antonio et al. 2012).

Uno de los lenguajes más utilizados para la descripción de arquitecturas software es UML, a pesar de que no se trata de un lenguaje de descripción de arquitecturas propiamente dicho (Clements et al. 2011). UML es el estándar *de facto* para el modelado de sistemas software. UML Tiene sus origines en la orientación a objetos, y en ocasiones, las abstracciones que posee no son el mejor medio para representar arquitecturas software ya que dificulta la descripción detallada de alguna de las vistas arquitectónicas, como por ejemplo, la vista componente-conector (Clements et al. 2011).

System Modeling Language (SysML) es un lenguaje de propósito general para el modelado de sistemas (hardware y software) que da soporte a las actividades de diseño en aplicaciones de la ingeniería de sistemas. Aunque no se trata de un lenguaje específicamente definido para la descripción de arquitecturas, sí que posee las abstracciones necesarias para su representación. Fue desarrollado inicialmente como un perfil UML, añadiendo las extensiones necesarias para el modelado de sistemas. SysML incorpora el concepto de vista y se pueden crear definiciones acordes al estándar ISO/IEC/IEEE 42010:2011.

El estándar *Architectural Analysis Design Method* (AADL) define un lenguaje para representar la arquitectura en ejecución de un sistema como un modelo basado en componentes. La arquitectura del sistema se define en términos de sistemas, subsistemas, conexiones, tareas, plataforma de ejecución y contexto físico. AADL dota de semántica específica a cada tipo de componentes arquitectónicos. Además, permite la definición de conjuntos de propiedades definidas por el usuario y anexos al lenguaje para poder llevar a cabo análisis ad-hoc de diferentes atributos de calidad de la arquitectura a partir de los modelos, como análisis de seguridad, de consumo de recursos o de prestaciones (Feiler 2007a; Clements et al. 2011).

EAST-ADL2 es un lenguaje de descripción arquitectónica definido por un consorcio de fabricantes de la industria automovilística. Provee de las abstracciones necesarias para definir los distintos artefactos con los que definir la arquitectura de sistemas empotrados (requisitos, características,

comportamiento, componentes software y hardware) y sus dependencias (refinamientos, despliegue o comunicaciones) (Cuenot et al. 2011). La especificación en EAST-ADL2 está organizada en cinco niveles de abstracción, siendo la cuarta una capa en la que se describen los detalles de implementación sobre una arquitectura compatible con el estándar AUTOSAR[3] y la última una representación del software y del hardware real del vehículo sobre el que se ejecuta el software definido en capas superiores.

2.2.2 El impacto de las arquitecturas en la calidad del software

Las arquitecturas software son el medio a través del cual es posible alcanzar los requisitos de calidad del sistema en desarrollo. En la propia definición de arquitectura propuesta por Bass et al. (1998) ya se explicita que arquitectura, son también las propiedades visibles de los componentes del sistema. Por propiedades se entienden los servicios que un componente ofrece al resto (funcionalidad), pero también características como rendimiento, manejo de errores, tolerancia frente a fallos o uso de recursos (Clements et al. 2002).

La Figura 2.3 muestra la relación existente entre los requisitos de calidad, la arquitectura y los atributos de calidad del sistema, aunque. Los requisitos no-funcionales del sistema, expresados en términos de atributos de calidad, guían las decisiones de diseño y el desarrollo de la arquitectura del sistema. Esas decisiones de diseño, en ocasiones, condicionan el proceso de desarrollo y determinan la calidad del software y las capacidades que finalmente tendrá.

En la literatura podemos encontrar distintas definiciones para el termino requisitos no-funcionales (RNF), recogidas en su mayoría por Glinz (2007). De entre todas ellas, en esta tesis seguimos la propuesta por (Robertson y Robertson 1999) dado que se ajusta mejor a la estructura de la ISO/IEC SQuaRE (ISO 2005), al considerar las características de rendimiento como una RNF más del sistema.

> *Requisitos no-funcionales, son las calidades que el producto ha de satisfacer, como apariencia, rendimiento o precisión.*

[3] AUTOSAR es un consorcio integrado por fabricantes de la industria automovilística, que ha establecido una arquitectura software de referencia para las unidades de control de vehículos (*Electronic Control Units*).

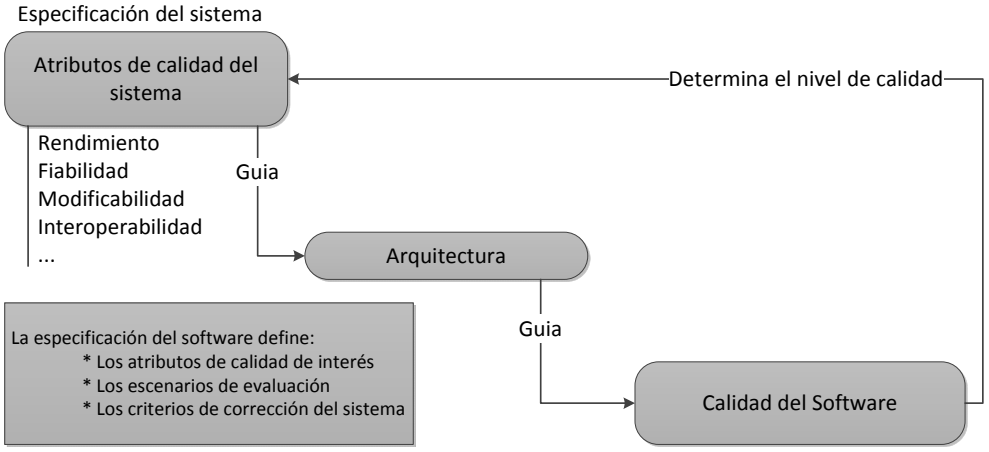


Figura 2.3 Relaciones entre requisitos de calidad del sistema y arquitecturas software (Bergey et al. 1999)

Los requisitos no-funcionales a nivel arquitectónico pueden ser clasificados en *operacionales* o de *desarrollo* (Bosch 2000).

> *Requisitos no-funcionales operacionales, son las propiedades del sistema en funcionamiento, como el rendimiento, la fiabilidad, robustez y tolerancia a fallos.*

> *Requisitos no-funcionales de desarrollo, son las propiedades de calidad del sistema desde el punto de vista de la ingeniería del software, como la mantenibilidad, reusabilidad o flexibilidad.*

La arquitectura abstrae los detalles de implementación para permitirnos razonar acerca de las propiedades del mismo, relacionadas con la calidad del sistema. Así, se puede evaluar los atributos de calidad de la arquitectura en fase de diseño. El hecho de tener que esperar a disponer del sistema completamente implementado para poder medir los valores reales de los atributos de calidad, es una aproximación ineficiente. Hay que invertir recursos en desarrollar por completo un sistema que puede que carezca de los requisitos de calidad deseados, para entonces evaluarlo y plantear las mejoras necesarias con el fin de que este cumpla con dichos requisitos de calidad.

Dependiendo del atributo de calidad de interés en cada caso, será posible medirlo con mayor o menor precisión sobre la arquitectura en fases tempranas. Es obvio que no tiene demasiado sentido medir determinados atributos de calidad sobre una especificación abstracta, y es por ello que se pueden emplear

distintos tipos de evaluaciones arquitectónicas, en función de la naturaleza de la arquitectura a evaluar y de los atributos de calidad de interés.

Los distintos atributos de calidad no son independientes entre sí, y se van a producir interacciones entre ellos (por ejemplo la modificabilidad, que puede afectar negativamente al rendimiento y positivamente a la fiabilidad). Las evaluaciones arquitectónicas van a permitir también analizar estos *trade-offs* entre atributos.

2.2.3 Arquitecturas software y líneas de producto

Cualquier arquitectura es una abstracción que permite múltiples instancias. Elevar el nivel de abstracción para centrarnos en detalles de diseño, permite posteriormente que ese diseño admita distintas implementaciones (Clements y Northrop 2001). Pero en el caso de las líneas de producto, se pretende que esas múltiples instancias vayan más allá de la implementación de un mismo diseño. La arquitectura de la línea de productos describe los conceptos y estructuras para alcanzar la variabilidad que da soporte a los distintos productos que conforman la línea de productos, mientras que trata de compartir el máximo de partes comunes en la implementación como sea posible (Jazayeri et al. 2000).

La arquitectura de la línea de productos, es el resultado de la ingeniería de dominio y se emplea como un activo software más en la ingeniería de aplicación (van der Linden et al. 2007). El diseño de la arquitectura de la línea de producto, tiene que llevarse a cabo a partir del análisis del alcance, para ser capaces de dar soporte a todas y cada una de las variantes soportadas por la línea de productos. Los cambios en la arquitectura de la línea de productos son pequeños y poco frecuentes, dado que cambios profundos en dicha arquitectura van a afectar a la integración y a la reutilización de los componentes de la misma.

La arquitectura de producto se deriva de la arquitectura de la línea de productos resolviendo la variabilidad prevista en esta última con el fin de cumplir los requisitos (tanto funcionales como no-funcionales) del producto en desarrollo. Durante la derivación de la arquitectura de producto, la arquitectura resultante contendrá un subconjunto de los elementos de la de arquitectura de la línea de productos, fruto de podar la arquitectura de la línea de productos suprimiendo aquellos elementos que no son necesarios en el producto en desarrollo (Bosch 2000). Además, en algunos casos, la arquitectura del producto podrá contener nuevos elementos que den soporte a las características específicas del producto en desarrollo.

En un nivel abstracto contamos con tres técnicas básicas para la manejar la variabilidad en la arquitectura las cuales se describen de forma esquemática en la Figura 2.4: adaptación, reemplazo y extensión, las cuales se describen en detalle a continuación (van der Linden et al. 2007):

- *Adaptación*: solamente hay disponible una implementación de un cierto componente, pero ofrece distintas interfaces para ajustar su comportamiento (ver Figura 2.4.(a)).
- *Reemplazo*: hay disponibles varias implementaciones de cada componente. Cada componente se adhiere a la especificación del componente en la arquitectura. En la ingeniería de aplicación, se elige una de estas implementaciones, o se desarrolla una implementación específica para ese producto, de nuevo, siguiendo las especificaciones (ver Figura 2.4.(b)).
- *Extensión*: la técnica de extensión requiere que la arquitectura cuente con interfaces que permitan añadir nuevas componentes. Los componentes pueden o no pueden ser específicos de producto. La diferencia con el remplazo, es que ahora, únicamente hay interfaces genéricos, permitiendo que se añadan distintos tipos de componentes mientras que en el reemplazo, el interfaz especificaba exactamente que debía hacer el componente y únicamente como se hace variable (ver Figura 2.4.(c)).

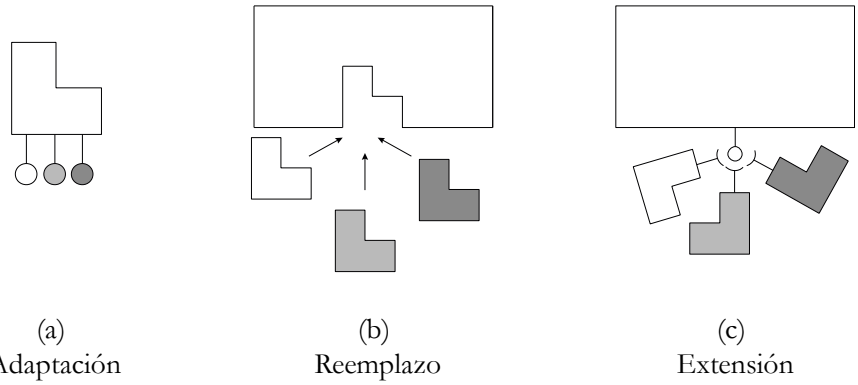


Figura 2.4 Técnicas básicas de variabilidad a nivel abstracto (van der Linden et al. 2007)

En la arquitectura de la línea de productos estas tres técnicas abstractas se despliegan en los puntos y mecanismos de variabilidad arquitectónica.

2.2.4 Variabilidad arquitectónica

La experiencia ha demostrado que el desarrollo de arquitecturas software mediante la conexión de componentes *as-is* (donde los componentes son reusados sin necesidad de modificarlos), es en gran medida ilusorio (Bosch 2000). Por ello, la variabilidad ha de ir más allá de la mera conexión de componentes y ha de permitir la introducción de cambios estructurales, de comportamiento o, en algunos casos, incluso de plataforma de ejecución. Por último, la variabilidad arquitectónica debe ser descrita en términos de las descomposiciones del diseño escogido, y éstas, no siempre se corresponden de manera univoca con la jerarquía de características representadas en el modelo de calidad (Loughran et al. 2008).

Hay dos causas principales para requerir la representación de alternativas en una arquitectura (Bachmann y Bass 2001):

1. Durante el diseño pueden aparecer diferentes alternativas que deberán ser capturadas si implican una selección posterior de las mismas.
2. En la arquitectura de una línea de productos se deberán representar la colección de alternativas que nos permita soportar la variación entre productos. Estas variantes y sus fundamentos deberán ser capturadas y plasmadas con el fin de poder expresar la lista de potenciales soluciones de las que elegir.

Con el fin de que dichas alternativas sean plasmadas en la arquitectura y puedan ser representadas en los modelos arquitectónicos, éstos deberán ser capaces de soportar la variabilidad arquitectónica.

La variabilidad arquitectónica es soportada a través de los denominados *puntos de variabilidad*.

> *Puntos de Variabilidad: lugares en la arquitectura en los que se brindan instancias específicas de flexibilidad* (Clements et al. 2011).

La flexibilidad se consigue dejando intencionalmente las decisiones arquitectónicas abiertas, para que sea posible limitar dichas decisiones más adelante en el ciclo de desarrollo. En el contexto de las líneas de producto, las decisiones arquitectónicas se dejan abiertas para limitarlas durante la ingeniería de la aplicación.

A nivel arquitectónico contamos con distintos mecanismos de variabilidad arquitectónica para soportar la selección de una determinada variante en un punto de variación. A continuación se describe la lista de mecanismos de variación inicialmente definida por (Bachmann y Bass 2001) y posteriormente ampliada por Clements et al. (2011):

- *Sustitución de elementos:* Consiste en el reemplazo de la implementación de un módulo o componente con una implementación alternativa del mismo interface. Esto dará lugar a una versión del sistema con una característica que se comporta de un modo, mientras que tendremos una segunda versión de la característica que se comporta de un modo diferente.

- *Replicación de componentes:* Consiste en la creación de múltiples instancias de un componente para dotarlo de mayor capacidad.

- *Inclusión de opciones:* Consiste en incluir en algunos productos un determinado componente que en otros productos se omitirá. Esto nos permite dotar a algunos productos de características específicas.

- *Frameworks:* Un framework es una abstracción que nos permite sobrescribir o especializar un código común, que provee una determinada funcionalidad, con un código de usuario, que provee otra funcionalidad especifica.

- *Parametrización:* Provee mecanismos de variación para un amplio rango de constructores. La parametrización permite variar nombres de fichero, URLs, valores mínimos y máximos, etc.

- *Composición de Elementos:* Consiste en el ensamblado de nuevos componentes por agregación de componentes existentes.

- *Plantillas:* Consiste en la definición de cuerpos que están casi, pero no totalmente, completos. Los diseñadores completarán las partes abiertas para dotar de la funcionalidad requerida. Este mecanismo es generalmente empleado para código, pero puede también ser usado en la arquitectura, por ejemplo, dejando algunas partes de un diagrama arquitectónico pendientes de rellenar.

- *Herencia:* Consiste en la definición de clases genéricas e interfaces, que posteriormente serán extendidas con distintas implementaciones, que darán lugar a subclases especificas o clases que *implementan* esos interfaces.

- *Generación:* Consiste en el empleo de un software (el generador) que toma como entrada una especificación del producto deseado y produce como salida un programa que cumple esa especificación.

Los distintos lenguajes de descripción de arquitecturas descritos en la sección 2.2.1, implementan diferentes variantes de los mecanismos de variabilidad arquitectónica anteriormente descritos:

- UML incorpora la noción de herencia y nos permite describir la sustitución mediante la relación *realizes* entre un interface y distintas implementaciones, además permite la multiplicidad de componentes mediante las composiciones y agregaciones tipadas y con multiplicidad (Clements et al. 2011).

- SysML soporta la variabilidad mediante el uso de la especialización de la herencia en la especificación de interfaces y componentes, así como el ensamblaje a través de interfaces, que permite la especificación de la variabilidad a distintos niveles (Shiraishi 2013).

- AADL soporta la variabilidad por un lado gracias a la especificación de la arquitectura a dos niveles: i) la definición externa de interfaces con la especificación de tipos y ii) la especificación interna de la implementación de la especificación de un tipo, y por otro lado gracias al mecanismo de extensión (*extends*) y de refinamiento (*refined to*) como mecanismos de herencia (Shiraishi 2010; Shiraishi 2013; Feiler 2007b).

- EAST-ADL2 provee de diferentes niveles de definición de un sistema para soportar la variabilidad: i) *VehicleFeatureModel* donde se representan las características de la electrónica del vehículo; ii) *AnalysisArchitecture* que contiene la descripción abstracta de la funcionalidad de la electrónica del vehículo; iii) *DesignArchitecture* que contiene la especificación y la arquitectura hardware del vehículo; iv) *ImplementationArchitecture* que contiene la arquitectura software y sus componentes, así como la arquitectura hardware del vehículo; v) *OperationalArchitecture* representa el software real y la electrónica en el vehículo manufacturado. Esta jerarquía de modelos y arquitecturas contiene nuevos modelos y elementos, así como relaciones que cruzan las fronteras de los distintos modelos/arquitecturas (ADLRelationships, ADLRealization y ADLSatisfy). Además EAST-ADL se basa en la definición de los detalles de implementación en otro lenguaje de modelado, AUTOSAR, un estándar de facto adoptado por un amplio conjunto de fabricantes de la industria automovilística que,

de nuevo, añade un nuevo nivel de abstracción para el soporte de la variabilidad.

2.3 Desarrollo de software dirigido por modelos

Elevar el nivel de abstracción en la descripción de los problemas y sus soluciones ha sido una de las constantes en la evolución de la industria del software desde sus inicios. Una muestra de ello es la irrupción del análisis estructurado en la década de los 70 y 80 o de la orientación a objetos a mediados de la década de los 90. Ello ha contribuido a aumentar la capacidad de entender y resolver problemas cada vez más complejos, así como a mejorar el proceso de desarrollo de software.

El modelado, es una herramienta clave en cualquier proceso científico o de ingeniería (Brambilla et al. 2012). En cualquier rama de la ingeniería se crean modelos que abstraen los detalles de interés de un determinado problema y permiten el análisis de las soluciones a dicho problema. Un modelo es creado con un determinado propósito, para contestar a un determinado tipo de preguntas, y las respuestas que se extraen del modelo serán las mismas que las extraídas de la realidad que modelan (Seidewitz 2003). En el campo de la ingeniería del software, el uso de modelos no es nuevo, han sido utilizados durante mucho tiempo para documentar tanto la estructura interna del software como la estructura de los sistemas, sus interfaces o las estructuras de datos subyacentes.

El paso de desarrollar software haciendo uso de modelos (con propósitos de documentación), a desarrollar el software a partir de modelos, significa un salto significativo en pos de elevar el nivel de abstracción. Éste es el objetivo que persigue el *Desarrollo de Software Dirigido por Modelos* (DSDM), en el que se aborda el desarrollo de sistemas software mediante el refinamiento o transformación sucesiva de modelos. El sistema se modela en términos del dominio del problema. Cada nuevo modelo, desciende en el nivel de abstracción, añadiendo nuevos detalles de la plataforma destino a los modelos definidos en niveles de abstracción superiores, hasta llegar a la obtención del código del sistema en desarrollo. En este escenario, los modelos pasan de ser meros elementos de documentación, a ser parte del software, constituyendo un factor decisivo para incrementar la velocidad de desarrollo y la calidad del software obtenido (Stahl et al. 2006).

El enfoque DSDM se apoya en el uso de modelos formales (o semi-formales), donde el modelo tiene exactamente el mismo significado y tratamiento que el

código de la aplicación, dado que éste finalmente se genera a partir de ellos por transformación sucesiva.

El uso de transformaciones automáticas de modelos a lo largo del ciclo de vida tiene múltiples beneficios:

- Fomenta la reutilización una vez definida la cadena de transformaciones que pasan de los modelos en un alto nivel de abstracción al código de la solución final. Éstas transformaciones pueden ser utilizadas una y otra vez para construir sistemas similares.
- Minimiza los errores en tareas repetitivas de creación de un modelo a partir de otro.
- Mejora la calidad de los modelos resultantes al encapsular las buenas prácticas en la propia definición de la transformación.
- Encapsula el conocimiento de los expertos del dominio en las transformaciones, una vez una transformación ha sido definida, esta puede ser empleadas por desarrolladores que no necesitan conocer el proceso de transformación en detalle.

Para la implantación industrial de esta estrategia, era necesaria la aparición de estándares de modelado, metodologías y herramientas, que diesen soporte al ciclo de desarrollo mediante herramientas interoperables entre sí.

2.3.1 Model Driven Architecture

Precisamente la falta de estándares y herramientas interoperables es lo que llevó al *Object Management Group* (OMG) a proponer la iniciativa *Model Driven Architecture* (MDA) (Object Management Group 2003), mediante la cual se establecen una serie de estándares para dar soporte al desarrollo de sistemas portables (independientes de plataforma) mediante herramientas interoperables siguiendo el enfoque DSDM. MDA es pues la visión particular de DSDM propuesto por la OMG en base a sus estándares y por tanto puede ser visto como un subconjunto de DSDM (Brambilla et al. 2012).

MDA define múltiples estándares aunque sus pilares básicos son:

- *Unified Modeling Languaje (UML)* es el estándar de facto para el modelado de sistemas independiente del dominio.

- *Meta Object Facility* (MOF) define el lenguaje común empleado para definir metamodelos[4]. Es el meta-meta modelo a partir del cual se definen todos los lenguajes de modelado, incluido UML.
- *Object Constraint Languaje* (OCL) es un lenguaje declarativo y sin efectos colaterales, que permite la definición de restricciones como reglas en los modelos basados en MOF.
- *Query View Transformation* (QVT) es el lenguaje estándar para definir transformaciones en modelos basados en MOF.
- *XML Metadata Interchange* (XMI) es el mecanismo de persistencia para el almacenamiento e intercambio de modelos entre herramientas compatibles con MOF.

MDA define, además, su propio proceso de desarrollo para producir el código ejecutable de un sistema a partir de la especificación, en términos de modelos, con una sintaxis y semántica formales. El proceso de desarrollo del sistema parte de los siguientes conceptos (Object Management Group 2003):

- *Plataforma*: conjunto de subsistemas y tecnologías que proveen un conjunto coherente de funcionalidad a través de interfaces y el uso de patrones de uso, que cualquier aplicación soportada por la plataforma podrá usar sin prestar atención a como la funcionalidad ofrecida esta implementada.
- *Modelo Independiente de Computación (Computation Independent Model o CIM):* Vista del sistema abstrayendo detalles de estructura/s, sistemas de procesamiento de datos, etc. Es un modelo del sistema que muestra el sistema en su entorno de operación, y muestra cuál será su cometido. Para su especificación se emplean términos del vocabulario común entre los profesionales del dominio de interés.
- *Modelo Independiente de Plataforma (Platform Independent Model o PIM):* Vista del sistema en el que se ocultan los detalles de la plataforma de ejecución y que permite describir el sistema de manera que, esta descripción, pueda ser utilizada con varias plataformas cuyo tipo sea similar.

[4] Metamodelo: descripción de la estructura que puede tener un modelo (Stahl et al. 2006); modelo de un lenguaje para expresar modelos (Favre 2004b).

- *Modelo de Plataforma (Platform Model):* Conjunto de conceptos técnicos que representan las distintas partes que conforman una plataforma y los servicios que ésta ofrece. También provee los conceptos que representan los distintos elementos para ser usados en la especificación del uso de la plataforma por una aplicación.
- *Modelo Específico de Plataforma (Platform Specific Model o PSM):* Vista del sistema en el que se combina la especificación del PIM con los detalles de cómo el sistema hace uso de plataforma específica.

El proceso de desarrollo parte de una especificación del sistema en modelos PIM, que será transformado mediante una o varias transformaciones en uno o varios modelos PSM. El código final de la aplicación se obtiene mediante una transformación de modelos PSM a código. La separación entre PIM y PSM, es clave en el framework MDA y es lo que permite la definición de sistemas portables, al abstraer en la primera definición del sistema los detalles de la plataforma. La definición de transformaciones de PIMs a PSMs de distintas plataformas es el mecanismo mediante el cual se podrán obtener versiones del mismo sistema para distintas plataformas.

2.3.2 Transformaciones de modelos en DSDM

Las transformaciones de modelos son uno de los puntos clave en el DSDM, guiando el proceso de desarrollo y derivando el código de la aplicación a partir de la especificación del sistema en modelos PIM y posteriormente PSM.

Una transformación de modelos es el proceso automatizado de generar un modelo destino a partir de un modelo origen de acuerdo a una definición de transformación que está expresada únicamente en términos de los metamodelos involucrados en dicha transformación (Kleppe et al. 2003; Stahl et al. 2006). La definición de una transformación consiste en una colección de reglas que especifican como un modelo, que se ajusta a un determinado metamodelo, puede ser transformado en otro modelo que se ajuste a su correspondiente metamodelo. Las reglas de transformación son especificaciones inequívocas de la forma en que un conjunto de elementos de un modelo que se ajusta a un determinado metamodelo, puede ser usada para crear un conjunto de elementos en otro modelo que se ajusta a su correspondiente metamodelo (Kleppe et al. 2003). La Figura 2.5 muestra la definición de una transformación vertical entre un PIM y un PSM y sus correspondientes metamodelos.

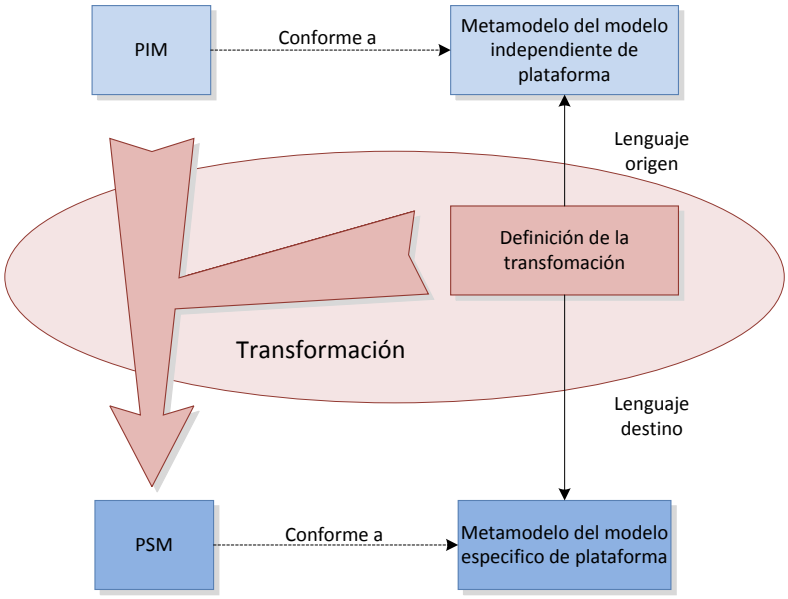


Figura 2.5 Ejemplo la definición de una trasformación de PIM a PSM

En un proceso de desarrollo DSDM, se pueden presentar distintos tipos de transformaciones. Mens et al. (2005) presentan una taxonomía que permite clasificar los distintos tipos de transformaciones, basándose en el objeto de la transformación. Establecen cuatro criterios principales:

1. *Transformaciones de programa y transformaciones de modelos:* si el objeto de la transformación son programas (código fuente), hablaremos de transformaciones de programa (código-a-código). Si por el contrario, al menos uno de los elementos de la transformación es un modelo, entonces hablamos de transformaciones de modelos que pueden ser de modelo-a-modelo (M2M) o transformaciones modelo-a-texto (M2T).
2. *Nivel de abstracción:* distinguiremos entre transformaciones horizontales (aquellas en que ambos modelos se encuentran al mismo nivel de abstracción), frente a las transformaciones verticales (aquellas en que el modelo origen está a distinto nivel de abstracción que el modelo destino), como son las transformaciones PIM-PSM descritas en la sección anterior.
3. *Lenguajes involucrados:* distinguiremos entre transformaciones exógenas (aquellas que se dan entre modelos que se ajustan a distintos metamodelos), frente a las transformaciones endógenas (aquellas en que ambos modelos se ajustan al mismo metamodelo).

4. *Direccionalidad:* por último, las trasformaciones pueden ser definidas como correspondencias unidireccionales (como las que se pueden definir empleando lenguajes de transformación de modelos como ATL o QVT-Operational), en oposición a las relaciones que definen transformaciones bidireccionales (como las que se pueden definir empleando lenguajes de transformación de modelos como QVT-Relations).

2.3.3 El lenguaje de transformación QVT

Con el objetivo de dar soporte a la transformaciones entre modelos acordes al meta-metamodelo MOF, la OMG definió el estándar *Query /View /Transformation* (QVT) (Object Management Group 2008a), que ofrece varios lenguajes de definición de transformaciones.

2.3.3.1 Lenguajes del estándar QVT

La especificación del lenguaje tiene una naturaleza dual declarativa e imperativa. Por un lado consta de dos lenguajes declarativos y por otro de dos implementaciones imperativas. En la vertiente declarativa, ofrece dos lenguajes a dos niveles de abstracción distintos, el lenguaje de alto nivel *QVT-Relations* y el lenguaje *Core*. En la vertiente imperativa, ofrece los lenguajes *QVT-Operational Mappings* y *Black Box*.

QVT-Relations permite la definición de expresiones de *unificación de patrones* (patrones de objetos) sobre conjuntos complejos de objetos. QVT-Relations crea y mantiene también los enlaces de trazabilidad entre los modelos involucrados en las transformaciones.

El lenguaje Core está definido mediante un súper-conjunto mínimo de los estándares MOF y OCL que provee las capacidades de transformación básicas. El soporte a la trazabilidad ha de ser gestionado de forma explícita, y no se deduce de la definición de la transformación, como ocurre en el caso de QVT-Relations.

El lenguaje QVT-Operational provee un estándar para la definición de implementaciones de transformaciones QVT con una sintaxis más cercana a los lenguajes imperativos. Mediante una extensión de las expresiones OCL, que permite que tengan efectos colaterales se pueden definir las transformaciones con un estilo procedural.

Por último, el estándar QVT permite la definición de implementaciones de caja negra (*black box*), con el fin de poder embeber operaciones sobre MOF con la misma signatura que una relación declarativa. Esto comporta las siguientes ventajas:

- Permite la implementación de algoritmos complejos en cualquier lenguaje de programación.
- Permite emplear librerías específicas para el cálculo de ciertas propiedades de los modelos.
- Permite ocultar la implementación de partes de una transformación.

Este tipo de integraciones entrañan ciertos riesgos, ya que la implementación de caja negra tiene acceso a las referencias de los objetos en el modelos y puede llevar a cabo acciones arbitrarias sobre esos objetos, que pueden derivar en la perdida de la encapsulación. Las implementaciones *black-box* no mantienen los enlaces de trazabilidad (como hacia QVT-Relations o la versión operacional del lenguaje).

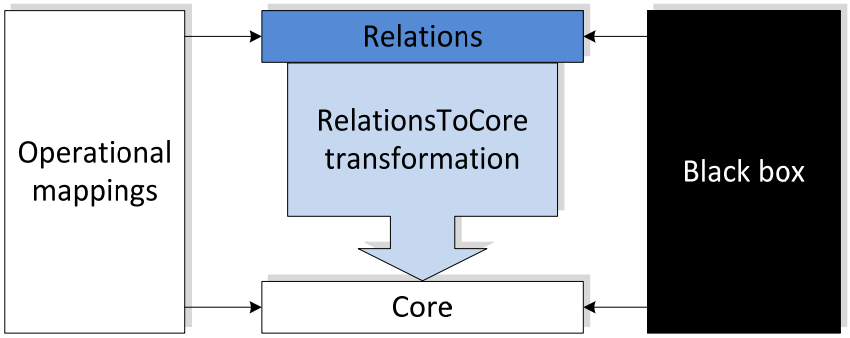


Figura 2.6 Relaciones entre los distintos lenguajes QVT

2.3.3.2 El lenguaje QVT-Relations

En el lenguaje QVT-Relations, una transformación entre dos o más modelos candidatos se especifica como un conjunto de relaciones que deben satisfacerse para que la transformación se lleve a cabo. Un modelo candidato es aquel que se conforme a un metamodelo. Los modelos candidatos tienen un nombre y los elementos que pueden contener, están restringidos a aquellos que se definen en el metamodelo al que son conformes.

Las relaciones se componen de dos o más *dominios* y un par de pre y post condiciones, definidos mediante predicados *when* y *where*.

> *Dominio: variable de tipo que puede ser instanciada en un modelo de un determinado metamodelo. Un dominio contiene un patrón, que puede ser visto como un grafo de nodos del objeto, sus propiedades y enlaces de asociación procedentes de una instancia del tipo del dominio. Alternativamente, un patrón puede ser visto como un conjunto de variables, y un conjunto de restricciones que los elementos del modelo con destino a esas variables deben satisfacer para calificar como un enlace válido del patrón. El dominio, además, estará calificado como checkonly o enforce, calificadores que determinarán el comportamiento de la transformación durante su ejecución.*

Una relación puede estar restringida por dos conjuntos de predicados, especificados en las cláusulas *when* y *where*. La cláusula *when,* describe las condiciones en que la restricción ha de satisfacerse. La cláusula *where*, especifica las condiciones que deben ser satisfechas por todos los elementos del modelo que participan en esa relación, y restringe todas las variables en la relación y sus dominios.

Una transformación contiene dos tipos de relaciones, las relaciones *top-level* y *non-top-level*. La ejecución de una transformación requiere que todas las relaciones *top-level* sean satisfechas, mientras que las relaciones *non-top-level* únicamente han de ser satisfechas cuando se invocan directa o transitivamente desde una cláusula *where* de otra relación.

La dirección de una transformación se define en tiempo de ejecución, seleccionando un determinado modelo candidato como modelo destino. El modelo destino podrá estar vacío o podrá contener elementos del modelo que serán tenidos en cuenta en la transformación. La ejecución de la transformación, primero comprueba si las relaciones se satisfacen y, para aquellas que no lo hacen, trata de que la relación sea satisfecha mediante la creación, modificación o borrado del modelo destino, *forzando* la relación. El que una relación pueda o no ser forzada, viene determinada por el calificador del dominio destino, que podrá estar marcado como *checkonly* o *enforce*. Cuando una transformación se ejecuta en dirección a un dominio marcado como *checkonly*, únicamente se comprobará que existe en el modelo destino un *pattern-matching* unificación válida que satisface la relación. Cuando la transformación se ejecuta en dirección a un dominio marcado como *enforce*, si la comprobación falla, el modelo se modificará para hacer que la relación sea satisfecha.

2.3.4 El estándar CVL para la representación de variabilidad

El lenguaje de representación de variabilidad común CVL (Common Varaibility Languaje) (Object Management Group 2012a) es un estándar de la OMG para la representación de la variabilidad a nivel de modelos, que se encuentra en fase de aprobación. Su propósito es facilitar la representación y resolución de la variabilidad sobre instancias de cualquier modelo o lenguaje definido utilizando un metamodelo basado en MOF. A ese modelo, sobre el que se especifica la variabilidad, pasa a denominarse *modelo base*.

La Figura 2.7 muestra como el lenguaje CVL se combina con otros lenguajes de propósito general para permitir la especificación y resolución de la variabilidad en dichos lenguajes. El modelo de variabilidad CVL especifica la variabilidad a nivel del modelo base y permite, mediante el uso de modelos de resolución definidos igualmente en CVL, obtener de forma automática modelos en el lenguaje del modelo base en los que la variabilidad ha sido resuelta. El proceso de transformación encargado de resolver la variabilidad y producir un modelo resuelto es el *proceso de materialización*.

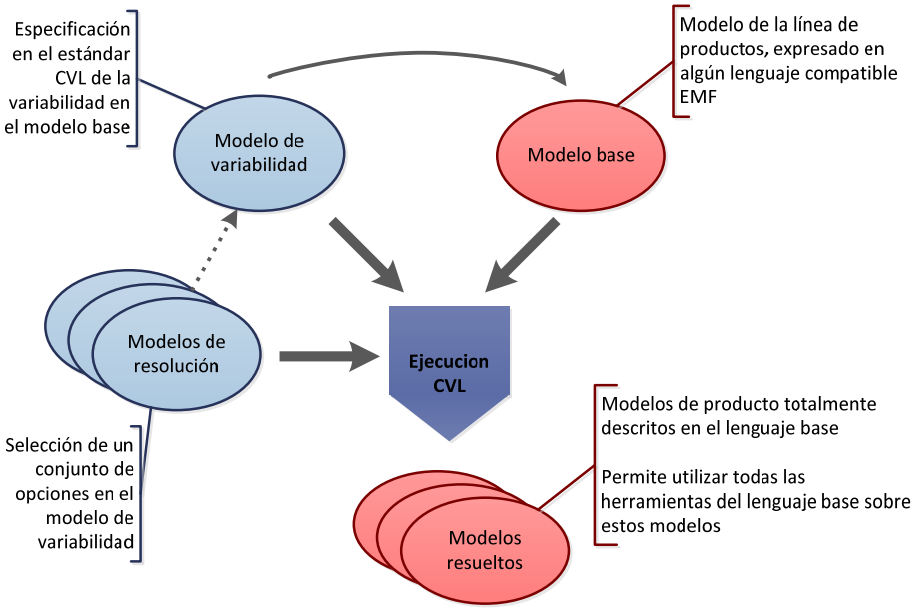


Figura 2.7 Especificación y resolución de la variabilidad mediante CVL

La arquitectura de CVL, que se muestra en la Figura 2.8, se organiza en varias partes:

- *Modelo base:* El modelo base, desde el punto de vista de la arquitectura de CVL, es un conjunto de elementos y enlaces entre ellos, expresado en un lenguaje de modelado distinto de CVL sobre el que se definirá la variabilidad.

- *Abstracción de la variabilidad:* Ofrece los constructos necesarios para especificar la variabilidad de forma abstracta, sin definir las consecuencias que dicha variabilidad tendrá sobre el modelo base. Aísla el componente lógico de CVL de las partes que manipulan el modelo base. La entidad principal de abstracción de la variabilidad es la *especificación de variabilidad* (VSpec), que representa una opción binaria, un parámetro o un elemento de especificación que se puede instanciar varias veces. Las VSpecs se organizan en arboles similares a los modelos de características.

- *Realización de la variabilidad:* Provee de constructos para especificar los puntos de variabilidad sobre el modelo base. Se entiende por punto de variación a una modificación que se aplica al modelo base durante el proceso de materialización. Los puntos de variabilidad son los que impactan directamente sobre el modelo base y se refieren a éste mediante manejadores del modelo base (*base model handles*). Los puntos de variabilidad se asocian a las VSpecs para definir qué noción de variabilidad abstracta se materializa con ese punto de variabilidad. Esas asociaciones forman el enlace entre la abstracción de la variabilidad y su realización, definiendo el efecto que la VSpec tiene sobre el modelo base.

- *Unidades configurables:* Sobre los constructos de abstracción y realización de la variabilidad se definen las unidades configurables. Permiten la especificación de componentes configurables y reusables. Proveen de modularidad soportando la definición composicional jerárquica, reflejando la estructura composicional del modelo base. Las unidades configurables exponen una interfaz de variabilidad, compuesta de VSpecs, que permite su configuración.

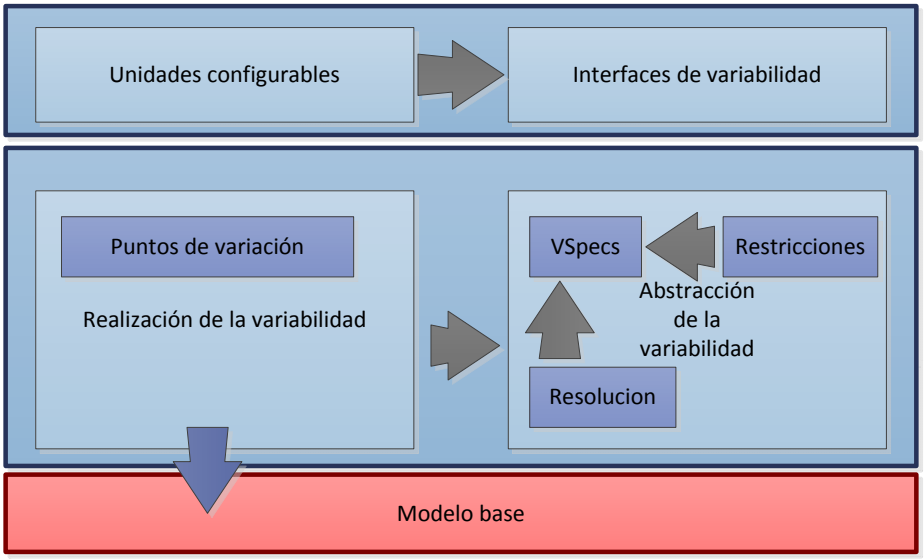


Figura 2.8 Arquitectura CVL

2.3.4.1 Abstracción de la variabilidad

La abstracción de la variabilidad ofrece cuatro tipos de VSpecs: *elección*, *variable*, *clasificador de variabilidad* y *especificación de variabilidad compuesta* cuyos detalles se describen a continuación:

- Una *elección* es una VSpec cuya resolución requiere una decisión si/no y cuyas consecuencias no se conocen a nivel de la abstracción de la variabilidad.
- Una *variable* es una Vspec cuya resolución implica proveer un valor de un tipo específico. Este valor está destinado a ser empleado en el modelo base pero, de la misma manera que en el caso de las elecciones, se desconoce, a este nivel, donde y como se utiliza.
- El *clasificador de variabilidad* (VClassifier) es un tipo de VSpec cuya resolución implica instanciar, proveer y resolver los sub-arboles de VSpecs para cada instancia.
- El último tipo de VSpec es la *especificación de variabilidad* compuesta (CVSpec), que se corresponde a contenedores del modelo base los cuales se tratan como unidades configurables. Permite la especificación de una colección de declaraciones de variabilidad dentro de un

contenedor del modelo base (por ejemplo, un paquete o componente UML), ocultando los detalles y ofreciendo una interfaz de variabilidad para su configuración.

La Figura 2.9 muestra el árbol de VSpecs CVL para un sistema de software empotrado para el control de los sistemas de seguridad de vehículos automóviles (sistema de control de vehículos), el cual va a servir como ejemplo ilustrativo a lo largo de todo el documento[5]. Un sistema de control de vehículos está compuesto de una serie de subsistemas, como pueden ser:

- Sistema de Antibloqueo de Frenos (ABS), que se asegura de que la máxima fuerza de frenado es transferida a cada rueda sin que estas se deslicen sobre el pavimento.
- Sistema de Control de Tracción, que previene el deslizamiento de las ruedas durante la aceleración.
- Sistema de Control de Estabilidad, que mantiene el vehiculo en la dirección de giro fijada por el conductor, previniendo el deslizamiento en los giros.
- Sistema de control de crucero, que mantiene una velocidad constante establecida por el conductor.
- Sensores de aparcamiento, capaces de detectar un obstáculo en la dirección de movimiento del vehículo y avisar, mediante indicadores acústicos y luminosos, a su conductor.

El árbol de VSpecs CVL contiene cinco VSpecs de tipo elección (*Sensor de giro, Actuador de freno, Controlador de ABS, Sensores de distancia y Sensor delantero*), un VClassifier que permite de cuatro a seis instancias del sensor trasero (que pueden ser, o bien sensores de efecto *doppler,* o bien sensores de infrarrojos) y dos VSpecs de tipo variable (el rango en metros del sensor delantero y el ángulo de los sensores traseros).

El significado de la estructura en árbol implica que un sub-árbol debajo de otro nodo representa una VSpec subordinada, en el sentido de que la resolución de un cierto nodo, restringe la resolución de los nodos de su sub-árbol (aunque el propio estándar permite la inclusión de restricciones explicitas).

[5] La descripción completa del sistema de control de vehículos y de los distintos subsistemas que lo integran y que se emplean como ejemplo a lo largo de todo el documento se detallan en el Apéndice A.

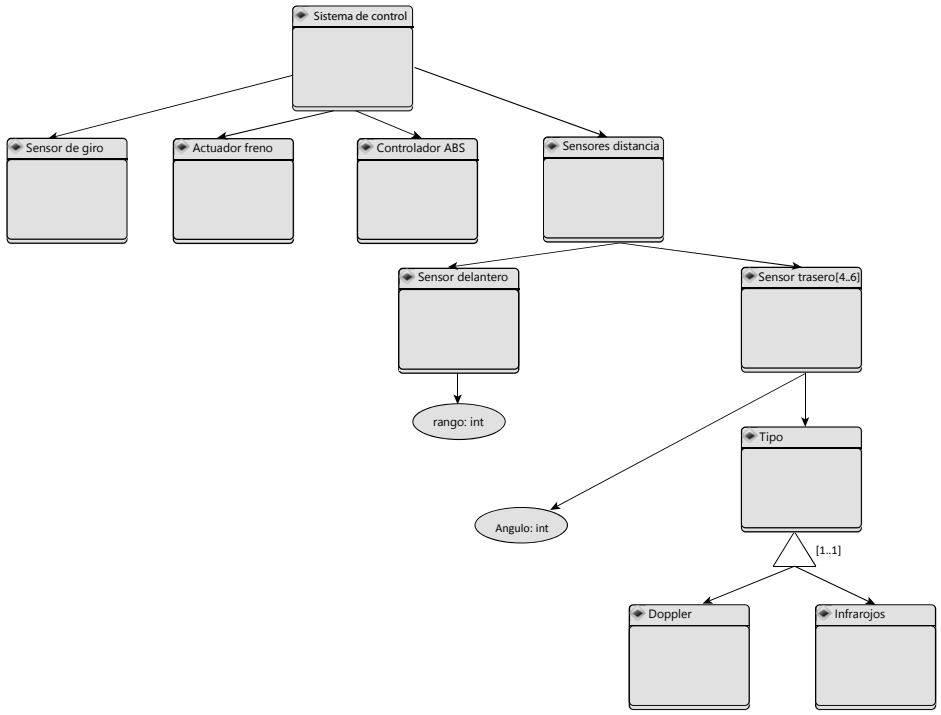


Figura 2.9 Árbol de VSpecs con VSpecs de elección, variables y VClassifiers

Las restricciones aplicables por la estructura del árbol son las siguientes:

- *Implicación de la resolución negativa:* la selección de una resolución negativa (no para una determinada VSpec) no implica seleccionar ni todas sus sub-elecciones ni todas sus VSpecs hijas (por ejemplo en la Figura 2.9, en el caso de resolver negativamente sensores de distancia, ni el sensor delantero ni ninguna instancia de sensor trasero podrán resolverse positivamente).

- *Implicación de la resolución positiva:* cada VSpec de tipo selección tiene un campo "*ImplicadoPorPadre*" que, cuando tiene valor *verdadero* indica que, si el padre es resuelto positivamente o la selección es la raíz, entonces debe ser decidida a verdadero. Por definición las VSpecs que no implican selección se resuelven siempre a verdadero.

- *Multiplicidad de grupo:* una VSpec puede tener una multiplicidad de grupo que especifica cuantas resoluciones positivas puede haber entre sus hijas, en el caso que la VSpec se resuelva positivamente (por ejemplo

en la Figura 2.9, si se seleccionan los sensores traseros, podrá haber como máximo una resolución positiva de las VSpecs bajo *Tipo*).

- *Multiplicidad de instancia:* cada VClassifier puede tener una multiplicidad de instancia que especifica cuantas instancias han de crearse (por ejemplo en la Figura 2.9, en el caso de resolver positivamente sensor trasero, se resolverán positivamente de cuatro a seis instancias configurándose con el sub-árbol bajo dicha VSpec).

2.3.4.2 Realización de la variabilidad

Los mecanismo de realización de la variabilidad hacen posible la materialización de modelos a partir de la descripción CVL mediante la transformación del modelo base en modelos en los que la variabilidad ha sido resuelta. La realización, define puntos de variabilidad que representan las modificaciones del modelo base para conseguir los modelos destino.

Hay cuatro formas básicas mediante las cuales se pueden definir las modificaciones al modelo base asociadas a un punto de variabilidad:

- **Existencia:** Indica que la existencia de un determinado objeto, enlace o valor en el modelo base, está en cuestión.
- **Sustitución:** Indica que un determinado objeto o un fragmento del modelo será sustituido por otro. La sustitución de objeto (*Object substitution*) involucra a dos objetos y significa la redirección de todas las conexiones en las que el primero es origen o destino y entonces el objeto es eliminado y sustituido por el nuevo objeto. La sustitución de extremo de conexión (*Link-end substitution*) implica una conexión y un objeto y significa redirigir el extremo de la conexión al objeto (de ahí la sustitución por el antiguo objeto en ese extremo). La sustitución de fragmentos (*Fragment substitution*) implica identificar un fragmento de modelo (*placement holder)* designando los elementos frontera del fragmento, tal como se muestra en la Figura 2.10(a). De este modo se crea un "agujero" en el modelo que podrá ser rellenado con un fragmento de reemplazo (*replacement holder*), de un tipo compatible, definido del mismo modo (mediante entidades y elementos de frontera), tal como se muestra en la Figura 2.10(b). El resultado de la sustitución se muestra en la Figura 2.10(c).
- **Asignación de valor:** Indica que un valor puede ser asignado a un espacio específico de un objeto del modelo base.

- **Punto de variación opaco:** Indica que el punto de variación se asocia a variabilidad de dominio especifico (definida por el usuario). La variabilidad se especifica empleando lenguajes de transformación de modelos como QVT.

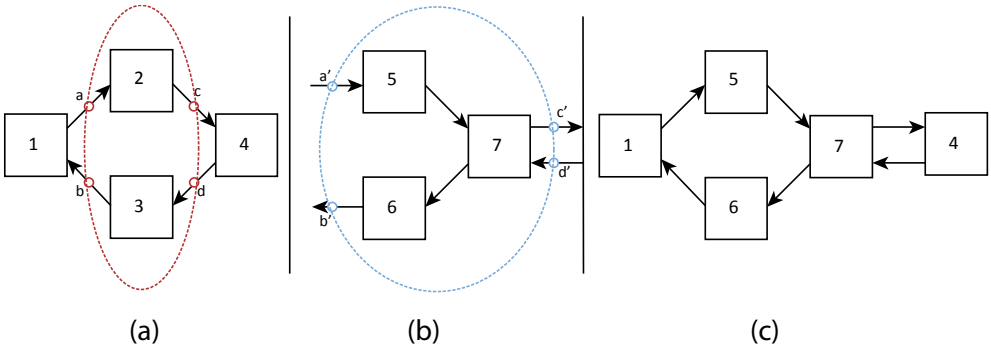


Figura 2.10 Reemplazo de fragmentos con la definición de puntos frontera

Todos los constructos descritos hasta este punto, permiten definir variabilidad mediante arboles de VSpecs y puntos de variación organizados localmente sobre el modelo base, pero todos ellos están asociados al nivel más bajo en la arquitectura CVL que se muestra en la Figura 2.8. CVL también permite agrupar declaraciones de variabilidad CVL correspondientes a contenedores del modelo base mediante unidades configurables. Las unidades configurables exponen un interfaz de variabilidad que permite su configuración. Las unidades configurables proveen además de mecanismos de modularidad y reutilización, puesto que las unidades configurables ofrecen la posibilidad de clonar y configurar varias veces una unidad configurable desde distintas partes de un mismo árbol de CVSpecs.

La Figura 2.11 muestra un modelo CVL definido sobre un modelo base que contiene un paquete AADL (parte inferior). Concretamente, el paquete AADL contiene dos versiones de un control de crucero (*control de crucero simple* y *control de crucero adaptativo)*. La variabilidad expresada supone que ambos sistemas son alternativos, es decir, que en el sistema de control tendremos uno u otro, pero no ambos. Esta estructura encapsula la variabilidad del paquete en una *unidad configurable*, cuya variabilidad se recoge en una CVSpec. De este modo, la variabilidad del paquete está encapsulada en una CVSpec que ofrece la interfaz de configuración del paquete.

Todos los mecanismos de abstracción y realización de la variabilidad, tienen como último objetivo, la materialización de modelos en los que la variabilidad

existente ha sido resuelta mediante un modelo de resolución que contiene un árbol de resoluciones de las distintas VSpecs. El modelo de resolución determina, para cada elección, si ha sido resuelta positiva o negativamente y los valores asociados a cada variable. Con este modelo de resolución se genera, de manera automática, el modelo resuelto. La Figura 2.12 muestra el ciclo completo de materialización CVL. En la parte derecha, el modelo de decisión contiene las elecciones y los valores de las VSpecs de selección y los valores de las VSpecs variables. En dicho modelo, se han resuelto positivamente el sensor de giro, el actuador de freno, el controlador ABS y el sensor de distancia delantero, con un rango máximo de 90 m. Como resultado, se obtiene el modelo resuelto que se muestra en la parte inferior derecha, en el que se han instanciado los objetos del modelo base enlazados con cada una de las VSpecs resueltas positivamente.

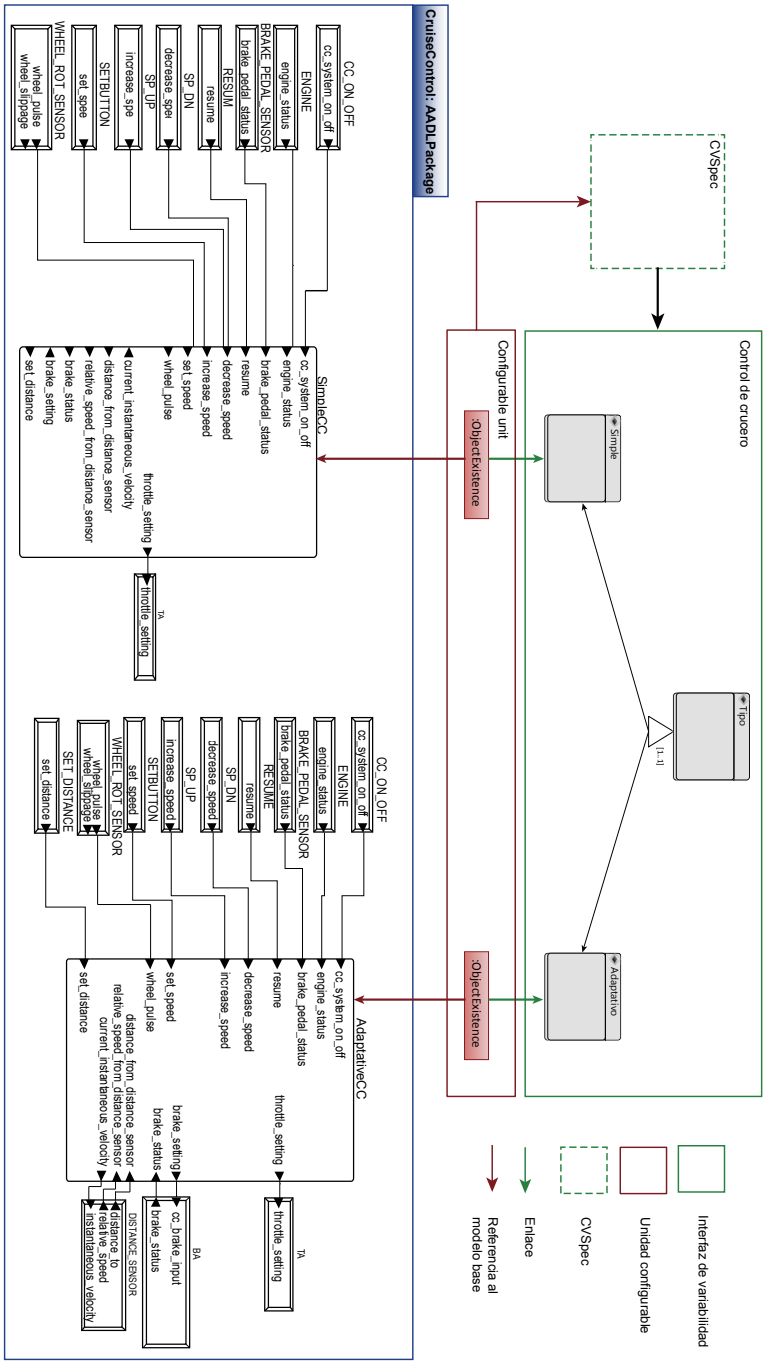


Figura 2.11 Unidad configurable enlazada a un CVSpec

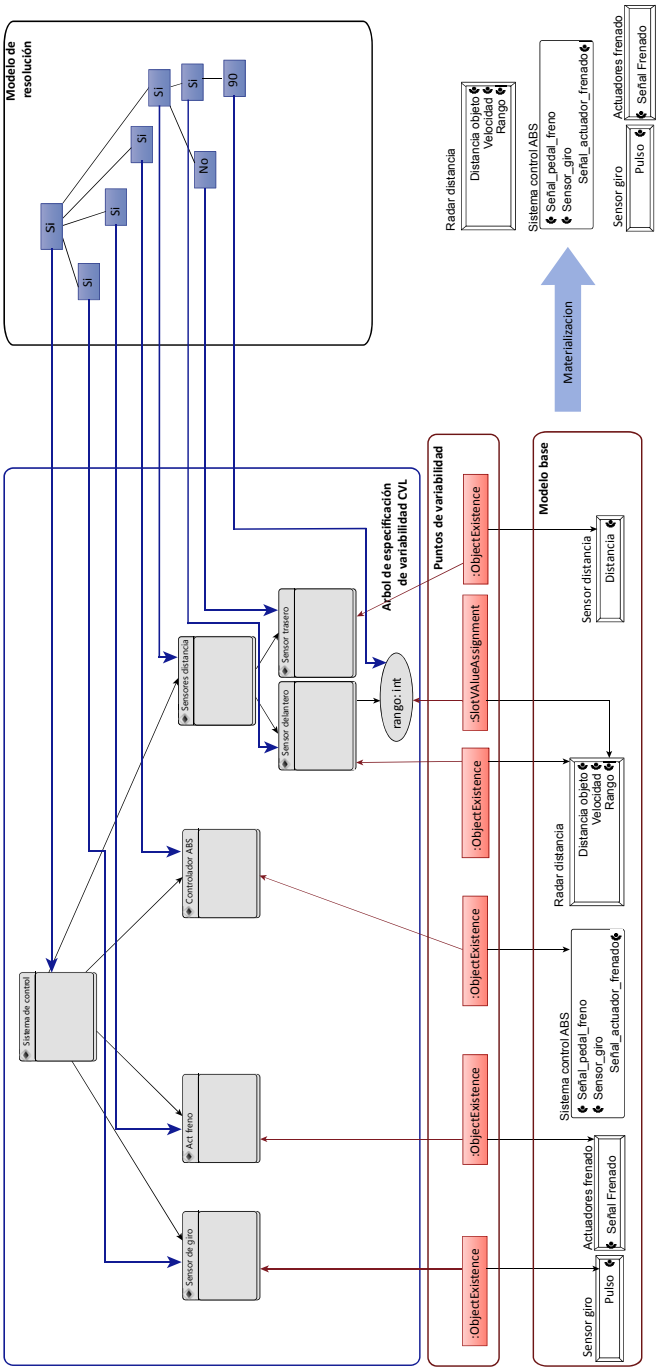


Figura 2.12 Árbol de variabilidad CVL, modelo de resolución y materialización

2.4 Aseguramiento de calidad en líneas de producto software

La aplicación de la aproximación de líneas de producto software tiene un gran impacto en la calidad de los productos resultantes (Clements y Northrop 2001). Cada nueva aplicación está compuesta por un conjunto de activos software maduro y testeado. Esto implica que cabe esperar que la densidad de defectos sea drásticamente menor que en productos que se desarrollan desde cero (van der Linden et al. 2007). Pero el hecho de que el nivel de calidad de los activos software sea mayor no implica que las tareas de aseguramiento de calidad en el desarrollo de líneas de productos tengan menor complejidad. Si en el desarrollo tradicional, conseguir ciertos niveles de calidad es un reto, en el desarrollo de líneas de productos este reto es aún más complejo, fruto de la variabilidad en los requisitos no-funcionales y las diferentes restricciones de calidad (Etxeberria et al. 2008). La importancia del aseguramiento de la calidad en los contextos de reúso, se debe a que un error en un activo reusable, va a propagarse a un gran número de productos.

Las peculiaridades del desarrollo de líneas de producto, hacen que no sea sencillo aplicar métodos de aseguramiento de calidad enfocados al desarrollo de software tradicional (Montagud et al. 2011). Por un lado, debido a la variabilidad de la línea de productos, normalmente no está claro qué selección de características va a conducir a qué propiedades no-funcionales (y si esas propiedades no-funcionales cumplen los requisitos no-funcionales) (Siegmund et al. 2011). Además esto se ve agravado por la no predictibilidad de algunos atributos de calidad en base a los atributos de sus componentes, tal como ocurre en el desarrollo de software basado en componentes (Crnkovic et al. 2004). Esto hace que, en ocasiones, sea difícil predecir si el conjunto de requisitos no-funcionales seleccionados va a poder cumplirse (por ejemplo, si se seleccionan muy altos niveles de seguridad es posible que alcanzar estos niveles de seguridad vaya en detrimento del rendimiento).

Las técnicas de aseguramiento de calidad, deberán ser aplicadas tanto en la ingeniería de dominio (para poder asegurar que el conjunto de activos software cumple con los requisitos de calidad), como en la ingeniería de aplicación (donde deberá evaluarse y controlarse la calidad de los productos que se desarrollen a partir de ese conjunto de activos). Para el aseguramiento de la calidad, se podrán emplear inspecciones, revisiones, testeo y evaluaciones,

dependiendo de la fase en la que se lleve a cabo y los atributos de calidad a evaluar.

2.4.1 Atributos de calidad y líneas de productos

En el desarrollo de líneas de producto, los distintos productos de la línea de productos van a requerir diferentes niveles de atributos de calidad, o la calidad puede ser opcional para determinados productos (si no es importante en todos los productos (Etxeberria et al. 2008). La variabilidad en atributos de calidad puede resumirse en los siguientes tipos (Niemelä 2005):

- *Atributos de calidad opcionales:* para un determinado producto un atributo de calidad puede ser crítico mientras que para otros puede no ser requerido.
- *Niveles de atributos de calidad:* los niveles requeridos para los distintos atributos de calidad puede variar en función de los productos, así como la prioridad de alcanzar los mismos.
- *Impacto de la variabilidad funcional sobre los atributos de calidad (variación indirecta)*: la variabilidad funcional puede causar indirectamente variación en los niveles de los atributos de calidad.

Por otro lado, existen atributos de calidad clave en este método de producción como son la variabilidad, la reusabilidad, la composicionalidad o la flexibilidad (Montagud et al. 2011) y que se añaden a los atributos de calidad de producto. La mayoría de estos atributos de calidad, podrían ser calificados como atributos de desarrollo si atendemos a la clasificación propuesta por Bosch (2000) o no observables si atendemos a la clasificación propuesta por Dolan (2001). Etxeberria y Sagardui (2005) proponen una clasificación como atributos propios de la línea de productos y atributos de calidad del dominio.

> *Atributos de calidad de la línea de productos, son aquellos inherentes o específicos de las líneas de productos que permiten que la arquitectura de la línea de productos sea base para un conjunto de productos relacionados, así como para futuros productos. Estos atributos están relacionados con la flexibilidad o la modificabilidad.*

> *Atributos de calidad del dominio, son aquellos observables vía ejecución que afectan al producto, como puedan ser el rendimiento, la fiabilidad o la seguridad.*

Los atributos de calidad propios de la línea de productos, serán evaluados bien como parte de los procesos de la ingeniería de dominio o bien en tiempo de evolución de la línea de productos. Los atributos de calidad de la línea de productos van a ser medidos a nivel de la arquitectura de la línea de productos, a nivel de activo software y a nivel del conjunto de activos software como tal. Sobre la arquitectura de la línea de productos, se podrán medir atributos como la variabilidad total de la arquitectura de línea de productos (Alves et al. 2008) o su modularidad (Rahman 2004). Sobre un activo, se pueden considerar atributos como la reusabilidad (número de veces que un determinado activo ha formado parte de productos) o su capacidad de configuración (Her et al. 2007). Por último, sobre el conjunto de activos como tal, podemos medir atributos como la tasa de reúso de componentes (que mide el ratio de utilización de los activos en los distintos productos) (Zhang et al. 2008).

2.4.2 Evaluación de calidad en líneas de producto software

En el desarrollo de software convencional, las evaluaciones de calidad se realizan durante el diseño, donde puede evaluarse la arquitectura para comprobar en qué grado soportará los requisitos de calidad sobre los artefactos (una vez éstos han sido obtenidos) y sobre el producto final.

En el desarrollo de líneas de productos, va a ser necesario llevar a cabo esas evaluaciones en distintas fases del proceso de desarrollo. Las evaluaciones de la arquitectura, tendrán que ser desdobladas para las fases de ingeniería de dominio e ingeniería de aplicación.

En la ingeniería de dominio, se llevaran a cabo *evaluaciones de la arquitectura de la línea de productos* para asegurar que ésta cumple los atributos de calidad del dominio, así como que presenta los atributos de calidad de línea de productos esperados (por ejemplo, si es lo suficientemente flexible para dar soporte a los productos que integran la línea de productos).

En la ingeniería de aplicación, se llevaran a cabo *evaluaciones de la arquitectura de producto* para asegurar los requisitos no-funcionales del producto en desarrollo. Esta evaluación se efectúa en tiempo de configuración (si se trata de comparar posibles configuraciones candidatas o alternativas de diseño), o en tiempo de derivación (si lo que se quiere es comprobar si la arquitectura obtenida cumple o no con los requisitos). En esta evaluación se analizan únicamente atributos de calidad del dominio.

Será de nuevo necesario realizar *evaluaciones de la arquitectura de la línea de productos en fase de evolución* (Etxeberria y Sagardui 2005), después de modificar el alcance de la línea de productos o los requisitos o propiedades no-funcionales de los mismos. En esta evaluación, de nuevo, será necesario analizar los atributos de calidad del dominio y los de línea de producto.

Por último, en caso de modificaciones de las arquitecturas, se llevaran a cabo *evaluaciones en fase de sincronización* (Etxeberria y Sagardui 2005) para analizar como las modificaciones o actualizaciones de la arquitectura de la línea de productos pueden afectar a los productos ya en el mercado y si hay que realizar modificaciones en los mismos. En este caso, los atributos de calidad a analizar serán los atributos de calidad del dominio y de línea de producto, analizando si alguno de los cambios afecta a estos atributos y hace que la arquitectura pueda no adaptarse a alguno de los productos en el mercado.

2.5 Resumen

En este capítulo hemos proporcionado una breve introducción a los principales conceptos y al marco tecnológico sobre los que se asienta esta tesis: las líneas de producto software, las arquitecturas software, el desarrollo dirigido por modelos y el aseguramiento de calidad en líneas de producto software.

Por un lado, se ha presentado la aproximación de líneas de producto software para el desarrollo de familias de sistemas que comparten una serie de características comunes. Se han introducido la ingeniería de líneas de producto como aproximación para soportar su desarrollo, el concepto de variabilidad y las notaciones de modelado de variabilidad más extendidas.

Por otro lado, se ha introducido el concepto de arquitectura software, nociones básicas de su diseño y su descomposición en vistas. Se han descrito las diferentes notaciones y lenguajes de descripción arquitectónica. Se ha introducido la vinculación existente entre arquitecturas software y la calidad del software, y el papel de las arquitecturas software en el desarrollo de líneas de producto software.

Además, se han introducido las bases del paradigma de desarrollo dirigido por modelos, los estándares propuestos por la OMG para dar soporte al ciclo de desarrollo, las transformaciones de modelos con las que se deriva el producto a partir de la especificación del sistema en modelos en un alto nivel de abstracción, los distintos lenguajes de transformación definidos dentro del estándar QVT propuesto por la OMG para soportar las distintas

trasformaciones que tendrán lugar a lo largo del ciclo de desarrollo del producto, haciendo especial énfasis en el lenguaje QVT-Relations (que es el lenguaje de transformación en el que se han definido las distintas trasformaciones de modelos que integran el proceso QuaDAI) y por último el estándar CVL propuesto por la OMG que facilita la definición y resolución de la variabilidad sobre modelos.

Por último, se ha introducido la problemática asociada al aseguramiento de calidad en el desarrollo de líneas de producto software haciendo énfasis en los atributos de calidad propios de las líneas de producto, así como de las evaluaciones de calidad a lo largo del ciclo de desarrollo.

Con todo ello, se ha formado el marco conceptual sobre el que se sustenta esta tesis y que permite entender con mayor detalle, el problema que se pretende resolver.

Capítulo 3

Estado del Arte

En este capítulo se va a analizar los métodos, técnicas y enfoques existentes para la derivación, evaluación y mejora de arquitecturas de producto y propuestas para el aseguramiento de calidad, que componen el cuerpo de conocimiento en el que se engloba el método propuesto.

En la sección 3.1 se analizan los métodos de derivación de arquitecturas de producto, a partir de la arquitectura de la línea de productos software, en entorno de desarrollo de líneas de producto software siguiendo el paradigma de desarrollo dirigido por modelos.

La sección 3.2 analiza los métodos de evaluación arquitectónica para líneas de producto haciendo énfasis en aquellos que pueden ser aplicados para evaluar la arquitectura del producto en tiempo de derivación.

La sección 3.3 describe las técnicas de aseguramiento de calidad en entornos de desarrollo de líneas de producto software, con el fin de ofrecer una visión de conjunto sobre la problemática específica del aseguramiento de la calidad en dichos entornos.

La sección 3.4 analiza los estudios empíricos llevados a cabo en el área de las arquitecturas software.

3.1 Métodos de derivación en el desarrollo de LPS

En comparación con la gran cantidad de investigación que se ha producido en el desarrollo de líneas de productos, relativamente pocos trabajos se han centrado a la utilización de las LPS para la derivación de productos individuales, es decir, el proceso de derivación de productos (Rabiser et al. 2011). Uno de los aspectos cruciales de ese proceso es la derivación de la arquitectura del producto.

En esta sección se van a analizar las propuestas de métodos, lenguajes y herramientas de derivación en general y de arquitecturas de producto en particular, en entornos de desarrollo de líneas de producto software que siguen el paradigma de desarrollo dirigido por modelos definidas en los últimos años. En las siguientes subsecciones se van a analizar los métodos de derivación de arquitecturas de producto, otros métodos y técnicas de derivación en LPS y los lenguajes y herramientas específicamente definidos para gestionar y automatizar el proceso de derivación.

3.1.1 Derivación de la arquitectura de producto

A pesar de que en la aproximación de líneas de producto software, la arquitectura software es uno de los artefactos clave (Clements y Northrop 2001) en la literatura se encuentran relativamente pocos trabajos centrados en definir métodos para la derivación de esta partiendo de la arquitectura de la línea de productos, siendo algunos ejemplos relevantes el método Kobra (Atkinson et al. 2000), Koalish (Asikainen et al. 2003), el framework BOM (Cabello 2008) o los métodos propuestos por Botterweck et al. (2009), Perovich et al. (2009), Duran-Limon et al. (2011) y Guana y Correal (2013). Los detalles de estos métodos se describen a continuación:

Kobra (Atkinson et al. 2000) es una aproximación sistemática para el desarrollo de frameworks para el desarrollo de líneas de producto software basado en componentes que permite la derivación de la arquitectura del producto como un paso intermedio del ciclo de desarrollo. En Kobra se distingue entre i) la ingeniería de framework, en la que se desarrolla el conjunto de componentes que darán soporte al desarrollo de la familia de productos, incluyendo la gestión de variabilidad y ii) la ingeniería de aplicación donde se deriva un producto específico. Los componentes Kobra se describen mediante tres modelos: el modelo estructural, el modelo de comportamiento y el modelo de decisión en el que se describen las distintas variantes de un componente.

Kobra define el modelado de los componentes empleando distintos modelos, la fase de derivación está definida en la aproximación, pero no hay soporte automatizado para la misma, los distintos modelos sirven únicamente para documentar las decisiones de diseño que se habrán de adoptar durante la derivación. No hay un soporte explícito a la configuración, se establecen las tareas a realizar, pero no se definen los artefactos que intervienen en ellas. No hay un tratamiento explícito a los requisitos no-funcionales, sino que estos se engloban, de forma genérica, en los requisitos del producto.

Koalish (Asikainen et al. 2003) describe la integración del modelado de variabilidad con el lenguaje de descripción de arquitecturas software para líneas de producto Koala (Ommering et al. 2000) en un metamodelo combinado con una semántica definida formalmente. Koalish añade nuevos mecanismos de variabilidad a Koala que solo cuenta en su definición con la parametrización y la compilación condicional. Koalish agrega la posibilidad de seleccionar el número y tipo de las partes de un componente a desplegar en la arquitectura. Los modelos arquitectónicos y de variabilidad son trasladados a un razonador que permite llevar a cabo la tarea de configuración, atendiendo únicamente a criterios funcionales. La salida del configurador es la descripción en Koala del producto.

Cabello (2008) define un framework (BOM), con dos implementaciones distintas (*BOM-Eager* y *BOM-Lazy)* para la derivación, mediante transformaciones QVT-Relations, de arquitecturas de producto expresadas en PRISMA (Perez 2006). A partir de un modelo de variabilidad que expresa las características del dominio se obtiene un modelo conceptual expresado como diagramas de clase UML con restricciones OCL. Con este modelo conceptual, junto con el modelo de funcionalidad del sistema se obtiene el modelo de la arquitectura de la línea de productos. Esta arquitectura de la línea de productos junto con las instancias del modelo conceptual del dominio de aplicación permite obtener la arquitectura decorada con las características del dominio de aplicación expresadas como modelos PRISMA. Las arquitecturas expresadas mediante el lenguaje PRISMA pueden ser posteriormente convertidas en código ejecutable mediante el compilador de modelos de PRISMA. En el desarrollo de las transformaciones se aplican patrones de buen diseño, tal como se describe en (Gómez 2012) para asegurar criterios de calidad en las transformaciones, pero estos patrones van encaminados a la calidad interna de los modelos generados y no a satisfacer a los requisitos de calidad del producto final.

Botterweck et al. (2009) proponen la automatización de la derivación de arquitecturas software mediante transformación de modelos en las que se explota la relación entre el modelo de características y componentes arquitectónicos formalizada por Janota y Botterweck (2008). En este caso, a pesar de que la relación permite la optimización de atributos expresados como valores numéricos, no se da soporte explícito a la derivación de la arquitectura teniendo en cuenta atributos de calidad. La variabilidad arquitectónica no se representa explícitamente sino que se establecen correspondencias entre una característica y su implementación mediante un lenguaje de modelado de la implementación (relación uno a muchos), sin permitir implementaciones alternativas.

Perovich et al. (2009) proponen una automatización de la derivación de la arquitectura del producto a partir de un modelo de configuración que representa las características seleccionadas para un producto. La premisa de partida es que la selección de una característica seleccionada en un modelo de configuración inspira un conjunto de decisiones arquitectónicas que guía la construcción de parte de la arquitectura del producto que incluye esa característica. Las decisiones se hacen de manera local a cada característica considerando únicamente su subárbol de características. Así, las decisiones arquitectónicas que tienen relación con atributos de calidad van ligadas a las características cercanas a la raíz del modelo de características mientras que las decisiones funcionales van ligadas a las características cercanas de las hojas. Esto implica que, una determinada característica va asociada a un requisito no-funcional y viceversa. Esto puede no ser cierto si se permiten implementaciones alternativas de una misma característica con distintas propiedades no-funcionales. En esta propuesta, la arquitectura de la línea de productos no existe como tal, sino un conjunto de transformaciones de modelos que, en base a la selección de características construirán la arquitectura del producto representada únicamente mediante la vista de componente-conector. Esto lleva asociado asumir que la variabilidad externa y la variabilidad de la línea de productos están relacionadas unívocamente, y que la configuración va a ser la ejecución monotónica de transformaciones para generar únicamente la vista componente y conector de la arquitectura, que representa únicamente un aspecto de la arquitectura de producto.

Duran-Limon et al. (2011) proponen una aproximación en la que la arquitectura del producto es derivada a partir de la arquitectura de la línea de productos mediante transformación de modelos. La variabilidad arquitectónica y el árbol de variabilidad se modelan mediante ontologías OWL (Horridge et al. 2004) y las reglas de transformación se seleccionan y componen mediante

consultas a la ontología. La salida de las consultas a la ontología es la entrada a una transformación de modelos ATL que genera la arquitectura de producto. La aproximación no tiene en cuenta los requisitos no-funcionales en el proceso de derivación y presupone que la configuración va a llevar aparejada la ejecución de transformaciones que agregarán los distintos componentes sobre el modelo arquitectónico.

Guana y Correal (2013) proponen una aproximación para obtener las configuraciones software (arquitectura y componentes) que satisface una serie de requisitos de calidad expresados mediante escenarios. La aproximación genera un modelo de conciliación a partir del modelo de activos software de la línea de productos y del modelo que contiene los puntos sensibles de la arquitectura. Posteriormente este modelo será analizado para seleccionar el conjunto más apropiado de componentes que implementan los componentes abstractos de la arquitectura con el fin de que el producto cumpla con los requisitos no-funcionales. La aproximación únicamente considera el impacto de diferentes implementaciones de un mismo componente abstracto de la arquitectura sobre los atributos de calidad. La variabilidad arquitectónica no se modela explícitamente, se establecen correspondencias entre el modelo de variabilidad y los componentes que realizan esa variabilidad externa. Es posible establecer relaciones entre n característica y n componentes. La propuesta solo permite desplegar o no un determinado conjunto de componentes, pero esto puede no ser suficiente para cubrir la variabilidad externa de la línea de productos (Bosch 2000).

3.1.2 Otras técnicas de derivación

En la literatura también podemos encontrar otras técnicas de derivación más centradas en modelos más generales, que representan de una u otra forma el producto, siendo algunos de los más relevantes la aproximación de Czarnecki y Antkiewicz (2005), la de Ziadi y Jézéquel (2006), el método PLUS-EE (Gomaa y Shin 2007), la aproximación de Perrouin et al. (2008), el framework de Schaefer et al. (2009) o la aproximación de Tawhid y Petriu (2011b). Los detalles de estas aproximaciones se describen a continuación:

Czarnecki y Antkiewicz (2005) presenta una aproximación para la derivación de modelos del producto en desarrollo a partir de la definición de plantillas de modelado, que constan de condiciones y meta-expresiones definidas en términos de las características y sus atributos y que se evalúan con respecto a las características seleccionadas en una determinada configuración. Para una

determinada configuración en función del resultado de dichas condiciones se decide que elementos permanecen en el modelo (si se evalúan a cierto) o se eliminan del modelo final (si se evalúan a falso). La resolución de las condiciones y meta-expresiones da lugar al modelo simplificado del producto, en la que se han eliminado los elementos del modelo que no forman parte de la configuración, aplicando una técnica de variabilidad negativa (Sánchez et al. 2007). En la propuesta no hay soporte a los requisitos no-funcionales ni a la variabilidad de los mismos.

Ziadi y Jézéquel (2006) presentan una aproximación con la que obtener el diagrama de clases y diagrama de secuencia del producto en desarrollo mediante transformaciones de modelos llevadas a cabo a partir de un lenguaje de transformación de modelos denominado MTL. Proponen un perfil UML para anotar los diagramas UML con información de variabilidad, definen las restricciones de la línea de productos (dependencia, exclusión mutua etc.) como restricciones OCL, la descripción de la configuración se lleva mediante modelos de decisión la aplicación del patrón *factoría abstracta* (Gamma et al. 1995) con el que pueden definir un interface con el que gestionar los puntos de variación. No hay soporte para expresar requisitos no-funcionales ni su variabilidad. La derivación no genera la arquitectura como tal sino una serie de modelos que representan algunos aspectos del producto en desarrollo.

PLUS-EE (Gomaa y Shin 2007) automatiza la derivación del producto por adaptación de la arquitectura de la línea de productos descrita mediante varios modelos UML interrelacionados. Para ello aplican el método PLUS (Gomaa 2004), y a partir de un conjunto de características seleccionadas sobre un modelo de características, resuelven las dependencias definidas entre características y clases. El sistema de transformación KBRET se ha desarrollado sobre la plataforma Rose RT. La aproximación únicamente ofrece soporte a los requisitos funcionales.

Perrouin et al. (2008) define una aproximación para la composición del producto en la que se integran el modelado de características y el modelo de los activos software de la línea de productos. La aproximación permite relacionar las características con los modelos UML de los activos. En base a la selección de características sobre el modelo de variabilidad, se lleva a cabo una pre-configuración del producto mediante la composición de los activos software, que da como resultado un modelo fusionado del producto en desarrollo. Posteriormente se lleva a cabo un segundo paso que consiste en personalizar el modelo para dar soporte a los requisitos funcionales específicos del producto, actividad que se lleva a cabo mediante transformaciones de modelos mediante

pre y post-directivas que se aplican al modelo fusionado. Además permite evaluar restricciones OCL definidas a nivel de los activos software con el fin de comprobar la validez de las modificaciones introducidas sobre el modelo fusionado. Esta aproximación flexibiliza el proceso de transformación permitiendo obtener productos más allá del alcance de la línea de productos, sin embargo la aproximación se limita únicamente a los atributos funcionales.

Schaefer et al. (2009) presenta un framework con el que dar soporte a la derivación de productos que añade una capa de diseño basada en modelos que enlaza la variabilidad de la línea de producto, descrita usando modelos de variabilidad, con la capa de implementación subyacente. La capa de diseño describe a grandes rasgos la arquitectura del producto derivado mediante diagramas UML extendidos. En la capa de diseño se resuelve la variabilidad mediante implementaciones núcleo y Δ-diseños, los cuales definen las modificaciones a los diseños núcleo para incorporar características específicas de producto. Las implementaciones núcleo y los Δ-diseños se representan haciendo uso del modelo de componentes orientado a objetos CoBox. Los CoBoxes son transformados en código mediante una transformación en dos pasos desarrollada en XVCL. La configuración y posterior derivación del producto se lleva a cabo teniendo en cuenta criterios estrictamente funcionales, sin tener en cuenta aspectos no-funcionales ni la variabilidad de estos últimos.

Tawhid y Petriu (2011b) definen una aproximación con la que obtener el modelo UML del producto por transformación de modelos. El modelo de la línea de productos se describe mediante un modelo de diseño con múltiples vistas en UML. Los requisitos funcionales se modelan como casos de uso, las dependencias y restricciones entre características se mapean mediante un diagramas de clase UML estereotipado. La variabilidad en los modelos UML se representa anotando explícitamente la información de variabilidad en los elementos del modelo. Para evitar colapsar los modelos con información de variabilidad, los autores proponen una anotación restringida, anotando únicamente la información de variabilidad estrictamente necesaria. La derivación se lleva a cabo mediante una transformación ATL que toma como entrada el modelo de la línea de productos y genera el modelo del producto (este sin anotaciones de variabilidad). Esta transformación tiene en cuenta la información de variabilidad implícita, es decir, si por ejemplo se incluye un caso de uso que tiene un punto de extensión, una relación *extend* o *include* el caso de uso asociado es automáticamente incluido sin necesidad de que hubiese una anotación en el modelo de la línea de productos. La aproximación solo tiene en cuenta los requisitos funcionales al llevar a cabo la derivación.

Además, hay diferentes propuestas que dan soporte teórico al proceso de derivación del producto en entornos de desarrollo de líneas de producto software como Pro-PD (O'Leary et al. 2012), Doppler (Rabiser y Dhungana 2007), PuLSE-I (Bayer et al. 1999; Bayer y Gacek 2000) o COVAMOF (Sinnema et al. 2006) o la propuesta de Deelstra et al. (2005).

3.1.3 Herramientas y lenguajes para la automatización de la derivación

Por último, se han definido en los últimos años algunas propuestas de lenguajes de modelado genéricos que permiten representar la variabilidad en modelos de cualquier naturaleza, como el propuesto por Sánchez et al. (2008), FeatureMapper (Heidenreich et al. 2008) o la herramienta de soporte a CVL propuesta por Haugen et al. (2010).

Sánchez et al. (2008) propone la definición de lenguajes para la especificación de los procesos de derivación de los modelos arquitectónicos del producto (los modelos que describen las distintas vistas arquitectónicas como la de componente conector). La aproximación se basa en la definición de relaciones entre el modelo de variabilidad y el diseño de la variabilidad (relaciones entre la variabilidad en el espacio del problema y la del espacio de la solución). Basándose en esas relaciones se construye el lenguaje a partir de una gramática EBNF, de la implementación de cada operador del lenguaje y de las plantillas con las transformaciones que implementan el proceso de derivación. Con el lenguaje generado se especifica la LPS como instancia del lenguaje definido que será compilado para poder ejecutar las derivaciones a partir de una configuración válida expresada en el lenguaje.

FeatureMapper (Heidenreich et al. 2008) ofrece soporte automatizado para la derivación de productos mediante la definición de correspondencias entre las características en el espacio del problema y los elementos de los modelos que realizan esas características en el espacio de la solución. La configuración que contiene las características seleccionadas se combina con el modelo de correspondencia y es interpretado por el componente de transformaciones que se encarga de derivar los modelos fusionados del producto. Se aplica una técnica de variabilidad negativa para llevar a cabo la derivación del producto.

Haugen et al. (2010) proponen una herramienta que implementa algunos de los mecanismos de variabilidad propuestos en el lenguaje estándar de variabilidad arquitectónica CVL (Object Management Group 2012a). La herramienta permite especificar variabilidad sobre *modelos base* (véase Sección 2.3.4) compatibles con MOF y derivar automáticamente el modelo de *producto* (véase

Sección 2.3.4) una vez resuelta al variabilidad. La herramienta permite la validación de consistencia de los modelos resueltos mediante un lenguaje de especificación de restricciones BCL (Basic Constraint Language) basado en OCL. La herramienta flexibiliza enormemente el proceso de derivación, ahora bien, aunque nada impide representar las características en un modelo CVL (Object Management Group 2012a), CVL no está inicialmente pensado para representar la variabilidad externa de la línea de productos por lo que se hace necesario proveer algún mecanismo de enlace con los modelos de variabilidad que representen la variabilidad externa.

A pesar de que estas aproximaciones ofrecen gran flexibilidad para automatizar la derivación de arquitecturas, no hay soporte explícito a la representación de los requisitos no-funcionales ni a su variabilidad. Estas aproximaciones requieren de artefactos externos para representar los aspectos no-funcionales y que estos intervengan en el proceso de configuración.

3.1.4 Discusión

A pesar de la importancia del proceso de derivación de productos, no existe todavía un consenso sobre cuáles son los requisitos/actividades criticas de éstos procesos. Para dar respuesta a esa carencia, Rabiser et al. (2010) han realizado una revisión sistemática con el objetivo de identificar los requisitos cruciales que los procesos de derivación de productos deben dar soporte y los ha validado mediante una encuesta con profesionales de la industria.

A continuación comparamos, siguiendo los criterios que se describen en la Tabla 3.1, las propuestas anteriormente descritas. Los criterios C5, C6, C11 y C12 han sido seleccionados directamente de la revisión sistemática de Rabiser et al. (2010). Los criterios C1 y C2 han sido propuestos a partir del requisito genérico "*soporte a la gestión de requisitos de aplicación*" y los criterios C3 y C4 han sido propuestos a partir del requisito genérico "*Resolución automática e interactiva de la variabilidad (e.j., soporte para la selección de características, soporte para la inclusión automática y personalización de activos software)*", propuestos ambos en la revisión sistemática y, finalmente, los demás criterios han sido introducidos para comparar los métodos existentes con nuestra propuesta.

De los resultados de dicha comparación, que se resumen en la Tabla 3.2, podemos extraer las siguientes conclusiones:

- La mayoría de las propuestas no tienen en cuenta atributos de calidad durante el proceso de derivación y aquellas que lo hacen no están

orientadas a describir cómo afectan la selección de características o partes de la arquitectura a los requisitos no-funcionales y atributos de calidad (Criterios 1 y 2).

- La mayoría de las propuestas dan soporte el proceso de configuración (Criterio 3), y automatizan el proceso de derivación de la arquitectura (Criterio 4).

- La mayoría de propuestas soportan la extensibilidad y la definición de nuevos puntos de variabilidad a medida que el alcance de la línea de productos evoluciona (Criterio 5). Las aproximaciones basadas en reglas de transformación, en las que no hay una representación explicita de la arquitectura y/o de la variabilidad se considera que ofrecen un soporte más pobre frente a la evolución del alcance (requerirán del desarrollo y o modificación de las transformaciones existentes).

- La visualización de variabilidad flexible sigue siendo una de las grandes carencias del proceso de derivación (Criterio 6). Casi todas las propuestas muestran ejemplos sencillos y no consideran el problema de la gestión y visualización de grandes modelos de características que deberán ser gestionados durante el proceso de derivación.

- La mayoría de las propuestas optan por representar explícitamente la variabilidad arquitectónica, directamente sobre los modelos (como en los casos de Ziadi y Jézéquel (2006) o Tawhid y Petriu (2011b), o a través de relaciones (directas) sobre el modelo de variabilidad como en el resto de propuestas (Criterio 7).

- Salvo en los casos de FeatureMapper (Heidenreich et al. 2008), la propuesta de Sánchez et al. (2008) o la implementación del lenguaje CVL presentado por (Haugen et al. 2010), no se trata de propuestas genéricas, que permitan definir la variabilidad sobre cualquier vista arquitectónica/lenguaje de descripción arquitectónica (Criterios 8 y 9).

- Muy pocas propuestas permiten validar la consistencia (Criterio 10) de la configuración y/o de los modelos arquitectónicos derivados.

- La mayoría de propuestas no detallan las actividades a llevar a cabo por el usuario final, a la hora de aplicar el método (Criterio 11). En el

método Kobra, la propuesta de Perrouin et al. (2008), la de Ziadi y Jézéquel (2006) y la de Sánchez et al. (2008) se describen, cuando menos parcialmente, el proceso de usuario, aunque en las dos primeras no se ofrecen guías de cómo aplicar dicho proceso en la práctica.

- Ninguna de las propuestas anteriormente descritas ofrece soporte a la gestión del proyecto (Criterio 12), con la definición de que tareas ha de llevar a cabo cada rol, empleando que artefactos etc.

Tabla 3.1 Criterios para comparación de los métodos de derivación

Criterio	*Descripción*
C1	Soporte a requisitos de calidad (RNFs)
C2	Representación explicita de RNFs/Atributos y relaciones con la variabilidad o los componentes arquitectónicos
C3	Soporte a la configuración
C4	Soporte automatizado de la derivación
C5	Adaptabilidad y extensibilidad (e.j., soporte al metamodelado, puntos de extensión para la integración de generadores específicos de dominio)
C6	Visualizaciones de variabilidad flexibles y específicas del usuario (soporte para filtrado, clasificación, ordenación de variabilidad en función de roles, usuarios, tareas, etc.)
C7	Representación explicita de la variabilidad arquitectónica
C8	Propuesta específica para una vista arquitectónica (+ para propuestas independientes de la vista)
C9	Específico para un LDA / Modelo (+ para propuestas independientes del lenguaje/modelo)
C10	Validación de consistencia de la configuración
C11	Guías para el usuario final
C12	Soporte a la gestión de proyecto (soporte a la gestión de tareas, roles, usuarios, etc.)

Tabla 3.2 Comparación entre los métodos de derivación analizados

	C1	*C2*	*C3*	*C4*	*C5*	*C6*	*C7*	*C8*	*C9*	*C10*	*C11*	*C12*
Kobra (Atkinson et al. 2000)	-	-	-	-	+	-	-	C&C	Propio	-	Parcialmente	-
Koalish (Asikainen et al. 2003)	-	-	+	+	+	-	+	C&C	Propio	-	-	-
Cabello (2008)	-	-	+	+	+	-	+	+	PRISMA	+	-	-
Botterweck et al. (2009)	-	-	+	+	+	-	+ (MC /C)	C&C	+	+ (MC/C)	-	-
Perovich et al. (2009)	+	-	+	+	-	-	-	C&C	+	-	-	-
Duran-Limon et al. (2011)	-	-	+	+	-	-	+ (OWL y MC)	C&C	+	+ (MC)	-	-
Guana y Correal (2013)	+	-	+	+	+	-	+	C&C	+	-	-	-
Czarnecki y Antkiewicz (2005)	-	-	+	+	+	-	+	+	+	+ (MC)	-	-
Ziadi y Jézéquel (2006)	+	-	+	+	+	-	Sobre el modelo	+	UML	+	Parcialmente	-
PLUS-EE- (Gomaa y Shin 2007)	-	-	+	+ (Código ejecutable)	-	-	Sobre el modelo	Múltiples vistas	UML	+	-	-
Perrouin et al. (2008)	-	-	+	+	+	-	+	-	UML	+	Parcialmente	-
Schaefer et al. (2009)	-	-	+	+	+	-	+	CoBoxes	CoBoxes	-	-	-
Tawhid y Petriu (2011b)	-	-	-	+	-	-	Sobre el modelo	Estructura	Marte	-	-	-
Sánchez et al. (2008)	-	-	-	+	+	-	+	+	+	-	Para definir lenguajes	-
FeatureMapper (Heidenreich et al. 2008)	-	-	+	+	+	-	+ (MC y modelos)	+	+	+	-	-
Haugen et al. (2010)	-	-	+	+	+	-	+	+	+	+	-	-

Leyenda:
MC: Modelo de características
C&C: Componente conector
MC/C: Modelo de características y modelo de componentes
+: Soportado
-: No soportado

3.2 Métodos de evaluación de arquitecturas para LPS

En el desarrollo de líneas de producto, donde un defecto en la calidad de la arquitectura puede ser replicado a múltiples productos, la evaluación de las arquitecturas software, tanto de la arquitectura de la línea de productos como la del producto, se convierte en una actividad crucial para el éxito o fracaso del proyecto software. De entre todos los métodos de evaluación para líneas de producto analizados en la revisión sistemática presentada por Montagud y Abrahão (2009) un 58% de las propuestas se centraban en la evaluación de la arquitecturas de la línea de producto mientras que solamente un 17% iban orientadas a la evaluación de la arquitectura de producto.

En la literatura se pueden encontrar múltiples propuestas para la evaluación de arquitecturas software propuestas en los últimos años. Hay numerosos estudios que analizan y clasifican estas propuestas atendiendo a diferentes criterios (Roy y Graham 2008; Falessi et al. 2009a; Ali Babar y Gorton 2004). Los métodos que cuentan con una mayor madurez son los métodos basados en escenarios que permiten la evaluación subjetiva de la arquitectura en tiempo de diseño como SAAM (Kazman et al. 1994) o ATAM (Kazman et al. 2000), frente a los métodos basados en modelos matemáticos para la evaluación en tiempo de diseño de atributos como el rendimiento como PASA (Williams y Smith 2002) o la fiabilidad con el método propuesto por Roshandel et al. (2006) o los métodos basados en métricas como el método de Lindvall et al. (2003), más centrados en evaluación tardía. Los métodos basados en escenarios son los que han alcanzado un mayor grado de madurez (Roy y Graham 2008) y concretamente ATAM es uno de los métodos más difundidos en la industria (Mårtensson 2006), aunque por otro lado, dado su carácter subjetivo, requieren de una mayor experiencia para llevar a cabo la evaluación.

La evaluación de arquitecturas software va a requerir además de una serie de características específicas, como tener en cuenta la variabilidad de los requisitos no-funcionales, soportar requisitos no-funcionales de la línea de productos y requisitos no-funcionales específicos del producto durante la evaluación. Además el resultado de la evaluación no recibe el mismo tratamiento que en el desarrollo de software convencional. En ocasiones el resultado de la evaluación de un producto puede llevar aparejado la evolución de la arquitectura de la línea de productos, modificaciones del alcance de la línea de productos (para dar soporte a un nuevo requisito no-funcional) o a una nueva evaluación de la arquitectura de la línea de productos. En esta sección, debido a la naturaleza del método, vamos a centrarnos en métodos específicamente definidos para

entornos de desarrollo de líneas de producto, con el fin de cubrir las necesidades específicas de evaluación de este tipo de entornos.

En las siguientes subsecciones se analizan en detalle los métodos para la evaluación de arquitecturas de la línea de producto, de la arquitectura de producto y aproximaciones hibridas que pueden ser utilizadas para evaluar ambos tipos de arquitecturas.

3.2.1 Métodos de Evaluación de arquitecturas de líneas de producto

La evaluación de arquitecturas de línea de productos ha sido cubierta por distintos métodos en función del propósito de la evaluación. Los métodos FAAM (Dolan 2001), D-SAAM (Graaf et al. 2005), ALMA Bengtsson et al. (2004), AQA (Matinlassi et al. 2002) o el framework de evaluación basado en métricas definido por Alves et al. (2008) para evaluación en tiempo de diseño, métodos de evaluación de arquitecturas existentes como el propuesto por Gannod y Lutz (2000) y métodos orientados a evolución para albergar nuevos requisitos como las propuestas de Maccari (2002) o Riva y Rosso (2003).

FAAM es un método para la evaluación de arquitecturas software que toma como base los métodos SAAM y ATAM para evaluar arquitecturas de línea de productos software. Principalmente se centra en dos atributos de calidad, interoperabilidad y extensibilidad. El método se centra en analizar como un conjunto de escenarios de evolución se alinean con los objetivos de evaluación. D-SAAM es otra extensión de SAAM que permite la ejecución de SAAM en equipos de evaluación distribuidos y que además es capaz de evaluar arquitecturas de referencia, como la arquitectura de línea de productos. ALMA es un método de evaluación basado en escenarios centrado en el análisis de la modificabilidad[6] de la arquitectura. AQA es un método que se integra en la metodología de diseño QADA (Matinlassi et al. 2002) descrito en la sección 3.3.2. El método, basado en escenarios, es una extensión de SAAM mejorado mediante la introducción de una base de conocimiento para el diseño y análisis para líneas de producto y categorías de escenarios.

También se han propuesto aproximaciones basadas en métricas para analizar los atributos de calidad de la arquitectura de la línea de productos como la propuesta de Alves et al. (2008), en la que se define un framework para la

[6] La modificabilidad de un sistema software es la facilidad con la que puede ser modificado para adaptarse a cambios en el entorno, requisitos o especificación funcional.

evaluación de arquitecturas de línea de productos basada en métricas. La variabilidad se describe explícitamente sobre los distintos modelos UML, empleando estereotipos y anotaciones UML. Con el framework es posible evaluar, por ejemplo, la complejidad asociada a cada variante o la complejidad total de la variabilidad de la línea de productos mediante una adaptación de la métrica de complejidad *Cyclomatic Complexity* (McCabe 1976).

Gannod y Lutz (2000) proponen un método que combina un análisis basado en escenarios similar a SAAM con técnicas de *model checking* para el análisis automatizado de arquitecturas de línea de productos existentes.

Por último, Maccari (2002) define un método para analizar como la arquitectura de una línea de productos se adapta a una serie de escenarios de evolución. El objetivo es analizar los problemas, riesgos y defectos que pueden surgir durante la evolución de una línea de productos. Este método fue posteriormente extendido por Riva y Rosso (2003) aplicando entrevistas para la extracción de problemas y riesgos asociados a los escenarios de evolución.

3.2.2 Métodos de Evaluación de arquitecturas de producto para LPS

En los últimos años se han definido solamente unos pocos trabajos aislados para la evaluación de arquitecturas de producto para entornos de desarrollo de líneas de producto software. En general se han intentado cubrir atributos de calidad aislados, como el rendimiento, con trabajos como el de Tawhid y Petriu (2011a) o el de Alonso et al. (1998), aunque podemos también encontrar propuestas de evaluación para múltiples atributos como el método de Guana y Correal (2011).

Tawhid y Petriu (2011a) proponen un método por el que obtener un modelo de rendimiento de un determinado producto. El método consta de dos procesos de transformación de modelos ATL, en el primero se obtiene el modelo del producto a partir del modelo de la línea de productos y posteriormente se obtiene en un segundo proceso de transformación el modelo de rendimiento. La variabilidad de la línea de productos se describe mediante un modelo de características con cardinalidades. El modelo de la línea de productos se describe mediante un modelo de diseño con múltiples vistas en UML, que representa la super-imposición de todas las variantes de la línea de productos tanto a nivel estructural como a nivel de comportamiento. Las anotaciones de rendimiento se introducen en el modelo de la línea de productos mediante anotaciones MARTE.

Alonso et al. (1998) define el método TPA (Timing Property Assessment), para analizar el rendimiento del sistema, que reutiliza modelos de análisis RMA (Rate Monotonic Analysis) de los componentes individuales para componer el modelo RMA global del sistema.

Guana y Correal (2011) definen un método basado en una cadena de transformaciones de modelos que les permite evaluar cuál es el conjunto de componentes que satisface los atributos de calidad del producto a desarrollar. Se modela la arquitectura de la línea de productos, la variabilidad de la línea de productos descrita mediante modelos de características con cardinalidades, el catálogo de componentes descrito mediante un metamodelo de descripción de componentes, los escenarios de calidad descritos mediante un metamodelo de escenarios de calidad y los puntos sensibles mediante un lenguaje de dominio especifico denominado *Sensitivity Point Definition Language* (SPDL). A partir de estos modelos se analiza todas las posibles combinaciones de componentes que se pueden desplegar en la arquitectura y verifica si las propiedades resultantes del producto satisfacen los requisitos de calidad expresados a través de escenarios de calidad.

3.2.3 Métodos híbridos de evaluación de arquitecturas para LPS

Por último, solo tres métodos afrontan la evaluación tanto de la arquitectura de la línea de productos como de la arquitectura del producto en desarrollo: E-ATAM (Kim et al. 2008), HooPLA (Olumofin y Misic 2007) y CaLiPro (Etxeberria y Sagardui 2008).

E-ATAM es una extensión del método ATAM para la evaluación de arquitecturas de líneas de producto y de arquitecturas de producto derivadas de ella. El método incorpora la noción de variabilidad y lleva el análisis en dos fases, en la primera fase se evalúa la arquitectura de la línea de productos analizando los atributos de calidad de la línea de productos junto con los atributos de calidad del dominio (ver Sección 2.4.1) mientras que en la segunda fase, se evalúan las distintas arquitecturas de producto que se derivan de ella, analizando únicamente los atributos de calidad del dominio aplicando el método ATAM original. Realmente en este método no se da soporte explícito a la variabilidad de requisitos no-funcionales ni a los requisitos específicos de producto, sino que simplemente se emplea ATAM para evaluar dos arquitecturas de distinta naturaleza.

HooPLA es otra extensión de ATAM que añade el análisis cualitativo de los puntos de variación y la generación dependiente de contexto, clasificación y

priorización de escenarios. En una fase inicial se evalúa la arquitectura de la línea de productos con una aplicación de ATAM. En una primera fase se ejecuta una variante de ATAM sobre la arquitectura de la línea de productos. Como salida se obtienen, además de las salidas de ATAM (enfoques arquitectónicos, atributos de calidad, escenarios, riesgos y puntos sensibles), los puntos de evolución y las guías de evolución para la arquitectura de la línea de productos. En la segunda fase se analizan las arquitecturas de producto, tomando como entradas los objetivos de negocio, el alcance de la línea de productos y la variabilidad. Para cada arquitectura de producto se obtiene como salida los enfoques arquitectónicos, atributos de calidad, escenarios, riesgos y puntos sensibles. El método prevé la evaluación de todos y cada uno de los productos dentro del alcance de la línea de productos, lo cual puede verse como altamente ineficiente o directamente inabordable en algunos contextos. Además la información de la evaluación no es empleada explícitamente en la derivación de nuevos productos, únicamente sirve para detectar riesgos y puntos sensibles del conjunto de productos. El método solo será escalable en líneas de productos con una variabilidad limitada y un conjunto pequeño y cerrado de posibles productos.

CaLiPro (Evaluación de atributos de Calidad en Líneas de Productos), es un método de evaluación de arquitecturas para líneas de producto software que puede emplearse a nivel de diseño, para la evaluación de la arquitectura de la línea de productos como a nivel de implementación, para evaluar los potenciales productos antes de su generación. Emplean un modelo de características extendido con una variación del árbol de utilidad de ATAM. Para reducir el esfuerzo de evaluación necesario para caracterizar la variabilidad a nivel de atributos de calidad, los autores proponen dos aproximaciones: i) crear un modelo de evaluación genérico para la línea de productos; o ii) seleccionar un conjunto de productos representativos a evaluar. Tras la evaluación la información de evaluación puede ser incorporada en el modelo de características extendido para guiar el proceso de derivación. El método ha sido aplicado usando el método de evaluación PASA aunque puede ser aplicado a otros métodos de evaluación arquitectónica (Etxeberria 2008). Utilizar la información de los productos más representativos puede ser una primera aproximación para guiar la derivación, aunque esta información puede ser imprecisa y sesgar, de forma incorrecta, el proceso de derivación de productos. Igualmente con el método se hace necesaria la evaluación de la arquitectura de producto tras la derivación para constatar que los requisitos no-funcionales han sido correctamente satisfechos.

3.2.4 Comparativa y discusión

A continuación comparamos los métodos de evaluación analizados. La Tabla 3.3 muestra el conjunto de criterios de clasificación empleado para la comparación de los métodos. Dicho conjunto se basa en la clasificación propuesta de Etxeberria y Sagardui (2005) extendido con el criterio *C4* para analizar la independencia del método con respecto a las vistas arquitectónicas y lenguajes de modelado arquitectónico.

Tabla 3.3 Criterios para la comparación de métodos de evaluación arquitectónica para líneas de producto software

Criterio	*Descripción*
C1	Soporte a la evaluación de producto
C2	Soporte a múltiples atributos de calidad
C3	Soporte a evaluación objetiva
C4	Independiente de las vistas arquitectónicas / lenguajes de descripción arquitectónica
C5	Soporte a la evaluación en tiempo de derivación

La Tabla 3.4 muestra la comparativa de los métodos de evaluación de arquitecturas atendiendo a los criterios descritos en la Tabla 3.3. Del análisis pormenorizado de las propuestas detalladas en las anteriores subsecciones, podemos concluir que:

- La mayoría de las propuestas se centran en la evaluación de las arquitecturas de líneas de productos y solamente encontramos tres métodos para la evaluación de arquitecturas de producto y otros tres aplicables a ambas arquitecturas.

- Gran parte de las propuestas (si atendemos a la fracción de métodos para arquitecturas de productos) solamente está enfocada a un atributo de calidad en particular, siendo muy limitado el número de métodos enfocados a la evaluación de múltiples atributos, de propósito general, en arquitecturas de producto.

- La mayoría de las propuestas se enfocan a la evaluación en fase de de diseño, empleando escenarios como técnica de evaluación. El uso de métricas para la evaluación objetiva de los requisitos no-funcionales.

- La mayoría de los métodos permiten la evaluación independientemente de la vista arquitectónica o el lenguaje de descripción arquitectónico empleado.

- Solamente un grupo reducido de los métodos puede ser aplicados en fase de derivación.

Tabla 3.4 Comparación entre los métodos de evaluación de arquitecturas analizados

Método	*C1*	*C2*	*C3*	*C4*	*C5*
FAAM (Dolan 2001)	-	+	Escenarios	+	-
D-SAAM	-	+	Escenarios	+	-
ALMA	-	- (Modificabilidad)	Escenarios	+	-
AQA	-	+	Escenarios	+	-
Alves et al. (2008)	-	+	+	+	-
Gannod y Lutz (2000)	-	+	-	+	-
Maccari (2002)	-	- Evolución	-	+	-
Riva y Rosso (2003)	-	- Evolución	+	+	-
Tawhid y Petriu (2011a)	+	- (Rendimiento)	+	-	+
Alonso et al. (1998)	+	- (Rendimiento)	+	+	+
Guana y Correal (2011)	+	+	-	+	+
E-ATAM (Kim et al. 2008)	+ (ALP/P)	+	-	+	+
HooPLA (Olumofin y Misic 2007)	+ (ALP/P)	+	-	+	+
CaLiPro (Etxeberria 2008)	+ (ALP/P)	+	-	+	+

3.3 Aseguramiento de calidad en el desarrollo de LPS

Las técnicas de aseguramiento de calidad no han de circunscribir únicamente al testeo de los productos obtenidos, o a considerar atributos de calidad durante el diseño, sino que han de cubrir el ciclo de vida en su extensión. El problema de la configuración de productos teniendo en cuenta atributos de calidad ha recabado gran interés por parte de los investigadores, así como la introducción de atributos de calidad en el diseño de las arquitecturas software y el testeo en

el desarrollo de líneas de producto. Aunque hay multitud de propuestas definidas específicamente para el testeo en el desarrollo líneas de producto, prácticamente ninguna aborda la validación de requisitos no-funcionales, con excepción de la propuesta de Reis et al. (2006) que se propone una técnica para llevar a cabo pruebas de rendimiento. En las siguientes subsecciones se analizan las propuestas que abordan la configuración de productos y el diseño de arquitecturas de línea de productos atendiendo a criterios de calidad.

3.3.1 Atributos de calidad en la configuración de productos

En el ámbito del desarrollo de líneas de producto software los esfuerzos a la hora de incorporar técnicas de aseguramiento de la calidad se suelen centrar en el modelado de los requisitos no-funcionales y la introducción de criterios de calidad en el proceso de configuración de productos como las propuestas de Bagheri et al. (2010), Siegmund et al. (2011), Roos-Frantz et al. (2011), Trendowicz y Punter (2003), Soltani et al. (2012), Jarzabek et al. (2006), Zhang et al. (2010) o Ghezzi y Sharifloo (2011).

Una de las técnicas para la introducción de análisis de modelos de metas (Jarzabek et al. 2006; González-Baixauli et al. 2004). Jarzabek et al. (2006) presentan un método para modelar y analizar los atributos de calidad en el contexto de las características de comportamiento en las líneas de producto software. La anotación propuesta, F-SIG Feature-softgoal interdependency graph), integra el grafo de interdependencia de *softgoals* (Softgoal Interdepency Graph - SIG), con la representación del modelo de características. La notación permite analizar cómo la selección de características afecta a los atributos de calidad del sistema en desarrollo. Los grafos F-SIG permiten expresar los impactos cualitativos entre características y atributos de calidad (y entre atributos de calidad). González-Baixauli et al. (2004) propone el uso del análisis orientado a metas para representar variaciones en los requisitos (conjuntos de requisitos diferenciados para cada producto de la familia). El modelo consta de dos sub-modelos, un modelo de metas funcionales y un modelo de *softgoals*.

El primero representa la funcionalidad que el sistema eventualmente será capaz de llevar a cabo. El modelo de *softgoals* representa los requisitos no-funcionales como condiciones o criterios que el sistema debe cumplir. El modelo permite introducir prioridades para cada *softgoal* en forma de porcentajes a la hora de llevar el análisis. Además entre *softgoals* se pueden establecer enlaces de correlación con diferentes etiquetas de influencia (--, -, ¿, +, ++). Con toda esta información el análisis es capaz de seleccionar las mejores variantes, analizar la

probabilidad de satisfacción o denegación, que permiten medir el intervalo de confianza del análisis, así como el grado de satisfacción de las *softgoals*. (Trendowicz y Punter 2003) presentan Prometheus, un método orientado a metas para definir modelos de calidad para el desarrollo de líneas de producto software que permite expresar tanto requisitos no-funcionales cualitativos como cuantitativos. El método comprende la identificación de las atributos de calidad, la definición de métricas para evaluar dichas atributos, y la identificación y representación de relaciones entre características de calidad a través de redes de Bayes (*Bayesian Belief Networks*).

Otras aproximaciones introducen los requisitos no-funcionales como prioridades y usan modelos de análisis jerárquico como herramienta para la toma de decisiones como las aproximaciones de Soltani et al. (2012), Zhang et al. (2010) o Zhang et al. (2013). Soltani et al. (2012) definen un enfoque para la planificación de la selección de características que satisfagan tanto los requisitos funcionales como los no-funcionales, proporcionando apoyo tanto a los requisitos no-funcionales cualitativos y cuantitativos. Las propiedades no-funcionales son asociadas a las características y las preferencias de los *stakeholders* se representan mediante la escala de prioridades Analytical Hierarchy Process (AHP) (Saaty 2008) y permiten la definición de restricciones específicas a los requisitos no-funcionales que el sistema debe cumplir. El modelo de características anotadas y las restricciones se traducen en Redes de Tareas Jerárquicas (Hierarchy Tasks Networks) para la planificación de la selección de las diferentes características. Zhang et al. (2010) proponen un enfoque basado en AHP para estimar la importancia relativa de cada característica funcional en relación a los atributos de calidad. El enfoque permite la obtención de una lista de configuraciones con un valor de importancia que expresa el soporte relativo de los atributos de calidad requeridos. La información de calidad se presenta como información complementaria en el modelo de características, siguiendo un esquema XML. Zhang et al. (2013) extiende el trabajo anterior proponiendo una aproximación que permite por un lado extraer y modelar los atributos de calidad en un modelo de características, cuantificar el impacto de las características en los atributos de calidad, aplicando el método AHP, analizar y representar las relaciones entre atributos de calidad y por otro configurar un producto haciendo uso de esa información. Para representar la información de calidad y las interacciones con el modelo de características emplean modelos de calidad extendidos con atributos de calidad. El proceso de validación permite guiar la selección de características y validar que los requisitos no-funcionales expresados por los *stakeholders* se pueden cumplir en el producto configurado.

Por ultimo utilizan una base de conocimiento para predecir los niveles de los atributos de calidad, pero no se brinda ningún tipo de soporte a la validación de que esos niveles se cumplen de forma efectiva mediante la medición sobre los artefactos obtenidos. Una incorrección en el modelo empleado, en la base de conocimiento o en el proceso de derivación llevaran a la obtención de productos que no cumplen los requisitos no-funcionales.

Otra técnica es el uso de razonadores previa transformación del problema a distintos modelos lógicos, como las lógicas difusas (Bagheri et al. 2010) o el uso de la programación por restricciones (Roos-Frantz et al. 2011). Bagheri et al. (2010) definen una aproximación con la que dar soporte a la configuración semiautomática de características en base a los requisitos funcionales, no-funcionales (restricciones fuertes) y preferencias (restricciones débiles) de los *stakeholders*. La aproximación se basa en la representación de las características y sus impactos en los atributos de calidad mediante lógica difusa proposicional y el uso de razonadores para este tipo de lógicas. Roos-Frantz et al. (2011) propone un enfoque para soportar el análisis de la calidad de las líneas de producto y el proceso de configuración del producto. El enfoque se basa en la representación de las propiedades de calidad de las diferentes variantes mediante un modelo ortogonal de variabilidad anotado con la información de calidad. Esos modelos se trasladas a un problema de programación con restricciones (*Constraint Programing*) con el fin de realizar el análisis de factibilidad y calcular atributos derivados que dependen de la selección de una determinada característica en un punto de variación. Las restricciones de los *stakeholders* se representan como restricciones al modelo y las preferencias como la función objetivo en el análisis. El análisis de optimización, llevado a cabo mediante una extensión de la herramienta FaMa (ISA Research Group 2011), es capaz de retornar el producto más representativo que cumple con las restricciones y los objetivos.

Por ultimo otras aproximaciones abordan la exploración de las diferentes configuraciones para analizar como la selección de distintas variantes mejora las propiedades de calidad del producto, como las propuestas de Ghezzi y Sharifloo (2011) o SPL-Conqueror (Siegmund et al. 2011). Ghezzi y Sharifloo (2011) proponen el análisis de los niveles de atributos de calidad de las distintas configuraciones de una línea de productos. El análisis toma como entrada diagramas de secuencia con puntos de variación. La información de calidad se adjunta al diagrama de secuencia mediante anotaciones. Los diagramas de secuencia se transforman de automáticamente en modelos de Markov para ejecutar un análisis probabilístico paramétrico. El resultado del análisis es el conjunto de posibles configuraciones de la línea de productos y los niveles de

atributos de calidad para cada configuración. La aproximación no considera requisitos no-funcionales cualitativos. Siegmund et al. (2011) proponen una aproximación, SPL-Conqueror para la optimización de propiedades no-funcionales en la ingeniería de líneas de producto. Para medir las distintas propiedades generan las distintas configuraciones y posteriormente se anota esa información sobre un modelo integrado de la línea de productos (Siegmund et al. 2008). En dicho modelo es posible representar explícitamente las correspondencias entre características y unidades de código y como estas influyen en las propiedades no-funcionales del sistema (atributos de calidad). Esta información sirve posteriormente para obtener la configuración que cumple con los requisitos funcionales y no-funcionales y para optimizar dicha configuración con la implementación alternativa que mejor se adapte a los requisitos no-funcionales.

3.3.2 Criterios de calidad en fase de diseño

Otra de las fases del proceso de desarrollo de líneas de producto en la que se han introducido técnicas de aseguramiento de calidad es el diseño de la arquitectura de la línea de productos con propuestas como QRF, QASAR, QADA y PuLSE-DSSA.

Bosch (2000) propone el método QASAR (*Quality Attribute-Oriented Software Architecture Design Method*). El método consta de dos procesos iterativos en el que el proceso más externo empieza con el núcleo de los requisitos de producto y entra en el otro proceso iterativo interno en el que se diseña la arquitectura. Cuando se tiene una primera versión de la arquitectura se extiende el conjunto de requisitos y se vuelve a entrar en el proceso interno, así hasta considerar todos los requisitos, tanto funcionales como de calidad. Para asegurar los atributos de calidad se propone el uso de transformaciones arquitectónicas, en las que se aplican patrones de diseño, patrones arquitectónicos o estilos arquitectónicos.

Thiel y Hein (2002) proponen QuASAR (*Quality Attribute-Software Architecture Design Method*), un framework de diseño de arquitecturas que soporta el diseño de arquitecturas de líneas de producto, empezando desde las decisiones tempranas, el modelado de las vistas arquitectónicas y la variabilidad en cada una de ellas. Finalmente incluye actividades de análisis e consistencia en la arquitectónica, cobertura de la variabilidad y cumplimiento de los requisitos de calidad.

Matinlassi et al. (2002) proponen QADA, un método que da soporte al diseño de arquitecturas de la línea de productos a partir de requisitos funcionales y no-funcionales. El método también hace uso de patrones y estilos arquitectónicos con el fin de dar soporte a los requisitos no-funcionales. Emplean una base de conocimiento para evaluar los distintos estilos arquitectónicos y asignarles una valoración que permitirá posteriormente asesorar sobre que patrón o estilo aplicar para conseguir alcanzar ciertos requisitos de calidad.

Bayer et al. (2000) proponen PuLSE-DSSA, un método para el diseño de arquitecturas de líneas de producto a partir del alcance de la línea de productos y de un modelo de dominio que describe los casos de negocio para el desarrollo de la línea de productos. La arquitectura resultante se evalúa siguiendo el plan de evaluación arquitectónica en el que se evalúa que la arquitectura resultante cubre los productos dentro del alcance.

Niemelä y Immonen (2007) proponen el método QRF (*Quality Requirements of a Software Family*) para el diseño de arquitecturas de líneas teniendo en cuenta los requisitos de los distintos *stakeholders* y los objetivos de negocio. El método permite extraer y representar los requisitos funcionales y no-funcionales mediante un modelo de dependencias que permitirá además describir la variabilidad en ambos tipos de requisitos. Se seleccionan los estilos y patrones arquitectónicos de manera que den soporte a los requisitos extraídos en las primeras fases del método. Se emplean perfiles UML para anotar los distintos modelos arquitectónicos con la información de calidad con el fin de poder posteriormente medir y trazar los requisitos no-funcionales.

3.3.3 Discusión

Del análisis pormenorizado de las propuestas descritas en las anteriores subsecciones podemos concluir que la problemática de la configuración atendiendo a requisitos no-funcionales ha atraído la atención de los investigadores con multitud de técnicas para obtener una configuración optima, aunque resulta sorprendente que, con la excepción de (Siegmund et al. 2011), ninguna otra aproximación permita la validación, a posteriori, del cumplimiento de dichos requisitos no-funcionales.

La otra disciplina que ha recibido gran atención por parte de los investigadores es el diseño de la arquitectura de la línea de productos para que esta incorpore ciertos niveles de calidad. Únicamente el método de Thiel y Hein (2002) hacen un análisis de cobertura de la variabilidad, pero sin analizar los niveles de calidad de los productos dentro del alcance. En el método PuLSE-DSSA se

evalúa la arquitectura para asegurarse de que cubre los productos dentro del alcance y en QRF se anotan los modelos arquitectónicos con la información de los atributos de calidad, pero no se genera como resultado de estos procesos una representación explicita de los umbrales de los requisitos no-funcionales o como determinadas decisiones de diseños van a afectar a ciertos atributos de calidad.

3.4 Estudios empíricos en el campo de las arquitecturas software

El incremento en el tamaño y complejidad de los sistemas software y la creciente demanda de sistemas de alta calidad, ha incrementado el interés en el campo de las arquitecturas software. En este contexto, se han definido múltiples métodos y herramientas que soportan las distintas actividades del proceso de diseño y evaluación de arquitecturas software. Sin embargo, se ha prestado poca atención a la validación empírica de los métodos y herramientas propuestos. En las siguientes subsecciones se presentan y discuten trabajos relacionados que presentan comparaciones entre métodos de evaluación de arquitecturas software y estudios empíricos que evalúan diferentes aspectos del proceso de evaluación de arquitecturas software.

3.4.1 Marcos de clasificación para la comparación de métodos de evaluación de arquitecturas software

Recientemente, se han presentado múltiples marcos de comparación de métodos de evaluación de arquitecturas software (Ali Babar y Gorton 2004; Roy y Graham 2008; Etxeberria y Sagardui 2005; Breivold et al. 2012). (Ali Babar y Gorton 2004), presentan una primera revisión de la literatura comparando cuatro métodos de evaluación de arquitecturas software basados en escenarios:

- *Scenario-Based Architecture Analysis* (SAAM)
- *Architecture Level Modifiability Analysis* (ALMA)
- *Performance Assessment of Software Architectures* (PASA)
- *Architecture Trade Off Analysis Method* (ATAM)

En un trabajo posterior los autores presentan una extensión del trabajo, donde establecen el marco de caracterización de métodos de evaluación de

arquitecturas software basado en una revisión de la literatura (Ali Babar et al. 2004). En este último estudio, los autores aplican el marco para clasificar ocho métodos de evaluación (SAAM, ATAM, ARID, etc.). En ambos estudios, los autores siguen un esquema de clasificación basado en quince criterios (como son la fase de madurez, la definición de arquitectura software que sigue el método, si hay soporte al proceso, las actividades recogidas por el método etc.). Este marco ha sido también evaluado mediante una encuesta de opinión de expertos presentado en Ali Babar y Kitchenham (2007a). El objetivo de la encuesta era analizar la idoneidad de los criterios del marco. Los resultados de la encuesta avalan la mayoría de los criterios y solamente unos pocos (soporte de herramienta, actividades del método y dominio de aplicación) condujeron a desacuerdo entre los participantes.

Roy y Graham (2008) presentan una encuesta en la que analizan treinta y siete métodos de evaluación de arquitecturas. También establecen una taxonomía para su clasificación basándose en la fase en la que se aplican (evaluación temprana en oposición a evaluación tardía), la técnica de análisis aplicada o los artefactos analizados (basados en escenarios, en modelos matemáticos o en métricas) y su capacidad para hacer frente a la evaluación de estilos y patrones arquitectónicos. Las principales conclusiones del estudio: i) que es difícil ser competente en el uso y aplicación de métodos de evaluación de arquitecturas; ii) hay una carencia de herramientas que den soporte a los métodos y iii) la mayoría de métodos (excepto SAAM y ATAM) no han sido validados empíricamente.

Etxeberria y Sagardui (2005), presentan un marco de comparación basado en una revisión de la literatura para clasificar dieciséis propuestas y técnicas de evaluación definidas para entornos de desarrollo de líneas de producto software. Este marco clasifica las propuestas de evaluación de arquitecturas en función del tiempo de evaluación (evaluación en tiempo de diseño frente a evaluación en tiempo de configuración o en fase de evolución), la arquitectura que se evalúa (la arquitectura de la línea de productos o la arquitectura de producto) y el propósito de la evaluación (evaluación de la evolución de la arquitectura de la línea de productos o evaluación de la arquitectura de producto en derivación).

Por ultimo Breivold et al. (2012) presentan una revisión sistemática de la literatura en la que, entre otros temas, cubren la evaluación de la calidad de arquitecturas software haciendo énfasis en aspectos de evolución. Los autores se centran en analizar los métodos de evaluación de arquitecturas software basados en escenarios, basados en la experiencia y basados en métricas que

afrontan el análisis de la capacidad de evolucionar de una arquitectura. Una de las principales conclusiones del estudio es que, las técnicas que apoyan el análisis de aspectos de calidad, ayudan en la identificación de los atributos de calidad clave en la fase inicial del diseño del software. Los autores también alientan a la definición de métodos y herramientas para el diseño (y administración) de arquitecturas software para sistemas altamente complejos (por ejemplo líneas de producto software).

Los estudios anteriormente descritos, ofrecen un análisis de las características de los métodos comparados, que pueden ayudar a los profesionales e investigadores a tener una visión holística de los métodos disponibles. Sin embargo, no ofrecen datos concretos sobre qué método es más eficiente, o más eficaz o más fácil de usar en un determinado tipo de proyecto o escenario de desarrollo. La mayoría de estos trabajos reportan estudios basados únicamente en información subjetiva sin seguir una metodología predefinida (con excepción del trabajo de Breivold et al. en la que se sigue una metodología bien definida).

3.4.2 Estudios empíricos analizando la evaluación de arquitecturas software

A pesar de que el interés en la sub-disciplina de arquitecturas software se ha visto incrementado en los últimos años, se han llevado a cabo relativamente pocos experimentos para el análisis de los distintos aspectos de los métodos de evaluación de arquitecturas (Ali Babar y Kitchenham y et al. 2007; Ali Babar y Kitchenham 2007b; Ali Babar 2008; Falessi y Cantone y et al. 2008; Falessi y Capilla y et al. 2008; Golden y John 2005; Martens et al. 2010) o validaciones empíricas a través de estudios de caso o informes de experiencias (Barbacci et al. 2003; Boucké et al. 2006; Reijonen et al. 2010; Svahnberg y Mårtensson 2007). A continuación se presenta un resumen de cada uno de estos estudios.

Golden y John (2005) reportó un experimento controlado con el que analizar el valor de las distintas partes de los patrones arquitectónicos orientados a usabilidad en la modificación del diseño de una arquitectura software. En el estudio se evalúa, cómo las distintas soluciones arquitectónicas resultantes del uso de una especificación más completa del patrón, dan un mejor soporte a la evaluación de los requisitos de usabilidad. El estudio demuestra que el uso de especificaciones más completas de los patrones aumenta la eficacia y la eficiencia de las evaluaciones de usabilidad.

Con respecto a la evaluación de la influencia del tamaño del equipo, organización y apoyo a la comunicación entre los miembros de los equipos, se han presentado diferentes estudios empíricos (Ali Babar y Kitchenham 2007b; Ali Babar y Kitchenham y et al. 2007; Ali Babar y Winkler y et al. 2007). Ali Babar y Kitchenham (2007b) presentan un experimento controlado en el que se analiza el impacto del tamaño del grupo en el resultado de un ejercicio de evaluación de arquitecturas software. En el estudio se analiza cómo el tamaño del grupo afecta a la calidad de los perfiles de escenario y a la satisfacción de los participantes con el proceso y con el resultado. Su principal resultado es que, efectivamente, el tamaño del grupo afecta a la calidad de los escenarios creados. También hubo desacuerdos entre el tamaño del grupo con el que los sujetos obtienen escenarios con mejor calidad (grupos de cinco participantes) y el tamaño del grupo con el que los participantes estaban más satisfechos (grupos de tres participantes).

Ali Babar, Kitchenham, et al. (2007) presentan un experimento que compara las reuniones distribuidas, frente a las reuniones cara-a-cara en el proceso de evaluación de arquitecturas software. El objetivo del estudio fue evaluar la eficacia del proceso de trabajo en grupo propuesto con respecto el desarrollo de escenarios de alta calidad durante el proceso de evaluación. En un trabajo similar, Ali Babar, Winkler, et al. (2007) reportaron los resultados de un experimento analizando la utilización de LiveNet, que es una herramienta de trabajo en grupo para dar soporte al proceso de evaluación de arquitecturas software. El objetivo del estudio era analizar la facilidad de uso y la utilidad percibidas por los participantes después de realizar diversas tareas de colaboración. Los resultados del primer estudio confirmaron que la calidad de los escenarios desarrollados por los equipos distribuidos, utilizando una herramienta de trabajo en grupo, fueron significativamente mejores que la calidad de los perfiles de escenarios desarrollado por los equipos cara-a-cara. Los resultados del segundo estudio mostraron que los participantes consideraron como positivo el uso de la herramienta de trabajo en grupo para dar soporte a las reuniones distribuidas.

Falessi, Cantone, et al. (2008) presentaron los resultados de un experimento controlado y de su réplica (Falessi y Capilla y et al. 2008) en el que se analiza la utilidad percibida de la información expuesta en la documentación de justificación de decisiones de diseño arquitectónico (*Architectural Design Decisions Rationale Documentation* o DDRD), un artefacto empleado para documentar decisiones de diseño arquitectónico. En el estudio, los participantes realizaban una serie de actividades (descritas mediante casos de uso DDRD) para después clasificar las diferentes categorías de información contenidas en los DDRDs

(por ejemplo, el problema, la decisión, el estado, las limitaciones, los requisitos relacionados etc…). Los resultados mostraron que la importancia percibida de cada una de las categorías de información contenida en los DDRDs, dependía en gran medida de la actividad que se estaba llevando a cabo (Falessi y Cantone y et al. 2008) y que la DDRD debía contener únicamente la información requerida para llevar a cabo una determinada actividad (Falessi y Capilla y et al. 2008).

Ali Babar (2008) presentó la evaluación del Marco para el Análisis del Nivel de Seguridad Arquitectónica (*Architectural Level Security Analysis Framework* ALSAF), que llevó a cabo mediante un piloto y un cuasi-experimento. El objetivo del estudio fue la identificación de los atributos de seguridad y patrones de diseño arquitectónico adecuados para el logro de esos atributos, basándose en una lista de propiedades de seguridad. En este experimento, el grupo de control tuvo acceso sólo a la especificación de requisitos del software (ERS), mientras que el grupo de tratamiento tuvo acceso tanto al ERS y a ALSAF. Los resultados muestran que los participantes obtienen resultados significativamente mejores utilizando ALSAF a la hora de identificar atributos de seguridad y patrones y percibieron la utilidad de ALSAF al realizar estas tareas.

Martens et al. (2010) presentó una serie de tres experimentos controlados que comparan el esfuerzo requerido al aplicar cuatro métodos de evaluación de rendimiento de arquitecturas software y la precisión de dichos métodos. El objetivo del estudio, fue estudiar tres métodos monolíticos de evaluación del rendimiento (SPE, PC y umlPSI) contra el método de evaluación del rendimiento basado en componentes (PCM). Los resultados muestran que en términos de precisión, PCM, SPE y CP produjeron resultados similares, y que umlPSI produjo sobreestimación, mientras que en términos de esfuerzo, la aplicación de PCM es la que requiere mayor esfuerzo.

A pesar de que los estudios anteriormente descritos tienen como objetivo reunir el conocimiento empírico a través de experimentos o cuasi experimentos, también se muestra que la mayoría de ellos se centran en aspectos específicos del proceso de evaluación de la arquitectura o en la evaluación de cómo un determinado tratamiento mejora el rendimiento, pero se constata una carestía de validaciones empíricas de los métodos y herramientas que se proponen, mediante la comparación con el cuerpo de conocimiento existente.

Por último, hay distintos trabajos reportando experiencias en la aplicación de ATAM con distintos propósitos (como por ejemplo, (Barbacci et al. 2003; Boucké et al. 2006; Svahnberg y Mårtensson 2007; Reijonen et al. 2010).

Barbacci et al. (2003) presentan un estudio de caso en el que se evalúa la arquitectura de una línea de productos del dominio de los sistemas de aviónica[7]. Boucké et al. (2006) presentan un informe de la aplicación de ATAM para la evaluación de una arquitectura de sistema multi-agente. Svahnberg y Mårtensson (2007) presentan un informe con sus experiencias en la aplicación de una variante de ATAM para evaluaciones de arquitecturas software en entornos académicos. En este último caso, se aplicó un método ligero de evaluación de arquitecturas software adaptando SAAM y ATAM. Las evaluaciones de la arquitectura se aplicaron tanto para la evaluación de las arquitecturas de los proyectos desarrollados por los alumnos, como para la enseñanza de la evaluación de arquitecturas software.

Reijonen et al. (2010) reportan sus experiencias en la aplicación de ATAM en once evaluaciones arquitectónicas de proyectos industriales reales. En este trabajo, los autores proporcionan una descripción detallada de la aplicación de los distintos pasos, de la planificación temporal seguida en cada evaluación, de los problemas a los que se enfrentaron durante cada evaluación y de los principales beneficios percibidos por los *stakeholders*.

Estos trabajos contribuyen a demostrar la viabilidad de ATAM y todos ellos, ponen de relieve la importancia de llevar a cabo evaluaciones de arquitecturas software, haciendo especial hincapié en la importancia del árbol de utilidad (aunque Reijonen et al. (2010) señalan las dificultades encontradas por los evaluadores al afrontar su creación) y cuán crítica es la calidad de los escenarios para el éxito de la evaluación.

3.4.3 Discusión

El análisis de los estudios descritos en las anteriores secciones, nos ha permitido identificar algunas limitaciones en la validación empírica de los métodos de evaluación de arquitecturas software, tales como:

i) El bajo número de estudios empíricos que evalúan los métodos y sus variantes, que se han definido.

ii) El hecho de que se ha descuidado la comparación con métodos existentes en términos cuantitativos y cualitativos.

[7] Aviónica: Aplicación electrónica de comunicaciones, navegación, control y visualización empleados en satélites artificiales y vehículos de la industria aeronáutica y aeroespacial.

iii) El hecho de que la mayoría de estudios empíricos tienden a ser aislados y raramente son replicados.

La primera limitación está en línea con las conclusiones de estudios de Ali Babar et al. (2011), de Falessi et al. (2009b) y Dyba et al. (2005) en la que se afirma que en el campo de las arquitecturas software, la mayoría de las propuestas presentadas carecen de validación empírica. Las pocas validaciones disponibles de las aproximaciones que se han propuesto están, en la mayoría de los casos, basadas en ejemplos, estudios de casos o informes excesivamente simplistas, tal como pone de relieve Qureshi et al. (2013). En este campo, además, podemos encontrar que la mayoría de artículos utilizan terminología incorrecta cuando se refieren a aspectos de validación empírica. En muchas ocasiones, se utiliza el término "experimento" cuando realmente presentan experiencias (como en el caso de los trabajos de (Niemelä y Immonen 2007; Mårtensson 2006; Wang y Chen 1999)) o el término "estudio de caso" cuando se documentan pruebas de conceptos, sin descripción de la metodología aplicada ni de los procedimientos de extracción de datos (Qureshi et al. 2013).

La segunda limitación se basa en la falta de comparaciones cuantitativas y cualitativas con los métodos existentes. Los métodos o herramientas que se proponen, no habían sido comparados con las alternativas existentes en el cuerpo de conocimiento de la evaluación de arquitecturas software. Tal como se describe en la Sección 3.4.1, existen diferentes marcos para la clasificación de los métodos de evaluación de arquitecturas software, sin embargo, hay una carestía de estudios empíricos en los que los autores analicen el comportamiento de los métodos o herramientas propuestos, en comparación a otros similares.

La tercera limitación va en la línea de otros estudios que se han realizado en el campo de la ingeniería del software, como el de Sjøberg et al. (2005). En este trabajo los autores afirman que sólo 20 de un total de 113 experimentos analizados, son repeticiones. Una réplica es la repetición de un experimento para confirmar los resultados o para asegurar la exactitud. Hay dos tipos de replicaciones (Lindsay y Ehrenberg 1993):

- Réplicas estrictas: réplicas en las que se trata de mantener las condiciones experimentales conocidas.
- Réplicas diferenciadas: réplicas que introducen variaciones en aspectos esenciales de las condiciones experimentales, como la ejecución de réplicas con diferentes tipos de participantes.

Ambos tipos de réplicas son necesarias para lograr una mayor validez a los resultados obtenidos a través de estudios empíricos. El manejo de replicaciones experimentales ha sido abordado también a través del concepto de *familia de experimentos*. Aunque, como se ha descrito en las anteriores secciones, en el campo de las arquitecturas software se han realizado distintos tipos de estudios empíricos, no se ha reportado ninguna familia de experimentos.

3.5 Conclusiones

Esta sección resume las conclusiones que se pueden extraer atendiendo a los trabajos analizados en las tres secciones anteriores.

La conclusión principal es que no se encuentran en la literatura metodologías genéricas que permitan la derivación y evaluación de las arquitecturas de producto de una manera integrada, mediante mecanismos de evaluación que nos permitan validar aquellos requisitos de calidad que formaban parte de la configuración. La mayoría de las técnicas analizadas se centran en obtener una solución óptima para el problema de la configuración, pero de nada servirá si el producto obtenido tras la configuración no cumple con las propiedades que se le suponían. Además, en muchas ocasiones no se tiene en cuenta la no-predictibilidad de ciertas propiedades de calidad (Crnkovic et al. 2004), que hace que ciertas propiedades no puedan ser modeladas como la suma de las propiedades de sus partes, con lo que siempre hay cierta incertidumbre que de la única forma en que puede ser resuelta es mediante la constatación empírica (mediante la medición) de los valores reales de dichas propiedades de calidad.

Sin métodos integrados que cubran el proceso completo se hace difícil el establecimiento de mecanismos de trazabilidad que nos permitan analizar las posibles fuentes de eventuales problemas de calidad que pueden ser detectados tras la derivación de la arquitectura.

Por otro lado, la mayoría de aproximaciones están definidas o para algún lenguaje de modelado concreto, o para alguna vista arquitectónica en particular o para arquitecturas modeladas mediante algún lenguaje de especificación arquitectónica concreto, sin que se encuentren, por ejemplo en el caso de los métodos de evaluación, métodos genéricos para la evaluación de arquitecturas de producto en tiempo de derivación.

Además, la gran mayoría de los nuevos métodos y técnicas definidos en el área de las arquitecturas software no cuenta con una validación empírica, lo cual

imposibilita la comparación de dichos métodos con el cuerpo de conocimiento existente.

Toda la problemática anteriormente descrita es lo que nos ha llevado a la definición y posterior validación de QuaDAI, un método integrado para la derivación, evaluación y mejora de arquitecturas de producto, en las que se integran los procesos de configuración, instanciación de la arquitectura, evaluación, y en caso necesario aplicación de transformaciones arquitectónicas para mejorar ciertos atributos de calidad de la arquitectura derivada. Dicho método es aplicable independientemente de los atributos de calidad a ser evaluados o de las vistas o lenguajes empleados para especificar la arquitectura.

Capítulo 4

QuaDAI, un Método para la Derivación, Evaluación y Mejora de Arquitecturas de Producto

En este capítulo se presenta QuaDAI (Quality-Driven Architecture Derivation and Improvement), un método de derivación, evaluación y mejora de arquitecturas de producto software en procesos de desarrollo de líneas de producto que siguen el enfoque de desarrollo dirigido por modelos. QuaDAI hace uso de un multimodelo que es capaz de captar las diferentes vistas de una línea de productos como artefacto vertebrador del proceso.

La sección 4.1 describe, en líneas generales, el método para la integración de la derivación, evaluación y, en caso necesario, aplicación de transformaciones arquitectónicas a las arquitecturas de producto derivadas de la arquitectura de la línea de productos mediante el uso de un multimodelo.

La sección 4.2 introduce el multimodelo, las vistas que lo componen, los tipos de relaciones que es posible definir entre elementos de esas vistas, así como el soporte tecnológico del mismo.

La sección 4.3 describe el proceso que soporta el método QuaDAI mediante la definición de sus fases y los resultados producidos en cada fase. Para la especificación de dicho proceso se ha utilizado el estándar SPEM v2.0 (*Software and Systems Process Engineering Metamodel Specification*) (Object Management Group 2008b) para la ingeniería de procesos.

4.1 Vista general del método

QuaDAI es un método integrado para la derivación, evaluación y mejora de arquitecturas software que define un artefacto (el multimodelo) y un proceso dirigido por transformación de modelos para obtener arquitecturas de producto software que cumplen, tanto los requisitos funcionales, como los no-funcionales.

El multimodelo permite representar las diferentes vistas (de variabilidad, arquitectónica, de calidad y de transformaciones) y definir relaciones entre los elementos de dichas vistas.

La Figura 4.1 muestra las líneas generales del proceso QuaDAI que consta de dos fases:

i) la fase de *derivación* en la cual se realiza la actividad de *derivación de la arquitectura de producto* a partir de la arquitectura de la línea de productos mediante una cadena de transformación de modelos.

ii) la fase de *evaluación y mejora*, donde la arquitectura será evaluada en la actividad de *evaluación* y, en el caso en que no se cumplan los requisitos no-funcionales, se aplicaran transformaciones arquitectónicas en la actividad de *transformación* para tratar de cumplir dichos requisitos. Los detalles de cada una de las fases y actividades se describen con más detalle en la Sección 4.3.

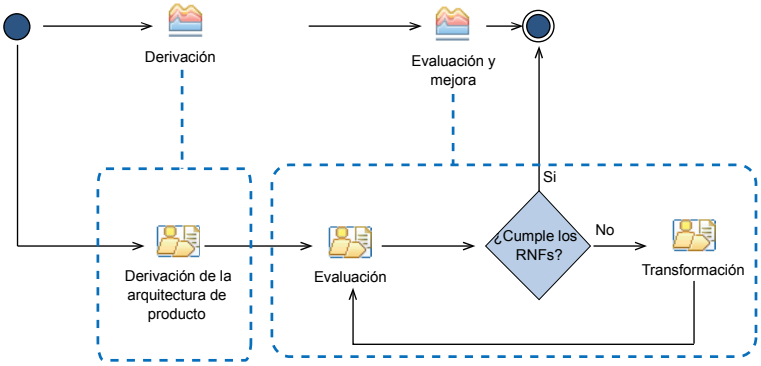


Figura 4.1 Vista general del proceso QuaDAI (Diagrama de actividad SPEM)

En la actividad de *derivación de la arquitectura de producto* esta se genera a partir de la configuración (que consta de las características y requisitos no-funcionales seleccionados por el ingeniero de aplicación), teniendo en cuenta las relaciones entre características, componentes de la arquitectura, requisitos no-funcionales

y atributos de calidad almacenadas en el multimodelo. La arquitectura derivada, es una primera versión de la arquitectura del producto en desarrollo que tendrá que ser evaluada con el fin de analizar el grado de cumplimiento de los requisitos no-funcionales.

En la fase de *evaluación y mejora*, la arquitectura será evaluada y, en caso necesario, se aplicaran transformaciones arquitectónicas con el fin de mejorarla con respecto a sus atributos de calidad. En la actividad de *evaluación* de la arquitectura, se aplicarán las métricas definidas en la vista de calidad de la arquitectura, con el fin de evaluar si se satisfacen o no los requisitos no-funcionales generándose un informe de evaluación. En el caso ideal, cuando los requisitos no-funcionales uedan ser alcanzados mediante los mecanismos de variabilidad arquitectónica no se pasará a la fase de evaluación, aunque en la práctica, sería razonable generar la arquitectura para comprobar el grado de cumplimiento de los requisitos no-funcionales e iterar de nuevo a la fase de configuración y derivación en caso necesario. Por el contrario, en los casos en que los requisitos no-funcionales no se puedan lograr ejerciendo los mecanismos de variabilidad arquitectónica previstos en la arquitectura de la línea de producto, en la actividad de *transformación*, se aplicarán transformaciones arquitectónicas a la arquitectura de producto con el fin de mejorar sus atributos de calidad y satisfacer así dichos requisitos no-funcionales. Las entradas para esta actividad son: la arquitectura de producto, la importancia relativa de los atributos de calidad, las transformaciones arquitectónicas y el impacto que estas tienen sobre atributos de calidad (que se almacenan como relaciones entre los atributos de calidad y las transformaciones en el multimodelo). La importancia relativa de cada atributo de calidad que el producto debe cumplir, se introduce como parámetro al ejecutar la transformación en forma de pesos normalizados que van de 0 a 1. El proceso de transformación utiliza la importancia relativa de los requisitos no-funcionales y las relaciones entre transformaciones y atributos de calidad, almacenadas en el multimodelo, para seleccionar la transformación arquitectónica que ha de ser aplicada en cada caso.

Los modelos que integran el multimodelo, permiten la especificación de la línea de productos desde los distintos puntos de vista de interés (de variabilidad, de calidad, arquitectónica y de transformaciones) y sobre todo, las relaciones entre los elementos de las distintas vistas, son las que permiten la integración de las actividades necesarias para obtener arquitecturas de producto de alta calidad. Además, estas relaciones permiten describir, de forma explícita, el conocimiento de los arquitectos y expertos del dominio, y su uso para seleccionar la alternativa más razonable en las decisiones de diseño en las que

intervienen múltiples factores de forma metódica y automatizada cubriendo asi una de las carencias detectadas en recientes estudios empíricos (e.g., Ameller et al. (2013)).

4.2 Multimodelo para la especificación de líneas de producto

El desarrollo de sistemas altamente complejos, como es el caso de las líneas de producto software siguiendo el enfoque de desarrollo dirigido por modelos, requiere el uso de modelos a diferentes niveles de abstracción como medio para describir el sistema, que más tarde serán transformados en el código final de la aplicación. La especificación de dichos modelos debe realizarse por separado, pero posteriormente es necesario definir las inter-relaciones entre dichos modelos, dado que estas van a representar los aspectos complementarios que no tienen cabida en ninguno de los modelos individuales.

En las siguientes subsecciones se va a introducir la definición de multimodelo, la descomposición en vistas y las relaciones identificadas para permitir la derivación, evaluación y transformación de las arquitecturas de producto, la arquitectura del metamodelo que soporte el multimodelo y la infraestructura desarrollada para dar soporte a la definición y manipulación de multimodelos.

4.2.1 Macromodelos, megamodelos y multimodelos

En los últimos años han surgido numerosas propuestas que abordan la cuestión de cómo capturar las dependencias entre modelos en diferentes lenguajes de modelado. Entre ellas destacan los *Megamodelos* propuestos por Bézivin et al. (2005) y Favre (2004a) o los *Macromodelos* propuestos por Salay et al. (2008) y el modelo único subyacente SUM (*Single-Underlying Model*) (Atkinson et al. 2012; Atkinson et al. 2013). Las definiciones de megamodelo y macromodelo, aunque se refieren a términos aparentemente distintos, van en la misma línea:

> *Megamodelo:* Modelo que contiene modelos y relaciones entre ellos (Hebig et al. 2012).
>
> *Macromodelo:* Modelo que consiste en elementos que representan modelos y enlaces que representan relaciones entre estos modelos, abstrayendo sus detalles internos (Salay et al. 2008).

Los megamodelos y macromodelos han sido aplicados en diferentes escenarios, como por ejemplo, la gestión de modelos (Bézivin et al. 2004), para facilitar la trazabilidad entre los modelos y los elementos que estos contienen (Barbero et al. 2007; Seibel et al. 2009). En el caso de Seibel et al. (2009), las relaciones encapsulan transformaciones que se aplican para resolver inconsistencias asociadas al cambio en los modelos. Perovich y Bastarrica y et al. (2009) emplean los megamodelos para diseñar arquitecturas software. El megamodelo define la arquitectura software, y las relaciones entre los modelos encapsulan las transformaciones de modelos asociadas a las distintas decisiones de diseño. Favre (2004a; 2004b) aplica los megamodelos para razonar acerca de las relaciones que pueden existir entre modelos en el desarrollo de software dirigido por modelos.

En el caso del modelo SUM este es creado a partir de transformaciones a partir de modelos de vistas (y viceversa), pudiéndose obtener vistas a partir del SUM. Los autores plantean además el uso de metamodelos ortogonales acoplando ontologías sobre el metamodelo de manera que se pueden establecer correspondencias entre elementos a nivel M2 en el modelo SUM para dar cobertura a instancias equivalente en modelos distintos, e incluso, a distintos niveles de abstracción.

Si bien no queda demasiado claro el propósito constructivo de la propuesta SUM, está más con propuestas como Archimate/TOGAF (Lankhorst et al. 2009), cuyo propósito es la descripción, a distintos niveles de abstracción y desde diferentes puntos de vista de sistemas software, no el de avanzar hacia la obtención de forma automatizada.

Con el fin de aumentar la expresividad que es posible plasmar con estos artefactos, nosotros proponemos el uso de multimodelos, que permiten la representación explicita de relaciones a nivel de elementos, en vez de a nivel de modelos. Este incremento de la expresividad es necesario para poder guiar el proceso de configuración del producto, la derivación y la evaluación y mejora de la arquitectura.

La definición de multimodelo que proponemos refina la propuesta por Barkmeyer et al. (2003) enfocándola al uso de múltiples lenguajes para describir múltiples puntos de vista del sistema y al establecimiento de relaciones entre sus elementos:

> ***Multimodelo:*** *Conjunto de modelos interrelacionados que representan distintos puntos de vista de un sistema o sistemas. Un punto de vista es una abstracción que produce la especificación del sistema con un propósito específico en mente. En un punto de vista es posible crear un modelo del sistema que contiene únicamente los objetos visibles desde ese punto de vista. Estos modelos se denominan vistas o modelos de vista. El multimodelo permite además, la definición de relaciones entre elementos definidos en vistas distintas, permitiendo así capturar la información que pueda perderse al separar la especificación del sistema atendiendo a diferentes intereses.*

La Figura 4.2 muestra la estructura genérica de un multimodelo[8]. Un modelo puede ser considerado como un grafo (Kühne 2005) que contiene entidades (nodos) y relaciones entre ellas (aristas) (Hebig et al. 2012).

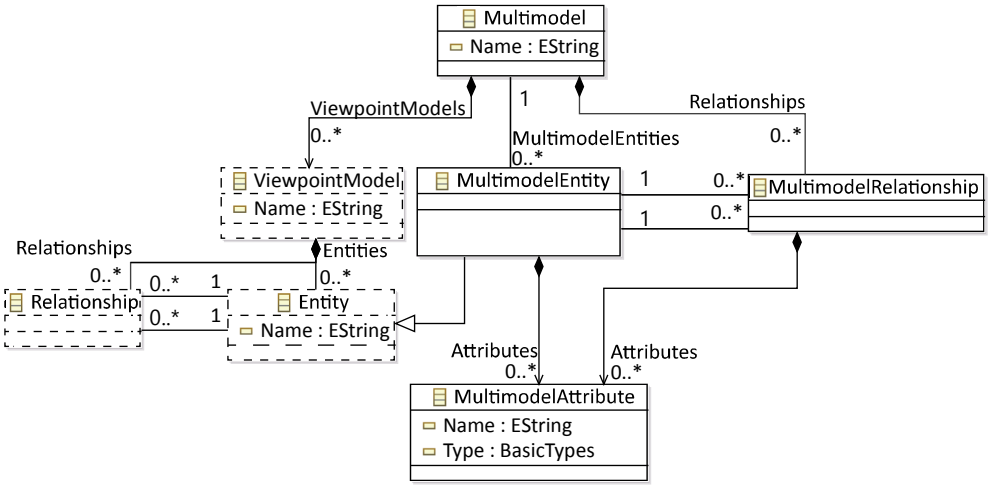


Figura 4.2 Estructura genérica de un multimodelo

El multimodelo estará compuesto por modelos de vista (*ViewpointModel*), que contienen entidades (*Entity*) y relaciones (*Relationships*). Habrá tantos modelos de vista como sean de interés en el dominio, y cada modelo de vista tendrá a su vez tantas entidades y relaciones como sean necesarias para representar los conceptos y relaciones en el correspondiente modelo de vista. Por este motivo, las clases ViewpointModel, Entity y Relationships se representan como clases genéricas (con línea de puntos).

[8] Empleamos una notación UML para ilustrar la estructura del multimodelo aunque en este caso la figura no se refiere a un metamodelo propiamente dicho.

Además, el multimodelo contendrá entidades propias (*MultimodelEntities*) que extienden las entidades en los modelos de vista para soportar las relaciones entre elementos de las distintas vistas (*MultimodelRelationship*). Esas relaciones involucran siempre a dos entidades del multimodelo (y no a entidades de los modelos de vista, manteniendo una baja cohesión entre entidades de los modelos de vista). Finalmente, tanto las entidades como las relaciones del multimodelo podrán contener atributos adicionales (*MultimodelEntities*) para capturar la información adicional asociada a esos constructos que no tendrían cabida en las entidades y relaciones de los modelos de vista.

Las relaciones entre los elementos de las diferentes vistas definidas en el multimodelo pueden tener diferentes semánticas, dependiendo de su naturaleza:

- **Implicación:** una entidad A en una vista, *implica* un elemento B en otra vista.
- **Co-implicación:** una entidad A en una vista, *implica* un elemento B en otra vista y viceversa.
- **Exclusión:** una entidad A en una vista, *excluye* un elemento B en otra vista y viceversa.
- **Relaciones de impacto:** un elemento de modelo A en una vista, impacta (positiva o negativamente) en un elemento B en otra vista. Se pueden asociar atributos adicionales con el fin de cuantificar dicho impacto o expresar su dirección.
- **Representación:** un elemento de modelo A en una vista, puede ser representado por un conjunto de elementos {B...N} en otra vista. Es una adaptación de la definición de las aristas-μ introducidas por Favre (2004a) para describir cómo los sistemas y modelos se pueden representar en otros modelos.
- **Descomposición (compuesto/parte):** una entidad compleja en una vista puede ser descompuesta en un conjunto de elementos {B...N} más sencillos en otras vistas. Esta relación se basa en la definición de aristas-δ introducidas por Favre (2005).

En el desarrollo de líneas de producto siguiendo la aproximación de desarrollo dirigido por modelos, el multimodelo va a jugar un papel dual. Por un lado, en la fase de ingeniería de dominio va a servir para especificar las diferentes vistas de interés, y por otro lado va a permitir la definición de relaciones entre elementos de dichas vistas con el fin de capturar y representar las carencias de información que podrían ser inducidas al representar el sistema/s por separado

desde distintos puntos de vista, atendiendo a distintos intereses. En la fase de ingeniería de aplicación se empleará el multimodelo para que las decisiones de diseño que se han de tomar durante la fase de derivación, evaluación y mejora se resuelvan atendiendo a criterios funcionales y no-funcionales.

4.2.2 Vistas del multimodelo para la especificación de líneas de producto

El problema de la derivación de arquitecturas de producto atendiendo a criterios funcionales y no-funcionales requiere, por un lado, de la representación de la variabilidad externa de la línea de productos en la vista de variabilidad, de la arquitectura (o al menos de la variabilidad de la misma) en la vista arquitectónica y, por otro lado, de los requisitos no-funcionales y de los atributos de calidad en la vista de calidad.

Para la evaluación de la arquitectura, se hará necesario representar las métricas que van a permitir analizar el grado de cumplimiento de los requisitos no-funcionales.

Por último, para la aplicación de transformaciones arquitectónicas atendiendo a como estas soportan los requisitos no-funcionales, será necesario representar, además, las posibles transformaciones arquitectónicas.

Así pues, para afrontar el problema que nos ocupa, el multimodelo estará compuesto de, al menos, cuatro vistas (de variabilidad, arquitectónica, de calidad y de transformaciones), cuyos detalles se describen en las siguientes subsecciones.

4.2.2.1 Vista de variabilidad

La vista de variabilidad del multimodelo, permite expresar la variabilidad externa de la línea de productos es decir, las características visibles por el usuario (ver Sección 2.1.2). Esta vista se ha definido utilizando un modelo de características con cardinalidades (Gómez 2012), dado que permite, no solamente la definición de relaciones entre características y multiplicidades, sino también la definición de restricciones mediante el lenguaje FMCL (*Feature Modeling Constraint Languaje*), un lenguaje formal con una sintaxis similar a OCL.

La Figura 4.3 muestra un extracto del metamodelo que soporta el modelo de características con cardinalidades. El modelo de características (*FeatureModel*) está compuesto de características (*Features*) que se pueden estructurar en grupos (*Groups*), relaciones (*Relationships*) y restricciones (*Constraints*). Es posible asociar atributos tanto a las características como a los grupos.

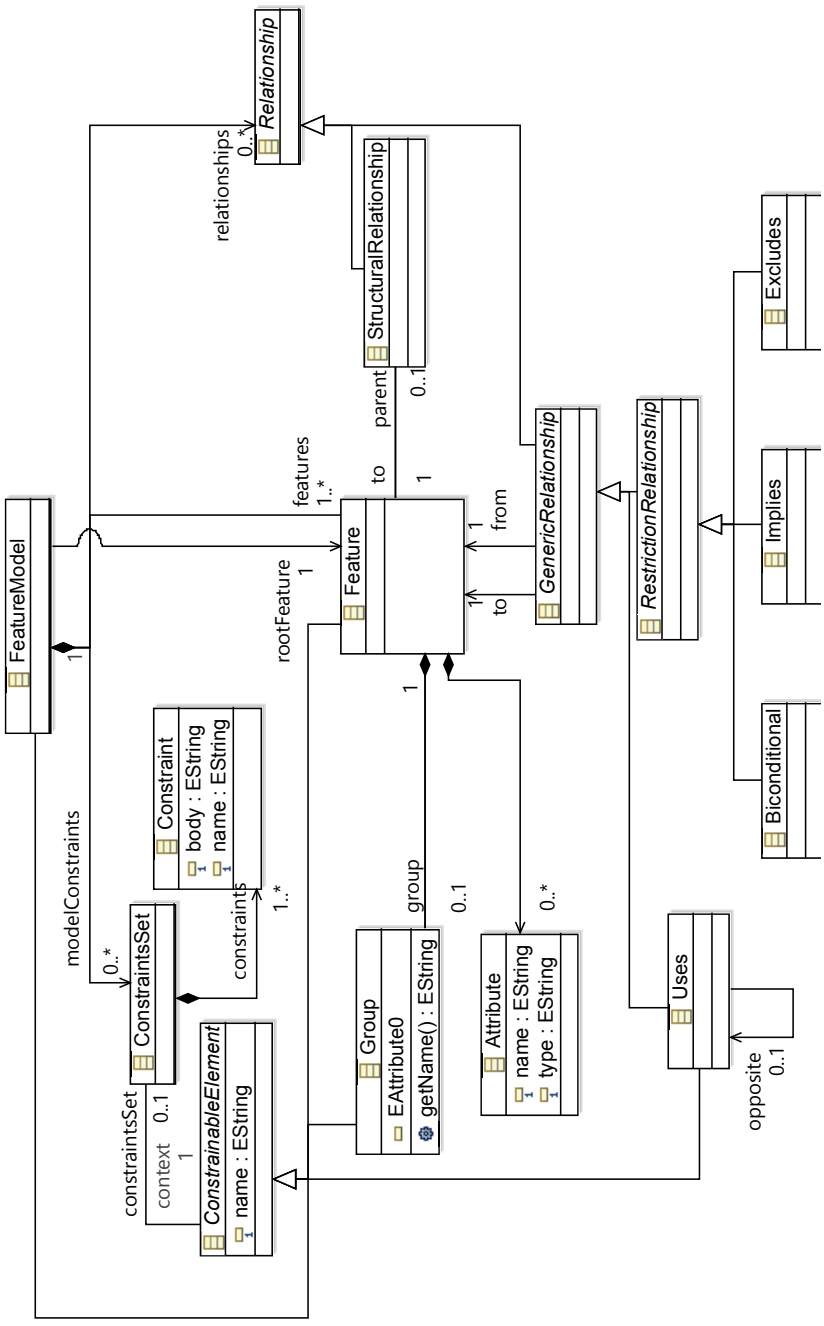


Figura 4.3 Extracto del metamodelo que da soporte a la vista de variabilidad (Gómez 2012)

4.2.2.2 Vista arquitectónica

Mediante la vista arquitectónica se expresa la variabilidad de la arquitectura de la línea de productos como variabilidad interna de la línea de productos (variedad en los activos software), permitiendo dar soporte a las diferentes características definidas en la vista de variabilidad. Se ha optado por representar la variabilidad de la arquitectura de la línea de productos empleando el estándar CVL. Esto permite i) emplear un lenguaje de modelado de variabilidad con diferentes abstracciones para su representación, que además provee de mecanismos para la derivación de los modelos (arquitectónicos), una vez la variabilidad has sido resuelta; y ii) proveer de una capa de abstracción adicional, de modo que el uso de QuaDAI se hace independiente de los lenguajes de descripción arquitectónica empleados y de los mecanismos de variabilidad que estos integran.La arquitectura de la línea de productos se representa explícitamente en el multimodelo mediante la vista de variabilidad arquitectónica, que en este caso es la vista de interés para el problema que nos ocupa. La arquitectura podrá representarse empleando otras vistas, y en el caso en que la variabilidad arquitectónica les afecte, estas serán tomadas como modelo base en la especificación de variabilidad arquitectónica empleando CVL. Es importante resaltar que es posible manejar las diferentes vistas arquitectónicas con el esquema propuesto, si estas están manejadas por un metamodelo monolítico (como es el caso de AADL, SysML o UML), puesto que el modelo base será el súper-conjunto de todas las vistas arquitectónicas y representaremos la variabilidad arquitectónica en todas y cada una de las vistas, asociando los puntos de variabilidad arquitectónica CVL de cada vista a una VSpec que podrá resolverse, de forma consistente, mediante los mecanismos de variabilidad de CVL descritos en la Sección 2.4.

La Figura 4.4 muestra un extracto del metamodelo de CVL que da soporte a la vista arquitectónica con la estructura de enlace entre las VSpecs, los puntos de variación y las entidades del modelo base. Una VSpec se asocia con varios puntos de variación (cuyos tipos se hayan descritos en la Sección 2.3.4). En el caso de las VSpecs configurables (*CVSpecs*) se asocia un interfaz de variabilidad (*VInterface*) y una unidad configurable (*ConfigurableUnit*), que a su vez pueden incluir múltiples VSpecs. En el caso de los puntos de variación de selección (*ChoiceVariationPoint*) se enlazaran con objetos (*ObjectHandle*), como en el caso del punto de variación de existencia de objeto (*ObjectExistence*), con enlaces (*ObjectLink*), como en el caso de existencia de enlace (*LinkExistence*), con una combinación de ambos, como en el caso de la sustitución de extremo de conexión (*LinkEndSubstution*), o con la especificación de un valor, en el caso de la asignación de una variable (SlotAssignment).

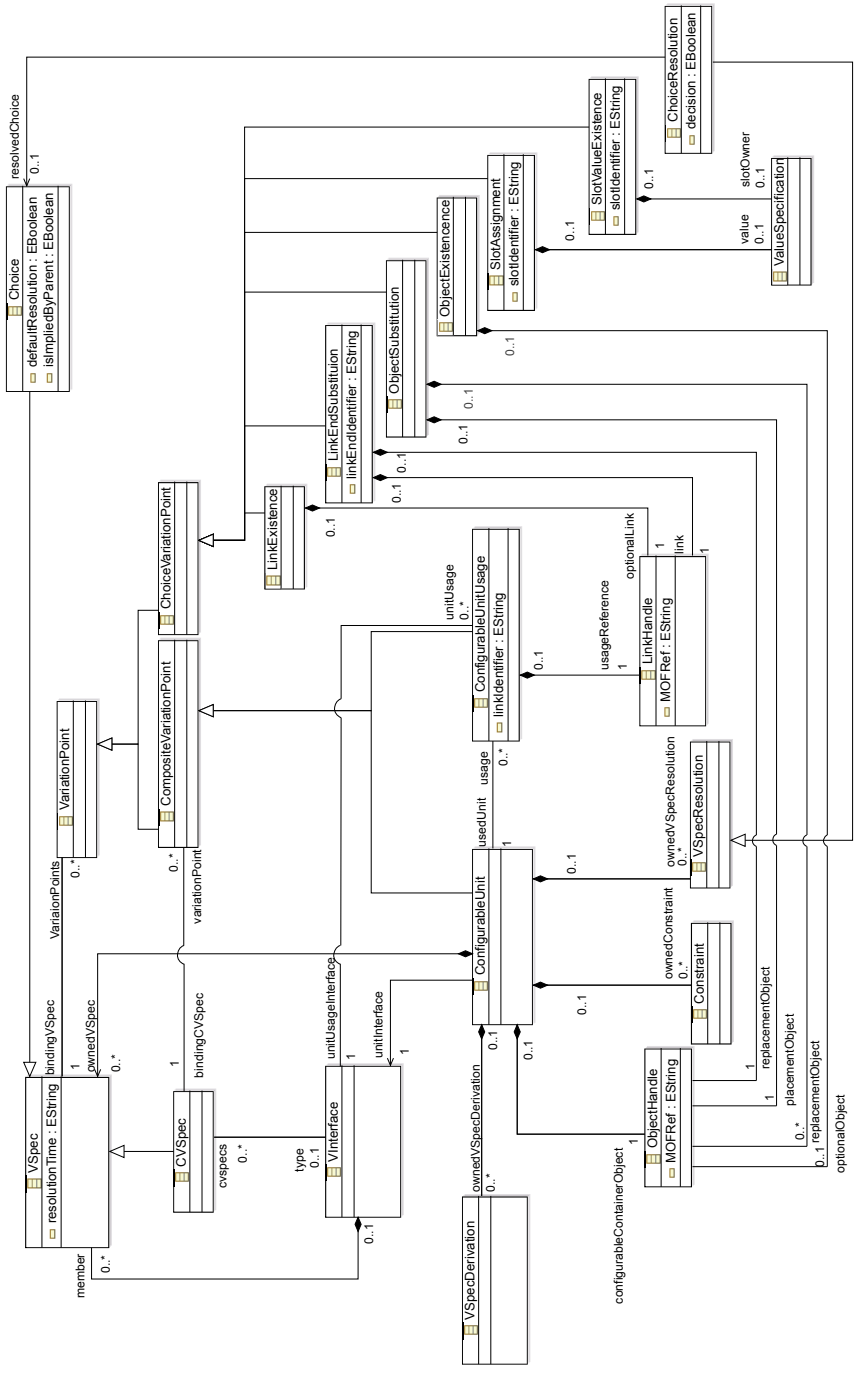


Figura 4.4 Extracto del metamodelo CVL: CVSpecs y unidades configurables

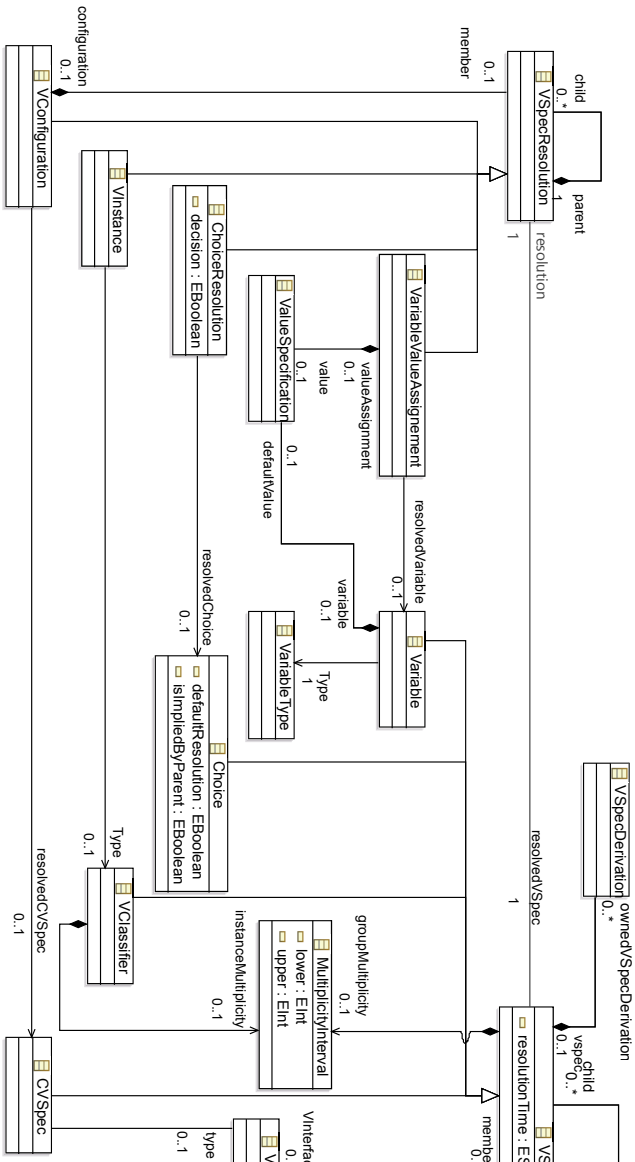


Figura 4.5 Extracto del metamodelo CVL: VSpecs y VSpecResolutions

La Figura 4.5 muestra un extracto del metamodelo del estándar CVL que soporta la vista arquitectónica del multimodelo. Las VSpecs son resueltas mediante las *VSpecResolutions*, las cuales se puede resolver positiva o negativamente en el caso de las VSpecs de elección, asociar un valor en el caso de las VSpecs variables o emplear una configuración VSpec (*VConfiguration*) (que a su vez se compondrá de un árbol de *VSpecResolutions)* en el caso de las VSpecs compuestas (*CVSpecs*).

4.2.2.3 Vista de calidad

La vista de calidad del multimodelo se ha definido a partir de una extensión del modelo de calidad para líneas de productos, definido inicialmente por Montagud (2009). Este modelo de calidad (Montagud 2009), está basado en la norma ISO/IEC 25000 (SQuaRE) (ISO 2005) para dar soporte a las actividades de aseguramiento y evaluación en el desarrollo de líneas de producto software y contiene la descomposición jerárquica de características de calidad en sub-características y cuyos elementos finales son los atributos de calidad, tal como se puede apreciar en el fragmento del metamodelo de la vista de calidad que se muestra en la Figura 4.6.

En dicho modelo es posible incluir las métricas empleadas para cuantificar cada uno de los atributos de calidad. El modelo permite definir dos tipos de métricas: métricas básicas (*base measure)* y métricas derivadas (*derived measure*). Una métrica derivada es el resultado de una combinación de métricas básicas o derivadas mediante una función de medición (*measurement function*). Cada métrica puede tener diferentes operacionalizaciones[9]dependiendo del artefacto software que se está midiendo en cada caso. También es posible definir las relaciones entre atributos de calidad mediante el concepto de impacto (*impact*) que representa, mediante un valor numérico, cómo un determinado atributo impacta sobre otro positiva o negativamente (Montagud 2009).

Hemos propuesto una extensión a este modelo de calidad para poder representar tanto los requisitos no-funcionales de la línea de producto, como los del producto en desarrollo. Los requisitos no-funcionales se representan como restricciones al modelo de calidad que habrán de ser satisfechos cuando se evalúa el artefacto software para el que han sido definidos.

[9] La operacionalización de una métrica consiste en establecer una correspondencia entre la descripción genérica de la métrica y los conceptos representados en el artefacto software a medir (ISO 2005; Fernandez et al. 2011).

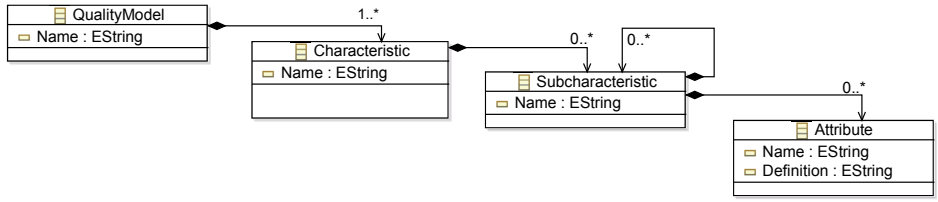


Figura 4.6 Jerarquía de características de calidad en la vista de calidad

La Figura 4.7 muestra un extracto del metamodelo que da soporte a la vista de calidad del multimodelo. Las metaclases por encima de la línea de puntos han sido añadidas o modificadas desde la versión inicial del modelo de calidad (Montagud 2009) con el fin de representar los conceptos asociados a los requisitos no-funcionales. Un requisito no-funcional (*NFR*) puede tener asociada una restricción, tanto en lenguaje natural como en OCL, y puede estar asociado a múltiples atributos de calidad (*Attribute*) si se trata de requisitos que implican una mezcla de funcionalidades o variaciones. Los requisitos no-funcionales se asociarán a un determinado tipo de artefacto (*EntityClass*), como pueda ser la arquitectura de producto o la interfaz de usuario.

Los requisitos no-funcionales también se podrán cuantificar, del mismo modo que los atributos de calidad, en términos de métricas (básicas o derivadas). Las métricas pueden tener asociadas unas escalas y unidades con las que expresar sus valores. Los requisitos no-funcionales cualitativos pueden ser transformados en escalas cuantitativas discretas para su gestión. Los requisitos no-funcionales más complejos pueden ser definidos en la vista de calidad mediante restricciones OCL que podrán ser evaluadas mediante evaluadores OCL como el que integra el *plug-in* OCLTools (Eclipse 2013b).

La vista de calidad permite establecer criterios de decisión (*DecisionCriteria*), que pueden llevar asociados umbrales (*threshold*) a los que se puede además asociar una acción en caso de incumplimiento del mismo. Dichos umbrales pueden ser de dos tipos:

- Restricciones fuertes (*HardConstraints*), que definen umbrales que han de ser satisfechos de manera obligatoria y en caso contrario, el producto pasaría a ser considerado configuración no valida. Están definidos mediante un nivel inferior y superior.

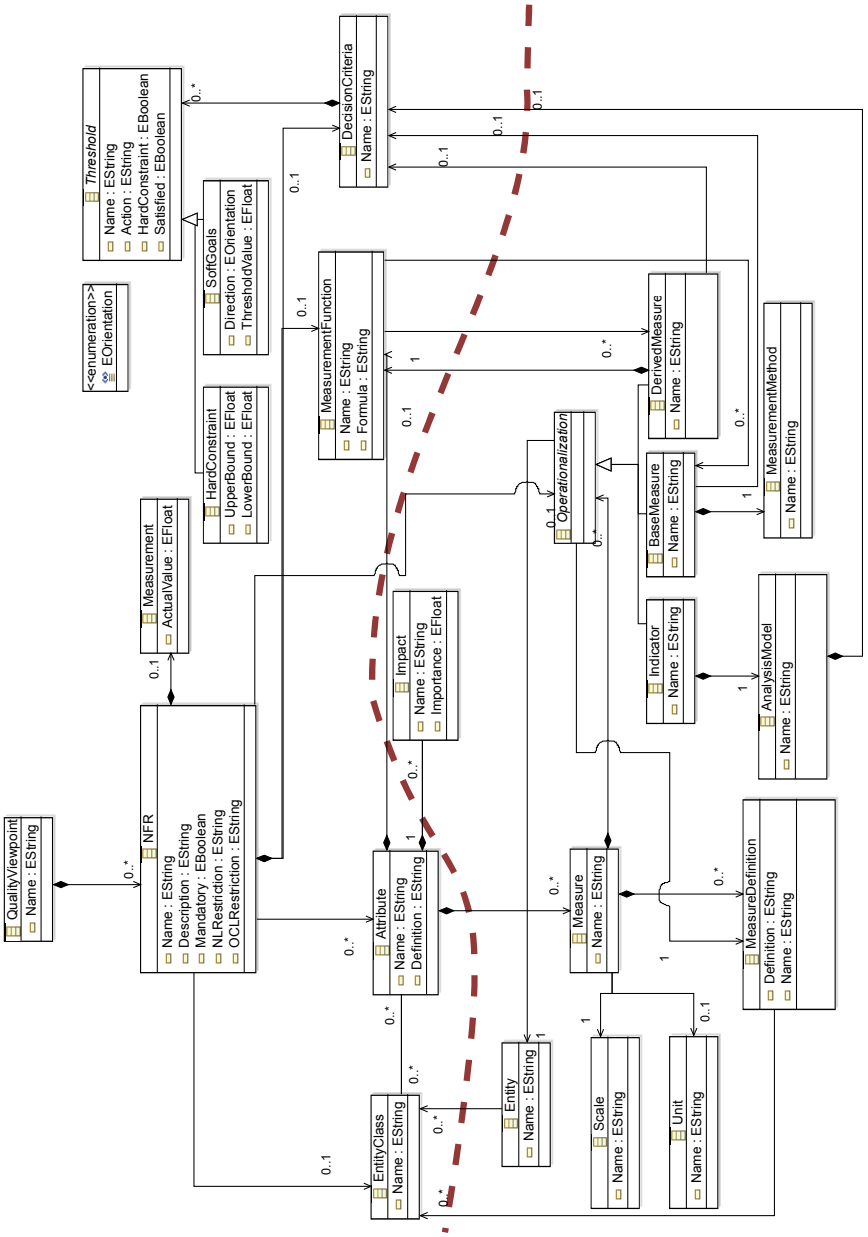


Figura 4.7 Extracto del metamodelo de la vista de calidad

- Metas (*SoftGoals*), empleados para expresar criterios decisión *deseables,* que no hacen que el producto sea válido o no, sino que expresan un determinado criterio a maximizar o minimizar. Se definen mediante el atributo de dirección que expresa la dirección en la que se considera que los valores mejoran (positivo para aquellos requisitos no-funcionales cuyo valor se pretende maximizar como la capacidad o la fiabilidad y negativo para aquellos que se pretende minimizar, como el coste o el tiempo de respuesta).

4.2.2.4 Vista de Transformaciones

La vista de transformaciones contiene la representación explícita de las transformaciones que integran el plan de producción en una línea de productos software en un entorno de desarrollo dirigido por modelos, incluyendo aquellas transformaciones que son alternativas en alguna decisión de diseño. De forma genérica, en un proceso de transformación pueden aparecer alternativas (decisiones de diseño) si un determinado conjunto de entidades del modelo origen admite diferentes representaciones en el modelo destino. La aplicación de cada alternativa puede generar modelos destino alternativos, que pueden ser equivalentes en cuanto a la funcionalidad pero pueden diferir en cuanto a sus atributos de calidad.

La Figura 4.8 muestra un extracto del metamodelo del modelo que da soporte a la vista de transformaciones, que describe la estructura jerárquica que nos permite describir procesos de transformación. Cada transformación (*Transformation*) puede contener diferentes decisiones de diseño (*DesignDecision*), que a su vez, tendrán como mínimo dos alternativas. Cada alternativa de diseño puede estar materializada por un conjunto de reglas (*TransformationRule*). La representación explícita de las reglas a nivel de proceso de transformación, nos permite que estas reglas puedan ser reutilizadas en diferentes alternativas o diferentes decisiones de diseño, aportando así modularidad y reutilización.

4.2.3 Relaciones inter-vistas

Las actividades del método QuaDAI van a requerir de la definición de relaciones entre elementos de la vista de variabilidad, arquitectónica y de calidad para el caso de la actividad de derivación, y de relaciones entre elementos de la vista de transformaciones y la vista de calidad.

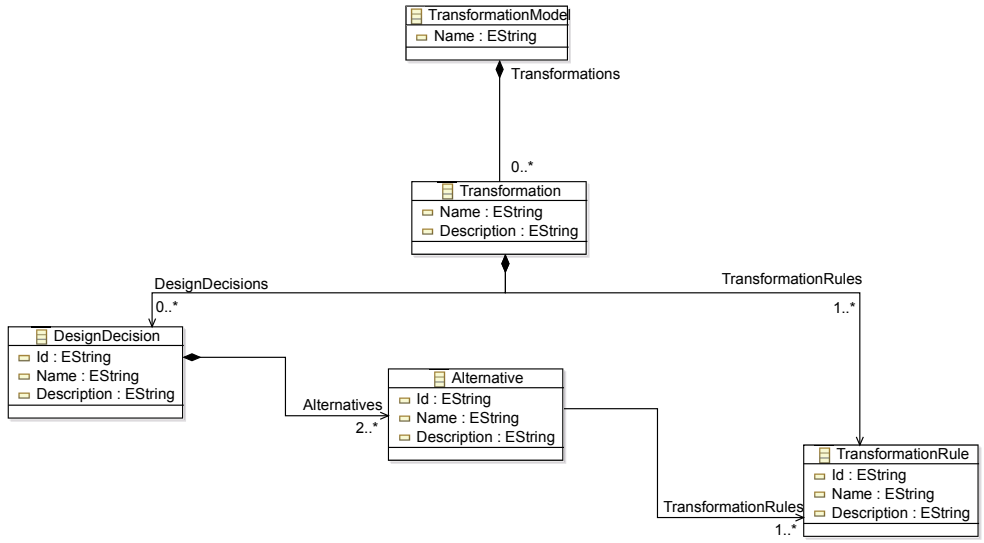


Figura 4.8 Vista de Transformaciones

Más concretamente para las tareas relacionadas con la actividad de derivación, se definen las siguientes relaciones:

- **Relación entre características y VSpecs:** Tal como se describe en la sección 4.2.2.1, la vista de variabilidad permite representar la variabilidad externa, aquellas características visibles por el usuario que distinguen a cada uno de los productos de la línea de productos. Estas características van a implicar cierta variabilidad a nivel arquitectónico, que dotará a la arquitectura de la flexibilidad necesaria para adaptarse a los requisitos funcionales (y no-funcionales) de la línea de productos, y que se representa en la vista arquitectónica (véase Sección 4.2.2.2). Esta variabilidad arquitectónica va a ser mapeada mediante una relación explícita en el multimodelo entre las características (*Features*) de la vista de variabilidad y las especificaciones de variabilidad (*VSpecs*) de la vista arquitectónica. Esta relación va a permitir la obtención de la resolución del modelo CVL mediante un proceso de transformación de modelos, a partir de la selección de características. Esta relación se cataloga como relación de *descomposición* atendiendo a la clasificación introducida en la Sección 4.2.1, aunque realmente se trata de un caso particular de relación RealizadoPor (*IsRealizedBy*) formalizada por Janota y Botterweck (2008) y que relaciona, con una semántica formal, características y componentes arquitectónicos, aunque en nuestro caso

estos componentes se abstraen mediante la capa de especificación de variabilidad arquitectónica CVL que da lugar a la vista arquitectónica del multimodelo.

- **Relación entre características y RNFs:** De forma análoga, un requisito no-funcional puede estar soportado por la selección de una serie de características que *realizan* dicho requisito no-funcional (por ejemplo, en una aplicación web un requisito no-funcional de seguridad puede verse satisfecho por la selección de la característica *protocolo https*). Esta relación va a ser empleada en el proceso de configuración, para seleccionar aquellas características que son necesarias para cubrir los requisitos no-funcionales.

- **Relación entre VSpecs y RNFs:** Al igual que en el caso de las características, un determinado punto de variación arquitectónica puede realizar por sí mismo o satisfacer un determinado requisito no-funcional. La relación será empleada en el proceso de derivación para resolver positivamente el VSpec asociado.

- **Relación entre características y atributos de calidad:** La configuración del producto atendiendo a criterios funcionales y no-funcionales, requiere establecer relaciones de impacto (ver Sección 4.2.1) entre las características del modelo de variabilidad y los atributos de calidad. Esta relación de impacto permite representar cómo una determinada característica influye de manera positiva o negativa sobre un atributo de calidad. Esta influencia se cuantifica mediante un peso que representa la importancia relativa de dicha influencia.

- **Relación entre VSpecs y atributos de calidad:** Del mismo modo que para el caso de las características, se hace necesario representar como un determinado punto de variación arquitectónica, representado mediante una VSpec, impacta en los atributos de calidad. Esta relación de impacto permite representar cómo la selección de un determinado punto de variación de la arquitectura, influye de manera positiva o negativa sobre un atributo de calidad. Esta influencia se cuantifica mediante un peso que representa la importancia relativa de dicha influencia. Esta relación va a permitir una mayor flexibilidad a la hora de establecer impactos, puesto que en ocasiones la variabilidad arquitectónica puede ser más rica que la variabilidad externa y las variantes pueden ir orientadas a ofrecer variabilidad en los atributos de calidad (mediante componentes alternativos de la arquitectura que

realizan la misma característica pero con distintos niveles de atributos de calidad).

Para dar soporte a las tareas de la actividad de transformación de la arquitectura se define la siguiente relación:

- **Relación entre transformaciones alternativas y atributos de calidad:** La aplicación de una determinada alternativa de transformación en una decisión de diseño va a dar lugar a modelos equivalentes desde el punto de vista funcional, pero diferentes con respecto a sus atributos de calidad. Es por ello que para permitir automatizar la selección de dichas transformaciones, se define la relación de impacto entre transformaciones alternativas de la vista de transformaciones y la vista de calidad que permite representar la influencia de cada alternativa sobre los atributos de calidad. Esta influencia se cuantifica mediante un peso que representa la importancia relativa de dicha influencia.

Todas estas relaciones se definen en el metamodelo que da soporte al multimodelo, sin necesidad de alterar la estructura de los metamodelos que soportan cada una de las vistas descritas en la sección 4.2.2, aplicando la arquitectura que se describe a continuación.

4.2.4 Arquitectura del metamodelo del multimodelo

Para la definición del metamodelo que soporta el multimodelo, se ha seguido una arquitectura en dos capas. Por un lado, se ha definido un metamodelo del *núcleo* del multimodelo *(CoreMultimodel)* que soporta la implementación en *Eclipse Modeling Framework* (*EMF*) (Eclipse 2013b)de la estructura descrita en la Figura 4.2. A la hora de definir un multimodelo *concreto* se extenderán las metaclases de este metamodelo núcleo. Esta arquitectura ha sido definida con el fin de automatizar la generación de editores de multimodelos. Además, se ha desarrollado una infraestructura[10] que permite obtener, de manera automática, un editor con el que gestionar el multimodelo una vez este ha sido definido extendiendo el núcleo del multimodelo, independientemente de la naturaleza de las vistas involucradas.

[10] Los detalles de la infraestructura se describen en la sección 7.1.

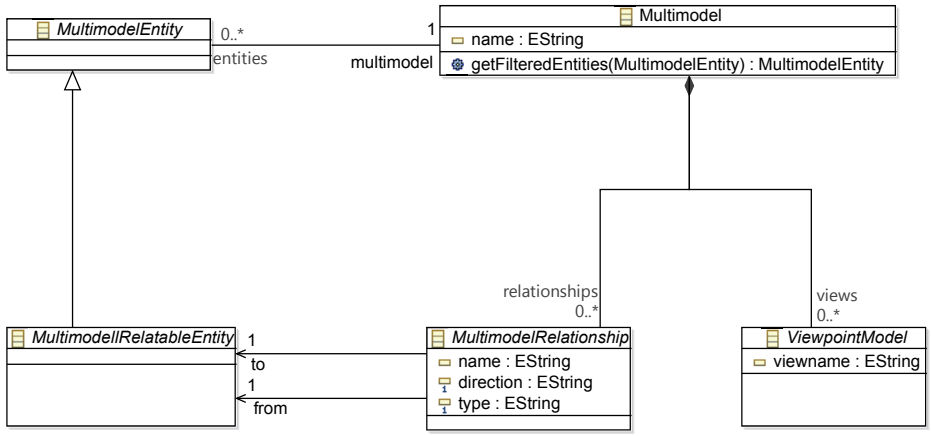


Figura 4.9 Metamodelo nucleo del multimodelo

Por otro lado, la segunda capa de la arquitectura definida es el propio metamodelo que soporta el multimodelo *concreto*, donde se van a representar las distintas vistas del multimodelo, las metaclases que participan en las relaciones entre elementos en las distintas vistas y las relaciones entre dichas metaclases. Los distintos metamodelos que soportan cada una de las vistas (de variabilidad, arquitectónica, de calidad y de transformaciones) se referencian como recursos externos en el multimodelo.

Para minimizar el acoplamiento entre los metamodelos, se aplica el patrón *proxy* (Gamma et al. 1995) para extender las metaclases que participan en dichas relaciones, así como las metaclases contenedoras de cada una de las vistas.

El metamodelo del multimodelo concreto incluirá una metaclase *proxy* que hereda de *ViewpointModel* y de la metaclase del metamodelo externo que representa el modelo en dicho metamodelo externo. Por ejemplo, en el caso de punto de vista de variabilidad, la metaclase en cuestión heredará de *ViewpointModel* y de *FeatureModel* (ver Figura 4.3).

Para cada clase que participa en una relación crearemos una metaclase *proxy*. En nuestro caso, se aplica el patrón a las metaclases *Feature* del metamodelo que da soporte a la vista de características, *VSpec* del metamodelo CVL que da soporte a la vista arquitectónica, *Alternative* del metamodelo que da soporte a la vista de transformaciones y, por último, *Attribute* y *NFR* del metamodelo que da soporte a la vista de calidad, dando como resultados las clases *EFeature*,

EVSpec, *EAlternative*, *EAtribute* y *ENFR*[11]. Estas clases también heredan de la metaclase *RelatableEntity* que soporta la definición de las relaciones entre elementos de las distintas vistas.

En el caso de las metaclases EFeature, EAtribute y ENFR, se ha añadido un atributo adicional *Selected* e *Importance* a cada una de ellas para capturar la información necesaria en el proceso de derivación.

En la Figura 4.10 se muestra un extracto del metamodelo que soporta el multimodelo. En el recuadro con línea punteada, se muestra la estructura de las diferentes vistas. El recuadro con línea de rayas, resalta las metaclases *proxy* resultantes de la aplicación del patrón *proxy* a las entidades de interés de las diferentes metaclases.

La definición de las relaciones se lleva a cabo creando una metaclase que extienda *MultimodelRelationship* y que, como parámetros, recibirá la metaclase origen y destino, así como la naturaleza (bidireccional o unidireccional) y el tipo de relación (de entre los tipos descritas en la Sección 4.2.1). Con esta información, la infraestructura que da soporte al multimodelo es capaz, en tiempo de ejecución, de definir relaciones entre las instancias de las metaclases involucradas. La Figura 4.11 muestra la definición de las relaciones (descritas en la sección 4.2.3), mediante las metaclases que extienden la metaclase *MultimodelRelationship*. Es posible definir atributos asociados a las relaciones, como es el caso de las relaciones de impacto *VSpectToQualityAttribute*, *FeatureToQualityAttribute* y *AlternativeTransformationToQualityAttribute* que se muestran en la Figura 4.11.

4.3 Proceso de derivación, evaluación y mejora de arquitecturas

En esta sección se describe el proceso que da soporte al método QuaDAI de derivación, evaluación y mejora de arquitecturas que emplea el multimodelo, introducido en la sección anterior, como artefacto conductor.

[11] Las metaclases resultantes de la aplicación del patrón proxy siguen la convención de nomenclatura *E+NombreClase*, donde *NombreClase* es el nombre de la clase en el metamodelo externo.

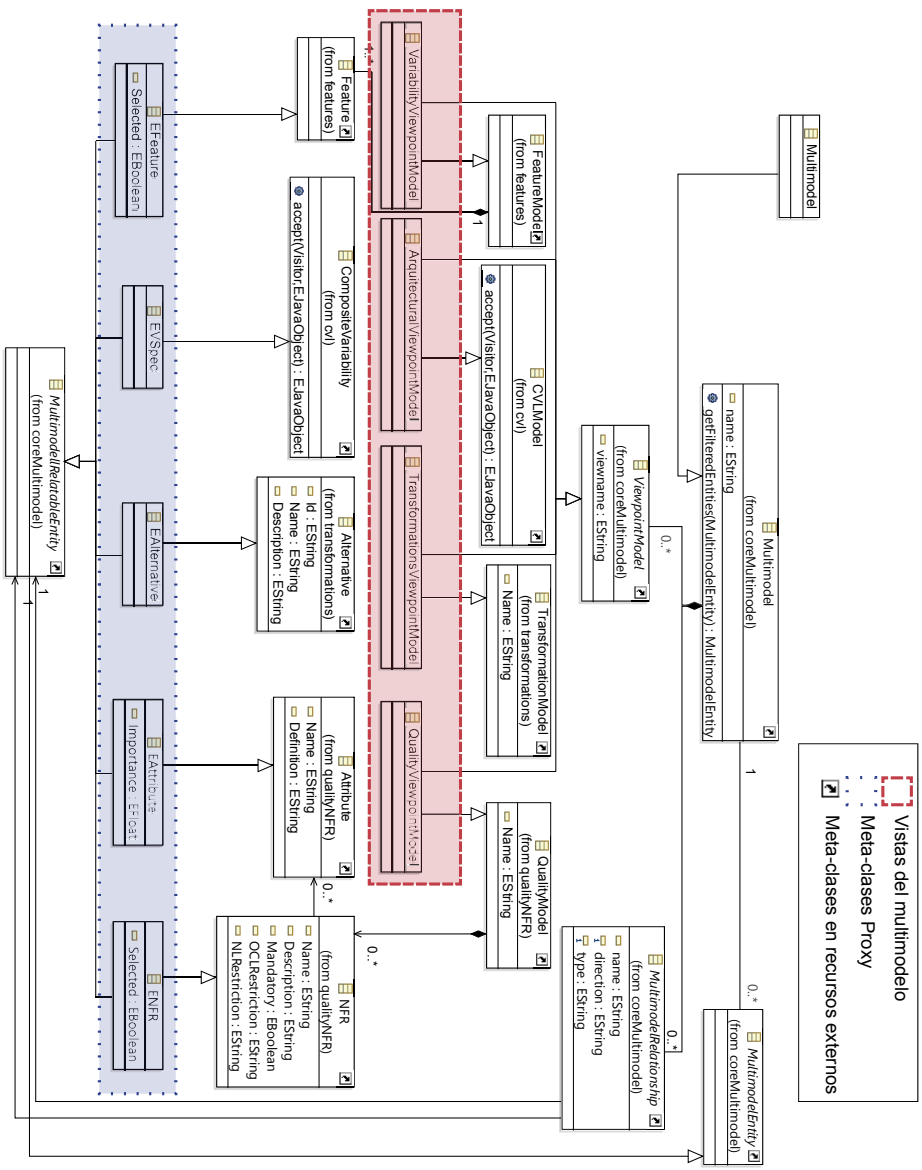


Figura 4.10 Extracto del metamodelo del multimodelo: Vistas y metaclases proxy

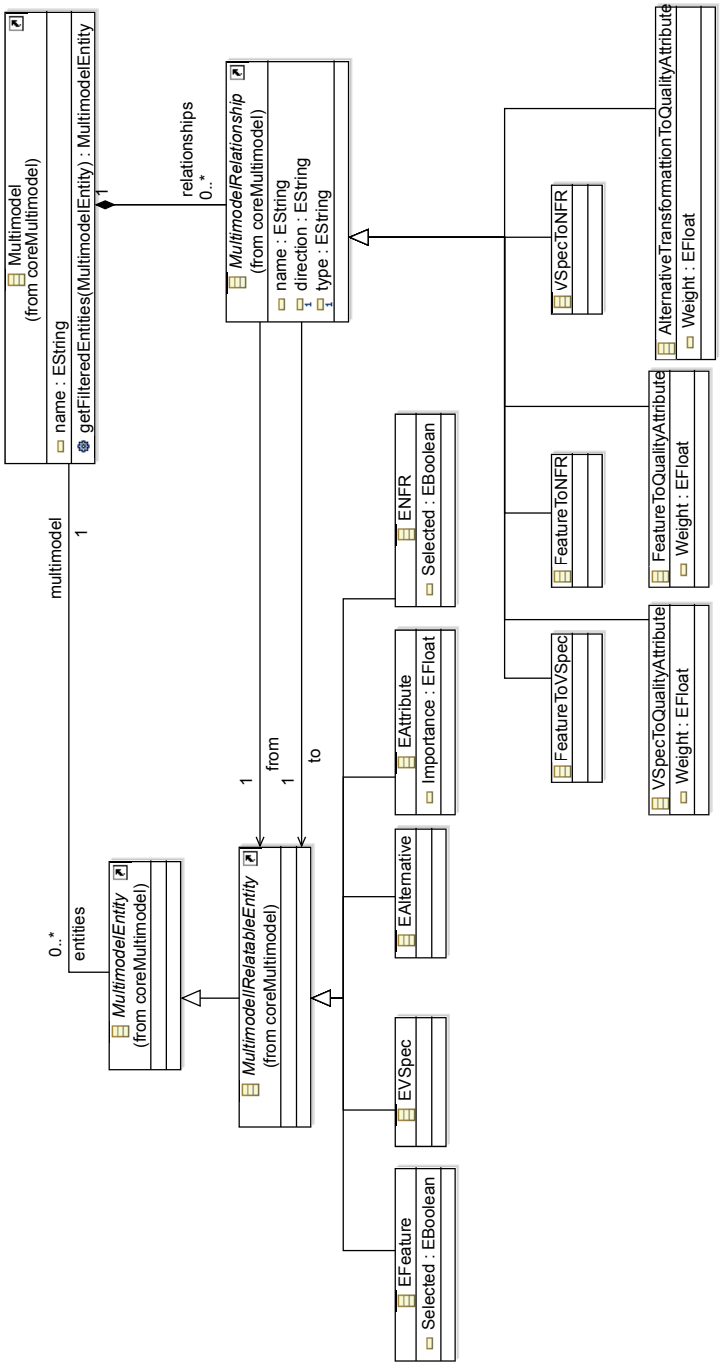


Figura 4.11 Extracto del metamodelo del multimodelo: relaciones entre elementos de las vistas

La definición del proceso se ha llevado a cabo haciendo uso del estándar para la ingeniería de procesos de la ingeniería del software y de sistemas SPEM v2.0 (*Software and Systems Process Engineering Metamodel Specification*) (Object Management Group 2008b) propuesto por la OMG.

SPEM es uno de los estándares de especificación de procesos más conocidos y aplicados en la industria software, destinado a proporcionar una definición detallada de los procesos que servirán de guía clara a los roles involucrados en la realización de los procesos. La descripción de procesos de forma completa, concisa y no ambigua es, en sí misma, una forma de mejorar la calidad asegurando la homogeneización de las tareas que constituyen el proceso.

La sección 4.3.1 introduce la notación SPEM v2.0, su estructura y las ventajas que aporta su uso como lenguaje de especificación de procesos.

En la sección 4.3.2 se describe cada una de las fases y actividades de QuaDAI.

4.3.1 Introducción a SPEM v2 como lenguaje de modelado de procesos software

El estándar SPEM v2 se basa en un metamodelo que define una notación precisa para la definición y especificación de modelos de procesos de ingeniería del software y de sistemas.

El alcance de SPEM v2 se limita a los elementos mínimos necesarios para definir dichos procesos, omitiendo la información propia del dominio o disciplina en particular, pero permite el modelado de métodos y procesos de diferentes estilos, culturas, nivel de formalismo o paradigmas de ciclo de vida. Es por ello que es un candidato para su uso, no únicamente para la especificación de procesos de desarrollo de software, sino también de las actividades de aseguramiento y evaluación de la calidad, en las que es necesario definir las guías y artefactos implicados en el proceso de manera clara y precisa.

La idea principal de SPEM v2 es representar los procesos en base a los tres elementos básicos que se muestran en la Figura 4.12: *Tarea, Rol* y *Producto de trabajo* (Ruiz y Verdugo 2008). Las *tareas* representan el esfuerzo a realizar, los *roles* representan la responsabilidad de realización de las tareas y los *productos de trabajo* representan las entradas requeridas y las salidas producidas por cada una de las tareas. Por tanto, la especificación del proceso se especifica en los términos "Quien (rol) realiza qué (tarea) con el fin de obtener, a partir de una serie de entradas (productos de trabajo), un resultado (productos de trabajo)".

La especificación de métodos y procesos empleando SPEM v2 se divide en la especificación, por un lado, de los contenidos del método como elementos reusables, y por otro, de la descripción del proceso. El contenido del método se expresa mediante las definiciones de los productos de trabajo, de los roles y

de las tareas. El elemento principal de la especificación de un proceso SPEM son las actividades que podrán ser anidadas para definir estructuras de desglose y ser relacionadas con otras actividades para definir el flujo de trabajo. Los elementos de orientación, tales como las guías, documentos técnicos, listas de verificación, ejemplos o planes de trabajo, se definen para dar soporte tanto al contenido del método como a los procesos específicos.

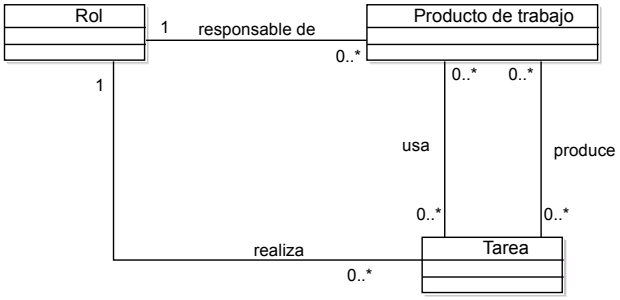


Figura 4.12 Elementos para la representación de procesos en SPEM v2.0

La Figura 4.13 muestra cómo los conceptos clave se posicionan para representar el contenido del método o el proceso en sí mismo. Los elementos de orientación, se definen en la intersección dado que van a proveer de cobertura, tanto a la definición del contenido del método, como al proceso.

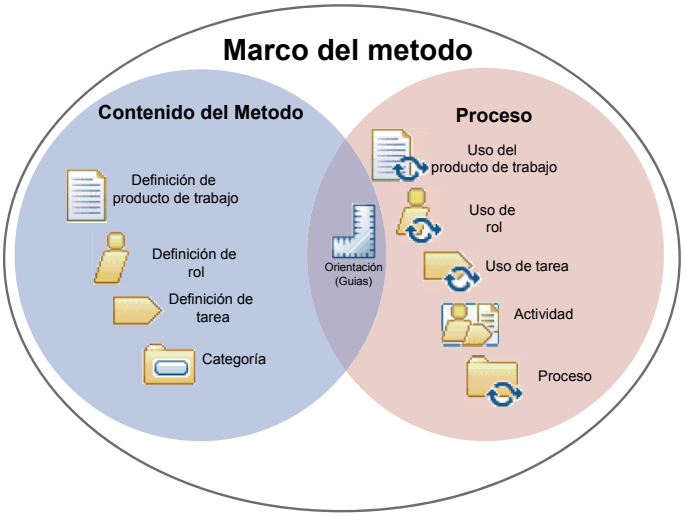


Figura 4.13 Terminología clave y su correspondencia en contenido del método y la definición de proceso en SPEM v2.0

Las Tabla 4.1 resume las primitivas para el modelado de contenidos del método y procesos definidos en el estándar SPEM v2.0.

Tabla 4.1 Primitivas de modelado de SPEM v2.0

Icono	*Nombre*	*Descripción*
	Definición de rol	Conjunto de habilidades, competencias y responsabilidades de un individuo o grupo.
	Definición de tarea	Describe la unidad de trabajo a ser asignada y manejada. Identifica el trabajo a ser desarrollado por los roles. Una tarea puede ser subdividida en varios pasos.
	Definición de producto de trabajo	Define el producto utilizado o producido por las tareas. Hay dos tipos de productos: Artefactos de carácter tangible (modelos, documentos, código fuente) Entregables que empaquetan productos para su entrega a un cliente interno o externo. Se pueden agrupar o asociar entre sí mediante relaciones de agregación o impacto.
	Categoría	Clasifica elementos como tareas, funciones y productos en base a criterios establecidos por el ingeniero de procesos. Hay diferentes tipos de categorías: grupos de roles (para roles), disciplinas (para las tareas) y dominios (para los productos).
	Guías	Proporciona información adicional con respecto a otros elementos. Los subtipos de guías pueden ser (entre otros) activos reutilizables, guías, documentación, plantillas etc.
	Uso de rol	Representa el rol que lleva a cabo la tarea o actividad dentro del proceso. Hace referencia a la definición del rol (contenido del elemento).
	Uso de tarea	Representa una tarea dentro del proceso. Hace referencia a la definición de la tarea (contenido del elemento).
	Uso de producto de trabajo	El producto de trabajo representa la entrada o salida de una actividad o tarea. Se refiere a la definición de un producto de trabajo (elemento contenido).
	Actividad	Una actividad representa un conjunto de tareas que se ejecutan en el proceso, junto con sus roles y productos de trabajo asociados. Si se representa solo el conjunto de tareas (sin roles o productos de trabajo) es posible emplear actividades o fases (este último elemento se incluye para soportar la compatibilidad con versiones anteriores). Si el conjunto de tareas se repite varias veces, es posible modelarlo mediante el elemento "iteración".
	Fase	
	Iteración	
	Paquete de procesos	Representa un paquete que contiene todos los elementos para un determinado proceso.
	Paso	Un paso describe una parte significativa y coherente de la labor general descrita para la definición de una tarea.

4.3.2 Proceso de derivación, evaluación y mejora de arquitecturas en SPEM v2.0

El método QuaDAI se apoya en un proceso conducido por transformación de modelos, que integran el conjunto de actividades y tareas para la obtención de arquitecturas que cumplan con los requisitos de calidad, siguiendo la aproximación de desarrollo de software dirigido por modelos.

La Figura 4.14 muestra el diagrama de actividades con la jerarquía de actividades del proceso y los roles responsable de llevar a cabo cada actividad. En el proceso hay tres roles principales involucrados:

- *Ingeniero de aplicación,* es responsable de obtener, en base a los requisitos funcionales y no-funcionales del producto en desarrollo, la configuración de características que mejor se adapte a dichos requisitos.
- *Arquitecto de aplicación,* es el responsable del desarrollo y obtención de la arquitectura de un producto concreto (van der Linden et al. 2007).
- *Evaluador,* es responsable de llevar a cabo la evaluación de la arquitectura, con el fin de detectar la capacidad de la arquitectura para soportar el comportamiento y los atributos de calidad requeridos (Clements et al. 2011).

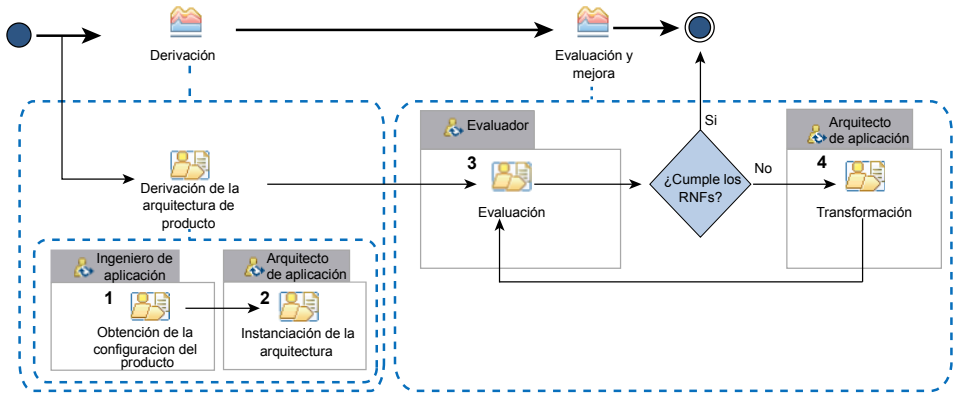


Figura 4.14 Diagrama de actividades general del proceso QuaDAI

El ingeniero de aplicación lleva a cabo la actividad de obtención de la configuración del producto (1). El arquitecto de aplicación lleva a cabo la instanciación de la arquitectura (2) y la aplicación de transformaciones arquitectónicas (4). El evaluador es el encargado de evaluar la arquitectura de producto (3) tras su derivación o tras cada proceso de transformación.

Las siguientes sub-secciones describen las fases principales y su descomposición en actividades y tareas.

4.3.2.1 Fase de derivación

El objetivo de esta fase es obtener una primera versión de la arquitectura del producto en desarrollo a partir de los requisitos especificados por el ingeniero de requisitos de aplicación (van der Linden et al. 2007). La Figura 4.15 muestra el diagrama de actividad con la descomposición en tareas de la actividad de derivación. La actividad de obtención de la configuración del producto (1), llevada a cabo por el ingeniero de aplicación, se desglosa en las tareas de configuración del producto (1.1) y validación de consistencia de dicha configuración (1.2). La actividad de instanciación de la arquitectura (2), llevada a cabo por el arquitecto, se desglosa a su vez en las tareas de generación del modelo de resolución CVL (2.1) y, a partir de este último, en la de materialización de la arquitectura (2.2).

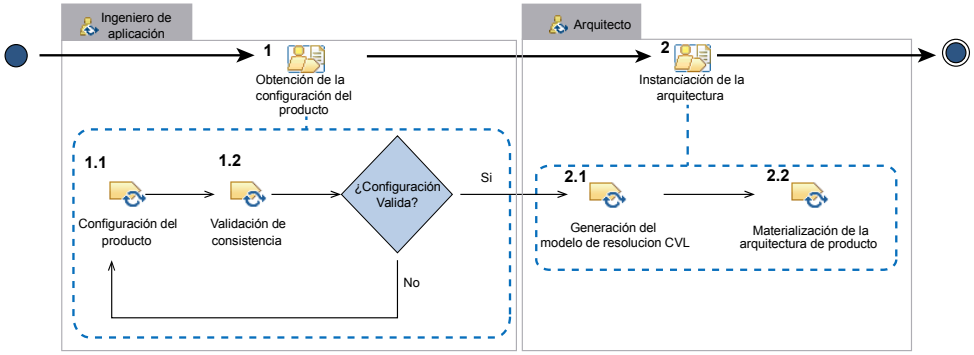


Figura 4.15 Diagrama de actividad de la fase de derivación

En la Figura 4.16 se muestra el diagrama de actividad detallado con las entradas, las salidas y el rol responsable de cada una de las tareas que forman parte de la fase de derivación.

La tarea de **configuración del producto** (1.1) toma como entradas:

- La vista de calidad del multimodelo, con las restricciones de requisitos no-funcionales de la línea de productos.
- La vista de variabilidad del multimodelo, con las características y restricciones entre ellas.

- Los requisitos funcionales y no-funcionales del producto en desarrollo especificados por el ingeniero de requisitos de aplicación.

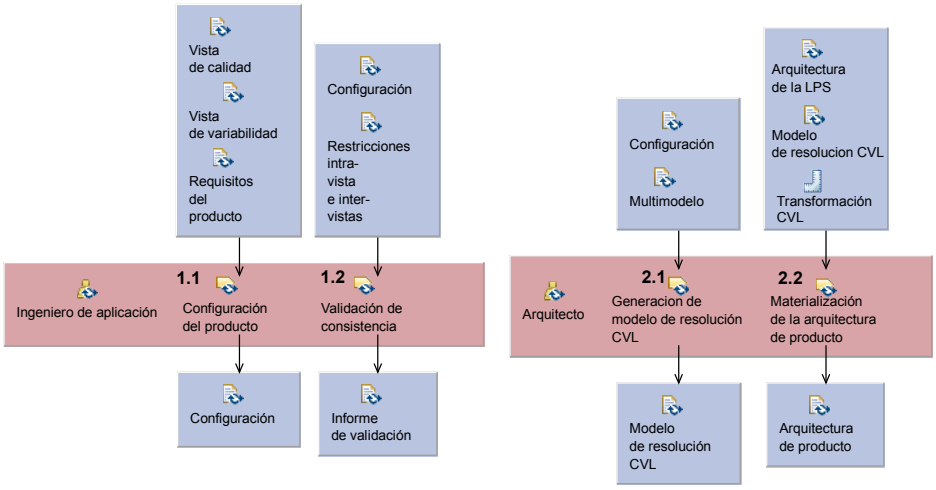


Figura 4.16 Diagrama de actividad detallado de la fase de derivación

Para reflejar la configuración se extrae una proyección de la vista de variabilidad y de calidad del multimodelo, sobre las que el ingeniero de aplicación realiza la selección de características, prioriza los atributos de calidad y define, en caso necesario, los requisitos no-funcionales específicos del producto, incluyendo los niveles umbral asociados a las métricas con las que se evalúa cada uno de los requisitos no-funcionales. La prioridad de los atributos de calidad y las relaciones entre las características y los atributos de calidad definidas en el multimodelo, pueden asistir o completar el proceso de selección de características. La proyección extraída de las vistas de variabilidad y de calidad del multimodelo que contiene la selección de características, los requisitos no-funcionales y la priorización de atributos de calidad, va a actuar como modelo de configuración del producto.

La tarea de **validación de la configuración** (1.2) toma como entradas:

- El modelo de configuración generado en la tarea (1.1).
- El multimodelo (sus restricciones intra-vista y las relaciones entre los elementos de las distintas vistas).

Esta tarea genera como salida un informe de validación y se lleva a cabo con la ayuda de procesos de transformación de modelos que transformarán el modelo de configuración en uno o varios modelos, que serán tomados como entrada

por las herramientas de validación. En caso de que el informe indique que se ha violado alguna de las restricciones (bien a nivel intra-modelo, bien alguna de las restricciones definidas mediante las relaciones entre elementos de las distintas vistas del multimodelo), se volverá a la fase de configuración hasta obtener una configuración válida (que cumple todas las restricciones a nivel de modelo y de multimodelo).

La tarea de **generación del modelo de resolución CVL** (2.1) toma como entradas:

- El modelo de configuración (validado) que proviene de la tarea (1.2).
- El multimodelo (las relaciones entre los elementos de las distintas vistas.

En esta tarea se genera, mediante un proceso de transformación de modelos, el árbol de resolución CVL que contiene la resolución de la variabilidad del producto en desarrollo, teniendo en cuenta las características seleccionadas en la configuración y las prioridades de los atributos de calidad, que se combinarán con las relaciones de impacto entre los puntos de variabilidad arquitectónica (VSpecs) y los atributos de calidad.

Por último la tarea de **materialización de la arquitectura de producto** (2.2) toma como entradas:

- El modelo de resolución CVL, generado en la tarea 2.1.
- Los modelos arquitectónicos de la línea de productos sobre los que se definió la variabilidad.

Por último, en esta tarea se obtiene de forma automática, la primera versión de la arquitectura del producto en desarrollo haciendo uso de la transformación CVL, que tendrá que ser evaluada durante la fase de evaluación y mejora, para verificar el cumplimiento de los requisitos no-funcionales.

4.3.2.2 Fase de evaluación y mejora

El objetivo de esta fase es la verificación de los requisitos no-funcionales del producto en desarrollo y en caso necesario, la aplicación de transformaciones arquitectónicas con el objetivo de adecuar los niveles de los atributos de calidad de la arquitectura a dichos requisitos no-funcionales. La Figura 4.17 muestra el diagrama de actividad de la descomposición en tareas de la actividad de evaluación y mejora. La actividad evaluación (3) que es llevada a cabo por el evaluador, se corresponde a la actividad de evaluación de la arquitectura de

producto (3.1). La actividad de transformación (4) llevada a cabo por el arquitecto de aplicación, se corresponde a la actividad de transformación de la arquitectura de producto (4.1).

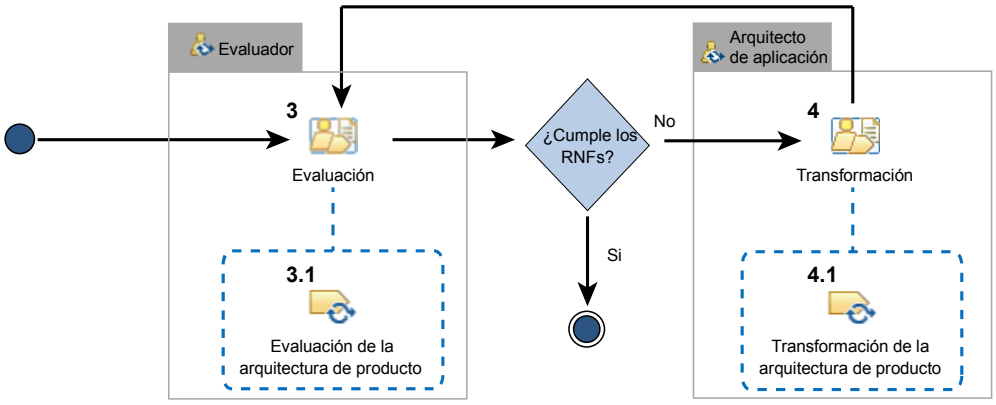


Figura 4.17 Diagrama de actividad de la fase de evaluación y mejora

En la Figura 4.18 se muestra el diagrama de actividad detallado con las entradas, las salidas y el rol responsable de cada una de las tareas que forman parte de la fase de evaluación y mejora.

La tarea de **evaluación de la arquitectura de producto** (3.1) toma como entradas:

- La arquitectura de producto (bien la obtenida en la tarea 2.2 o bien la que proviene de la tarea 3.2 después de la aplicación de transformaciones arquitectónicas).
- La vista de calidad del multimodelo, donde se hallan descritas las métricas para evaluar los distintos atributos de calidad asociados a los diferentes requisitos no-funcionales.
- La configuración (requisitos no-funcionales) del producto en desarrollo a evaluar.

Tras la aplicación de las métricas (que puede realizarse automáticamente empleando herramientas de medición, mediante procesos de transformación de modelos o manualmente) se obtiene un informe de evaluación que detallará el valor de las métricas que miden cada uno de los requisitos no-funcionales y si se encuentra entre los umbrales definidos en el modelo de configuración.

La tarea de transformación de la arquitectura de producto toma como entradas:

- La arquitectura de producto (que incumple alguno de los requisitos no-funcionales).
- La configuración del producto (las prioridades de los atributos de calidad).
- El multimodelo, en concreto las relaciones entre las transformaciones alternativas y los atributos de calidad.
- Las trasformaciones arquitectónicas.

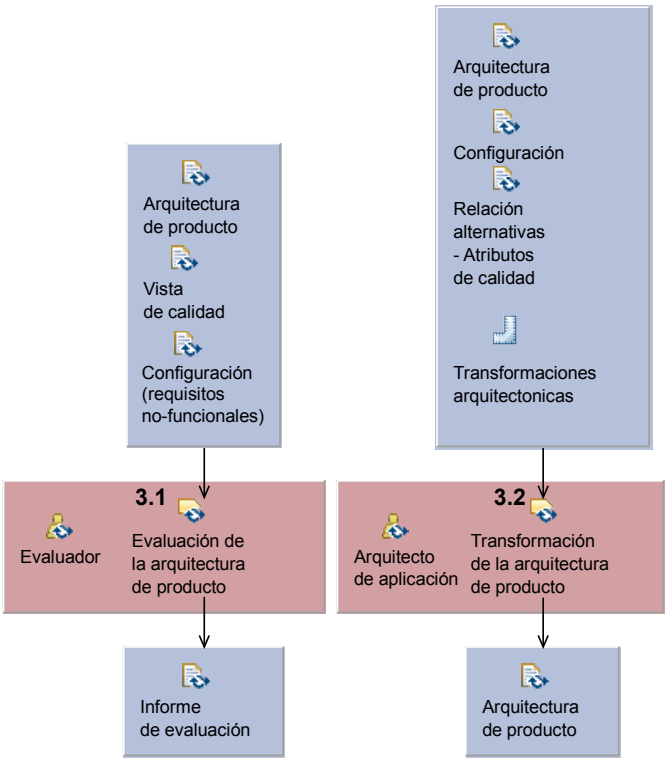


Figura 4.18 Diagrama de actividad detallado de la fase de evaluación y mejora

En esta actividad, mediante un proceso de transformación de modelos, se seleccionan y aplican (en el caso en que la transformación sea susceptible de ser aplicada automáticamente) las transformaciones arquitectónicas que mejor se adaptan a la prioridad de los atributos de calidad y a las relaciones de impacto

entre transformaciones alternativas y atributos de calidad establecidas en el multimodelo.

4.4 Conclusiones

En este capítulo hemos presentado la contribución metodológica de esta tesis: la definición de un método que se compone de un artefacto (el multimodelo) y de un proceso que permite la obtención de arquitecturas de producto que cumplen con los atributos de calidad deseados.

El método propuesto da soporte, de manera integrada, a la derivación evaluación y mejora de arquitecturas cuando estas no cumplen con los niveles de calidad deseados.

Con respecto al multimodelo, no solo se trata de la definición de un artefacto para soportar el proceso descrito, sino que la arquitectura del metamodelo propuesto permite crear multimodelos para múltiples propósitos sobre cualquier lenguaje de modelado soportado por MOF, sin que sea necesario alterar la estructura de los metamodelos que soportan a dicho metamodelo. Esto permite crear conglomerados de modelos y definir relaciones explicitas entre los elementos de dichos modelos. Todo ello contando con el soporte que nos permite editar y gestionar dichos metamodelos de manera genérica.

La estructura de especificación de variabilidad en dos niveles, mediante la vista de variabilidad para la variabilidad externa y la vista arquitectónica para representar la variabilidad interna de la arquitectura, permite simplificar ambos modelos, añadiendo mayor expresividad, y mejorando la mantenibilidad de ambos. Además nos permite, primero, ser capaces de establecer las relaciones entre la variabilidad externa de la línea de productos y la variabilidad arquitectónica (variabilidad interna) para expresar la realización de la variabilidad externa en puntos de variabilidad arquitectónica. Además podremos establecer las consecuencias que ejercer de un modo u otro dicha variabilidad interna y externa tienen sobre la calidad del producto final, mediante las relaciones de impacto entre los puntos de variabilidad (VSpecs) y los atributos de calidad o entre las características y los atributos de calidad. Por último, permite describir como los puntos de variación satisface los requisitos no-funcionales.

Por otro lado el multimodelo permite hacer explicitas las relaciones entre decisiones arquitectónicas y requisitos no-funcionales, cubriendo una de las carencias detectadas en recientes estudios empíricos (Ameller et al. 2013).

Entendemos además que el hecho de emplear CVL para la representación de la variabilidad hace que la propuesta sea aplicable independientemente del dominio, de la naturaleza de la arquitectura de la línea de productos y de los lenguajes de representación arquitectónica empleados (siempre y cuando sean compatibles con MOF). CVL añade un nuevo nivel de dirección que aísla el multimodelo y la propuesta de la especificación de la arquitectura de la línea de productos.

El uso de CVL enlazado al modelo de características puede no estar circunscrito únicamente a la representación de la variabilidad arquitectónica, sino que puede utilizarse sobre otros artefactos empleados en el ciclo de desarrollo, siempre que estos artefactos puedan ser modelados empleando lenguajes de modelado basados en MOF.

Por otro lado, el multimodelo en sí mismo puede reflejar inconsistencias, al poder definir relaciones entre elementos en vistas que a su vez pueden tener relaciones incompatibles con otros elementos en esa u otras vistas. Esto puede ser mitigado mediante la traducción a lógica proposicional y tratar de utilizar razonadores para razonar acerca de las posibles relaciones que pueden dar lugar a inconsistencias en el multimodelo.

Los capítulos siguientes profundizan en la definición de cada una de las dos fases principales de QuaDAI: la de derivación y la de evaluación y mejora de arquitecturas de producto.

Capítulo 5

Derivación de la Arquitectura de Producto

Cada vez es mayor el número de organizaciones que adoptan el uso de las líneas de producto para aumentar la productividad y desarrollar productos de mayor calidad a menor coste. En este contexto las actividades de derivación juegan un papel crítico, ya que representan tareas complejas (Griss 2000), con un alto coste tanto temporal como económico (Sinnema et al. 2006).

En este capítulo se describe la primera de las fases del método QuaDAI, la derivación de la arquitectura de producto a partir de la arquitectura de la línea de productos, en el que esta se obtiene por transformación a partir de los requisitos (funcionales y no-funcionales) que el producto ha de cumplir.

En la sección 5.1 se describe en detalle la obtención de una configuración válida para el producto en desarrollo. En el modelo de configuración se definen tanto las características como los requisitos no-funcionales y la prioridad de los atributos de calidad del producto en desarrollo, extraídas de su especificación de requisitos. Este modelo de configuración deberá ser validado para asegurar que representa una configuración de producto válida atendiendo a las restricciones especificadas en el multimodelo.

La sección 5.2 describe el proceso por el cual, a partir del modelo de configuración se obtiene el modelo arquitectónico del producto en desarrollo a partir del modelo de configuración y de la arquitectura de la línea de productos.

5.1 Obtención de la configuración del producto

Previamente a la derivación de la arquitectura del producto se define una configuración válida del producto compuesto por los requisitos funcionales, representados por la selección de características y los requisitos no-funcionales que el producto debe cumplir junto con las preferencias de los atributos de calidad demandados.

Para dar soporte la tarea de configuración se emplea un modelo de configuración, una proyección de la vista de variabilidad y de calidad del multimodelo, sobre el que se seleccionarán las características, requisitos no-funcionales y se priorizarán los atributos de calidad. Este modelo de configuración es la entrada del proceso de validación de consistencia. Durante la configuración, se va a utilizar las relaciones definidas entre características, los requisitos no-funcionales y los atributos para asistir al proceso.

En la tarea de validación de consistencia se emplearán transformaciones de modelos y restricciones OCLs para validar que el modelo de configuración es conforme a las restricciones y a las relaciones definidas a nivel de modelo (mediante la validación de consistencia intra-vista) y las definidas entre elementos a nivel del multimodelo (mediante la validación inter-vistas). La Figura 5.1 muestra la descomposición de pasos de las tareas de configuración del producto y validación de la consistencia, con la retroalimentación a los diferentes pasos en el caso en que alguna de las validaciones no sea satisfactoria. En el caso de la consistencia inter-vistas, la realimentación va hacia todos los pasos, puesto que puede implicar correcciones en todos ellos, aunque las funcionalidades de validación de consistencia de la herramienta de soporte permiten detectar que relación o que entidad de que vista provoca la inconsistencia.

5.1.1 Configuración del producto

El modelo de configuración es el principal artefacto empleado para definir la configuración a partir de los requisitos del producto a desarrollar. Las clases extendidas *EFeature*, *ENFR* y *EAttribute* (véase Figura 4.10) y los atributos definidos en dichas metaclases van a permitir al ingeniero de aplicación definir la configuración del producto.

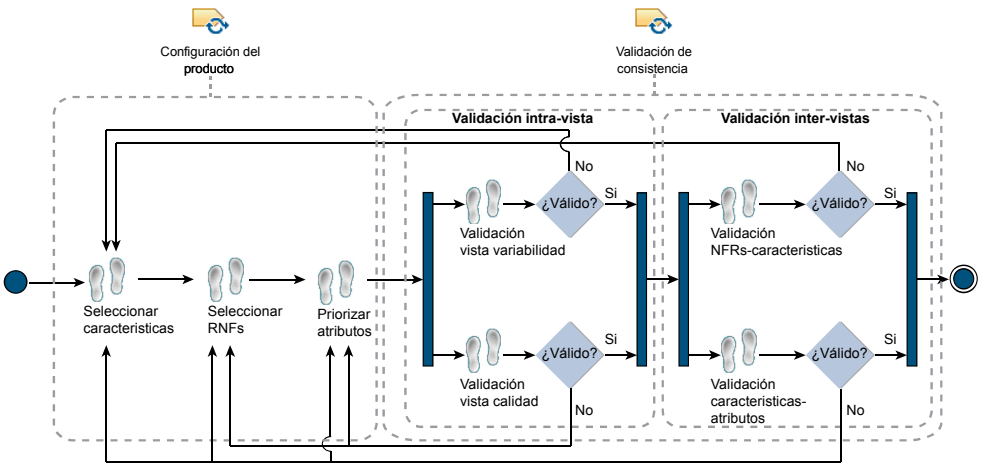


Figura 5.1 Descomposición en pasos de las tareas de configuración y validación de consistencia

La configuración estará compuesta por:

- Las características seleccionadas, (mediante el campo *selected* de la metaclase *EFeature*) en la vista de variabilidad del modelo de configuración.
- Los requisitos no-funcionales seleccionados (mediante el campo *selected* de la metaclase *ENFR*) en la vista de calidad del modelo de configuración.
- La priorización de los atributos de calidad, descrita mediante la importancia relativa (atributo *importance* de la meta clase *EAttribute*), en la vista de calidad del modelo de configuración. Esta importancia relativa se expresa en valores en un rango de 0 a 1 (0 para un atributo sin ninguna prioridad y 1 para la máxima prioridad).

Además, en este proceso se utilizan varias de las relaciones entre vistas definidas en el multimodelo, descritas en detalle en la sección 4.2.3, las cuales asisten en la selección de atributos de características y atributos de calidad de diferentes formas:

- La relación entre características y requisitos no-funcionales permite que, a través de la infraestructura de gestión del multimodelo, si se selecciona un determinado requisito no-funcional, se seleccione de forma simultánea las características asociadas.

- La relación de impacto entre características y atributos de calidad no implica la selección automática de la característica. Si un determinado atributo de calidad ha sido establecido como prioritario se calcula el producto de la prioridad por el impacto y se utiliza el valor para sugerir su selección si el valor resulta por encima de cierto umbral.

La tarea de configuración del producto se soporta por la infraestructura definida para la edición de multimodelos, en la que es posible editar el modelo de configuración e implementa los mecanismos de asistencia anteriormente descritos para asistir en la obtención del modelo de configuración.

Dicho modelo será empleado en la fase de validación de consistencia para analizar si la configuración, definida por el ingeniero de aplicación, representa una configuración de producto válida, de acuerdo a las restricciones de la línea de productos definidas en el multimodelo.

5.1.2 Validación de consistencia

El objetivo de la tarea de validación de consistencia es analizar si la configuración definida en la tarea de obtención de la configuración se ajusta a las restricciones definidas en los modelos que soportan cada una de las vistas así como las posibles restricciones definidas entre elementos de las vistas de calidad y de variabilidad del multimodelo. Esta validación de consistencia se lleva a cabo en dos fases, en la primera fase se válida la consistencia intra-modelo, es decir si el modelo cumple las restricciones definidas en la correspondiente vista del multimodelo y en la segunda fase se válida la consistencia inter-modelo, es decir, si la configuración se puede satisfacer teniendo en cuenta las relaciones definidas a nivel del multimodelo.

5.1.2.1 Validación de consistencia intra-vista

Validación de consistencia de la vista de variabilidad

Si atendemos únicamente a la vista de variabilidad, una configuración válida será aquella que satisface las restricciones definidas entre características (cardinalidades, relaciones de exclusión, implicación, etc.).

Un modelo de características M_c es un conjunto de configuraciones donde $M_c \subseteq C$ siendo C el conjunto de posibles combinaciones[12] de característica (Janota y Botterweck 2008).

Una configuración $c = \{c_1, c_2, \ldots, c_n\} \in C$ se dice que es una configuración válida de Mc (es conforme a Mc) si, y solo si, $c \in Mc$ (Janota y Botterweck 2008).

El análisis de consistencia a nivel de vista de características, tal como se muestra en la Figura 5.2 se lleva acabo proyectando la vista de calidad del multimodelo (las relaciones y restricciones del modelo de características) y la configuración (conjunto c de características seleccionadas) a la herramienta FaMa (ISA Research Group 2011), mediante transformaciones del modelos.

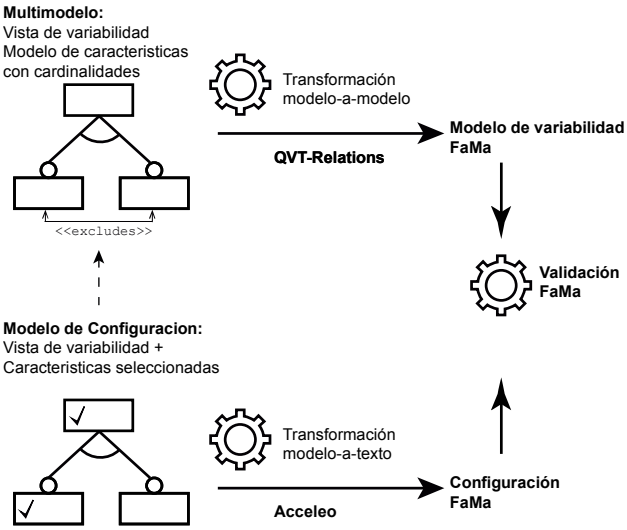


Figura 5.2 Esquema de transformación para la validación de consistencia en la vista de variabilidad

La herramienta FaMa permite, tanto validar que el modelo de variabilidades sea factible (puede dar lugar a configuraciones validas), como generar el conjunto Mc por extensión o identificar cuando una determinada configuración c es válida. Esta última funcionalidad es la que vamos a explotar para la evaluación

[12] Estas combinaciones no son generadas por extensión para comprobar la conformidad de una dada, por el contrario, la formalización de las relaciones nos permite su verificación sin necesidad de generar todas ellas. Este principio es aplicable a toda las validaciones de consistencia tanto intra-vista como inter-vistas.

de la consistencia intra-modelo a nivel del modelo de características. Mediante la transformación *modelo-a-modelo* que establece la equivalencia entre el metamodelo que da soporte a la vista de variabilidad y el metamodelo de FaMa (Khachan 2012; Gómez 2012) generamos la representación en el metamodelo FaMa de la vista de características. Mediante una transformación *modelo-a-texto* generamos la configuración seleccionada (que en el caso de FaMa no es más que la lista de nombres de las características seleccionadas). Con esta información FaMa es capaz de dilucidar si la selección de características cumple con las restricciones del modelo de características.

Validación de consistencia de la vista de calidad

El modelo de calidad M_q puede ser visto como un conjunto de configuraciones donde $M_q \subseteq Q$ siendo Q el conjunto de posibles configuraciones expresadas como priorizaciones de atributos.

Una configuración $q = \{q_1, q_2, \dots, q_n\} \in Q$ de atributos de calidad representados por su importancia relativa $q_i \in \mathbb{N}$[13] se dice que es una configuración válida de M_q (es conforme a M_q) si y solo si:

$$\forall q_i, \forall q_j \in q\ (q_i > 0\ \wedge impacto_q(q_i, q_j, X) \wedge X > 0 \rightarrow q_j \geq 0) \qquad (1)^{14}$$

Siendo el $impacto_q(q_i, q_j, X)$ la relación de impacto definida entre los atributos de calidad q_i y q_j con valor de impacto X.

El análisis de consistencia a nivel de vista de calidad valida que las relaciones entre atributos (definidas a nivel del modelo de calidad) no hagan incumplir la prioridad de dichos atributos. La idea que subyace es que el signo de las prioridades ha de ir acorde con las relaciones de impacto entre atributos. La Figura 5.3 muestra dos ejemplos de priorización de atributos de calidad. La configuración de la Figura 5.3(a) se considera válida, puesto que el atributo *Qa*, que ha sido seleccionado por el ingeniero de aplicación como de cierta

[13] Con el fin de formalizar las relaciones supondremos los valores de impactos y pesos como discretos y la cardinalidad del conjunto de posibles configuraciones como finito.

[14] En el caso de que no existan relaciones de impacto definidas explícitamente en el multimodelo, diremos que esa relación es igual a 0, y por tanto satisfaría igualmente la restricción.

prioridad, tiene un impacto positivo en el atributo *Qc* el cual también ha sido seleccionado como prioritario, y el atributo *Qb*, a pesar de ser impactado negativamente no ha sido seleccionado como prioritario (peso=0). Por el contrario, la configuración de la Figura 5.3(b) no se considera valida, puesto que el atributo *Qa*, que ha sido seleccionado por el ingeniero de aplicación como de cierta prioridad, tiene un impacto negativo en un atributo de calidad que ha sido considerado por al ingeniero de aplicación como de mayor prioridad, por lo que no sería posible satisfacer esos dos atributos al mismo tiempo.

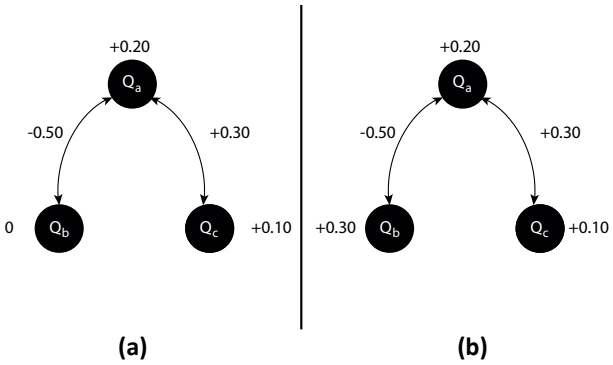


Figura 5.3 Relaciones de impacto entre atributos de calidad

El Listado 5.1 (líneas 5-12) muestra la definición en OCL, de la restricción expresada en la formula (1). Esta restricción OCL se valida en tiempo de ejecución por el intérprete OCL del componente *OCLTools* (Eclipse 2013b) del *Eclipse Modeling Framework* (Eclipse 2013a) sobre el modelo de configuración.

Listado 5.1 Restricción OCL para validar la consistencia de los impactos descritos en el modelo de calidad

```
1  class EAttribute extends coreMultimodel::MultimodellRelatableEntity, qualityNFR::Attribute
2  {
3  attribute Importance : ecore_0::EFloat[?];
4
5  invariant WrongQualityImpact:
6  if
7      self.Importance>0
8  then
9      self.Impacts->select(Importance*ImpactsOn.oclAsType(EAttribute).Importance<0)->size()=0
10 else
11 true
12 endif;
13 }
```

La Figura 5.4 muestra un ejemplo de validación de la consistencia de la vista de calidad, haciendo uso del ejemplo del dominio de los sistemas de control de

vehículos. La Figura 5.4(a) muestra un fragmento de la vista de calidad, que incluye tres atributos con prioridades positivas y un impacto, que expresa como la tolerancia a fallos impacta negativamente al *uso de CPU* (mejores valores de tolerancia a fallo van a traer como consecuencia peores valores de uso de CPU, puesto que habrá que, cuanto menos, hacer comprobaciones adicionales o replicar componentes). Este impacto no satisface la expresión (1) y por tanto la formula OCL asociada se evalúa a falso. La Figura 5.4(b) muestra la salida de la validación de la formula OCL en la infraestructura de soporte al multimodelo.

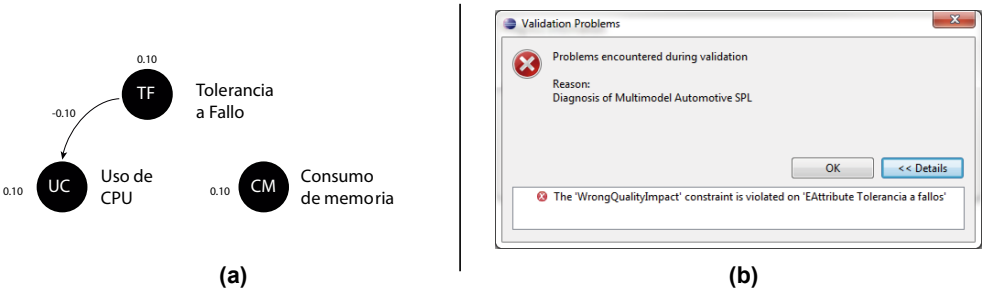


Figura 5.4 Ejemplo de validación de consistencia haciendo uso del validador OCL

5.1.2.2 Validación de consistencia inter-vistas

En cuanto a la consistencia inter-vistas, se llevarán a cabo validaciones de dos relaciones entre vistas: la relación de impacto entre características y atributos de calidad y la relación *RealizadoPor* entre requisitos no-funcionales y características, las cuales se muestran de forma esquemática en la Figura 5.5.

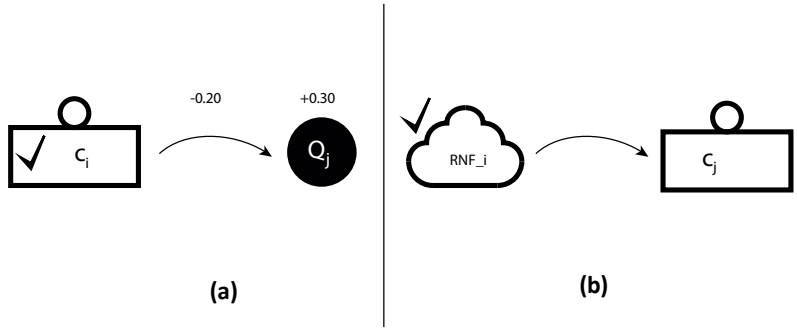


Figura 5.5 Relaciones que intervienen en la validación de consistencia entre vistas

Para formalizar dichas relaciones entre vistas primero será necesario extender la definición del modelo de calidad, para poder incluir los requisitos no-funcionales definido en la sección 5.1.2.1:

$M_q{}'$ se define como un conjunto de configuraciones de la vista de calidad donde $M_q{}' \subseteq Q'$, siendo Q' el conjunto de posibles priorizaciones de atributos $q_i \in \mathbb{N}$ y el conjunto de RNFs seleccionados $rnf_i \in \mathbb{B}$.

Y definimos configuración de la vista de calidad como:

$$q' = \{q_1, q_2, \dots, q_n\} \cup \{rnf_1, rnf_2, \dots, rnf_n\} \in Q' \quad (2)$$

A continuación definimos el multimodelo M_m como un conjunto de configuraciones donde $M_m \subseteq M$ siendo M el conjunto de posibles características seleccionadas, priorizaciones de atributos, y requisitos no-funcionales, que vendrá dada por:

$$m = q' \cup c$$

$$c = \{c_1, c_2, \dots, c_n\} \in C$$

En la validación de consistencia de la primera relación, se comprueba la consistencia con respecto a las relaciones de impacto entre las características seleccionadas en la configuración y los atributos priorizados por ingeniero de aplicación. La finalidad de dicha comprobación es la de identificar una determinada característica c_i que tienen un efecto negativo sobre un atributo q_i con prioridad positiva en la vista de calidad (Figura 5.5(a)).

La restricción viene dada por la expresión:

$$\forall c_i \in c, \forall q_j \in q' \, (c_i \wedge impacto_{cq}(c_i, q_i, X) \wedge X > 0 \rightarrow q_i \geq 0) \quad (3)$$

Siendo el $impacto_{cq}(c_i, q_j, X)$ la relación de impacto definida entre los la característica c_i y el atributo de calidad q_j con valor de impacto X.

En la validación de consistencia de la segunda relación, se comprueba la consistencia de las relaciones *RealizadoPor* entre características y requisitos no-funcionales. En caso de seleccionarse un determinado requisito no-funcional que *se realiza* mediante una o más características estas deberán ser forzosamente seleccionadas. Una configuración válida contendrá pues, todas las

características para las que se hayan definido relaciones *RealizadoPor* con requisitos no-funcionales y éstos hayan sido seleccionados (Figura 5.5 (b)).

La restricción viene dada por la expresión:

$$\forall rnf_i \in q', \forall c_j \in c\ (rnf_i\ \wedge realiza_{rnf-c}(\ rnf_i, c_j) \rightarrow c_i) \qquad (4)$$

Al igual que en el caso de la consistencia para los atributos de calidad, esto se lleva a cabo mediante el cálculo de la fórmula OCLs en tiempo de ejecución a través del validador de *OCLTools*. El Listado 5.2 (líneas 5-13) muestra la definición, como fórmula OCL, de la restricción expresada en la formula (3) para el caso de la relación entre características y atributos de calidad.

Listado 5.2 Fórmula OCL a evaluar para validar la relación entre características y atributos de calidad

```
1  class FeatureToQualityAttribute extends coreMultimodel::MultimodelRelationship
2  <EFeature,EAttribute,coreMultimodel::BidirectionalDirection,coreMultimodel::Impact>
3  {
4  invariant Feature_Selected_Negaive_Impacts_On_PrioritizedQA:
5
6  not
7    (
8       self.from.oclAsType(EFeature).Selected=true
9    and
10      self.Weight<0 and self.to.oclAsType(EAttribute).Importance>0
11   );
12   attribute Weight : ecore_0::EFloat[?];
13 }
```

El Listado 5.3 (líneas 5-6) muestra la definición, como restricción OCL, de la restricción expresada en la formula (4) para el caso de la relación *RealizadoPor* características y requisitos no-funcionales.

Listado 5.3 Fórmula OCL a evaluar para validar la relación entre RNFs y características

```
1  class FeatureToNFR extends
2  coreMultimodel::MultimodelRelationship
3  <EFeature,ENFR,coreMultimodel_0::BidirectionalDirection,coreMultimodel_0::Decomposition>
4  {
5  invariant NFR_Selected_Features_UnSelected:
6  not(self.from.oclAsType(EFeature).Selected=false and self.to.oclAsType(ENFR).Selected=true);
7  }
```

En el caso de la consistencia inter-vistas se ha previsto además un mecanismo que permite hacer una comprobación de manera local para un determinado elemento (característica, atributo de calidad o RNF), para asistir en la resolución de los posibles conflictos. Para llevar a cabo estas comprobaciones de manera local, la infraestructura lleva a cabo, en tiempo de ejecución la validación de restricciones, mediante el intérprete de *OCLTools* para una

determinada entidad del modelo de configuración (RNFs, atributos de calidad o características).

La Figura 5.6 muestra una validación de consistencia con resultado negativo para una relación entre características y atributos de calidad haciendo uso del ejemplo del dominio de los sistemas de control de vehículos. La Figura 5.6(a) muestra en la parte superior, un fragmento de la vista de variabilidad del modelo de configuración, en la que se especifica que un vehículo puede tener tres características opcionales: CC (control de velocidad de crucero), CT (control de tracción) y ABS (control de antibloqueo de frenos). Esta última, que ha sido seleccionada para el producto en cuestión, tiene además definida una relación de impacto negativo con el atributo *tolerancia a fallo*, para reflejar que este característica es propensa al fallo y afectará a la tolerancia total del sistema. En la parte inferior de la Figura 5.6(a) se muestra un extracto de la vista de calidad del modelo de configuración, que contiene las prioridades de los atributos de calidad, entre las que se especifica que la tolerancia a fallo cuenta con cierta prioridad. La combinación de la selección de la característica, la prioridad del atributo y el impacto de la característica en el atributo de calidad no satisface la expresión (3) y por tanto, la fórmula OCL del Listado 5.2 se evalúa a falso. La Figura 5.6(b) muestra la salida de la validación de la fórmula OCL en la infraestructura de soporte al multimodelo.

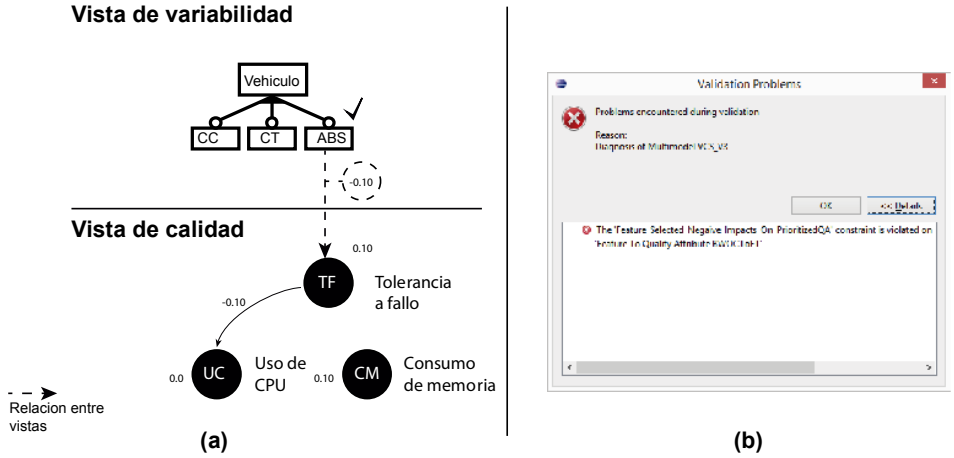


Figura 5.6 Ejemplo de validación de consistencia entre caracteristicas y atributos de calidad haciendo uso del validador OCL

La Figura 5.7 muestra una validación de consistencia con resultado negativo para una relación entre características y requisitos no-funcionales. La Figura

5.7(a) en la parte superior muestra, un fragmento de la vista de variabilidad del modelo de configuración, pero en este caso ninguna de las características ha sido seleccionada para estar presente en el producto en cuestión. En la parte inferior de la Figura 5.7(a) se muestra un extracto de la vista de calidad del modelo de configuración, en la que esta seleccionado el requisito no-funcional RNF_001 seguridad de usuario, el cual *es realizado por* la característica ABS. La combinación de la selección del requisito no-funcional y la no selección de la característica que lo realiza (ABS) no satisface la expresión (4) y por tanto la fórmula OCL descrita en el Listado 5.3 se evalúa a falso. La Figura 5.7(b) muestra la salida de la validación de la OCL en la infraestructura de soporte al multimodelo.

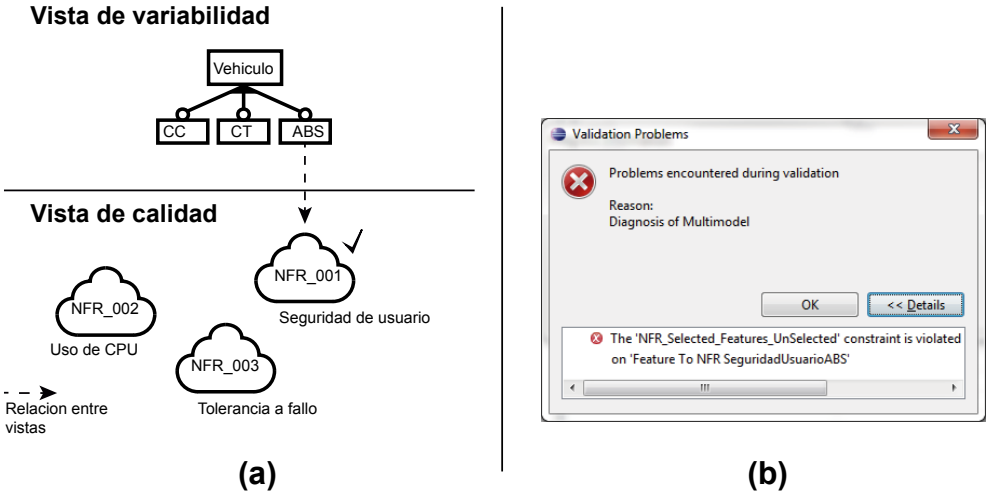


Figura 5.7 Ejemplo de validación de consistencia entre características y requisitos no-funcionales haciendo uso del validador OCL

5.2 Instanciación de la arquitectura de producto

El objetivo de esta actividad es obtener una primera versión de la arquitectura del producto en desarrollo, a partir del modelo de configuración, del multimodelo y de los modelos base y de librería que contienen los componentes arquitectónicos de la arquitectura de la línea de producto. El proceso se divide en dos tareas principales: la generación del modelo de resolución CVL y la transformación CVL. En la primera tarea se lleva a cabo la

selección de qué VSpecs van a ser resueltas positiva o negativamente y esta selección se basará en las relaciones entre las VSpecs de la vista arquitectónica con las características de la vista de variabilidad y los requisitos no-funcionales y atributos de calidad de la vista de calidad. La segunda tarea es totalmente transparente y, a partir del modelo de resolución CVL se ejecuta la transformación CVL que genera el modelo de la arquitectura del producto.

En la fase de ingeniería de la aplicación se parte del multimodelo que contiene las relaciones entre los elementos de variabilidad, de la vista arquitectónica y de la vista de calidad. Estas relaciones, así como el propio modelo CVL representado en la vista arquitectónica del multimodelo es el resultado de las actividades de modelado de la variabilidad arquitectónica en el multimodelo en la fase de ingeniería de dominio.

En las siguientes subsecciones se va a describir en detalle las actividades destinadas a obtener el modelo CVL que representa la variabilidad arquitectónica así como la obtención de los modelos arquitectónicos que representan la arquitectura del producto en desarrollo.

5.2.1 Modelado de la variabilidad arquitectónica en el multimodelo

El proceso de derivación de la arquitectura de producto, descrito en la sección 4.3.2.1 parte de la premisa de que se dispone de una especificación de la arquitectura de la línea de productos, en uno o varios modelos, y con una especificación CVL que representa la vista de variabilidad arquitectónica en dichos modelos, así como de un conjunto de relaciones con los elementos de las vistas de variabilidad y calidad en el multimodelo.

Para crear la especificación de la variabilidad en el modelo CVL y establecer las relaciones con la vista de variabilidad y de calidad, es necesario llevar a cabo un proceso por el que, a partir de la arquitectura de la línea de productos y del modelo (vista) de variabilidad, se construye la especificación CVL que representa la variabilidad en la arquitectura de la línea de productos.

Dicho proceso es llevado a cabo por el arquitecto del dominio, que es el responsable del desarrollo y mantenimiento de la arquitectura de la línea de productos, de los componentes y artefactos reusables así como de los mecanismos de variabilidad y configuración (van der Linden et al. 2007).

La Figura 5.8 muestra un resumen de la detección y especificación de la variabilidad arquitectónica en el multimodelo. Este proceso se basa en el proceso para la generación de modelos de producto descrito en Haugen et al.

(2010). Entre otras diferencias, en este proceso se han sustituido las fases de obtención de modelos de producto por las de conexión con las vistas del multimodelo. En nuestra aproximación la generación de productos se realiza a partir de un modelo de resolución obtenido externamente mediante transformación de modelos.

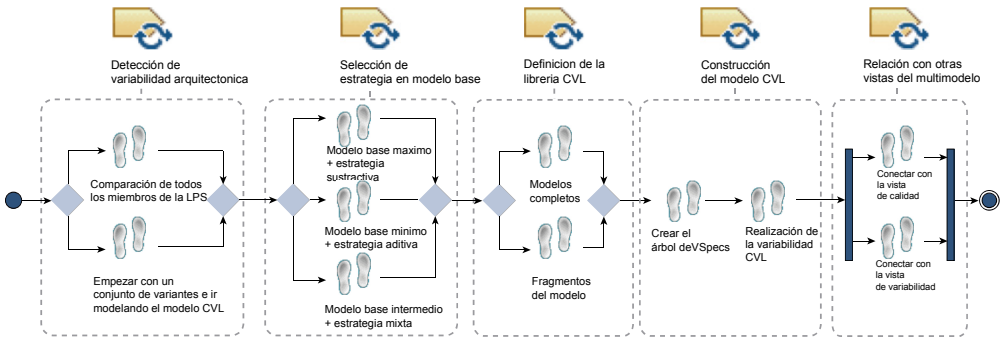


Figura 5.8 Extracto del proceso de especificación de la variabilidad arquitectónica en el multimodelo

El proceso parte de una arquitectura de línea de productos existente, no siendo necesario que la información de variabilidad esté representada explícitamente. La tarea de *detección de variabilidad* consiste en definir o detectar la variabilidad en los modelos arquitectónicos que representan dicha arquitectura. Existen aproximaciones que emplean CVL combinado con herramientas de comparación de modelos, como la presentada por Zhang et al. (2011), que emplea EMF Compare (Brun y Pierantonio 2008) para identificar la variabilidad de un conjunto de modelos, y construir automáticamente la especificación CVL que permite sintetizar modelos de producto a partir de dicho modelo de variabilidad. Otra alternativa es comenzar con un conjunto de variantes y modelar el modelo de variabilidad CVL de manera incremental. En el caso de la especificación incremental del modelo de variabilidad, otro de los artefactos a tener en cuenta al ejecutar la tarea es el modelo de características, para estructurarlo de manera lógica, así como para detectar posibles variantes aisladas de la arquitectura (que no dan servicio a ninguna configuración) o a posibles realizaciones alternativas de una misma característica, que puede tener distintos impactos en los atributos de calidad.

La tarea de *selección de estrategia en el modelo base* va a fijar tanto la estructura del modelo base (a partir del que se define la variabilidad) como la estrategia de variabilidad a seguir al definir los puntos de variación del modelo de variabilidad arquitectónica CVL. Se puede optar por las siguientes opciones:

1. **Modelo base máximo y estrategia sustractiva:** se basa en tener un modelo base completo (en el que están presentes todas las posibles variantes) y aplicar la aproximación negativa (Sánchez et al. 2007) o sustractiva (Haugen et al. 2010) que consiste en eliminar los elementos que no están presentes en una configuración determinada.
2. **Modelo base mínimo y estrategia aditiva:** es la estrategia opuesta, se basa en tener un modelo base en el que solo están presentes las partes comunes (y ninguna variante) y aplicar la estrategia positiva (Sánchez et al. 2007) o aditiva (Haugen et al. 2010) para añadir a la arquitectura aquellos elementos que han de estar presentes en una determinada configuración.
3. **Modelo base intermedio y estrategia mixta:** una estrategia mixta en la que algunos elementos opcionales estarán presentes en el modelo base y tendrán que ser sustituidos o eliminados, mientras que otros tendrán que ser agregados por no hallarse presentes en la configuración.

La decisión de la estructura y estrategia de variabilidad no condiciona únicamente la estructura del modelo base, sino también condiciona la existencia del modelo de librería CVL. CVL crea los modelos resultantes copiando el modelo base y realizando las pertinentes sustituciones o eliminaciones (o agregaciones de elementos que no existiesen en ese modelo base). Cuando un conjunto de elementos *A* del modelo base se reemplaza con otro conjunto de elementos *B*, el origen de este segundo conjunto de elementos *B* puede ser o bien el propio modelo base u otro modelo (o conjunto de modelos) denominado modelo de librería. En el caso de aplicar la estrategia aditiva, los elementos a agregar habrán de estar contenidos en un/os modelo/s de librería.

En la tarea de *definición de la librería CVL* se define la estructura de la librería CVL Los elementos de la librería pueden ser fragmentos de modelo o bien modelos en sí mismos que serán enlazados o agregados al modelo base por referencia (y que eventualmente pueden contener variabilidad anidada mediante la definición de variabilidad compuesta).

La Figura 5.9 muestra las tres posibles estrategias de variabilidad arquitectónica y tres ejemplos de modelo base y modelo de librería para un modelo arquitectónico en AADL de un sistema de software de control de un automóvil. Este sistema se compone del sistema antibloqueo de frenos (ABS), el actuador de frenado, el sensor de giro de los neumáticos (que permite al ABS

detectar cuando las ruedas están deslizando y por tanto bloqueadas) y los sensores de aparcamiento frontal y trasero[15].

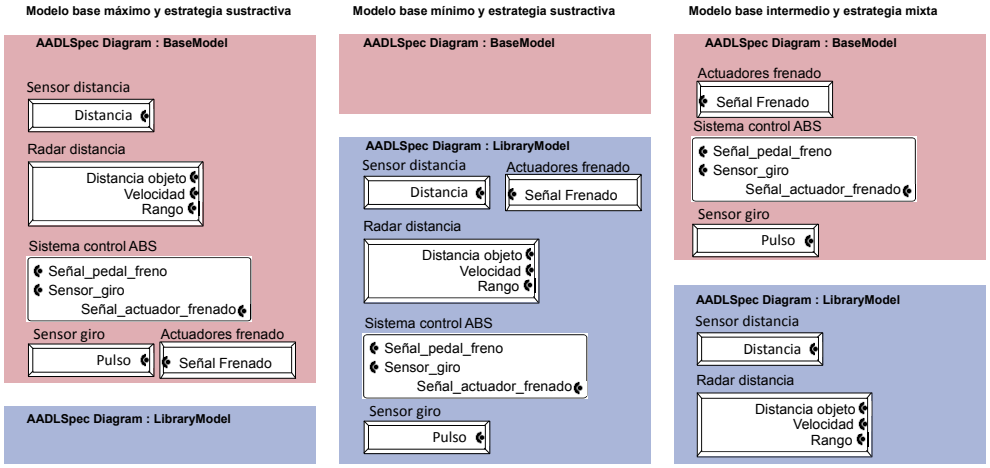


Figura 5.9 Estrategias de variabilidad arquitectónica y la estructura del modelo base y de libreria

El sistema ABS requiere del sensor de giro de los neumáticos (para detectar el deslizamiento de alguno de los neumáticos) y de los actuadores de frenado. En el primer caso, todos los sistemas y sus variantes están en el modelo base, y se eliminaran aquellos que no correspondan a la selección durante el proceso de materialización. En el segundo caso no hay ningún componente en el modelo base y todos se incluyen desde el modelo de librería, de modo que en caso necesario se copiaran al modelo base en la materialización. Por ultimo en el tercer caso el modelo base contiene el ABS, el actuador de frenado y el sensor de giro y los sensores de aparcamiento están ubicados en el modelo de librería.

La Figura 5.10 muestra el árbol de CVL con la variabilidad arquitectónica y su enlace con el modelo base, que en este caso es un modelo máximo con estrategia sustractiva. Aquellas VSpecs que se resuelvan negativamente en el modelo de resolución serán eliminadas de la copia del modelo base generada en la materialización.

[15] En las representaciones de los modelos arquitectónicos se han omitido intencionadamente las conexiones para simplificar los sistemas descritos.

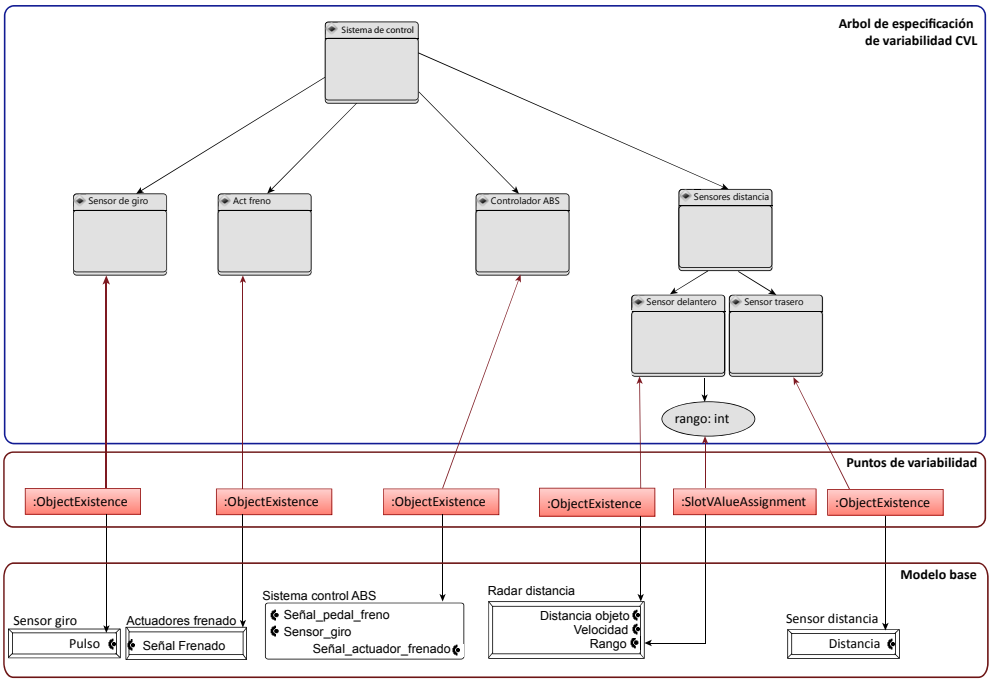


Figura 5.10 Modelo de CVL enlazado con el modelo base (sustractivo)

Por ultimo en la tarea de *relación con otras vistas del multimodelo* se definen las relaciones de las VSpecs con las características de la vista de variabilidad y con los atributos de calidad de la vista de calidad mediante las relaciones al efecto previstas en el multimodelo (ver Sección 4.2.3). Dichas relaciones se establecerán siguiendo los criterios que se detallan a continuación, con el fin de permitir el correcto funcionamiento de la transformación de generación de la resolución CVL:

- Se deberá establecer una relación *RealizadoPor* entre la VSpec *raíz* del árbol de VSpecs.
- Siempre que una VSpec realice una característica se deberá crear una relación *RealizadoPor* entre la VSpec y la característica.
- Siempre que una VSpec realiza un determinado requisito no-funcional se creara una relación *RealizadoPor* entre la VSpec y el requisito no-funcional.
- Se crearán relaciones de impacto con un atributo de calidad siempre que se estime que la resolución positiva de la VSpec tiene (o

potencialmente pueda tener) algun efecto ya sea positivo o negativo, sobre los niveles de dicho atributo de calidad.

- Se permitirá que una VSpec tenga más de una relación entrante con distintos elementos.
- Se permitirá que una VSpec no tenga relaciones entrantes si y solo si:
    a. Tiene el campo *ImplicadoPorPadre* a verdadero y la VSpec que la contiene tiene relaciones entrantes o el campo *ImplicadoPorPadre* a verdadero.
    b. Se trata de una VSpec hoja en cuyo caso podrá considerarse como VSpec *muerta* a nivel del proceso de generación del modelo de resolución, en el sentido de que nunca será resuelta positivamente mediante la transformación, aunque podrá ser posteriormente resuelta positivamente de forma manual (ver sección 5.2.2.2).

La Figura 5.11 muestra un ejemplo de dichas relaciones. La característica ABS del modelo de variabilidad de la línea de productos se enlaza con los VSpecs controlador ABS, sensor de giro y actuador de freno (1 a muchos). La característica sensor de obstáculos y sensor de aparcamiento delanteros se enlazan con la VSpec sensor delantero muchos a uno) permitiendo la reutilización de VSpecs (que a su vez van enlazados a los componentes arquitectónicos). Por último el sensor de aparcamiento trasero se enlaza con la VSpec sensor trasero. También se muestran las relaciones entre las VSpecs de la vista arquitectónica y los atributos de calidad, para en caso necesario, seleccionar las VSpecs en función de la prioridad de los atributos de calidad definida en el modelo de configuración.

5.2.2 Generación del modelo de resolución CVL

El proceso de generación del modelo de resolución va a hacer uso del modelo de CVL variabilidad arquitectónica así como de las relaciones de sus elementos con la vista de calidad y de la vista de variabilidad fruto del modelado de la variabilidad arquitectónica en el multimodelo.

En este proceso de trasformación se crean las instancias de resolución donde se asignan los valores verdadero/falso a las resoluciones de las VSpecs de tipo selección, se asignan valores a las VSpec variables (que puede provenir de un atributo de una característica), o se crean y asignan la resolución, asociando el

valor verdadero/falso en el caso de los clasificadores de variabilidad y las VSpecs compuestas, aunque en estos últimos casos la resolución positiva implica la configuración de cada una de las instancias para el caso de los clasificadores de la variabilidad y la configuración, mediante la interfaz de variabilidad en el caso de las VSpecs compuestas.

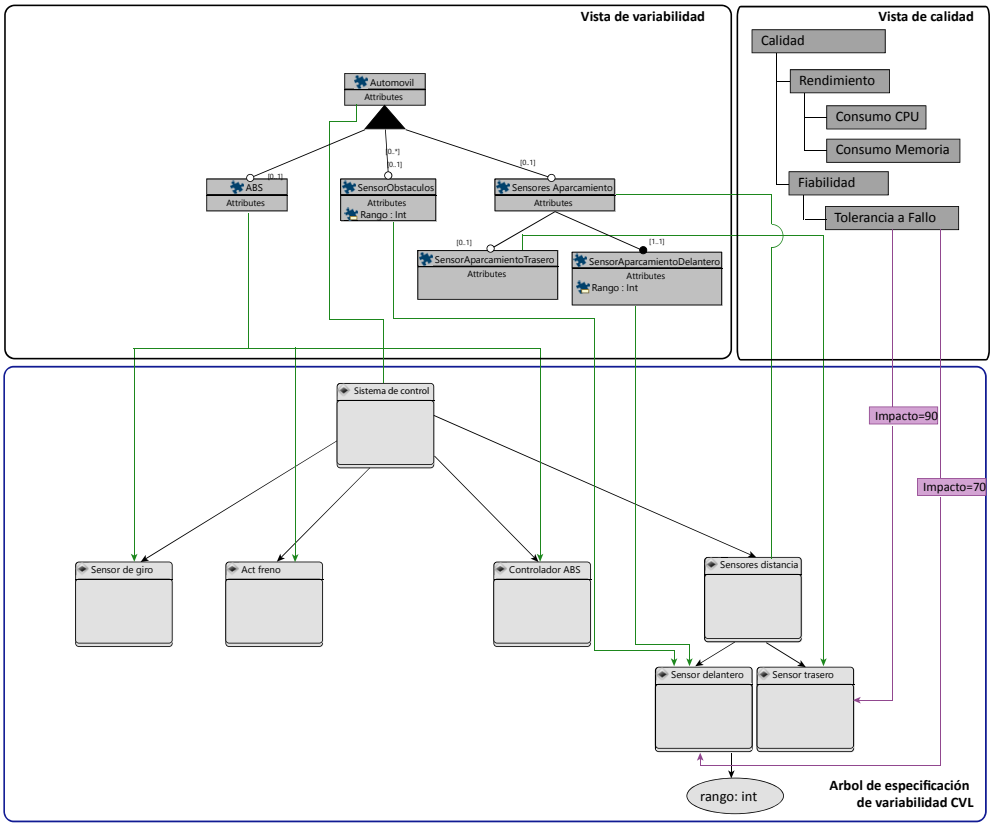


Figura 5.11 Relaciones entre las VSpecs, características y atributos de calidad

Para ilustrar el proceso vamos a centrarnos en el caso de las VSpecs de elección. El caso del clasificador variable pueden verse como un caso particular de las VSpecs de elección en las que en lugar de asignarse un valor lógico verdadero/falso se asignan valores extraídos de algún atributo de las características de la vista de variabilidad. Las VSpecs compuestas (CVSpecs) también pueden considerarse un caso particular del caso de la VSpec de elección dado que en este caso la resolución positiva de un VSpec implicará llevar a cabo una serie de pasos adicionales, los cuales podrán automatizarse o requerir de la intervención del arquitecto para configurar la variabilidad interna de la CVSpec mediante su interfaz de configuración de variabilidad. El caso de

los clasificadores de variabilidad puede verse de nuevo como un caso particular en el que se controlan el número de ejecuciones de la regla de transformación para generar VSpecs hijas y al igual que para las CVSpecs, su configuración podrá o no estar automatizada. La Figura 5.12 muestra los cuatro tipos de puntos de VSpecs y su resolución.

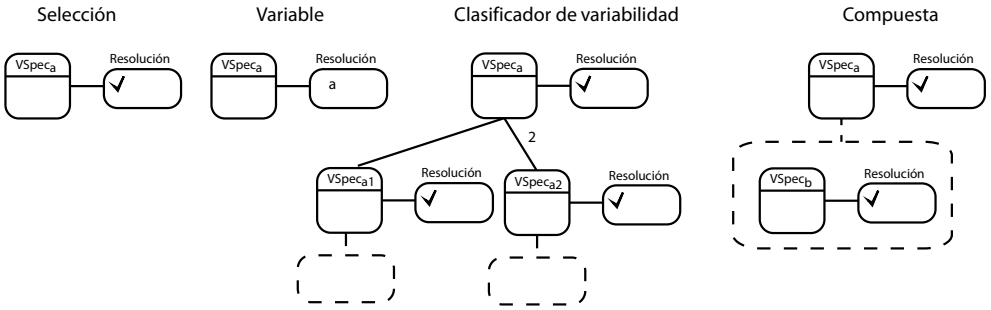


Figura 5.12 Tipos de VSpecs con su resolución asociada

5.2.2.1 Definición del proceso de transformación para la generación del modelo de resolución CVL

Definimos el proceso de transformación de modelos encargado de la generación del modelo de resolución CVL mediante un conjunto de relaciones QVT con dos dominios origen *checkonly*: i) el del metamodelo que soporta el modelo de configuración (Cdomain) y ii) el del metamodelo que soporta el multimodelo (MMdomain) y un dominio *enforce* el del metamodelo que soporta CVL (CVLdomain) y que permiten establecer la relación de equivalencia entre una determinada configuración y el modelo de resolución CVL resultante de dicha configuración.

En este conjunto de relaciones la equivalencia entre EVSpecs del multimodelo y las VSpecs del modelo CVL y las ENFRs, atributos de calidad y características entre el multimodelo y el modelo de configuración se realiza mediante el atributo nombre. A continuación se describen, haciendo uso de la *notación grafica de QVT-Relations* (Object Management Group 2008a), las diferentes relaciones QVT que integran dicha equivalencia para la obtención del modelo de resolución VSpec.

I. **Relación RootFeatureToRootVSpec**: La relación *RootFeatureToRootVSpec* establece que se creará (si no existe ya) la entidad contenedora del modelo CVL (*CVLModel*) y se resolverá positivamente la VSpec$_a$ hija del modelo CVL y equivalente a la EVSpec$_a$ si y solo si esta seleccionada la característica *padre* del modelo de características de la vista de variabilidad en el modelo de configuración; la cual es realizada por la EVSpec$_a$.

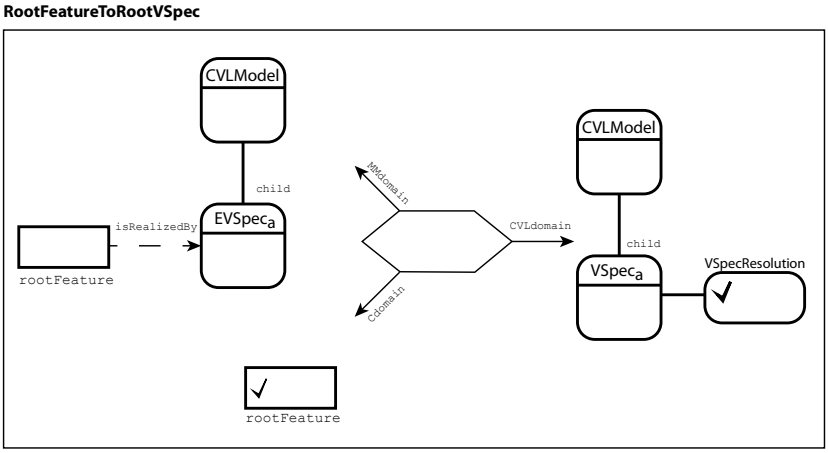


Figura 5.13 Relación RootFeatureToRootVSpec

II. Relación SelectedFeatureToSelectedVSpec: La relación *SelectedFeatureToSelectedVSpec* establece que se creará (si no existe ya) y se resolverá positivamente toda VSpec ($VSpec_b$), equivalente a la $EVSpec_b$ que realiza una determinada característica F^a seleccionada en el modelo de configuración; y que sea hija de otra EVSpec ($EVSpec_a$). En la precondición de la regla (cláusula *when*) se establece que previamente la $VSpec_a$, equivalente a la EVSpec padre $EVSpec_a$, tiene que haber sido resuelta positivamente.

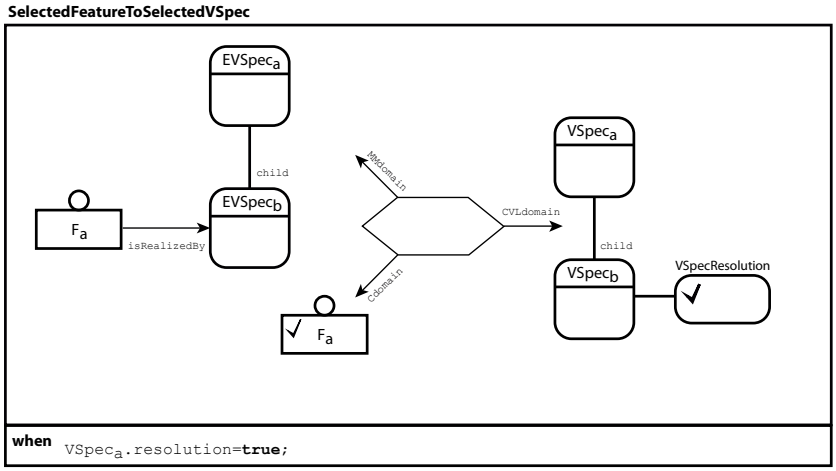


Figura 5.14 Relación SelectedFeatureToSelectedVSpec

III. Relación SelectedNFRToSelectedVSpec: La relación *SelectedNFRToSelectedVSpec* establece que se creará (si no existe ya) y se resolverá positivamente toda VSpec ($VSpec_b$), equivalente a la $EVSpec_b$ que realiza un determinado requisito no-funcional NFR_i seleccionado en el modelo de configuración; y que sea hija de otra EVSpec ($EVSpec_a$). En la precondición de la regla (cláusula *when*) se establece que previamente la $VSpec_a$, equivalente a la EVSpec padre $EVSpec_a$, tiene que haber sido resuelta positivamente.

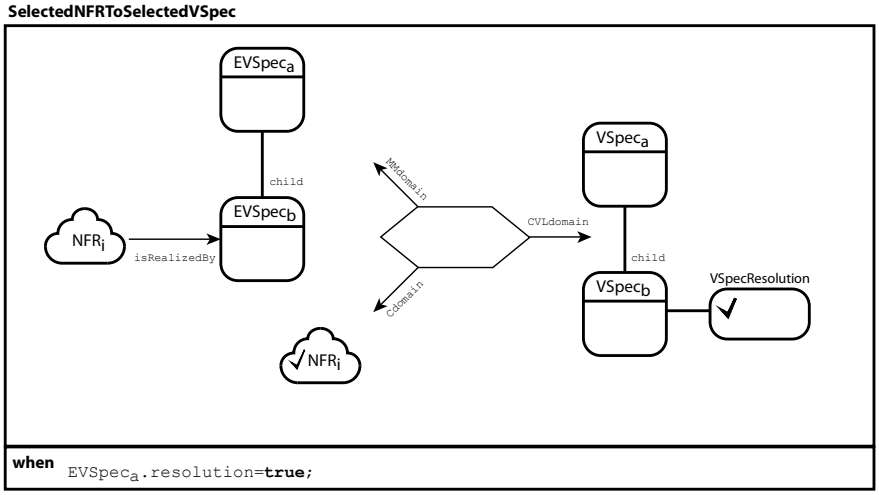


Figura 5.15 Relación SelectedNFRToSelectedVSpec

IV. Relación PrioritizedQAToSelectedVSpec: La relación *SelectedNFRToSelectedVSpec* establece que se creará (si no existe ya) y se resolverá positivamente toda VSpec ($VSpec_b$), equivalente a la $EVSpec_b$ que tiene una relación de impacto con un determinado atributo de calidad qique ha sido establecido como prioritario en el modelo de configuración ($q_i > 0$) y que es hija de otra EVSpec ($EVSpec_a$). En la precondición de la regla (cláusula *when*) se establece que previamente la $VSpec_a$, equivalente a la EVSpec padre $EVSpec_a$, tiene que haber sido resuelta positivamente y además que la $EVSpec_b$ es de entre las hijas de $EVSpec_a$ la que mejor soporta los atributos de calidad priorizados en el modelo de calidad, lo cual se expresa mediante la llamada a la consulta BestQualitySupport, que devolverá la EVSpec que mejor soporte los requisitos de calidad priorizados en la vista de configuración del modelo de calidad. Dicha

consulta calcula el ranking R de la $EVSpec_i$ hija de $EVSpec_a$ mediante la expresión:

$$R_i = \sum_{j=0}^{n} q_j * impacto_{cvlq}(\text{EVSpec}_i, q_j) \quad (5)$$

Donde $impacto_{cvlq}(\text{EVSpec}_i, q_j)$ es el valor numérico del campo *Weight* de la metaclase *VSpecToQualityAttribute* mediante la cual se definen los impactos entre VSpecs y los atributos de calidad (ver Seccion 4.2.3).

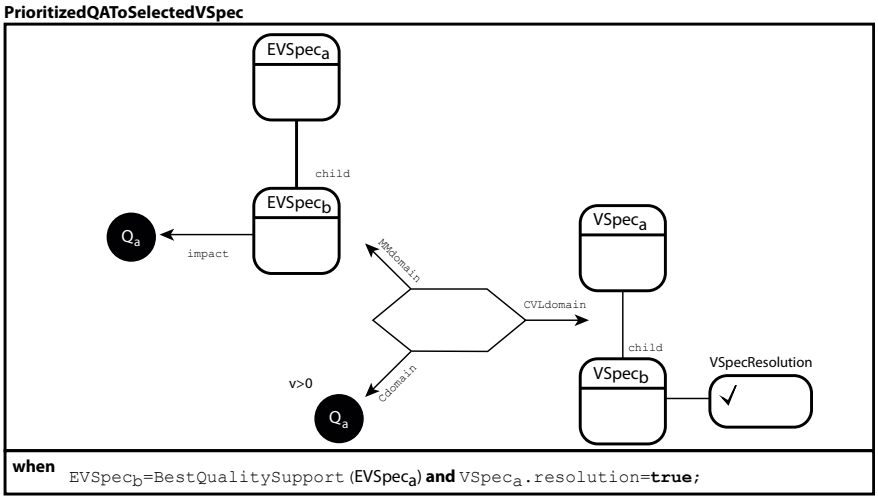


Figura 5.16 Relación PrioritizedQAToSelectedVSpec

V. Relación VSpecImpliedByParent: La relación *SelectedNFRToSelectedVSpec* establece que se creará (si no existe ya) y se resolverá positivamente toda VSpec ($VSpec_b$), equivalente a la $EVSpec_b$ que tiene el campo ImplicadoPorPadre a verdadero (IsImpliedByParent) y que es hija de otra EVSpec ($EVSpec_a$). En la precondición de la regla (cláusula *when*) se establece que previamente la $VSpec_a$, equivalente a la EVSpec padre $EVSpec_a$, tiene que haber sido resuelta positivamente.

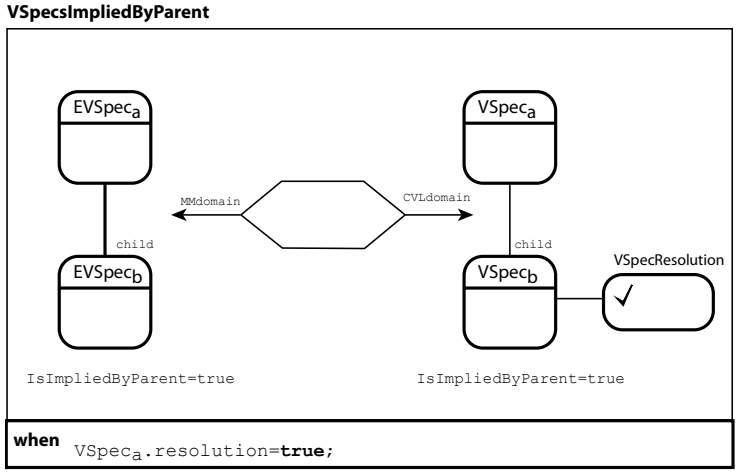


Figura 5.17 Relacion VSpecImpliedByParent

Estas relaciones nos van a permitir obtener un el árbol de resolución CVL correspondiente al modelo de configuración seleccionado. De la definición del metamodelo que soporta al multimodelo (y por tanto al modelo de configuración, el cual es una proyección del primero), es posible demostrar que las relaciones anteriormente descritas no son mutuamente excluyentes (es decir una determinada porción del modelo origen podría unificar con varias reglas distintas a la vez), y todas ellas se ejecutarían, sin que podamos a priori predecir o establecer el orden de ejecución de las mismas, lo cual podría dar lugar a problemas de confluencia (Huet 1980).

Si abstraemos el conjunto de reglas descritas anteriormente como un sistema de reescritura de reglas (Dershowitz y Jouannaud 1990) R denotado por $T(F,X)$ podemos afirmar que:

- *R* es *localmente confluente*, si para cualquier par de términos $s,t,t' \in T(F,X)$ y para cualquier par de reglas $s \to t$ y $s \to t'$, hay un $u \in T(F,X)$ tal que $t \overset{*}{\to} u$ y $t' \overset{*}{\to} u$.
- *R* es *confluente*, si para cualquier par de términos $s,t,t' \in T(F,X)$ y para cualquier par de reglas $s \overset{*}{\to} t$ y $s \overset{*}{\to} t'$, hay un $u \in T(F,X)$ tal que $t \overset{*}{\to} u$ y $t' \overset{*}{\to} u$.

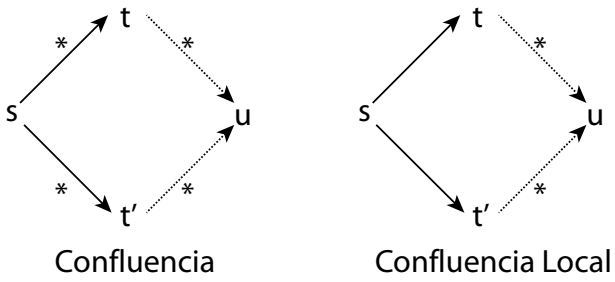


Figura 5.18 Confluencia y confluencia local

Intuitivamente es posible demostrar que el proceso de transformación definido por el conjunto de relaciones de transformación anteriormente descritas es confluyente, puesto que el resultado será el mismo, independientemente de cual sea el orden de aplicación de las reglas (la primera pondrá a verdadero la resolución de una determinada VSpec y a continuación otra podría ponerla de nuevo a verdadero). La Figura 5.19 muestra una configuración de modelo origen (lado izquierdo en la Figura 5.19) que daría lugar a dos secuencias posibles de ejecución de reglas {*SelectedNFRToSelectedVSpec*-*SelectedFeatureToSelectedVSpec*, *SelectedFeatureToSelectedVSpec* – *SelectedNFRToSelectedVSpec*}, aunque sea cual sea la secuencia de ejecución (y las unificaciones posteriores de las reglas que se produjesen a continuación) el resultado sería el mismo (lado derecho de la Figura 5.19).

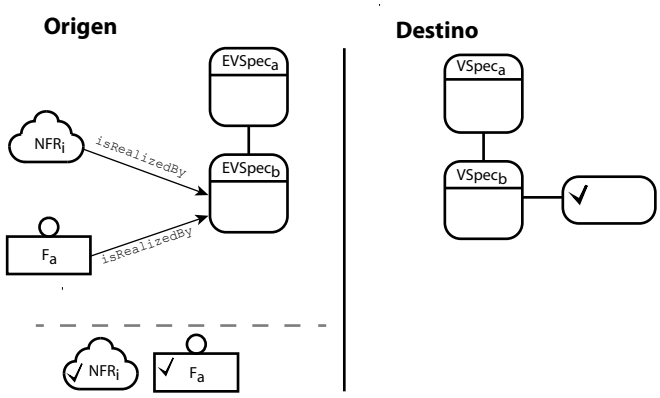


Figura 5.19 Ejemplo de modelo origen que permite la unificación simultanea de dos reglas

La transformación se ejecutará tomando como modelos de entrada el multimodelo y el modelo de configuración y como salida se sobrescribirá una copia del modelo CVL que se utilizó para generar la vista arquitectónica del multimodelo (con todas las VSPecs resueltas negativamente). De este modo el modelo resultante contendrá la especificación CVL cuyo árbol de resolución

CVL únicamente contendrá resoluciones positivas correspondientes a la configuración utilizada como entrada.

Es importante puntualizar que es posible que el modelo de resolución generado tras la transformación sea un modelo no completo, puesto que el arquitecto puede omitir intencionadamente ciertas partes del modelo CVL en el multimodelo con el propósito de que estas queden fuera del proceso de derivación automático para forzar que determinadas partes de la resolución CVL sean definidas de forma manual tras la obtención del modelo de resolución CVL. Esto tiene sentido, puesto que hay ciertas decisiones arquitectónicas que no van a ser fácilmente automatizables incluso con los mecanismos que brinda el método QuaDAI.

5.2.2.2 Consistencia de la resolución CVL generada

El proceso de obtención del modelo de resolución VSpec puede generar árboles de resolución inconsistentes desde el punto de vista de CVL. El estándar CVL tiene sus propios mecanismos para definición de relaciones y restricciones entre sus elementos, mediante un lenguaje de definición de restricciones basado en OCL. Con este lenguaje se pueden definir relaciones de implicación (*implies*), cuantificaciones y restricciones sobre conjuntos de VSpecs, así como definir VSpecs cuya resolución es combinación del resultado de otras VSpecs (Object Management Group 2012a).

Para llevar a cabo la validación de consistencia de los modelos de resolución CVL generados, hemos optado por una aproximación perezosa (Lazy), en el sentido de que permitimos generar un modelo de resolución no valido, que no cumple las restricciones impuestas en el modelo CVL y delegamos su validación a las herramientas de soporte de CVL.

La Figura 5.20 muestra, en la parte izquierda, un fragmento de un modelo de configuración, que a la vista de los elementos que se muestra en ella, se consideraría una configuración de producto válida siguiendo los criterios establecidos en la sección 5.1.2. En la parte derecha de la figura, se muestra el modelo CVL generado con una relación de implicación insatisfecha, lo cual en términos de la especificación CVL sería considerado un modelo de resolución inconsistente, hecho que sería detectada por el validador de CVL.

En el caso en que hayamos generado un modelo inconsistente desde el punto de vista de CVL, las relaciones explícitas del multimodelo y los modelos de trazas generados en la transformación QVT nos proveen de la herramientas de trazabilidad necesarias para detectar el origen de esa inconsistencia. Estos

mecanismos nos permiten detectar por qué el modelo CVL generado es inconsistente y proceder a su corrección, modificando la selección de características, de requisitos no-funcionales o la prioridad de los atributos de calidad. El origen de esta inconsistencia también puede deberse a inconsistencias o errores en las relaciones entre elementos de las distintas vistas en cuyo caso, la producción de un modelo de resolución VSpec incorrecto y su validación mediante la herramienta de soporte de CVL nos permite dar realimentación al multimodelo.

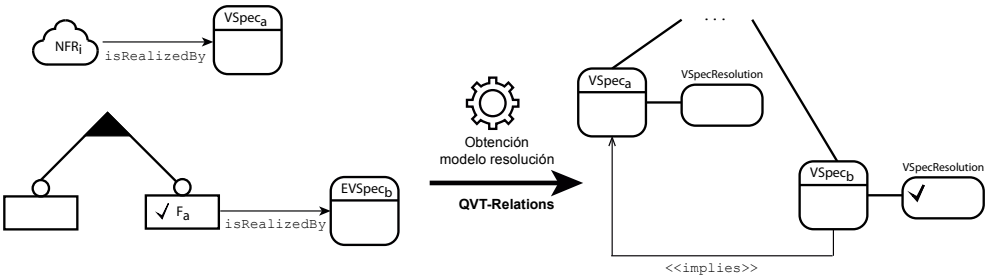


Figura 5.20 Ejemplo de generación de modelo de resolución CVL inconsistente

El hecho de optar por una aproximación perezosa en el caso de la validación de la consistencia CVL se debe además al intento de desacoplar los roles de arquitecto e ingeniero de aplicación. La configuración es llevada a cabo por el ingeniero de aplicación que puede tener un conocimiento limitado de la arquitectura y por tanto no saber interpretar de forma correcta las restricciones del modelo CVL. Será en este caso el arquitecto, el responsable de tomar las decisiones de diseño pertinentes, o bien corregir el modelo de resolución CVL para que genere un modelo valido, acorde con las restricciones de la especificación CVL, o bien informar al ingeniero de aplicación para que sea este el que modifique la configuración inicial, partiendo de la información de trazabilidad, para hacerla acorde con las restricciones de la especificación CVL y que genere de forma automática un nuevo modelo de resolución valido.

Es por todo ello que pensamos que la aproximación seguida para la validación de consistencia del modelo CVL generado resulta robusta para la generación de árboles de resolución VSpec correctos a partir del modelo de configuración.

5.2.3 Materialización de la arquitectura de producto

Por último, durante la materialización de la arquitectura de producto, el modelo de resolución VSpec validado y que representa una resolución correcta desde el punto de vista VSpec, podrá ser ejecutado empleando el mecanismo

de transformación CVL mediante las herramientas que dan soporte a CVL, dando como resultado el modelo que representa la arquitectura del producto en desarrollo.

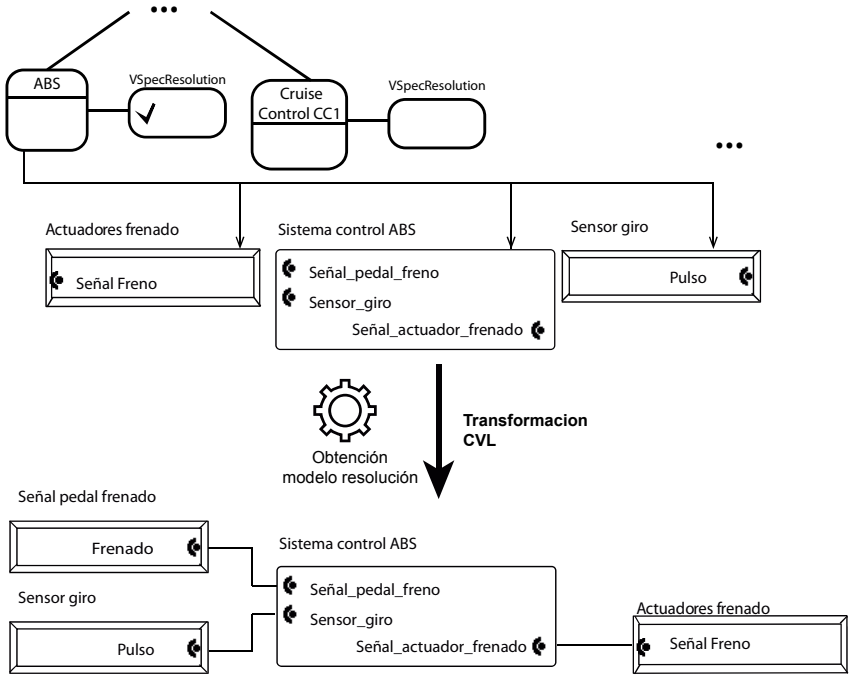


Figura 5.21 Esquema de obtención del modelo arquitectónico a partir de la resolución CVL

Esta arquitectura de producto será la entrada de la actividad de evaluación de la arquitectura con el fin de validar el grado de cumplimiento de los requisitos no-funcionales que hemos definido en la vista de calidad del multimodelo.

5.3 Conclusiones

Este capítulo ha presentado en detalle la primera de las fases del proceso QuaDAI, la derivación de la arquitectura de producto. Se ha presentado el proceso de modelado de la vista de variabilidad arquitectónica, la tarea de configuración del producto, se ha introducido la problemática de la consistencia entre vistas y se han definido las restricciones de consistencia necesarias para poder validar la consistencia en el modelo de configuración, la derivación del modelo de resolución CVL y su posterior transformación en la arquitectura del producto.

Con respecto a la configuración del producto se ha formalizado el concepto de configuración y las restricciones que nos permiten asegurar la consistencia de los modelos de configuración. Además estas restricciones han sido definidas en la herramienta de soporte al metamodelo lo que nos permite validar la consistencia de los modelos generados de manera automática. Entendemos que el mecanismo impaciente (*eager*) de validación de relaciones mediante restricciones propuesto puede ser complejo y que en determinados escenarios complicaría el proceso de configuración y el hecho de que el modelo pase a ser *no-valido* por violar alguna restricción de consistencia puede ser visto como demasiado inflexible, dado que en la mayoría de ocasiones cada producto desarrollado probablemente añadirá un nuevo delta de comportamiento o de calidad a los productos que pueden ser desarrollados a partir de la línea de productos. Ahora bien, el soporte a la configuración descrito en este capítulo establece los artefactos, los mecanismos y el proceso para llevarla a cabo. La validación de la consistencia mediante restricciones no es más que una operacionalización de uno de los mecanismos. Una de las extensiones a este trabajo que planteamos como trabajos futuros es el estudio de nuevas formalizaciones de la consistencia (con restricciones menos estrictas que las descritas en este capítulo). Como alternativas al mecanismo de las restricciones descritas nos planteamos el uso de lógicas difusas o cadenas de Markov y explorar la vía probabilística.

Con respecto a la instanciación de la arquitectura, se ha definido el proceso de definición del modelo de variabilidad arquitectónica que va a soportar la vista de variabilidad. Este modelo además de soportar la vista arquitectónica va a ser el artefacto que recibe la resolución de la variabilidad arquitectónica como resultado del proceso de transformación que toma como entradas las relaciones del multimodelo y el modelo de configuración. Además hemos ilustrado la estructura de las reglas de transformación de dicho proceso de transformación mediante pseudocódigo QVT. Hemos introducido la problemática de la consistencia en los modelos de resolución CVL y justificado el uso de los validadores propios de las herramientas de soporte a CVL para su validación. Por último se ha ilustrado la estructura que tendría el proceso de transformación CVL que genera la primera versión de la arquitectura de producto.

Con respecto al proceso de obtención del modelo de resolución CVL y la posterior materialización de la arquitectura de producto, éstos han sido definidos de forma genérica a partir de la última especificación disponible del estándar (Object Management Group 2012a). A día de hoy hay disponible una implementación de este estándar definida por Haugen et al. (2010) que se basa

en un metamodelo propio compatible, en cierto grado, con el estándar. Es por ello que hemos preferido en la contribución metodológica de esta tesis trabajar con el estándar lo cual hace posible que los conceptos, procesos y artefactos sean adaptados, estableciendo las correspondencias entre conceptos necesarias, a cualquier futura implementación de dicho estándar.

Capítulo 6

Evaluación y Mejora de Arquitecturas

El presente capitulo define la fase de evaluación y mejora de arquitecturas software de QuaDAI, que emplea el multimodelo para evaluar y, en caso necesario, transformar la arquitectura. En esta fase se llevará a cabo la medición de los atributos de calidad y se validará automáticamente que los valores medidos cumplen con los valores especificados en los requisitos no-funcionales por el ingeniero de aplicación durante la configuración. En aquellos casos en que la arquitectura no cumpla dichos requisitos, se aplicarán transformaciones arquitectónicas para tratar de mejorar los valores de los atributos de calidad.

La sección 6.1 presenta el procedimiento de evaluación para arquitecturas de producto basado en métricas que permite la validación del cumplimiento de los requisitos no-funcionales definidos.

La sección 6.2 presenta el proceso de transformación dirigida por atributos de calidad, incluyendo las directrices de diseño para el desarrollo de transformaciones dirigidas por atributos de calidad y el proceso de *tradeoff* entre transformaciones alternativas.

6.1 Evaluación de arquitecturas de producto

En la fase de evaluación de la arquitectura, se aplican las métricas definidas en el modelo de calidad mediante distintos medios:

- **Medición mediante restricciones OCL:** en aquellos casos en que las métricas se puedan operacionalizar en este leguaje, los modelos arquitectónicos (o el propio modelo de resolución que la representa) pueden ser medidos mediante restricciones OCL e interpretadas directamente sobre los modelos a nivel M1 mediante el *plug-in* OCLTools (Eclipse 2013b). Estas restricciones pueden ser empleadas como validación de consistencia del modelo arquitectónico obtenido (por ejemplo, para validar el consumo de memoria de todos los componentes que integran la arquitectura mediante atributos derivados).

- **Medición mediante las herramientas de modelado arquitectónico:** en aquellos casos en los que las herramientas de modelado arquitectónico poseen mecanismos para la medición, ésta se delegará en dichos mecanismos. Uno de los beneficios adicionales de emplear CVL, es aislar la definición de la variabilidad arquitectónica de los lenguajes y herramientas de definición y modelado arquitectónico y permitir el uso de las herramientas nativas con el fin de llevar a cabo el análisis de la arquitectura, tras su derivación y/o transformación.

- **Medición mediante procesos de transformación de modelos:** para aquellas métricas que requieren un procesamiento más complejo, se aplicará la medición mediante transformaciones aplicando aproximaciones como la de Mora et al. (2008), Gómez (2012), en las que se automatiza el proceso de medición mediante transformaciones de modelos QVT dando como resultado un modelo de medición.

- **Transformación de los modelos arquitectónicos para hacerlos compatibles con otros lenguajes o formalismos:** en aquellos casos en los que el proceso de medición sea más sencillo si éste se realiza empleando otros formalismos o lenguajes, los modelos arquitectónicos se transformaran a estos lenguajes. La nueva representación de los modelos arquitectónicos nos permite derivar los valores de las métricas o aplicar métricas empleando herramientas de terceros.

El soporte que el método QuaDAI brinda al procedimiento de medición es que los requisitos no-funcionales pueden ser validados en relación con los valores de las métricas obtenidas siguiendo los procedimientos descritos anteriormente. Para ello, la infraestructura de gestión del multimodelo permite introducir el resultado de los procesos de medición en la metaclase *Measurement*

y efectuar la validación de las restricciones OCL de los requisitos no-funcionales en tiempo de ejecución mediante el validador de OCLTools.

Este soporte, además de automatizar la parte de validación, nos brinda un mecanismo de trazabilidad que nos permite ver cuál de los requisitos no-funcionales no se cumple tras la medición y si ese incumplimiento implica que la configuración no es válida (violación de una restricción fuerte) o, si por el contrario, se trata de un requisito no-funcional de meta, que expresa criterios deseables.

En las siguientes subsecciones se van a ilustrar el proceso de medición y su posterior validación, de dos requisitos no-funcionales, RNF_001 y RNF_002, sobre dos sistemas de control de vehículos: el sistema ABS en el primer caso y el control de crucero básico en el segundo. Dichos atributos de calidad hacen referencia a la *tolerancia a fallo* (en el caso del RNF_001) y al *tiempo de latencia* (en el caso del RNF_002), los cuales se definen a continuación:

> ***Tolerancia a fallo:*** *Grado en que un sistema, producto o componente funciona como se espera a pesar de la presencia de fallos hardware o software* (ISO 2005).
>
> ***Tiempo de latencia:*** *Tiempo transcurrido entre la activación de un suceso de entrada y la obtención de una respuesta* (Bass et al. 2003).

El requisito RNF_001 especifica que la probabilidad ha de ser menor que 5*10e-5 y el RNF_002 especifica que es deseable (se ha definido como meta) que el tiempo de latencia sea menor que 50 μs. La Figura 6.1 muestra el modelo de configuración con la definición en la vista de calidad de los dos RNFs.

Estos requisitos no-funcionales se van a evaluar mediante dos métricas: la probabilidad de fallos (extraída del árbol de fallos) y el análisis del tiempo de latencia. La primera a partir de la transformación de la especificación AADL en otro formalismo (los árboles de fallos) y la segunda mediante la herramienta OSATE (Lewis et al. 2012), que soporta el lenguaje de descripción arquitectónica AADL.

6.1.1 Medición mediante transformación a otros formalismos

El lenguaje de descripción de arquitecturas AADL (Feiler et al. 2006), incluye una serie de mecanismos de extensión del lenguaje para definir conjuntos de propiedades y lenguajes anexos (Feiler y Gluch 2013). Uno de esos lenguajes es la librería del anexo de errores (*Error Annex Library*) que permite la definición de modelos de error.

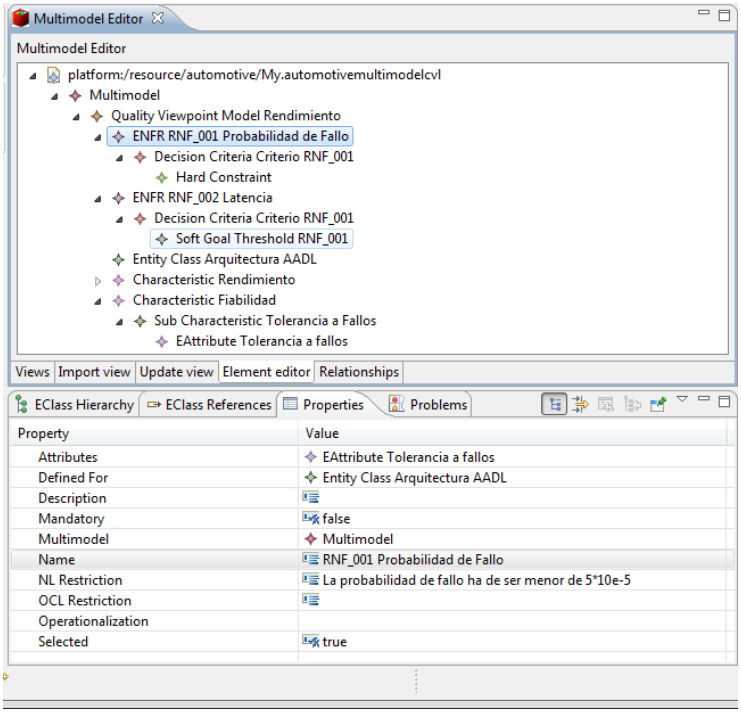


Figura 6.1 Especificación de los RNF_001 y RNF_002 en el modelo de configuración

Con el anexo de errores se pueden definir modelos de error como máquinas de estados. El Listado 6.1 muestra la definición de un modelo de fallos con tres estados: libre de fallos (*Error_Free*), parado (*Stopped*) y fallo activo (*Active_Fault*), además de los eventos que transitan a los estados y las probabilidades de ocurrencia de cada evento, así como la propagación de los errores.

Listado 6.1 Definición de modelo de errores mediante el anexo de errores de AADL

```
1  package Error_Models
2  public
3    annex Embedded_Error_Model {**
4                     error model Three_State
5                     features
6                               Fail_Stop: error event {ocurrence => poisson 10E-4};
7                               Fail_Active: error event {ocurrence => poisson 10E-5};
8                               No_Data, Sensor_error, No_Output_data, Aactuator_error,:
9                                                 in out error propagation;
10                              Error_Free: initial error state;
11                              Stopped, Active_Fault: error state;
12                    end Three_State;
13
14           **};
15 end Error_Models;
```

Estos modelos de estados pueden ser referenciados desde cualquier modelo AADL utilizando la subcláusula *annex* en cualquier especificación de tipo o de implementación de tipo (Feiler y Gluch 2013), tal como se muestra en el Listado 6.2.

A partir de los modelos con anotaciones con el anexo de errores, (Y. Li et al. 2011) proponen un método, basado en el algoritmo de *trace-route* con el que extraer el árbol de fallos del sistema y con el que realizar el análisis de la tolerancia a fallos.

El análisis del árbol de fallos es una técnica analítica donde se especifica un estado no deseado (normalmente desde el punto de vista de seguridad o fiabilidad), y se analiza entonces el sistema en el contexto de su ambiente de trabajo y sus modos de operación para encontrar, de una manera realista, todas las maneras en las que se puede alcanzar ese estado no deseado (Stamatelatos et al. 2002). El árbol de fallos es, en sí mismo, un modelo de las distintas combinaciones paralelas o secuenciales que llevarán a que el estado no deseado ocurra. Los fallos pueden ser eventos asociados a componentes hardware, errores humanos, componentes software o cualquier otro tipo de eventos que puedan derivar en el estado no deseado (Stamatelatos et al. 2002).

Listado 6.2 Ejemplo de declaración AADL con subcláusula *annex*

```
system implementation Sistema_ABS.impl
          subcomponents
          pedal: device pedal_freno;
          rotacion: device sensor_rotacion_rueda;
…

    annex Embedded_Error_Model
          {**
          model => Embedded_Error_Model::Three_State;
          Ocurrence => poisson 10E-3 applies to Fail_Stop;
          Ocurrence => poisson 10E-4 applies to Fail_Active;
          **};
end Sistema_ABS.impl;
```

Las posibles causas (eventos) del estado no deseado se unen mediante puertas lógicas (AND, OR, XOR,…). La Tabla 6.1 muestra la sintaxis gráfica, junto con la descripción y la fórmula de acumulación de probabilidades.

Aplicando el método propuesto por Y. Li et al. (2011) a la arquitectura AADL de la Figura 6.2 se obtiene el árbol de fallos asociado, que se muestra en la Figura 6.3.

Tabla 6.1 Elementos principales del árbol de fallos

Elemento	Descripción	Cálculo de la probabilidad
	Evento básico: describe un fallo iniciador básico que no requiere más descomposición.	P(A)
	AND: la puerta AND indica que el fallo de salida *Q* solamente ocurre si ocurren todos los fallos de entrada de esa puerta *(A, B).*	P(Q) = P(A) P(B)
	OR: la puerta OR indica que el fallo de salida *Q* ocurre si se da al menos uno de los fallos de entrada de esa puerta *(A, B).*	P(Q) = P(A) + P(B)
	Combinación: el evento de salida *Q* ocurre únicamente si n fallos de entrada ocurren simultáneamente *(A, B, C).*	P(Q)=P(A)*P(B) + P(A)*P(C) + P (B)*P(C) En el caso en que P(A)=P(B)=P(C) entonces P(Q)=3*(P(A))^2

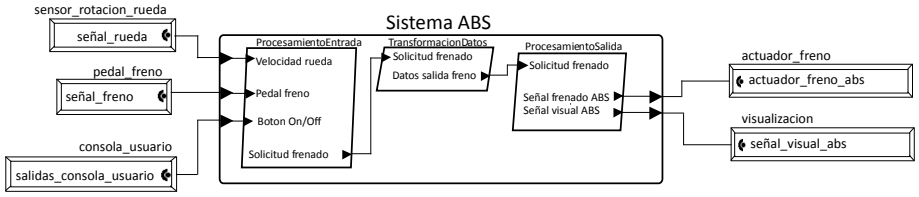


Figura 6.2 Arquitectura AADL del sistema ABS

La probabilidad de fallo asociada se obtiene aplicando el cálculo de probabilidades asociado a cada uno de los elementos del árbol de fallos, que se muestran en la Tabla 6.1, mediante la fórmula (1). Si consideramos para ese cálculo todos los sucesos igualmente probables con una probabilidad de 10^{-4}, la probabilidad de fallo total del sistema ABS será de 0,00006. Dicho valor se introduce en el atributo *ActualValue* de las instancias de la metaclase *measurement*, asociadas al requisito no-funcional de *tolerancia a fallos* en la vista de calidad del multimodelo. A partir de ese valor se procederá a validar el grado de cumplimiento de dicho requisito.

$$P(FalloABS) = P(FalloSensor) + P(FaltaEntrada) + P(FalloProcesamientoEntrada) + P(Actuador) \quad (1)$$

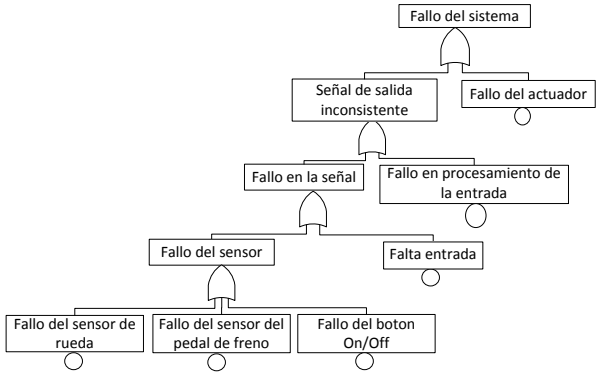


Figura 6.3 Árbol de fallos del sistema ABS

La medición sistemática de atributos de calidad haciendo uso de otros lenguajes o formalismos es de gran utilidad para la estrategia de derivación y transformación, puesto que nos permite evaluar las arquitecturas obtenidas tras los procesos de derivación y transformación mediante la agregación de información de los diferentes componentes arquitectónicos, disponible en la descripción arquitectónica.

6.1.2 Medición mediante herramientas de modelado arquitectónico

Las herramientas de modelado y análisis de arquitecturas software incluyen, en la mayoría de los casos, módulos de análisis que permiten la evaluación y medición de las arquitecturas generadas, como es el caso de OSATE (Lewis et al. 2012), que es la herramienta que soporta el lenguaje de descripción de arquitecturas AADL.

AADL permite la definición de propiedades definidas por el usuario para poder llevar a cabo el análisis de dichas propiedades de manera automática (Feiler y Gluch 2013). Además, incluye algunas propiedades de manera nativa como es el caso del análisis del tiempo de latencia.

Para ello OSATE cuenta con un módulo que permite el análisis del tiempo de latencia (Feiler 2007a), a partir de la especificación AADL con anotaciones para su cálculo.

El Listado 6.3 muestra un resumen de la definición en AADL del control de crucero básico (cuya estructura se muestra en la Figura 6.4), en las líneas 10-11 se especifica el flujo de datos de la señal de frenado que emerge del sensor de

frenado y su conexión al sistema software de control de crucero hasta la conexión al actuador de frenado. Este flujo ha sido anotado con la latencia esperada (50 μs). El sistema calculará este valor a partir de la latencia de los distintos flujos de los componentes internos (dispositivos, sistemas, procesos, *threads*…) que componen el sistema.

Listado 6.3 Definición de las propiedades de latencia del control de crucero básico

```
1   system implementation cc1_app.impl
2     subcomponents
3       CC: system cruise_control_CC1.impl;
4       BRAKE_PEDAL_SENSOR: device brake_pedal;
5       …
6     connections
7       …
8       C1: data port BRAKE_PEDAL_SENSOR.brake_pedal_status -> CC.brake_pedal_status;
9       C2: data port CC.throttle_setting -> TA.throttle_setting;
10      …
11    Flows
12      …
13      CC_BrakeSignalFlow: end to end flow BRAKE_PEDAL_SENSOR.Flow1 -> C1 -> CC.brake_flow_1
14         -> C2 -> TA.Flow1 {latency=>50Us;};
15      …
16  end cc1_app.impl;
```

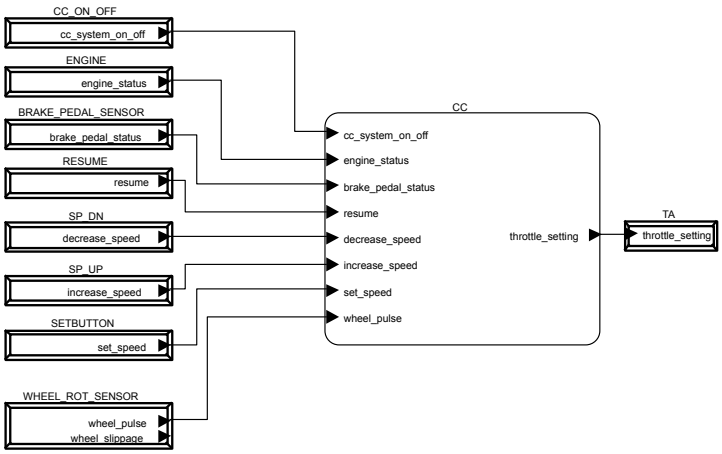


Figura 6.4 Arquitectura AADL del sistema de control de crucero básico cc1_app.impl

El Listado 6.4 muestra la especificación AADL de la implementación del sistema *cruise_control_CC1.impl* (su estructura se muestra en la Figura 6.5), que es uno de los componentes integrados en el sistema *cci1_app.impl* (Listado 6.3 – línea 3). En dicha especificación se detallan de nuevo los tiempos de latencia esperados para cada uno de los flujos de datos que integran la implementación de este sistema.

Listado 6.4 Definición de las propiedades de latencia sobre la implementación para el componente *cruise_control_CC1.impl*

```
1  system implementation cruise_control_CC1.impl
2    subcomponents
3      In_Control: process in_control;
4      Compute_Velocity: device compute_velocity;
5      C_D_S: process compute_desired_speed;
6      C_T_S: process compute_throttle_setting;
7    Connections
8      …
9      C1: data port cc_system_on_off -> In_Control.cc_system_on_off;
10     …
11   flows
12     brake_flow_1: flow path brake_pedal_status -> C2 -> In_Control.FS1
13        -> C9 -> C_D_S.FS1
14        -> C12 -> C_T_S.FS1
15        -> C13 -> throttle_setting {latency=>40 Us;};
16     decrease_speed_flow1: flow path decrease_speed -> C5 -> In_Control.FS2
17        -> C14 -> C_D_S.FS3
18        -> C12 -> C_T_S.FS1
19        -> C13 -> throttle_setting {latency=>55 Us;};
20 …
21 end cruise_control_CC1.impl;
```

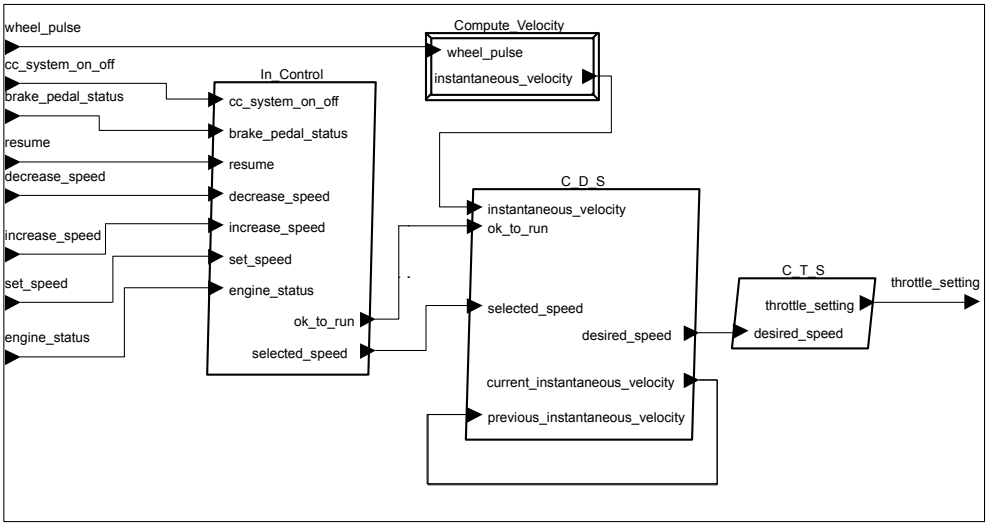


Figura 6.5 Estructura interna del componente *cruise_control_CC1.impl*

El análisis de latencia irá jerárquicamente calculando las latencias de cada uno de los elementos hasta llegar a los componentes elementales y para cada componente, asigna como tiempo de latencia el mayor de entre los tiempos de latencia de los flujos de datos (o eventos) de ese componente. Una vez calculados los compara con los valores esperados con el fin de verificar si alguno incumple la especificación.

OSATE realiza la medición del tiempo de latencia sobre la instancia del sistema y genera como salida un informe de errores y un fichero CSV con los resultados medidos frente a los esperados. La Figura 6.6 muestra el resultado de la medición del tiempo de latencia y el incumplimiento del valor esperado para el flujo de la señal de frenado (Listado 6.3 – línea 13).

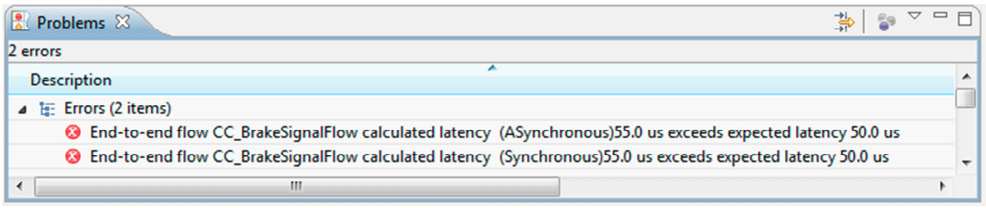


Figura 6.6 Informe de medición del tiempo de latencia en OSATE

El valor final (tiempo de latencia del flujo más lento, en este caso *CC_BrakeSignalFlow*) se introduce en el atributo *ActualValue* de las instancias de la metaclase *measurement*, asociadas al requisito no-funcional de *tolerancia a fallos* en la vista de calidad del modelo de configuración. A partir de ese valor se procederá a validar el grado de cumplimiento de dicho requisito.

La medición utilizando mecanismos como el que ofrece OSATE para el análisis de propiedades de la arquitectura (en este caso, el tiempo de latencia), nos provee del grado de flexibilidad necesaria para evaluar las arquitecturas una vez han sido derivadas y/o transformadas tras la actividad de transformación de arquitectura aprovechando las características propias del lenguaje. En el caso de este ejemplo el lenguaje AADL, en el que se especifica la arquitectura, tanto de la línea de productos con del producto en desarrollo nos ofrece las abstracciones necesarias para la medición y evaluación de ciertas propiedades no-funcionales del sistema. De forma general, la evaluación del cumplimiento de los requisitos no-funcionales se llevará a cabo mediante los mecanismos previstos en el multimodelo, aunque como en el caso del ejemplo OSATE también es capaz de validar el cumplimiento de los mismos.

6.1.3 Validación de los requisitos no-funcionales

Tras los distintos procesos de medición y una vez incorporados los datos de medición en la vista de calidad del multimodelo, la infraestructura de soporte del multimodelo permite la validación de aquellos requisitos no-funcionales a los que se les haya asociado una fórmula OCL. La Figura 6.7 muestra la vista de calidad del modelo de configuración con los valores medidos para el requisito no-funcional RNF_001 y RNF_002.

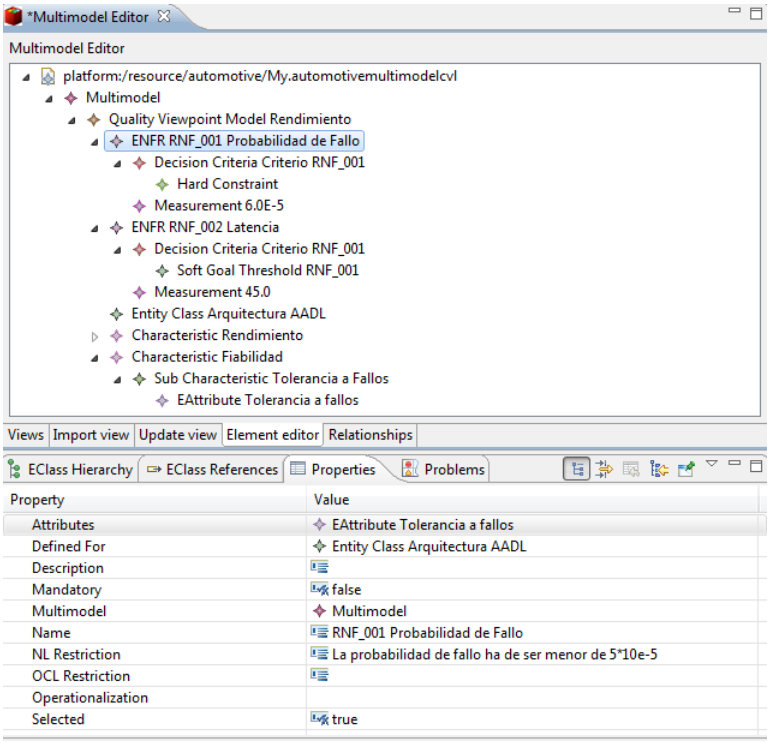


Figura 6.7 Resultado de la medición en el modelo de configuración

En el caso de los RNFs de tipo *Softgoal* se verifica si está por encima o por debajo del valor del umbral, en función de si el umbral tiene orientación positiva o negativa. En el caso de *HardConstraints*, se comprueba que el valor medido está entre los dos valores establecidos en el umbral.

Esta acción puede realizarse para requisitos no-funcionales individualmente o para el modelo completo, en cuyo caso únicamente se validaran aquellos requisitos no-funcionales seleccionados para el producto en cuestión (aquellos que tienen a verdadero el campo *Selected)*. La validación en el caso del ejemplo, será de incumplimiento en ambos casos.

- En el caso del RNF_001, el valor medido 6*10e-5, es mayor que el valor umbral (5*10e-5) y con ello se viola el requisito, cuyo umbral especifica una restricción fuerte.
- En el caso del RNF_002 el valor medido 55 μs, es mayor que el umbral de 50 μs, lo cual viola el requisito. Pero en este caso, dicho requisito

solo expresa un valor deseable, por lo que el producto en cuanto a este requisito, podría considerarse aceptable.

Para el primer caso, será necesaria la transformación de la arquitectura con el fin de tratar de mejorar la fiabilidad de la misma sin afectar a su funcionalidad, mediante la aplicación de transformaciones arquitectónicas.

6.2 Transformación de arquitecturas dirigida por atributos de calidad

La tarea de transformación de arquitecturas, seleccionará y aplicará aquellas transformaciones arquitectónicas que den un mejor soporte a los atributos de calidad priorizados por el ingeniero de aplicación en la configuración del producto. Para ello, será necesario adaptar los procesos de transformación para permitir que las decisiones sean tomadas en base a los requisitos de atributos de calidad. Con este fin, se llevarán a cabo dos actividades en la fase de ingeniería de dominio: la aplicación de las *directrices de diseño para la definición de transformaciones arquitectónicas dirigidas por atributos calidad* y el *tradeoff entre transformaciones alternativas*. La Figura 6.8 muestra el diagrama de actividades con cada una de las actividades involucradas en la definición y aplicación de transformaciones arquitectónicas dirigidas por atributos de calidad.

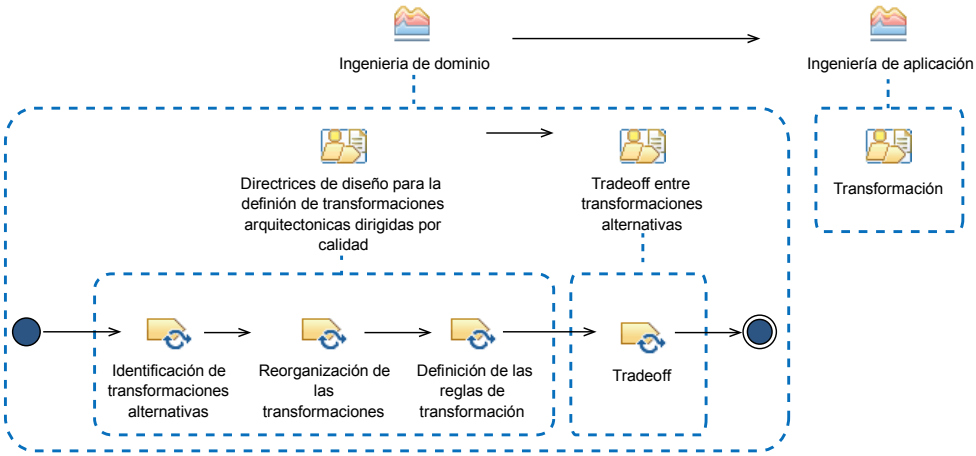


Figura 6.8 Diagrama de actividad para las directrices de diseño, *tradeoff* y transformación

La Figura 6.9 muestra el diagrama de actividad detallado con las entradas, salidas y el rol responsable de cada una de las tareas que forman parte de la fase de las actividades relacionadas con las directrices de diseño y el *tradeoff*.

El punto de partida de la aplicación de las directrices de diseño es un conjunto de transformaciones (y sus correspondientes reglas de transformación) en el lenguaje de transformación de modelos seleccionado. El resultado de las directrices de diseño será un conjunto de procesos de transformación adaptados al esquema de transformaciones dirigidas por atributos de calidad, así como la vista de transformaciones del multimodelo. En el *tradeoff* entre alternativas, se definirá como cada alternativa soporta los atributos de calidad de interés, estableciendo las relaciones preceptivas entre alternativas y atributos de calidad en el multimodelo. En la fase de ingeniería de aplicación, la actividad de transformación empleará los procesos de transformación resultante de las actividades de la ingeniería de dominio, junto con las prioridades de los atributos de calidad para seleccionar y aplicar las transformaciones que mejor se adapten a dichos atributos.

Las siguientes sub-secciones describen en detalle las directrices de diseño, la actividad de *tradeoff* y por último, el proceso de transformación de la arquitectura de producto.

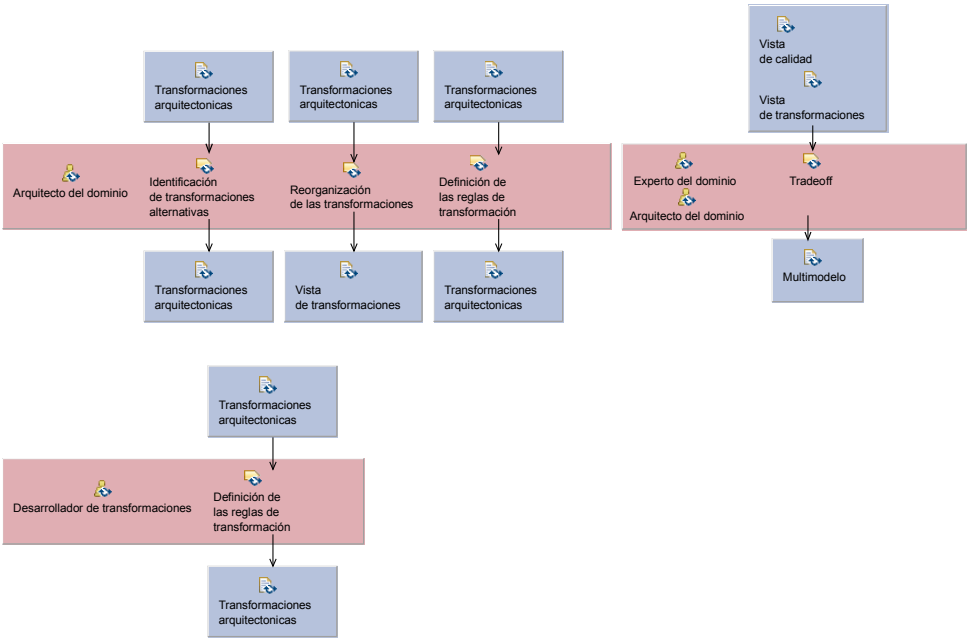


Figura 6.9 Diagrama de actividad detallado para las directrices de diseño y la actividad de *tradeoff*

6.2.1 Directrices de diseño para la definición de transformaciones arquitectónicas dirigidas por atributos de calidad

Esta sección presenta las directrices de diseño para la definición de transformaciones arquitectónicas dirigidas por atributos de calidad, como vía para mejorar los niveles de los atributos de calidad en casos en los que los mecanismos de variabilidad arquitectónica no permiten alcanzar los requisitos no-funcionales del producto en desarrollo. Estas directrices facilitan la definición de procesos de transformación de modelos en los que se automatiza la selección y aplicación de transformaciones arquitectónicas en base a los atributos de calidad que se pretenden mejorar. El objetivo de la aplicación de estas directrices de diseño, es construir un conjunto de procesos de transformación (cuyas decisiones de diseño y transformaciones alternativas estarán descritas en la vista de transformaciones del multimodelo) que podrán ser aplicadas cada vez que sea necesario mejorar los niveles de atributos de calidad de una arquitectura de producto porque esta no cumple con los requisitos de calidad.

Las transformaciones arquitectónicas se pueden clasificar en tres tipos principales (Bosch 2000):

- *Estilos arquitectónicos:* describen soluciones a problemas arquitectónicos comunes y recurrentes. Son un medio para mejorar ciertos atributos de calidad como la flexibilidad, la modularidad o la fiabilidad (Shaw y Garlan 1996). Algunos ejemplos son, el estilo en capas, el de tubos y filtros (*pipes and filters*) o el estilo centrado en datos (Shaw y Garlan 1996).

- *Patrones arquitectónicos:* especifican cómo el sistema afrontará algún aspecto de su funcionalidad, impactando directamente en los atributos de calidad. No ocupan un lugar destacado en la definición de la arquitectura, al contrario que los estilos, y pueden ser combinados con estos últimos. Algunos ejemplos son, los esquemas de distribución, de concurrencia, el uso de *brokers*, RMI (invocación a métodos remotos), etc… (Kruchten 1999; Douglass 2002).

- *Patrones de diseño:* especifican un medio para mejorar los atributos de calidad de un sistema, sin afectar a su funcionalidad. En general, la aplicación de patrones de diseño afecta únicamente a un número limitado de elementos de la arquitectura, siendo posible su aplicación de manera local, para mejorar algún atributo de calidad. Algunos

ejemplos son el patrón factoría abstracta, el patrón proxy o el patrón *façade* (Gamma et al. 1995).

Las transformaciones arquitectónicas se representan como transformaciones de modelos que van a ser aplicadas por reglas de transformación en forma de relaciones de grafos entre elementos de los modelos de la parte derecha y de la parte izquierda.

Las directrices de diseño se organizan de la siguiente manera: i) identificación de las transformaciones arquitecturales; ii) reorganización de las transformaciones arquitectónicas; iii) definición de las reglas de transformación, cuyos detalles se describen en las siguientes subsecciones.

La aplicación de las directrices se ilustra mediante un ejemplo de sistemas software de control del automóvil que contienen diferentes características, como el control de antibloqueo de frenos ABS, el sistema de control de tracción o el control de crucero. Dichas característica comprenden una serie de sistemas software para la captura de las señales de entrada de los sensores, su procesamiento y transformación y el envío de la salida procesada a un actuador que potencialmente puede afectar al estado de otros sistemas o partes mecánicas del vehículo (como la posición del acelerador, el motor, el airbag o los frenos).

Las directrices de diseño se ilustran empleado la sintaxis de QVT-Relations, pero se pueden aplicar empleando cualquier otro lenguaje de transformación (como ATL o QVT-Operational) estableciendo las correspondencias entre conceptos que sean necesarias.

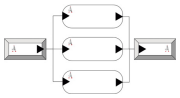
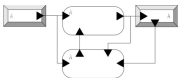
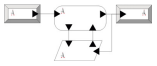
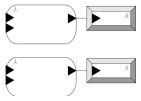
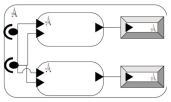
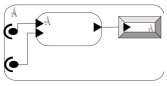
6.2.1.1 Identificación de transformaciones alternativas

Las transformaciones arquitectónicas son un conjunto de transformaciones genéricas que pueden ser aplicadas a una arquitectura. Las transformaciones alternativas se tendrán en consideración en función de los atributos de calidad de interés del dominio y especialmente aquellos que con más probabilidad la arquitectura de producto no vaya a satisfacer. Es posible que determinadas transformaciones arquitectónicas no sean susceptibles de ser aplicadas en este contexto, por tratarse de estilos arquitectónicos que afectan a la organización esencial de la arquitectura (como algunos estilos arquitectónicos), los cuales deberán ser considerados a nivel de la propia arquitectura de producto y no como transformación tras la derivación.

Además, dependiendo del dominio de interés y de la propia topología o naturaleza de la arquitectura, el conjunto de transformaciones a considerar puede verse reducido.

En el ejemplo de los sistemas de control de vehículos, vamos a centrarnos en un conjunto de patrones para sistemas empotrados que se representan como transformaciones arquitectónicas. En la Tabla 6.2 se muestra el catálogo de transformaciones que se ha considerado, el cual está formado por distintos patrones relacionados con la fiabilidad del sistema (patrón de triple redundancia modular, patrón *Sanity check*, y patrón *Watchdog*) y otros relacionados con la mantenibilidad y reusabilidad de la arquitectura (patrón de control jerárquico y el contenedor recursivo) (Douglass 2002).

Tabla 6.2 Catalogo de transformaciones arquitectónicas

Id	*Nombre*	*Descripción*	*Parte Izquierda*	*Parte Derecha*
T1	Triple redundancia modular	Sólo detecta errores aleatorios. Dado que los canales son homogéneos, cualquier fallo sistemático en un canal debe estar presente en todos los demás, aunque se trata de un patrón que ofrece robustez y es ampliamente empleado para solventar problemas de fiabilidad.		
T2	*Sanity check*	Detecta desviaciones graves entre el valor de control (salida) y el del actuador. Proporciona una cobertura mínima contra fallos.		
T3	*Watchdog*	Patrón muy ligero que provee de una cobertura mínima frente a fallos. Comprueba que los cálculos internos se están llevando a cabo satisfactoriamente, por lo que un amplio rango de fallos no será detectado.		
T4	Control jerárquico	Permite la visualización de sistemas complejos en muchos niveles de abstracción. El uso de interfaces de control y funcionales proporciona un enfoque sencillo, escalable y útil para definir sistemas altamente configurables.		
T5	Contenedor recursivo	Empleado en sistemas altamente complejos, permite mejorar la escalabilidad mediante la creación, de manera recursiva componentes opacos cada vez más pequeños que encapsulan su funcionalidad.		

6.2.1.2 Reorganización de las transformaciones

El segundo paso consiste en distribuir las transformaciones seleccionadas en diferentes procesos de transformación, que serán reorganizadas a su vez en decisiones de diseño. La reorganización de transformaciones en procesos de transformación diferenciados, permite programar la secuencia en que se aplica

cada tipo de transformaciones arquitectónicas, respondiendo al mismo principio por el que los procesos anexos a los activos software (Clements y Northrop 2001) pueden especificar la secuencia en que se deben resolver los puntos de variación de los distintos activos software (McGregor 2010).

Las transformaciones arquitectónicas se agrupan en una misma decisión de diseño si representan transformaciones alternativas, bien porque concuerdan con el mismo conjunto de entidades del modelo origen o bien por que representan decisiones excluyentes (por ejemplo la aplicación de asignación estática o dinámica).

Esta jerarquía de procesos de transformación, las decisiones de diseño en los distintos procesos de transformación, las alternativas en esas decisiones de diseño y las reglas capaces de aplicar dichas transformaciones se representan en la vista de transformaciones del multimodelo (véase Sección 4.2.2.4). La representación de las reglas de transformación permite su reúso desde otras alternativas, en la misma o en otras decisiones de diseño de la transformación en la que se definen.

El catálogo de transformaciones para el ejemplo de los sistemas de control de vehículos que se muestra en la Tabla 6.2 se reorganiza en dos procesos de transformación. El primer proceso de transformación (*redundancia y monitorización)*, contiene la decisión *fiabilidad* que define la agrupación de las tres transformaciones alternativas {T1, T2 y T3}, ya que tienen que ver con los aspectos fiabilidad y, tal como se aprecia en la Tabla 6.2, comparten el mismo lado izquierdo de las transformaciones (están definidas sobre el mismo conjunto de entidades del modelo origen). El segundo proceso de transformación, (o*rganización)*, contiene la decisión *mantenibilidad* que define la agrupación de las dos transformaciones alternativas {T4 y T5}, dado que están relacionadas con aspectos de mantenibilidad, y al igual que en el caso anterior, comparten el lado izquierdo de la transformación.

Tras la reorganización de las transformaciones, estas se representarán en la vista de transformaciones del multimodelo, que contendrá las dos transformaciones. La transformación "*redundancia y monitorización"* se ha definido como instancia de la metaclase *transformación*, que contiene una instancia de la metaclase *DesignDecision "fiabilidad"* con las tres instancias *EAlternative* {T1, T2 y T3} (*"triple redundancia modular"*, *"sanity check"* y *"watchdog"*). Para la transformación *"organización"* se ha creado una estructura análoga. La Figura 6.10 muestra el editor de la vista de transformaciones de la infraestructura de edición del multimodelo, con la descomposición jerárquica de alternativas de transformación del ejemplo.

En adelante nos vamos a centrar en la decisión de diseño *Fiabilidad*[16] para ilustrar el resto de pasos de las directrices y el *tradeoff* entre alternativas y atributos de calidad.

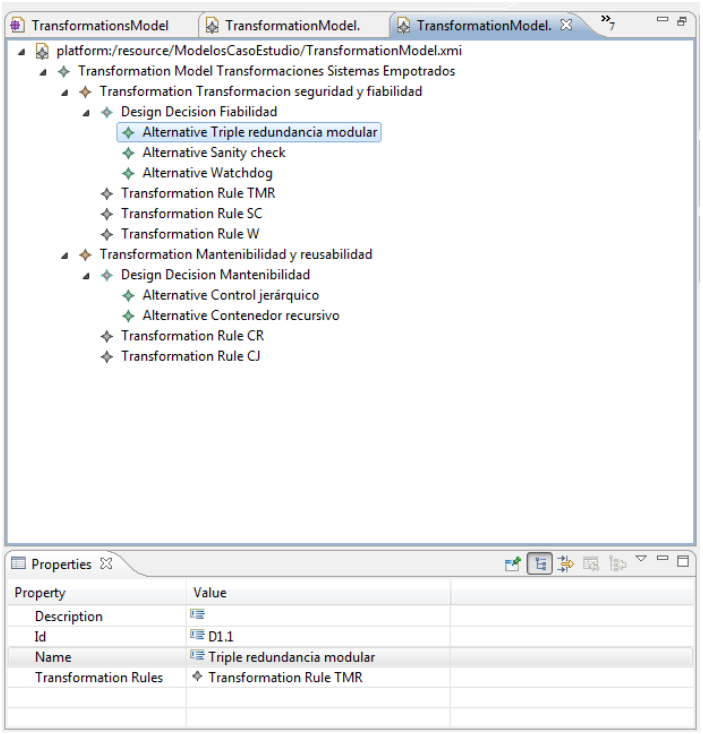


Figura 6.10 Jerarquía de transformaciones en el multimodelo

En esta aproximación distinguimos dos tipos de reglas, las reglas *patrón* y las reglas *discriminante*. Las reglas de transformación existentes, encargadas de la aplicación de cada una de las transformaciones tras la *Identificación de transformaciones alternativas* (ver subsección 6.2.1.1), son adaptadas a la estructura de las *reglas patrón* y por cada decisión de diseño se añade una nueva regla *discriminante*. La estructura de cada uno de los tipos de reglas se describe a continuación:

- **Reglas patrón:** las reglas patrón son responsables de la creación o actualización de los elementos del modelo destino. Las reglas patrón no

[16] A pesar de que se ilustra el proceso para una decisión de diseño con tres alternativas, es posible manejar *n* decisiones de diseño con *m* alternativas en cada decisión de diseño aplicando el mismo esquema.

se ejecutarán de forma automática, han de ser invocadas desde una regla discriminante. Constan de dos dominios (véase Listado 6.5):

- El domino del **modelo origen**, que describe las estructuras que constituyen la parte izquierda de la regla.
- El dominio del **modelo destino**, que especifica las estructuras creadas en el modelo destino.

Las reglas patrón son adaptaciones de las reglas originales que aplicaban cada uno de los patrones, a los que se les elimina la capacidad de ejecutarse automáticamente por unificación con el modelo origen, y para que se ejecuten será necesario que sean invocadas desde una regla discriminante.

- **Reglas discriminante:** se añadirá una regla discriminante por cada decisión de diseño, agrupando todas sus alternativas. Las reglas discriminantes se ejecutan automáticamente cada vez que el conjunto de estructuras en su parte izquierda unifica con un fragmento del modelo origen. Son las encargadas de seleccionar una de las alternativas en la decisión de diseño, en función de la forma en que soporta a los atributos de calidad. Constan de tres dominios (véase Listado 6.6):
  - El dominio del **modelo origen** en el que se define las estructuras que constituyen la parte izquierda de la regla.
  - El dominio del **modelo transformaciones**, en el que se especifica la decisión de diseño a la que pertenece la regla discriminante.
  - El dominio del **modelo destino** que especifica las estructuras creadas en el modelo destino.

  La precondición de la regla, especifica una llamada a una consulta (*Query*) que selecciona la alternativa que da mejor soporte a los atributos de calidad requeridos *Qa*..*Qn*, que provienen de la prioridad de los atributos de calidad del modelo de configuración y son introducidos, como parámetros externos, en el proceso de transformación. Dicha consulta tiene acceso a:
  - Los resultados del *tradeoff* que expresan el soporte que brinda cada alternativa a los atributos de calidad.
  - A los parámetros que expresan la prioridad de los atributos de calidad a ser satisfechos.

La postcondición de la regla lleva a cabo la invocación a la regla patrón correspondiente en base al valor retornado por la consulta de la precondición.

En el ejemplo de los sistemas de control de vehículos, habrá tres reglas patrón asociadas a cada uno de las alternativas de esta decisión de diseño: *TripleModularRedundant* (la cual se muestra en el Listado 6.5), *SanityCheck*, y *Watchdog*.

Listado 6.5 Estructura de la regla patrón

```
1  relation TripleModularRedundant
2  {
3  checkonly domain Source: ...
4  {
5        ...
6  };
7  enforce domain Target: ...
8  {
9        ...
10 };
11 }
```

Por último, se definirá la regla discriminante *Reliability* para la decisión de diseño *Fiabilidad* agrupando todas sus alternativas. Esta regla (cuya estructura se muestra en el Listado 6.6) seleccionará la alternativa que de un mejor soporte a los atributos de calidad de entre las tres posibles alternativas. En el Listado 6.6 puede apreciarse que la regla agrupa las tres posibles transformaciones alternativas y que contiene la invocación a la consulta *BestRankedAlternative* (Listado 6.6 – línea 18) en la cláusula *when* de la regla. Esta consulta accede a los resultados del *tradeoff* entre transformaciones alternativas y a la prioridad de los atributos *Qa…Qn,* que son introducidos como parámetros externos a partir de los valores expresados por el ingeniero de aplicación en el modelo de derivación. En base a esas dos entradas, selecciona la alternativa que da mejor soporte a dichos requisitos. Por último la cláusula *where* (Listado 6.6 – línea 20) realiza las invocaciones a las reglas correspondientes en base al valor retornado por la consulta.

6.2.2 Tradeoff entre transformaciones alternativas

El último paso para la definición de transformaciones arquitectónicas dirigidas por atributos de calidad, es llevar a cabo el análisis *tradeoff* entre alternativas y atributos de calidad. Este proceso, que se ejecuta como parte de las actividades de la ingeniería de dominio, es llevado a cabo bien por un experto del dominio o bien, por el propio arquitecto de dominio y consiste en determinar cómo las diferentes transformaciones alternativas soportan los diferentes atributos de

calidad del dominio, basándose bien en datos empíricos o bien, en su propia experiencia. El análisis *tradeoff* se lleva a cabo aplicado el proceso analítico jerárquico AHP (*Analytic Hierarchy Process*) (Saaty 2008), una técnica para la toma de decisiones ampliamente aplicada para resolver conflictos en los que es necesario hacer frente a comparaciones de múltiples criterios. El resultado del análisis AHP es un valor (o peso) que indica el soporte relativo de una determinada alternativa con respecto a un atributo de calidad. Este resultado se almacenará en el atributo *Importance* de la relación *AlternativeTransformationToQualityAttribute* definida entre transformaciones alternativas y atributos de calidad en el multimodelo (ver Sección 4.2.3). La consulta de la precondición de las reglas discriminantes accederá a esos valores para seleccionar la alternativa que mejor se adapte a la prioridad de los atributos de calidad.

Listado 6.6 Estructura de la regla discriminante

```
1   top relation Reliability
2   {
3      best_quality_support: String;
4      checkonly domain Source: ... {...};
5      enforce domain Target: ... {...};
6      checkonly domain MultiModel Decision1: Transformations::DesignDecision
7      {
9            Id='T1',
10           Name='Reliability',
11           Alternatives=best_alternative: MultiModel::Alternative
12            {
13              Id= best_quality_support
14           }
15     }
16     when
17     {
18           best_quality_support=BestRankedAlternative(Decision1,Qa,Qb,…,Qn);
19     }
20     where
21     {
22           if (best_quality_support ='T1') then
23                     TripleModularRedundancy(S,T)
24              else
25                     if (best_quality_support ='T2') then
26                               SanityCheck(S,T)
27                          else
28                               WatchDog(S,T)
29                     endif;
30           endif;
31     }
32  }
```

En el análisis AHP, se comparan las alternativas de una decisión de diseño en relación a los Q atributos de calidad. Para cada atributo Q_a se comparan las n

transformaciones alternativas *A* en comparaciones dos a dos. Para realizar la comparación empleamos un valor de la escala AHP (Saaty 2008), para expresar cómo una determinada alternativa *Ax* soporta el atributo Qa comparado con el soporte que ofrece *Ay*. La escala AHP es una escala numérica que permite indicar, de forma genérica, cuanto mejor es un elemento con respecto a otro mediante un valor, en un rango que va desde 1 a 9. La definición y descripción de los valores de la escala se detalla en la Tabla 6.3.

Tabla 6.3 Escala AHP para la comparación de alternativas (Saaty 2008)

Intensidad de la importancia	*Definición*	*Descripción*
1	Igual importancia	Las dos alternativas contribuyen de igual modo el atributo de calidad
2	Importancia ligera o débil	
3	Importancia moderada	La experiencia o el juicio favorecen ligeramente una alternativa con respecto a la otra
4	Moderada plus	
5	Importancia fuerte	La experiencia o el juicio favorecen fuertemente una alternativa con respecto a la otra
6	Fuerte plus	
7	Muy fuerte o demostrada	Una alternativa es favorecida de manera muy fuerte. Su dominancia ha sido demostrada en la practica
8	Muy, muy fuerte	
9	Extrema	La evidencia a favor de una alternativa sobre la otra es el orden mayor que se puede afirmar

Los valores de la escala AHP asociados a una comparación son recíprocos, es decir, si en un caso, a la alternativa *Ax* se le asigna una intensidad de 5 respecto a la alternativa *Ay*, la alternativa Ay recibirá un valor 1/5 con respecto a la alternativa *Ax*.

En el ejemplo de los sistemas de control de vehículos nos vamos a centrar en dos atributos de calidad, *tolerancia a fallo* y *tiempo de latencia*[17], definidos en la sección 6.1.

[17] A pesar de que se va a ilustrar el proceso de tradeoff únicamente para dos atributos de calidad y tres alternativas, el mismo esquema puede ser aplicado a *n* atributos de calidad y *m* alternativas.

El experto del dominio (o el arquitecto) realiza la comparación entre las tres alternativas de la decisión de diseño *Fiabilidad,* en relación a la tolerancia a fallos en comparaciones dos a dos empelando la escala AHP. Los resultados de dichas comparaciones se introducen en una matriz de comparación con la que se normalizarán los valores para obtener el resultado final del proceso AHP (la matriz de comparación AHP para la tolerancia a fallos se muestra en la Tabla 6.4(a)). Los ejemplos de las comparaciones dos a dos se detallan a continuación:

- El patrón de triple redundancia modular (TRM) favorece más (con una importancia fuerte) la tolerancia a fallos que el patrón *Watchdog* (W) y se le asigna un valor 5 (Tabla 6.4(a), fila TRM, columna W).
- El patrón de triple redundancia modular (TRM) favorece mucho más (con una importancia muy fuerte) la tolerancia a fallos que el patrón *Sanity Check* (SC) y se le asigna un valor 7 (Tabla 6.4(a), fila TRM, columna SC).
- El patrón *Watchdog* (W) favorece más (con una importancia fuerte) la tolerancia a fallos que el patrón *Sanity Check* (SC) y se le asigna un valor 5 (Tabla 6.4(a), fila W, columna SC).

De manera análoga se lleva a cabo la comparación entre las tres mismas alternativas en relación al atributo de calidad *tiempo de latencia* (la matriz de comparación AHP para la latencia se muestra en la Tabla 6.4 (b)). Los ejemplos de las comparaciones dos a dos se detallan a continuación:

- El patrón *Watchdog* (W) favorece más (con una importancia fuerte) el tiempo de latencia que el patrón triple redundancia modular (TRM) y se le asigna un valor 5 (Tabla 6.4(b) fila W, columna TMR).
- El patrón de *Sanity Check* (SC) favorece más (con una importancia fuerte) la latencia que el patrón triple redundancia modular (TRM) y se le asigna un valor 5 (Tabla 6.4(b) fila W, columna TMR).
- No hay diferencias en cuanto al tiempo de latencia entre los patrones W y SC, por lo que se asigna un valor 1 (Tabla 6.4(b) fila W, columna SC).

Tabla 6.4 Matriz de comparación entre alternativas

(a)	*Tolerancia a Fallos*		
	TRM	***W***	***SC***
TRM	1	5	7
W	1/5	1	5
SC	1/7	1/5	1
Sum	1,34	6,20	13,00

(b)	*Latencia*		
	TRM	***W***	***SC***
TRM	1	1/5	1/5
W	5	1	1
SC	5	1	1
Sum	11,00	2,20	2,20

La reciprocidad de las comparaciones se refleja en que el valor de una celda de la matriz de comparación *C* es la inversa del valor de la celda de la posición traspuesta, tal como refleja la expresión (2). Al ser los elementos de la diagonal 1 por definición (sería el resultado de comparar algo consigo mismo) el número de comparaciones a llevar a cabo para *n* alternativas, se calcula según la expresión (3). La complejidad del proceso es considerable si tenemos en cuenta *m* atributos de calidad, pero esta es otra de las razones para organizar las transformaciones atendiendo no solo a su estructura sino también a los atributos de calidad a los que impactan, como se describe en la sección 6.2.1.2, para que el número de atributos de calidad a considerar en los procesos de *tradeoff* se vea reducido. Así, se podrá llevar a cabo el análisis *tradeoff* con los atributos de calidad Q_a que se ven afectados de manera directa por la transformación, junto con aquellos atributos de calidad Q_b que tienen relaciones de impacto con los atributos de calidad Q_a, información esta última que se refleja en la vista de calidad del multimodelo (ver Sección 4.2.2.3).

$$C[i,j] = \frac{1}{C[j,i]} \tag{2}$$

$$Comparaciones = Fibonacci(n-1) \tag{3}$$

A partir de la matriz de comparaciones *C* la matriz normalizada C_{norm} se calcula aplicando la expresión (4)[18].

El resultado de la normalización para las matrices de comparación de los atributos tolerancia a fallos y latencia se muestra en la Tabla 6.5.

[18] El valor del sumatorio que aparece como denominador en la formula (4) corresponde a la fila Sum de la Tabla 6.4.

Tabla 6.5 Matrices de comparación normalizadas

(a)	*Tolerancia a Fallos*		
	TRM	***W***	***SC***
TRM	$\frac{1}{1.34}$	$\frac{5}{6.20}$	$\frac{7}{13}$
W	$\frac{1/5}{1.34}$	$\frac{1}{6.20}$	$\frac{5}{13}$
SC	$\frac{1/7}{1.34}$	$\frac{1/5}{6.20}$	$\frac{1}{13}$

(b)	*Latencia*		
	TRM	***W***	***SC***
TRM	$\frac{1}{11}$	$\frac{1/5}{2.20}$	$\frac{1/5}{2.20}$
W	$\frac{5}{11}$	$\frac{1}{2.20}$	$\frac{1}{2.20}$
SC	$\frac{5}{11}$	$\frac{1}{2.20}$	$\frac{1}{2.20}$

El *Impacto* que la transformación alternativa *Ai* tiene sobre el atributo de calidad Q_j se calcula aplicando la expresión (5) sobre la matriz normalizada C_{norm} dando como resultado la matriz I de tamaño *nxm* siendo n el número de alternativas y m el número de atributos de calidad. Dicha matriz contiene valores en un rango de 0 a 1. El elemento $I[i,j]$ expresa en qué medida la alternativa *Ai* soporta el atributo de calidad Q_j. La Tabla 6.6 muestra los impactos relativos de cada alternativa con respecto a los atributos de calidad, tolerancia a fallos y latencia.

$$C_{norm}[i,j] = \frac{C[j,i]}{\sum_{k=0}^{n} C[k,j]} \tag{4}$$

$$I[i,j] = \frac{\sum_{k=1}^{n} C_{norm}[i,k]}{n} \tag{5}$$

Tabla 6.6 Impacto de las transformaciones sobre los atributos de calidad

	Tolerancia a Fallos	*Latencia*
TRM	0,696531334	0,090909091
W	0,231613959	0,454545455
SC	0,071854707	0,454545455

Estos valores se almacenan en el correspondiente atributo *Importance* de la relación entre transformaciones alternativas y atributos de calidad *AlternativeTransformationToQualityAttribute* del multimodelo. La Figura 6.11

muestra el editor de relaciones de la infraestructura para la gestión del multimodelo con los resultados del *tradeoff* del ejemplo. Dichos resultados serán empleados en la fase de ingeniería de aplicación para guiar las transformaciones arquitectónicas en el caso en que, tras la evaluación de la arquitectura, esta no cumpla los requisitos no-funcionales.

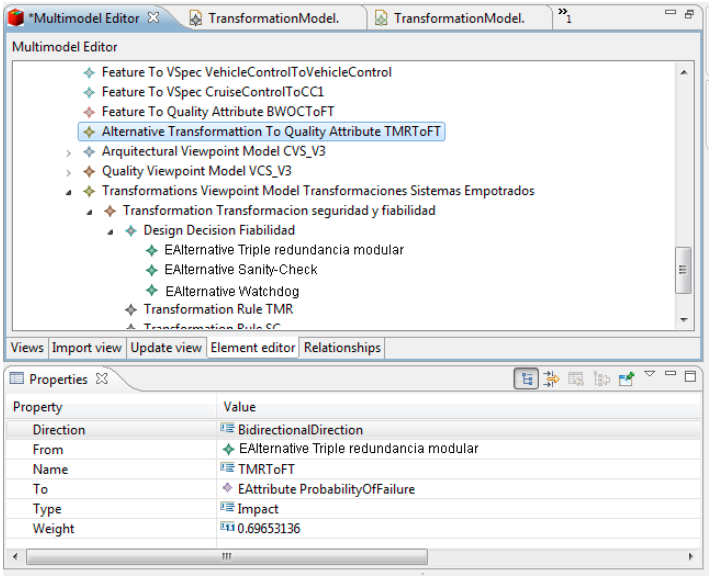


Figura 6.11 Introducción de los resultados del *tradeoff* en el multimodelo

6.2.3 Transformación de la arquitectura

En la fase de ingeniería de aplicación, se ejecuta el proceso (o los procesos) de transformación de modelos asociado a la tarea de *transformación de la arquitectura*. Dicho proceso toma como entradas:

- Las transformaciones resultantes de la aplicación de las directrices de diseño descritas en la sección 6.2.1 (incluyendo la vista de transformaciones construida como resultado de la reorganización de las transformaciones descrita en la sección 6.2.1.2).
- Las relaciones entre alternativas y atributos de calidad resultado de la aplicación del *tradeoff* descrita en la sección 6.2.2.
- Las prioridades de los atributos de calidad Q definidas en el modelo de configuración como resultado de la actividad de obtención de la configuración del producto descrita en la sección 5.1.

- La arquitectura de producto (que proviene directamente de la derivación o de procesos de transformación anteriores.

Las prioridades de los *k* atributos de calidad *Q* se introducen en el proceso de transformación como parámetros externos. Cuando la parte izquierda de una regla de transformación discriminante unifica con un conjunto de elementos del modelo origen, la función *BestRankedAlternative* (Listado 6.6 - línea 18) selecciona la *Ai* con mejor valor *R* aplicando la formula (6).

$$R[i] = \sum_{j=0}^{n} Qj * I[i,j] \qquad (6)$$

En el ejemplo de los sistemas de control de automóviles, se va a aplicar el proceso de transformación a la arquitectura del sistema ABS, cuya arquitectura en AADL se muestra en la Figura 6.12.

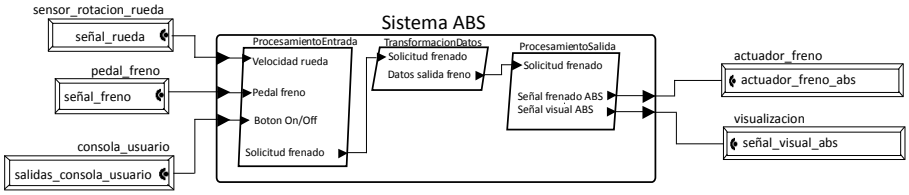


Figura 6.12 Arquitectura del sistema ABS

Si el ingeniero de aplicación seleccionó el valor *Qt*=1 como prioridad para la tolerancia a fallos y *Ql*=0,60 para el tiempo de latencia (la tolerancia a fallos es de máxima prioridad y el tiempo de latencia no tiene ninguna prioridad), el procesos de transformación seleccionará y aplicará el patrón TRM. Los resultados del cálculo tras aplicar la formula (6) empleando los impactos *I* de la Tabla 6.7 se muestran en la (a). La Figura 6.13 muestra el resultado de aplicar la transformación TRM a la arquitectura de la Figura 6.12.

Tabla 6.7 Valores de R para las distintas alternativas

(a)	*R* $Q_t=1\ Q_l=0$
TRM	**0,70**
W	0,23
SC	0,07

(b)	*R* *Qt=0.4 Ql=0.6*
TRM	0,33
W	**0,37**
SC	0,30

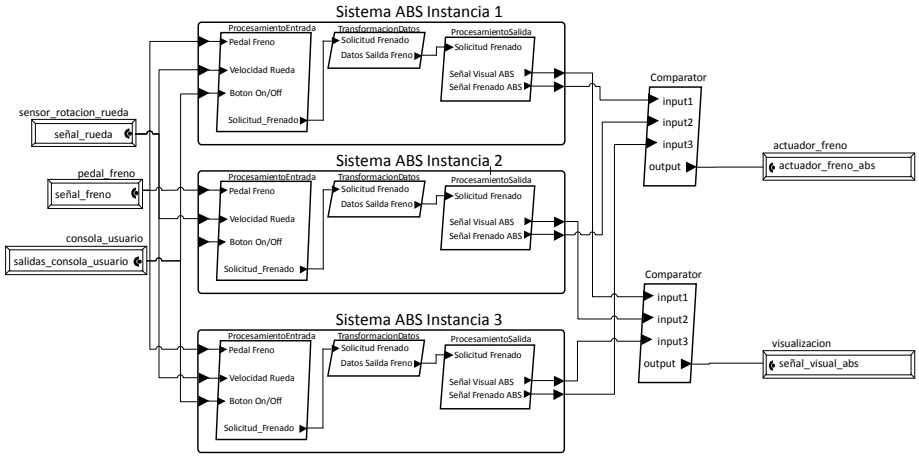


Figura 6.13 Arquitectura resultante tras la aplicación del patrón Triple redundancia modular

Si por el contrario, el ingeniero de aplicación seleccionó el valor Qt=0,4 como prioridad para la tolerancia a fallos y Ql=0,6 para el tiempo de latencia (la latencia es ligeramente más prioritaria que la tolerancia a fallos), el proceso de transformación seleccionará y aplicará el patrón W, tras aplicar la formula (6) empleando los impactos *I* de la Tabla 6.7. Los resultados del cálculo tras aplicar la formula (6) empleando los impactos *I* de la Tabla 6.7se muestran en la (b). La Figura 6.14 muestra el resultado de aplicar la transformación W a la arquitectura de la Figura 6.12.

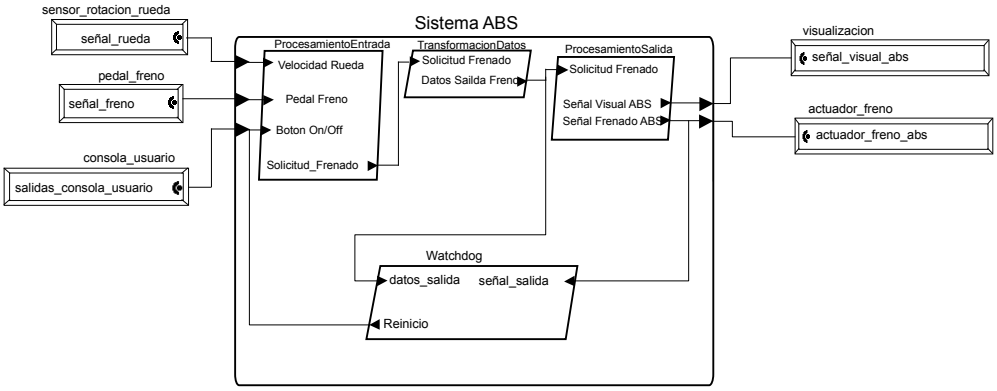


Figura 6.14 Arquitectura resultante tras la aplicación del patrón *Watchdog*

6.3 Conclusiones

En este capítulo hemos presentado en detalle la fase de evaluación y mejora de arquitecturas del proceso QuaDAI. Se han presentado las alternativas para la medición de los atributos no-funcionales y la validación automática de los valores en base a los valores especificados por el ingeniero de aplicación durante la configuración. Así mismo, se ha presentado la automatización de la selección y aplicación de transformaciones arquitectónicas atendiendo a criterios de calidad.

El uso de la aproximación de evaluación basada en métricas, permite la validación de los requisitos no-funcionales, de manera objetiva (para aquellos casos en los que los requisitos no-funcionales puedan ser expresados en base a métricas y umbrales) y en fases tempranas, justo tras la derivación de la arquitectura de producto. Además, las relaciones definidas en el multimodelo proveen de un mecanismo de trazabilidad que nos permitirá dar con el origen del no cumplimiento, su resolución, si hay alternativas para la configuración seleccionada, o bien su renegociación con los *stakeholders* pudiendo valorar la posibilidad de relajar algún requisito no-funcional si éste no es alcanzable.

Las directrices de diseño sistematizan la definición de transformaciones arquitectónicas, basadas en patrones, que emplean los atributos de calidad como factor de decisión cuando existen distintas alternativas de transformación. El proceso de selección pretende documentar y sistematizar la toma de decisiones del arquitecto/experto del dominio, que en ocasiones va a tener más conocimiento del problema del que somos capaces de plasmar en los modelos, no obstante, es un primer paso, no excluyente para adoptar una solución al problema. Asimismo, la aproximación no pretende abordar la optimización de atributos no-funcionales, como en las aproximaciones de R. Li et al. (2011); Etemaadi et al. (2013), sino reutilizar información "*a priori*" para adoptar una solución de entre una serie de alternativas, en una decisión de diseño. Este reúso proviene del hecho de plasmar la información de impactos entre transformaciones alternativas y atributos de calidad en el multimodelo, que permitirá que esta información sea empleada en procesos de transformación por arquitectos con menos experiencia.

La representación de las relaciones entre alternativas de calidad y transformaciones alternativas permite además, que esas relaciones sean flexibles y que puedan aprovechar la información del proceso de evaluación posterior a la transformación para realimentar el multimodelo, permitiendo la corrección de posibles desviaciones si los procesos de medición constatan que

alguna de estas transformaciones no producen los efectos deseados en los atributos de calidad.

Por otro lado, somos conscientes de que la aproximación empleada para llevar a cabo el *tradeoff* entre alternativas y atributos de calidad y la posterior selección entre alternativas carece, probablemente, de la potencia expresiva que se requeriría en algunos casos. Este es uno de los puntos a mejorar, con la adopción de modelos más expresivos como por ejemplo, las cadenas de Markov, las lógicas difusas o técnicas de programación lineal como el algoritmo Simplex (Dantzig 1951) o técnicas de optimización no lineal como el método de Nelder y Mead (1965), que nos permitan mayor precisión a la hora de definir el objetivo y seleccionar la mejor alternativa en base a los criterios de calidad. No obstante, el acoplamiento del proceso definido con el apartado de decisión de la alternativa es relativamente bajo (la selección se realiza en función de una *query* en las reglas discriminantes), lo cual hace que en el futuro se puedan explorar nuevas alternativas con relativamente poco esfuerzo.

Por todo ello, consideramos que la aproximación propuesta supondría una mejora en los procesos de desarrollo de líneas de producto siguiendo el enfoque de desarrollo dirigido por modelos en los que existe una alta variabilidad en los requisitos de calidad y en los que ir más allá del alcance de la línea de productos para los requisitos no-funcionales sea frecuente.

El capítulo 9 describe la validación empírica del método mediante una familia de experimentos controlados realizados con estudiantes de grado en informática, ingeniería informática y master en Italia, Paraguay y España en los que se analiza la efectividad, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso futuro de las actividades y tareas que integran la fase de evaluación y mejora del método QuaDAI.

Capítulo 7

Herramienta de Soporte al Método QuaDAI

Este capítulo presenta el prototipo completamente funcional de la herramienta desarrollada en el contexto de esta tesis doctoral, que da soporte al método y que ha permitido, por un lado analizar la aplicabilidad de las distintas actividades y procesos que integran el método propuesto y por otro lado, validar empíricamente la fase de derivación mediante el estudio del caso descrito en la sección 8.

La herramienta, desarrollada como un conjunto de plug-ins de *Eclipse Modelling Framework* (Eclipse 2013a), da soporte a las actividades de creación y edición de multimodelos, a la configuración del producto, a la validación de la consistencia de la configuración, a la derivación del modelo CVL a partir de una configuración valida y a la comprobación del grado de cumplimiento de los diferentes requisitos no-funcionales tras la medición de la arquitectura obtenida.

En las siguientes secciones se van a presentar en detalle cada una de las distintas funcionalidades que integran la herramienta:

En la sección 7.1 se describe la infraestructura desarrollada para la edición de multimodelos a partir de modelos existentes.

En la sección 7.2 se describe el conjunto de funcionalidades desarrollado para dar soporte a la derivación de la arquitectura de producto, como son: el soporte a la configuración, la validación de la consistencia y la derivación del modelo de resolución CVL.

En la sección 7.3 se describe el mecanismo de soporte a la evaluación de la arquitectura derivada.

7.1 Infraestructura para la edición de multimodelos

Con el fin de soportar la definición de multimodelos compuestos de dos o más vistas y el establecimiento de relaciones entre elementos de dichas vistas, se ha definido una infraestructura para la edición y gestión de multimodelos que implementa la estructura genérica de multimodelos (que se muestra en la Figura 4.2 de la sección 4.2.1), y la arquitectura descrita en la sección 4.2.4. En esta sub-sección se describe cómo se llevó a cabo la implementación de dicha infraestructura.

La implementación ha sido llevada a cabo utilizando el marco tecnológico Eclipse. Eclipse define Ecore, una implementación del estándar Essential MOF (EMOF) (Object Management Group 2006). Las herramientas de EMF para Eclipse permiten la definición de metamodelos que pueden instanciarse en modelos de instancia. Esta solución emplea los niveles M1 y M2 previstos en la arquitectura MOF, es válida para la mayoría de los modelos. En el caso en que sea preciso realizar cambios en el metamodelo, el desarrollador tiene que realizar dichos cambios, crear un generador nuevo y volver a crear el código con este generador. Algunos cambios pueden dejar inservibles las instancias previamente generadas, de modo que es responsabilidad del desarrollador realizar modificaciones que no sean incompatibles con instancias creadas anteriormente al cambio.

Sin embargo, cuando pretendemos editar un multimodelo, esta solución resulta insuficiente. En un multimodelo es un requisito necesario proveer la flexibilidad necesaria para que sea posible añadir o eliminar vistas y relaciones en el multimodelo sin perder instancias previamente generadas en herramientas de edición de terceros y sin alterar la estructura de los metamodelos que soportan cada una de las vistas (que pueden a su vez estar dotadas de herramientas de edición de terceros).

Así pues para conseguir una gestión eficaz de multimodelos es preciso satisfacer los siguientes requerimientos:

- Permitir añadir o eliminar vistas.
- Permitir añadir, eliminar o modificar tipos de relaciones entre los diferentes elementos del multimodelo.

- Permitir crear, eliminar y modificar relaciones entre elementos del multimodelo.
- Permitir importar vistas a partir de modelos existentes en el multimodelo.
- Permitir actualizar vistas del multimodelo a partir de modelos existentes.

En todas estas operaciones se debe asegurar la consistencia de todas las instancias del multimodelo creadas con anterioridad. Se valoraron tres alternativas de diseño: i) definir un metamodelo para el multimodelo que se modifica a medida que las necesidades del cliente cambian, ii) elevar el nivel de abstracción utilizando un metamodelo cuyas instancias serían metamodelos de multimodelos y iii) utilizar una jerarquía de dos niveles (multimodelo *Core* y multimodelo concreto).

La primera aproximación consiste en definir un metamodelo para el problema en el cual añadiríamos o eliminaríamos vistas dependiendo de las necesidades del problema. Cada vez que se realizara un cambio habría que volver a generar el código de los editores que trabajan con los modelos instancias de este metamodelo. Esta solución cuenta con la desventaja de que se invalidarían frecuentemente las instancias generadas con versiones anteriores del metamodelo.

La segunda aproximación consiste en elevar el nivel de abstracción. Dado que Eclipse sólo permite generar metamodelos e instancias (dos niveles), una solución factible es crear un nuevo metamodelo *Base*. Este metamodelo base contendría los elementos necesarios para cargar vistas, crear elementos y definir relaciones. Un modelo instancia basado en este metamodelo básico contendría toda la información necesaria para crear un metamodelo para un multimodelo. Aplicando transformaciones de modelos podríamos, a partir de un modelo instancia del metamodelo básico, obtener el meta-modelo para un multimodelo específico. Esta solución nos permitiría no tener que estar trabajando directamente sobre el metamodelo. Sin embargo, la solución no resolvería el problema de que cada vez que se haga un cambio exista el riesgo de invalidar las instancias existentes y la posible des-sincronización si una de las vistas es modificada externamente con un editor de terceros.

Finalmente la solución adoptada se basa en una arquitectura de dos niveles. Esta idea se basa en el soporte para trazabilidad de requerimientos propuesto para la herramienta de gestión de modelos MOMENT (Gómez et al. 2006). En

esa solución, se pretendía dar soporte a metamodelos de trazabilidad genérico que pudieran ser extendidos por el usuario mediante la creación de un metamodelo de trazabilidad personalizado. Este diseño permitió dar soporte a la trazabilidad de forma genérica sin depender del metamodelo personalizado por cada usuario, pero a su vez sin tener que renunciar a la flexibilidad que esto proporciona. Con el objetivo de dar soporte a la edición de multimodelos aplicando este diseño es necesario:

1. La definición del metamodelo genérico *Core* que contenga la información necesaria para representar un multimodelo: las vistas que contiene, los elementos del multimodelo y las relaciones entre dichos elementos del multimodelo.
2. Un editor que trabaje directamente sobre este metamodelo genérico. De esta forma, aunque el usuario define un metamodelo para un problema concreto, éste metamodelo siempre será una extensión del metamodelo genérico o *Core,* de modo que el editor podrá reconocer los elementos de los modelos instancia del metamodelo personalizado.

El editor desarrollado se implementa siguiendo un diseño basado en múltiples pestañas. Cada una de las pestañas contiene un editor dedicado a una funcionalidad concreta: editar vistas, importar modelos, actualizar modelos, editar elementos y editar relaciones, tal como se muestra en la Figura 7.1. El editor nos permite crear una vista concreta desde cero. En este caso se añade una nueva vista vacía al multimodelo y el usuario se encarga manualmente de añadir las entidades y vistas.

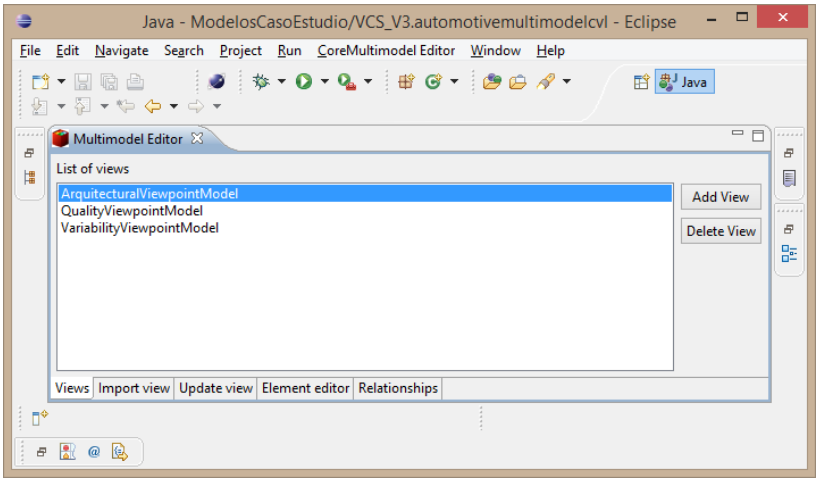


Figura 7.1 Editor de un multimodelo personalizado como extensión del editor core

Sin embargo en el caso de que ya dispusiéramos de un modelo creado, también es capaz de cargar una vista a partir de ficheros XMI. Mediante el código EMF dinámico se carga un modelo en memoria y se añade al multimodelo como una vista más. Si existen en dicho modelo entidades que se deben extender en el metamodelo, se lleva a cabo una conversión de tipo (*casting*) de dicha entidad. De esta forma, la entidad original es reemplazada por una entidad extendida dentro de la vista. Esta representación se basa en el polimorfismo de inclusión (también llamado de redefinición o sub-tipado), es decir las entidades del multimodelo heredan todos los atributos y relaciones del tipo base (perteneciente a la entidad original). El Listado 7.1 muestra el pseudocódigo del proceso de importación de vistas.

Listado 7.1 Pseudocódigo del proceso de importación de vistas

```
1  VistaAImportar = CargaVista();
2  if(not Multimodel.contiene(VistaAImportar){
3            VistaAdaptada = ConvierteVista(VistaAImportar);
4            Multimodelo.vistas.Añade(VistaAdaptada);
5  }
```

El método *ConvierteVista* recorre todas las entidades de una vista dada en dos pasadas. En la primera, crea una copia de todas las entidades en una vista nueva y en la segunda pasada convierte las entidades que se deben extender en entidades extendidas. El Listado 7.2 muestra el pseudocódigo del método *ConvierteVista*.

Listado 7.2 Pseudocódigo del método *ConvierteVista*

```
1  ConvierteVista(VistaOriginal){
2            VistaConvertida = CrearVistaVacía(TipoVista);
3            VistaConvertida.atributos = VistaOriginal.atributos;
4            VistaConvertida.características = VistaOriginal.características;
5            for(VistaOriginal.entidades)
6            {
7                     EntidadExtendida = ConvertidadEndidad(Entidad);
8                     if(DebeExtenderse(Entidad)
9                     {
10                      VistaConvertida.Añadir(EntidadExtendida);
11                    }
12
13           }
14          Devuelve VistaConvertida;
15 }
```

La infraestructura incorpora un editor en árbol que permite la edición de las entidades del multimodelo, soportando tanto la creación, como la modificación y el borrado de entidades del multimodelo instancias de las metaclases proxy correspondientes (ver sección 4.2.4). La Figura 7.2 muestra la pestaña de edición de entidades del multimodelo.

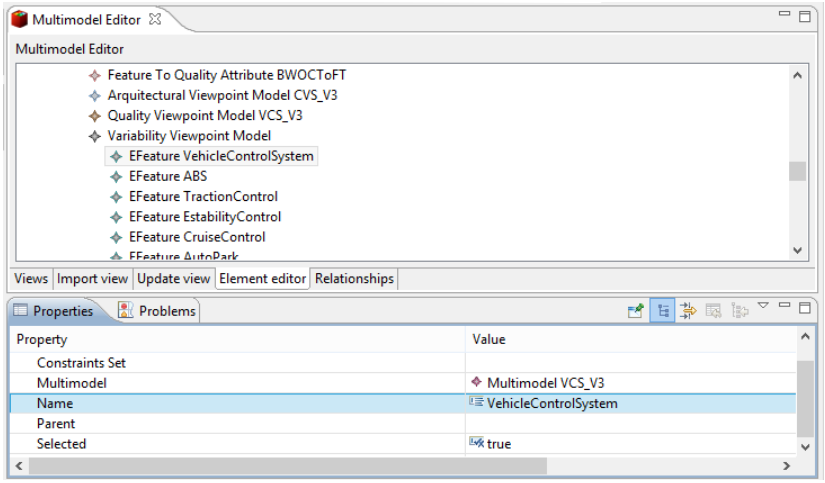


Figura 7.2 Editor de entidades del multimodelo

El editor de multimodelos también incluye una pestaña que permite editar las relaciones entre elementos de las distintas vistas. En la parte inferior se pueden editar los atributos heredados del multimodelo Core y los definidos para este tipo de relación en el metamodelo personalizado, tal como se muestra en la Figura 7.3.

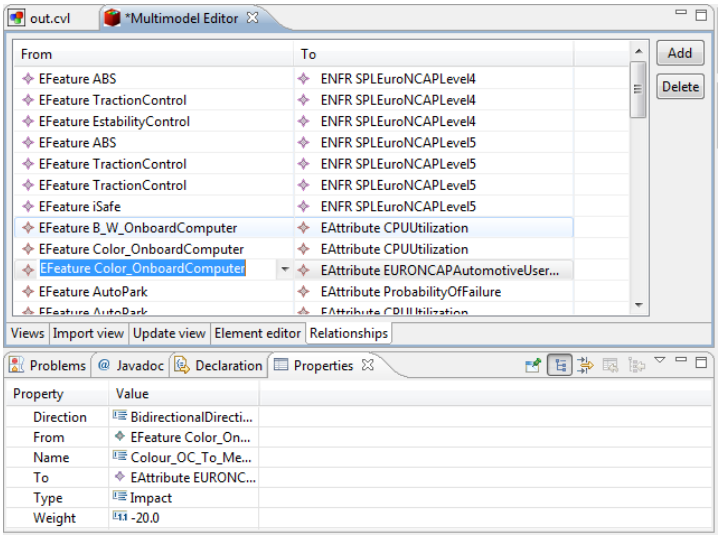


Figura 7.3 Editor de relaciones

Por último, el editor también permite actualizar una vista ya existente del multimodelo. Para ello se ha integrado una herramienta llamada EMF-Compare (Brun y Pierantonio 2008), que provee una utilidad para comparar modelos. Cuando se ha de actualizar una vista, primero se cargará dicha vista mediante el asistente y a continuación permite cargar un modelo mostrando las diferencias entre el modelo cargado y la vista correspondiente del metamodelo. El editor de diferencias permite seleccionar qué cambios se desean integrar en el multimodelo, tal como se puede ver en la Figura 7.4. El panel izquierdo nos muestra las diferencias, el panel central la vista contenida en el multimodelo y el panel derecho nos muestra la vista importada a partir del fichero. En la parte inferior se nos muestra una ventana con los cambios detectado. Esto permite editar con herramientas externas modelos que han sido importados al multimodelo, aun cuando sus entidades formen parte de relaciones en dicho multimodelo.

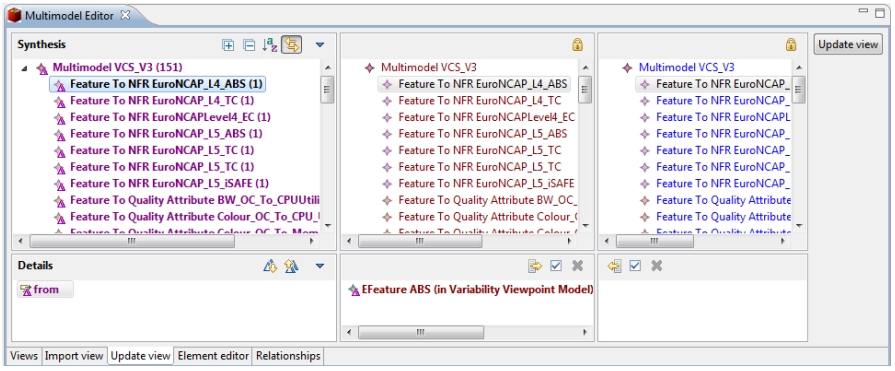


Figura 7.4 Actualización de vista existente en el multimodelo

7.2 Soporte a la derivación de arquitecturas de producto

Para dar soporte a las actividades de derivación de la arquitectura, se han implementado una serie de funcionalidades para la configuración del producto en desarrollo, la validación de la consistencia de dicha configuración y la obtención del modelo de resolución CVL a partir de dicha configuración. La validación de consistencia de la configuración a nivel de la vista de calidad, así como la consistencia inter-vistas, se han implementado mediante las fórmulas OCL descritas en la sección 5.1.2.1 y 5.1.2.2, mientras que la validación de consistencia a nivel de la vista de variabilidad se ha implementado, junto con la derivación del modelo de resolución, en un *plug-in* Eclipse que integra ambas acciones de interfaz.

7.2.1 Configuración del producto en desarrollo

Para editar la configuración de un determinado producto, el editor de la infraestructura permite la selección de características de la vista de variabilidad requisitos no-funcionales de la vista de calidad y la priorización de los atributos de calidad, permitiendo introducir los pesos que expresan dicha prioridad en el campo *Importance* de las instancias de *EAttribute* (ver sección 4.2.4). La Figura 7.5 muestra la interfaz de cuantificación de la prioridad de los atributos de calidad en el campo *Importance*.

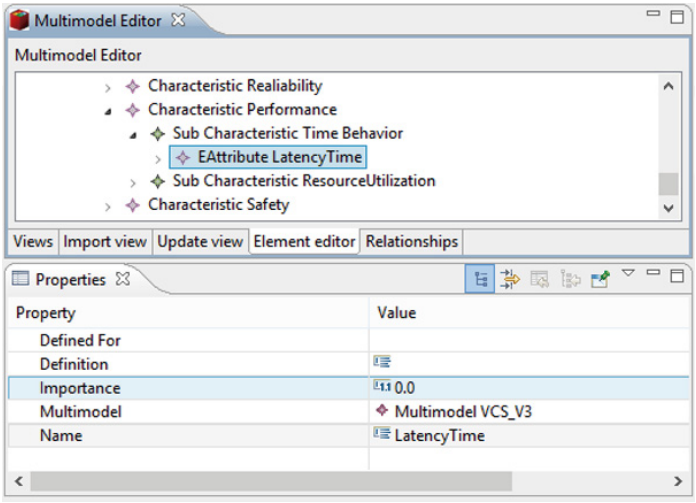


Figura 7.5 Priorización de atributos de calidad en la herramienta

En el caso de la configuración, se plantearon dos alternativas de diseño. La primera alternativa consistía en definir un nuevo metamodelo y su correspondiente editor, extendiendo el multimodelo descrito en la sección 4.2, lo que permitiría la visualización únicamente de las vistas de calidad y variabilidad; y añadiendo los campos necesarios para la configuración (los campos *Selected* e *Importance*). La segunda alternativa de diseño consistía en añadir, en las metaclases extendidas del propio multimodelo descrito en la sección 4.2, los atributos necesarios para acometer la configuración del producto. La primera opción, si bien más modular, incrementa la complejidad de la solución, dado que requiere el desarrollo de editores propios para dar soporte a las actividades de configuración y requerirá que las actividades que tienen lugar tras la configuración, tengan que tomar como entradas tanto el multimodelo genérico como la extensión para llevar a cabo la generación del

modelo de resolución CVL, para la materialización de la arquitectura, su posterior evaluación y en caso necesario, mejora de la misma. En la segunda opción (la que finalmente hemos adoptado en el desarrollo), se añaden los campos necesarios para las tareas de configuración, tal como se describe en la sección 4.2, y se emplea para la actividad de configuración el propio multimodelo que describe la línea de productos. Así, a nivel de implementación, en lugar de emplear una proyección del multimodelo se ha optado por emplear el propio multimodelo (aunque en un futuro se puede optar por restringir el modelo de configuración únicamente a las vistas y metaclases de interés).

7.2.2 Validación de la consistencia

Para soportar la validación de la consistencia, se implementaron las formulas OCL descritas en la sección 5.1.2. Dicha implementación ofrecía dos alternativas de diseño. La primera de ellas consiste en llevar a cabo llamadas a los validadores en tiempo de ejecución de *OCLTools* (Eclipse 2013b), pasando por parámetro el código de las formulas OCL con las que validar las restricciones definidas en la sección 5.1.2. La segunda alternativa consiste en definir las restricciones como invariantes de las propias relaciones entre vistas (véase Listado 5.2 y Listado 5.3). Esta última opción, que fue a la postre la opción implementada en la herramienta, aporta flexibilidad y mantenibilidad, puesto que nos permite desacoplar la definición de las restricciones, de manera que puedan ser modificadas directamente sobre el metamodelo, sin necesidad de modificar el código fuente de los editores.

Para la validación de la consistencia intra-vista de la vista de variabilidad, se ha implementado una acción adicional en el menú para invocar a la transformación encargada de generar el archivo de configuración e invocar al validador de FaMa. La Figura 7.6 muestra la invocación de dicha validación desde el menú contextual de la herramienta.

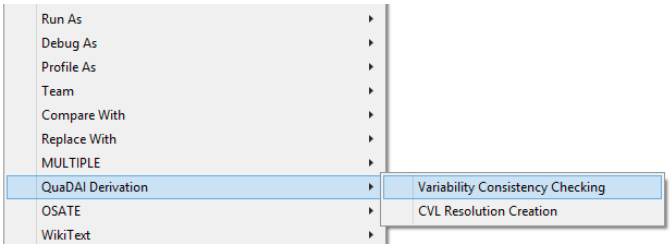


Figura 7.6 Invocación de la validación de consistencia de la vista de variabilidad

Al validador FaMa se le envía la consulta *ValidProduct* (product valido), consumiendo el servicio OSGI (OSGI Alliance 2012) ofrecido por FaMa. Para ello empleamos el puente FaMa definido en la herramienta Multiple (Gómez 2012).

El Listado 7.3 muestra la invocación de dicha validación mediante la creación del objeto FaMa a través del puente que hace de intermediario con el framework FaMa (líneas 4-15), la designación del modelo de características de FaMa, creado por transformación a partir de la vista de variabilidad del multimodelo (líneas 17-19), la creación del objeto *Product* que contiene la configuración de características seleccionadas en el multimodelo (líneas 27-40) y finalmente la invocación de la consulta (líneas 41-44).

Listado 7.3 Validación de la consistencia de las características seleccionadas

```
1   public void FeaturesConfigurationValidation(IAction action) {
2
3           ...
4           //FaMa Osgi invocation //
5           FamaDplfwBridePlugin fama = new FamaDplfwBridePlugin();
6           try
7           {
8                   fama.start(InternalPlatform.getDefault().getBundleContext());
9           }
10          catch (Exception e)
11          {
12                  showError("Feature Model Error","FAMA Plug-in Can't be loaded.");
13          }
14          //Creates FaMa question trader//
15          QuestionTrader qt = fama.getQuestionTrader();
16          //Loads the feature model into FaMA
17          File f = new File(filename);
18          VariabilityModel fm = qt.openFile(f.getAbsolutePath());
19          qt.setVariabilityModel(fm);
20          // Configuration load //
21          String features = readFile(featuresFileName);
22          // FaMa Valid Product question invocation //
23          String question = "ValidProduct";
24          ValidProductQuestion vq;
25          vq = (ValidProductQuestion) qt.createQuestion(question);
26          // Creates a FaMa Product with the configuration //
27          Product p = new Product();
28          // Creates a FaMa Product with the configuration //
29          FeatureList list = new FeatureList(fm.getElements());
30          StringTokenizer tokenizer = new StringTokenizer(features);
31          // Creates a FaMa Product with the configuration //
32          while(tokenizer.hasMoreTokens())
33                  {
34                          String aux = tokenizer.nextToken();
35                          VariabilityElement ve = list.getByName(aux);
36                          if(ve!=null)
37                          {
38                                  p.addFeature((GenericFeature) ve);
39                          }
40                  }
41          // Sets the FaMa to p//
42          vq.setProduct(p);
43          // Launches the question//
44          qt.ask(vq);
45          showInfo("Is the product valid?"," Result: "+ vq.isValid());
46
47          ...
48  }
```

7.2.3 Derivación del modelo de resolución CVL

El modelo de resolución CVL se genera por transformación a partir de una configuración valida plasmada en el modelo de configuración. Para ello se han implementado las reglas de transformación descritas en la sección 5.2.2.1. El proceso se invoca desde una opción del menú contextual de la herramienta, tal como se muestra en la Figura 7.7.

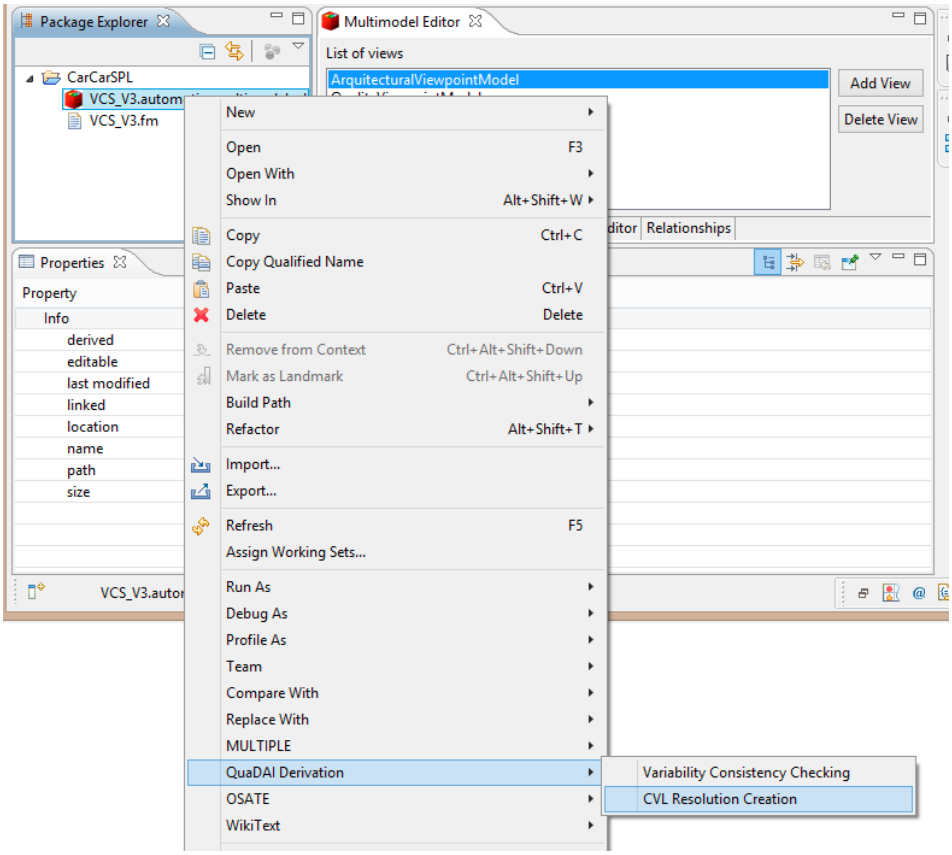


Figura 7.7 Invocación de la funcionalidad de generación del modelo e resolución CVL

El proceso de transformación genera el archivo CVL que contiene la especificación de variabilidad arquitectónica contenida en la vista arquitectónica del multimodelo y además añade el árbol de resolución asociado a la configuración representada en el modelo de configuración. Dicho árbol de resolución estará compuesto por un conjunto de *ResolutionElements* que representan la resolución positiva de los VSpecs de tipo selección, la asignación de valores a las VSpecs de tipo variable y las VSpecs asociadas a las VSpecs

compuestas. La Figura 7.8 muestra un ejemplo de modelo de resolución CVL generado para el caso en el que se seleccione únicamente el sistema ABS y el sistema multimedia FM-CD en el modelo de características del sistema de control de automóviles descrito en el Apéndice A.1. En este caso concreto, el modelo de resolución contiene las resoluciones positivas de los puntos de variabilidad arquitectónica.

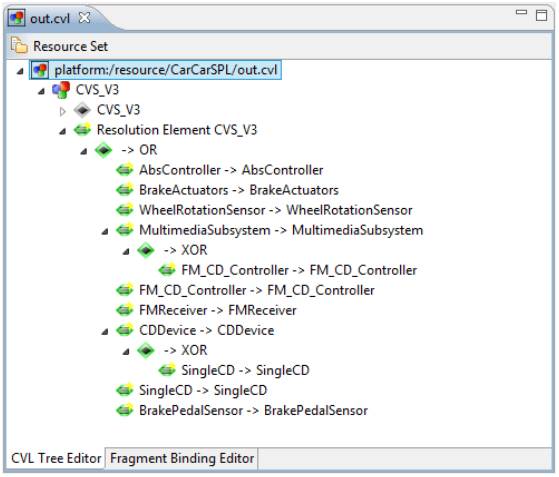


Figura 7.8 Modelo de resolución CVL generado

A partir del modelo de resolución CVL, la herramienta que da soporte a CVL (Fleurey et al. 2009), permite la obtención del modelo de la arquitectura de producto resultante de la derivación.

7.3 Soporte a la evaluación

Para dar soporte a la tarea de evaluación de la arquitectura de producto obtenida, el editor permite almacenar los requisitos no-funcionales tanto de la línea de productos como los requisitos no-funcionales específicos del producto en desarrollo. Además, se ha previsto un mecanismo que tras el proceso de medición, permite comprobar el grado de cumplimiento de dichos requisitos no-funcionales. Para ello se ha introducido la validación de los requisitos no-funcionales descrita en la sección 6.1.3.

La infraestructura realiza llamadas al validador de OCLTools pasándole como parámetros la OCLs definidas en el campo *OCLRestriction* de la meta-clase *RNF*, para aquellos requisitos no-funcionales en los que se haya definido una OCL de usuario, o bien, ejecuta una OCL genérica, que accede a la

información del umbral del RNF y valida el valor obtenido en relación a ese umbral, obteniendo la validación del cumplimiento del mismo. El Listado 7.4 muestra las OCL empleadas por la infraestructura para la validación de los RNFs.

Listado 7.4 OCL para la validación de RNFs tras la medición

```
if ((self.DecisionCriteria.Thresholds.oclIsTypeOf(qualityNFR::SoftGoal)) and (self.Selected))
then
   if ((self.DecisionCriteria.Thresholds.oclAsType(qualityNFR::SoftGoal).Orientation=
      qualityNFR::EOrientation::Negative))
   then
         self.Meassurement.ActualValue<=
         self.DecisionCriteria.Thresholds.oclAsType(qualityNFR::SoftGoal).ThresholdValue
   else
         self.Meassurement.ActualValue>=
         self.DecisionCriteria.Thresholds.oclAsType(qualityNFR::SoftGoal).ThresholdValue
   endif
else
     (self.Meassurement.ActualValue>=
      self.DecisionCriteria.Thresholds.oclAsType(qualityNFR::HardConstraint).LowerBound))
   and
     (self.Meassurement.ActualValue<=
      self.DecisionCriteria.Thresholds.oclAsType(qualityNFR::HardConstraint).UpperBound)
endif
```

Además, se ha implementado la validación de los requisitos no-funcionales de tipo *HardConstraint* como invariantes de la clase NFR. De este modo, es posible comprobar si los valores medidos se hallan entre los dos valores establecidos en el umbral validando el modelo. El Listado 7.5 muestra la formula OCL del invariante encargado de validar el cumplimiento de los RNFs definidos como *HardConstraints*, lo cual se lleva a cabo invocando a la validación del propio multimodelo, tal como se muestra en la Figura 7.9.

Listado 7.5 Invariante de para la validación de RNFs de tipo *HardConstraint*

```
class ENFR extends coreMultimodel_0::MultimodellRelatableEntity, qualityNFR_0::NFR
{
attribute Selected : Boolean[?];

invariant NFR_Not_Accomplised:
  if (self.Selected) then
     if (self.DecisionCriteria.Thresholds.oclIsTypeOf(qualityNFR::HardConstraint)) then
        (self.Meassurement.ActualValue>=
           self.DecisionCriteria.Thresholds.oclAsType(qualityNFR::HardConstraint).LowerBound)
        and
        (self.Meassurement.ActualValue<=
           self.DecisionCriteria.Thresholds.oclAsType(qualityNFR::HardConstraint).UpperBound)
     else
        true
     endif;
  else
    true
  endif;
}
```

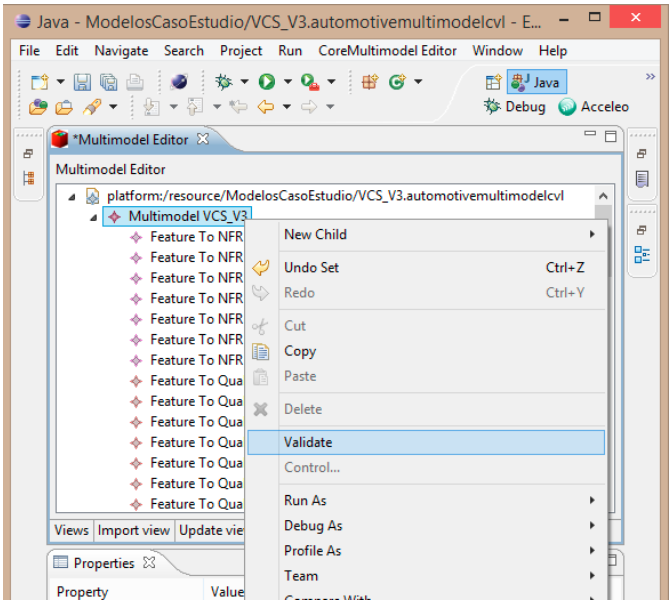


Figura 7.9 Invocación a la validación del modelo desde el menú contextual

7.4 Conclusiones

En este capítulo se ha presentado la herramienta de soporte al método QuaDAI y que cubre las actividades de configuración, derivación y evaluación.

La herramienta, desarrollada como un conjunto de plug-ins para ser integrados en Eclipse EMF, incluye un editor de multimodelos, una serie de restricciones OCL para la validación de la consistencia y la evaluación del cumplimiento de los requisitos no-funcionales tras el proceso de medición de la arquitectura, así como la transformación que permite la obtención del modelo de resolución.

El desarrollo integra diferentes herramientas y mecanismos del desarrollo dirigido por modelos para conseguir resolver un problema altamente complejo, delegando ciertas partes a herramientas de terceros como el framework FaMa (ISA Research Group 2011) para la validación de consistencia en modelos de características o la herramienta de soporte a CVL desarrollada en el Sintef (Haugen et al. 2010) en el caso de la derivación de la arquitectura de producto a partir del modelo de resolución CVL. Se hace uso de distintos estándares de la OMG como OCL o QVT en los procesos de validación y transformación, lo que alinea la herramienta con los estándares MDA.

El desarrollo de la herramienta, que se encuentra en estado de prototipo, nació con la intención de permitir la creación y edición de multimodelos y de soportar los distintos procesos sin una intención de comercializar la misma. Actualmente, y tras la validación empírica de la fase de derivación, surgieron algunas mejoras que afectan sobre todo a la usabilidad de la herramienta, así como a partes que han quedado pendientes de implementar, como el conjunto completo de reglas de transformación para la obtención de modelos de resolución CVL. Una vez desarrolladas todas estas mejoras, la herramienta podría convertirse en un producto comercial. Dada la naturaleza de la misma y del uso que se hace de herramientas de terceros que se encuentran bajo Licencia Publica Eclipse (EPL)[19], nos hace concebir su posible difusión bajo licencia EPL y que las actividades comerciales que puedan derivarse de la misma se limiten a las actividades de consultoría de implantación y posibles desarrollos ad-hoc para clientes individuales.

[19] EPL: Licencia de software de código abierto utilizada por la Fundación Eclipse, que sustituye la Licencia Publica Común y elimina ciertas condiciones relativas a los litigios sobre patentes (Eclipse 2013c).

Capítulo 8

Estudio de Caso para la Validación Empírica del Proceso de Derivación de Arquitecturas

Este capítulo presenta un estudio de caso con el que analizar la aplicabilidad del método en su fase de derivación. Para ello, se ha llevado a cabo un estudio de caso exploratorio que nos permitirá obtener retroalimentación de la opinión de profesionales e investigadores del área de la ingeniería del software y de la ingeniería de las líneas de producto software, así como analizar la facilidad de uso, utilidad percibida e intención de uso.

Se ha seleccionado un estudio de caso para la validación de esta fase por varias razones: por un lado, al no existir métodos integrados que soporten tanto la configuración como derivación de arquitecturas de producto atendiendo a criterios de calidad, se hace difícil ejecutar experimentos en los que comparar la eficacia y eficiencia de la fase de derivación del método de QuaDAI con otros métodos análogos. Por otro lado, se trata de una primera validación en la que se pretende obtener realimentación de los potenciales usuarios del método con el fin de mejorar el mismo, al tiempo que nos permite poner en práctica toda la definición del método, lo que en sí mismo ya es un mecanismo de mejora. Por último, los estudios de caso son una herramienta empírica que provee de cierto grado de libertad para definir un estudio adaptándose a las necesidades de los aspectos a validar, lo cual tratándose de una validación en fase temprana es un ventaja de relevancia.

En las siguientes secciones se va a presentar en detalle el estudio de caso:

En la sección 8.1 se describe el diseño y planificación del estudio de caso, los objetivos, hipótesis y variables a analizar.

En la sección 8.2 se describe la preparación y ejecución del estudio de caso, detallando las tareas que lo integran y los materiales definidos para soportarlas.

En la sección 8.3 se describe el protocolo de recogida de datos para cada una de las tareas.

En la sección 8.4 se analizan los resultados obtenidos tras la ejecución del estudio de caso.

En la sección 8.5 se lleva a cabo un análisis pormenorizado de las amenazas a la validez.

En la sección 8.6 se describen las lecciones aprendidas a partir de las experiencias recabadas tras la ejecución del estudio de caso.

Por último, en la Sección 8.7 se presentan las conclusiones y observaciones finales.

8.1 Diseño del estudio de caso

Un estudio de caso es un tipo de estudio empírico más flexible que los experimentos controlados, pero ello no implica que no sea necesaria una planificación (Runeson y Höst 2008; Wohlin et al. 2012; Runeson et al. 2012). Este estudio de caso ha sido diseñado teniendo en cuenta la metodología de diseño de estudios de caso en el ámbito de la ingeniería del software definida por Runeson et al. (2012).

8.1.1 Planificación del estudio de caso

La planificación de un estudio de caso debe contener, al menos, los siguientes elementos:

- **Objetivo:** el objetivo del estudio de caso, de acuerdo con el paradigma *Goal-Question-Metric* (Basili et al. 1994), es **analizar** la fase de derivación de QuaDAI **con el propósito** de evaluarla **con respecto a** su facilidad de uso, utilidad e intención de uso para la configuración e instanciación de arquitecturas software a partir de la arquitectura de la línea de productos y de un conjunto de requisitos, tanto funcionales como no-funcionales, **desde el punto de vista** de profesionales e investigadores

de la UPV en España y de la UFBA en Brasil en el ámbito del desarrollo de líneas de producto software.

- **Marco conceptual:** el marco conceptual que enlaza con el fenómeno a estudiar, es la derivación de arquitecturas de producto a partir de la arquitectura de la línea de productos y de los requisitos.
- **El caso:** el caso estudiado es la configuración e instanciación de la arquitectura para un producto de la línea de productos *Sistemas de Control de Vehículos CarCarSPL*. Los participantes partían de una serie de requisitos funcionales y no-funcionales que la aplicación ha de cumplir y debían seleccionar qué características, qué requisitos no-funcionales debían estar presentes en el producto, así como la prioridad para los atributos de calidad, para con ellos derivar la arquitectura.
- **Preguntas de investigación:** las preguntas de investigación que se pretende abordar son:
    - **R1:** *¿Cómo perciben los miembros de un equipo de desarrollo de líneas de producto la fase de derivación de QuaDAI en la práctica?*, *¿es percibida como fácil de usar, útil y ven factible utilizarla en el futuro?*, *¿estarían los miembros del equipo de desarrollo de líneas de producto dispuestos a usar, en caso necesario QuaDAI en el futuro?*
    - **R2:** *¿Qué limitaciones tiene el método desde el punto de vista de los participantes?*
- **Estrategia de muestreo:** la estrategia de muestreo del estudio de caso está basado en la ejecución de un único estudio de caso en dos contextos diferenciados (Yin 2009). Los estudios de caso pueden ser holísticos si el caso se estudia como un todo, por ejemplo la aplicación de QuaDAI en una compañía o entorno concreto, o incrustados, si el contexto son los distintos entornos y el objeto es el estudio es la aplicación de QuaDAI. En este caso, dado que lo que evaluamos es el método en dos contextos concretos consideramos que el estudio de caso se considera de *tipo embebido*.

Para contestar a las preguntas de investigación, la fase de derivación de QuaDAI fue aplicada al estudio de caso Sistemas de Control de Vehículos CarCarSPL. Hemos seleccionado este ejemplo porque es una línea de productos con una complejidad media. La vista de variabilidad de dicha línea

de productos consta de 15 características, dos de ellas obligatorias, con 14 relaciones entre características y 240 posibles configuraciones válidas[20], tal como se muestra en la Tabla 8.1. La vista de calidad contiene 6 requisitos no-funcionales para la línea de productos y 6 requisitos de calidad (con relaciones de impacto negativo definidas entre ellos). La vista arquitectónica está compuesta por 28 puntos de variación, siendo 7 de ellos de remplazo de fracciones del modelo arquitectónico destino. El multimodelo consta de 45 relaciones entre las vistas de calidad, de variabilidad y arquitectónica, utilizadas tanto en el proceso de configuración del producto, como en la posterior derivación del modelo de resolución CVL.

Tabla 8.1 Resumen de la estructura del modelo de características

Numero de Características	*Características obligatorias*	*Relaciones entre características (implicación/exclusión)*	*Numero de posibles configuraciones validas*
15	2	14 (13/1)	240

En el estudio se han analizado tres variables dependientes subjetivas: facilidad de uso percibida, utilidad percibida e intención de uso, basadas en el modelo de aceptación tecnológica (*Technology Acceptance Model TAM)* (Davis 1989), un modelo teórico para el análisis de la aceptación de nuevas tecnologías y del comportamiento. Las variables se describen a continuación:

- *Facilidad de Uso Percibida (FUP),* se refiere al grado en el que los participantes creen que el aprendizaje de un método en particular podrá llevarse a cabo con un esfuerzo minimo.
- *Utilidad Percibida (UP),* se refiere al grado en que los participantes consideran que el uso de un método especifico aumentará su rendimiento dentro de un contexto organizacional.
- *Intención de Uso (IU)*, se refiere a la medida en que un determinado participante tiene la intención de utilizar, en el futuro, un método particular. Esta última variable representa un juicio perceptual de la eficacia del método, es decir, si realmente es rentable. Esta variable es comúnmente empleada para predecir la probabilidad de aceptación de un determinado método en la práctica.

Para medir dichas variables se definió un cuestionario Likert (véase Apéndice A.5) con un conjunto de 13 preguntas cerradas: 3 para la facilidad de uso

[20] Información obtenida mediante la opción #*products* de la herramienta FaMa que devuelve el número de configuraciones válidas para un determinado modelo de características.

percibida, 6 para la utilidad percibida y 4 para la intención de uso. Las respuestas fueron codificadas mediante una escala Likert de 5 puntos con el formato de afirmaciones y respuestas escaladas de *totalmente de acuerdo* a *totalmente en desacuerdo*. El orden de las preguntas fue alterado de forma aleatoria con el fin de evitar el sesgo de respuesta sistemática y se reformularon algunas preguntas para convertirlas en enunciados negativos en la parte izquierda con el fin de evitar así las respuestas monótonas (Hu y Chau 1999). Se incluyeron también 4 preguntas abiertas con el fin de recabar la opinión de los participantes acerca de los cambios a realizar para mejorar el método, cuáles serían las razones que les moverían a usar el método en el futuro y qué mejoras sugieren para el método y para la infraestructura de soporte del multimodelo que se utilizó durante el estudio de caso.

Además se incluyeron 8 variables dependientes objetivas para analizar el grado en el que los participantes aplicaban el método de manera eficiente y efectiva:

- *Efectividad_RNFs:* número total de RNFs correctamente definidos (entendiendo por RNFs correctamente definidos aquellos que son acordes con los requisitos descritos en el boletín del estudio de caso).
- *Eficiencia_RNFs:* calculado como el ratio entre el número de RNFs correctamente identificados dividido por el tiempo total empleado en su definición.
- *AC_Correctos:* variable bi-valuada [0,1] que indica si la prioridad de los atributos de calidad definida por el participante es acorde a las restricciones del modelo de calidad.
- *RNFs_Correctos:* variable bi-valuada [0,1] que indica si los RNFs seleccionados por el participante cumplen con los requisitos descritos en el boletín del estudio de caso.
- *Características_Correctas:* variable bi-valuada [0,1] que indica si las características seleccionadas por el participante cumplen con los requisitos descritos en el boletín del estudio de caso.
- *Características_Válidas:* variable bi-valuada [0,1] que indica si las características seleccionadas por el participante cumplen con las restricciones definidas en la vista de variabilidad.
- *ModeloCVL_Correcto:* variable bi-valuada [0,1] que indica si el modelo de resolución CVL daría lugar a una arquitectura que cumple, tanto los

requisitos funcionales, como los no-funcionales descritos en el boletín del estudio de caso.

- *DuraciónTotal:* duración total de las tareas que integran el estudio de caso.

8.1.2 Selección de los participantes

Los sujetos del estudio fueron escogidos de entre un grupo de investigadores del campo de la ingeniería del software y profesionales del área del desarrollo de software. Uno de estos profesionales, a pesar de no tener conocimientos teóricos acerca del desarrollo de líneas de producto, cuenta con cinco años de experiencia en una compañía de desarrollo de software en la que su producto estrella es un software de gestión ERP en el que hay un tratamiento explícito de la variabilidad.

El estudio de caso se llevó a cabo en dos sesiones, la primera con siete estudiantes de Doctorado, un Doctor en Informática y un profesional con experiencia en desarrollo de software, todos ellos formados en la Universitat Politècnica de València. La segunda sesión se llevó a cabo con seis estudiantes de Doctorado y Máster de la Universidad Federal de Bahía. Se detectaron distintos perfiles de participantes, los cuales se describen a continuación, ordenados por sesiones:

- Universitat Politècnica de Valencia:
    - **Perfil UPV1:** estudiantes de Doctorado en Informática, todos ellos investigan actualmente temas relacionados con distintas áreas de la ingeniería del software y tienen nociones del desarrollo de líneas de producto software.
    - **Perfil UPV2:** estudiantes de Doctorado en Informática, todos ellos investigan actualmente temas relacionados con distintas áreas de la ingeniería del software y sin ninguna noción del desarrollo de líneas de producto software.
    - **Perfil UPV3:** Doctores en Informática que han desarrollado tareas de investigación en temáticas relacionadas directamente con el desarrollo de líneas de producto.
    - **Perfil UPV4:** Ingenieros en Informática con experiencia en la industria del desarrollo de software.

- Universidad Federal de Bahía:
    - **Perfil UFB1:** estudiantes de segundo y tercer año de doctorado en informática, con experiencia en desarrollo de líneas de producto (especificación de requisitos para LPS, arquitecturas y LPS e implementación aplicando la aproximación LPS).
    - **Perfil UFB2:** estudiantes de Máster en Informática con experiencia en desarrollo de líneas de producto (especificación de requisitos para LPS, arquitecturas y LPS e implementación aplicando la aproximación LPS).
    - **Perfil UFB3:** estudiantes de Máster en Informática sin experiencia en desarrollo de líneas de producto, matriculados en la asignatura de "Reúso en la Ingeniería del Software" en la que se estudian los conceptos propios de la aproximación de líneas de producto software.

La Figura 8.1 muestra la distribución por perfiles de las dos sesiones del estudio de caso.

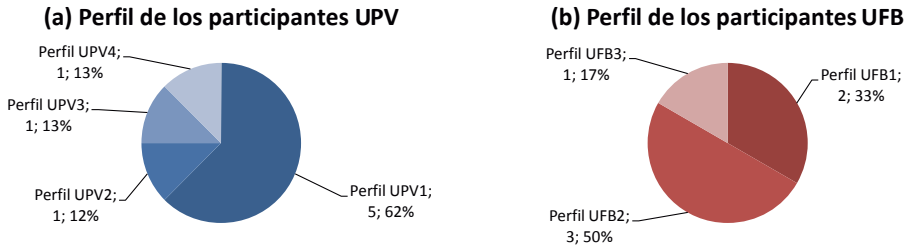


Figura 8.1 Distribución de los perfiles de los participantes en cada sesión

8.2 Preparación y ejecución del estudio de caso

En la Tabla 8.2 se muestra la planificación del estudio de caso. Antes de que los sujetos abordasen el estudio de caso, tuvo lugar un entrenamiento en el que se introdujo el método QuaDAI, el concepto de multimodelo, las distintas notaciones empleadas para expresar cada una de las vistas y las actividades de configuración del producto y de derivación de la arquitectura, con una duración total de 60 minutos. Tras el entrenamiento tuvo lugar la ejecución del estudio de caso con una duración prevista de 90 minutos, aunque se permitió que los participantes continuasen con el estudio de caso aun cuando pasasen de

esos 90 minutos, con el fin de eliminar el posible *efecto techo* (Sjøberg et al. 2003).

Tabla 8.2 Planificación del estudio de caso

Entrenamiento (60 min)	Introducción a QuaDAI Explicación de las tareas
Estudio de caso (90 min)	Tarea 1: Definición de los RNFs específicos del producto Tarea 2: Configuración del producto Tarea 3: Derivación del modelo de resolución CVL Cuestionario de satisfacción

El estudio de caso consistía en tres tareas:

- **Tarea 1:** definición de los requisitos no-funcionales del producto empleando la infraestructura de soporte al multimodelo[21]. A partir de los requisitos especificados textualmente, los participantes habían de plasmar aquellos requisitos no-funcionales que fuesen más restrictivos que los propios de la línea de productos.
- **Tarea 2:** configuración del producto en desarrollo. El propósito de esta tarea es la obtención de una configuración válida del producto. A partir de los requisitos, tanto funcionales como no-funcionales, los participantes debían seleccionar las características, requisitos no-funcionales (de la línea de productos o aquellos definidos específicamente para el producto en desarrollo) y priorizar los atributos de calidad. Para asegurarse de que la configuración obtenida cumplía las restricciones establecidas en el multimodelo, los participantes empleaban los módulos de validación de la consistencia previstos en la infraestructura de soporte al multimodelo.
- **Tarea 3:** derivación del modelo de resolución CVL. A partir de la configuración obtenida en la tarea 2 obtenían, mediante la infraestructura de soporte al multimodelo, el modelo de resolución CVL que resuelve los puntos de variabilidad de la arquitectura de la línea de productos.

Se diseñaron múltiples documentos como instrumentación para el estudio de caso. El material de entrenamiento consiste en un conjunto de diapositivas que contiene la descripción del método, distintas notaciones empleadas para

[21] La herramienta, los materiales del estudio del caso así como el proyecto Eclipse con el caso completo, están disponibles en: http://users.dsic.upv.es/~jagonzalez/CarCarSPL/

expresar cada una de las vistas y las actividades de configuración del producto y de derivación de la arquitectura.

La documentación de las tareas del estudio de caso incluye:

- Un boletín que contiene la descripción de la línea de productos CarCarSPL y los requisitos del producto en desarrollo. Además, en el boletín se detallan todas y cada una de las tareas a llevar a cabo por los sujetos. El Apéndice A.1 contiene el boletín completo del estudio de caso.

- Un anexo en el que se describen en detalle las diferentes características de los productos de la línea de productos CarCarSPL y que incluye el modelo de características de dicha línea de productos. El Apéndice A.2 contiene la especificación completa junto con el modelo de características.

- Un anexo en el que se detallan las relaciones de impacto entre atributos de calidad. El Apéndice A.3 contiene el diagrama con los impactos entre los atributos de calidad de interés del dominio.

- Un anexo donde se detallan las relaciones entre los elementos de las distintas vistas, el cual se muestra en el Apéndice A.4. Dichas relaciones restringen el número de configuraciones consideradas válidas.

Además, se definió un cuestionario que contiene las preguntas cerradas con las que los participantes pueden expresar su opinión respeto a la facilidad de uso, utilidad e intención de emplear el método en el futuro. En dicho cuestionario se incluyeron cuatro preguntas abiertas con las que los participantes pueden expresar sus impresiones, tres preguntas enfocadas el método y una a la infraestructura de soporte al multimodelo. El Apéndice A.5 contiene todas las preguntas cerradas relacionadas con las tres variables dependientes y las cuatro preguntas abiertas. El orden de las preguntas fue alterado de forma aleatoria con el fin de evitar el sesgo de respuesta sistemática y se reformularon algunas preguntas para convertirlas en enunciados negativos en la parte izquierda con el fin de evitar así las respuestas monótonas (Hu y Chau 1999).

Con respecto a la ejecución, el estudio tuvo lugar en una única habitación, y no se permitía la interacción entre los participantes. Las preguntas que surgieron durante la sesión fueron aclaradas individualmente por los propios instructores durante la sesión.

Con el fin de analizar la influencia que los diferentes perfiles pudieran tener sobre la percepción del método, se definió un cuestionario de nivelación, para que los participantes los rellenasen antes de comenzar la ejecución del estudio de caso. Dicho cuestionario consta de 7 preguntas de las que es posible extraer un indicativo de hasta qué punto el participante tiene conocimientos acerca de las líneas de producto software que le permitirán ejecutar el estudio de caso. El Apéndice A.6 contiene las preguntas del cuestionario de nivelación.

8.3 Recogida de datos

Los datos para este estudio de caso fueron recogidos durante la ejecución de la fase de derivación de QuaDAI. La captura de los datos se llevó a cabo empleando un formulario online. En él, los participantes introducían la información resultante de la aplicación del método, tal como se detallaba en el boletín del estudio de caso.

En la tarea 1, los participantes debían introducir los datos de los requisitos no-funcionales del producto en desarrollo que habían introducido previamente en la infraestructura de soporte al multimodelo utilizando el formulario que se muestra en la Figura 8.2.

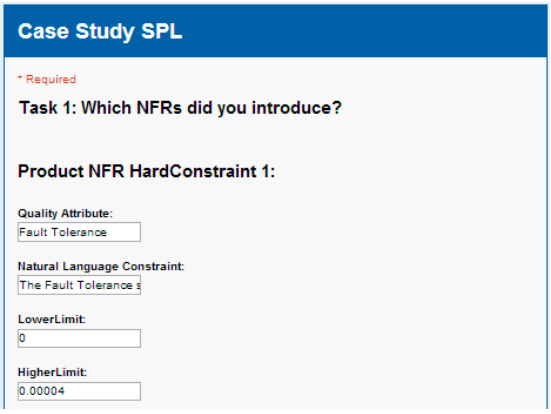


Figura 8.2 Formulario de captura de datos de la tarea 1

Con respecto a la tarea 2, los participantes debían llevar a cabo la configuración del producto y validación de la consistencia de dicha configuración, empleando la infraestructura de soporte al multimodelo. Una vez obtenida una configuración que satisfacía tanto los requisitos funcionales como los no-funcionales, los participantes debían introducir la información de la

configuración (prioridad de los atributos de calidad, requisitos no-funcionales y características seleccionadas) utilizando el formulario que se muestra en la Figura 8.3.

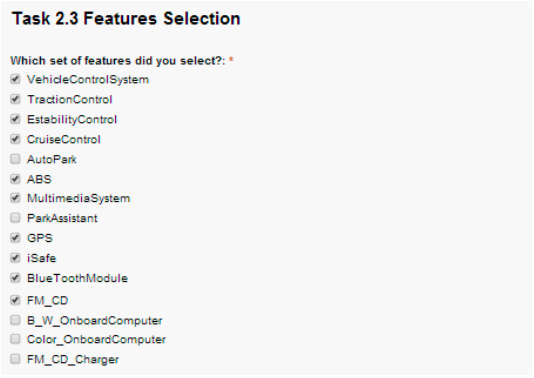


Figura 8.3 Formulario de captura de datos de la tarea 2

Con respecto a la tarea 3 , los participantes tenían que derivar el modelo de resolución CVL a partir de la configuración obtenida en la tarea 2 mediante la infraestructura de soporte al multimodelo y comprobar la configuración obtenida con alguna de las configuraciones validas propuestas en el boletín. En el boletín se incluían los 8 ocho modelos de resolución CVL más completos, aunque era posible obtener un total de 16 modelos de resolución CVL distintos a partir de configuraciones válidas. La Figura 8.4 muestra el formulario mediante el que los participantes introducían el resultado de la derivación del modelo de resolución CVL.

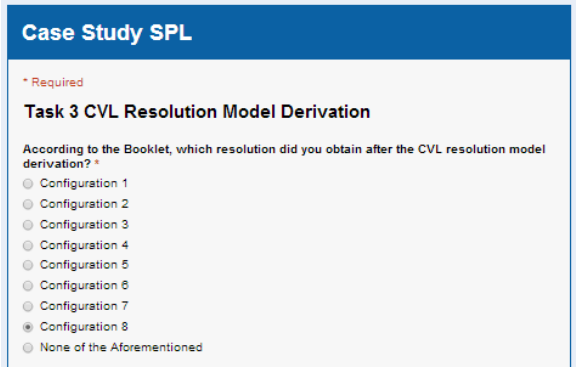


Figura 8.4 Formulario de captura de datos de la tarea 3

8.4 Análisis de los resultados

En esta sección se discuten los resultados extraídos de la ejecución del estudio de caso.

Todos los test estadísticos que se presentan en esta sección fueron llevados a cabo utilizando SPSS v20 con un nivel de significancia estadística $\alpha = 0.05$. Para el análisis, se utilizaron estadísticos descriptivos, diagramas de frecuencia, diagramas de cajas y pruebas estadísticas con el fin de analizar los datos obtenidos de cada estudio individual.

Hemos llevado a cabo dos análisis de los datos recogidos: un análisis cualitativo y un análisis cuantitativo. En el análisis cualitativo se analiza el grado en el que el método se aplicó correctamente y si los diferentes pasos daban como resultado especificaciones de RNFs, configuraciones y modelos de resolución CVL válidos, tanto desde el punto de vista sintáctico (respetan las restricciones de consistencia definidas en el multimodelo), como desde el punto de vista semántico (reflejan los requisitos de cliente descritos en el boletín para el producto en desarrollo). Esto nos permite identificar tanto deficiencias en el método como eventuales inconsistencias en la definición del estudio de caso. El análisis cuantitativo se basa en las respuestas del cuestionario para testear las hipótesis definidas en la sección 8.1.1.

8.4.1 Análisis Cualitativo

En primer lugar, con respecto a las preguntas del cuestionario de nivelación relativas al conocimiento de aspectos relacionados con la aproximación de líneas de producto software, el resultado fue mejor del esperado. Solo dos participantes no contestaron correctamente una de las preguntas, mientras que el resto (un 88%), contestó correctamente todas las preguntas, por lo que se descartó el uso de esta variable como factor en el análisis.

Partiendo de las respuestas a las preguntas del cuestionario de nivelación relativas a la experiencia previa, se estableció una valoración del perfil. Esto permite analizar las diferencias que pudiesen existir entre los valores de las variables dependientes debidas al factor *Perfil*. Dicho factor se calculó agregando los valores de las preguntas 1, 2 y 3 del cuestionario de nivelación, aplicando la fórmula (1). Para dicha agregación, se asignaron los valores a la respuesta de la preguntas 1 y 3 de acuerdo con la Tabla 8.3. Se calculó el número de años de experiencia en la aplicación de LPS como resultado de dividir el valor de la respuesta de la pregunta 2 del cuestionario por 12.

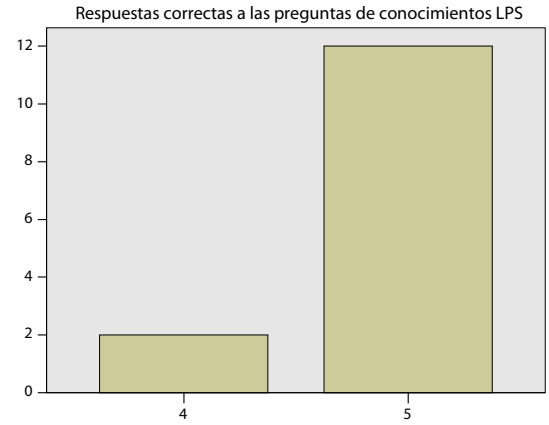


Figura 8.5 Respuestas correctas al cuestionario de nivelación relativas a conocimientos previos acerca de las líneas de producto software

La Figura 8.6 muestra la distribución de la variable para los participantes en el estudio, el sujeto 8, que se muestra como dato anómalo en esta distribución, es un participante que cuenta con 3 años de experiencia en la aplicación de la aproximación de LPS en el entorno industrial y académico.

$$Valoracion_{pregunta1} + Valoracion_{pregunta3} + \frac{Pregunta2}{12} \quad (1)$$

Tabla 8.3 Valoración de las respuestas a las preguntas del cuestionario de nivelación

	Respuesta	***Valor***
Pregunta 1	He participado en equipos de desarrollo aplicando la aproximación LPS	3
	Soy un investigador trabajando en temas relacionados con el desarrollo de LPS	2
	Sé lo que son las LPS pero nunca he participado en un proyecto software aplicando LPS	1
	No he oído hablar de las LPS con anterioridad	0
Pregunta 3	He aplicado la aproximación LPS sólo en el ámbito de la investigación	1
	He aplicado la aproximación LPS sólo en el ámbito industrial	2
	He aplicado la aproximación LPS sólo en el ámbito industrial y en el de la investigación	4
	No he aplicado la aproximación LPS con anterioridad	0

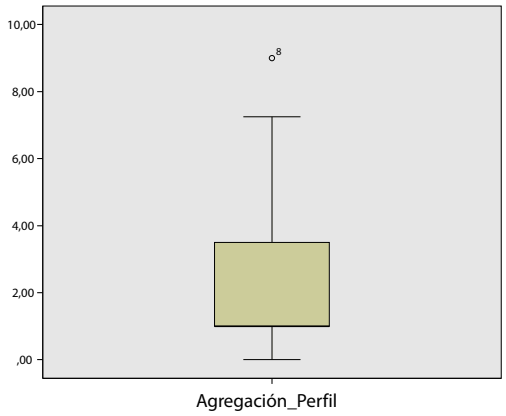


Figura 8.6 Distribución de la variable *agregación perfil*

Para considerar que un participante tenía ciertos conocimientos acerca de la aproximación LPS y asignar un valor 1 al factor *Perfil*, consideramos un valor umbral de 3 (al menos cuenta con un año de experiencia como investigador en el área). La Figura 8.7 muestra la distribución de participantes atendiendo a dicho factor.

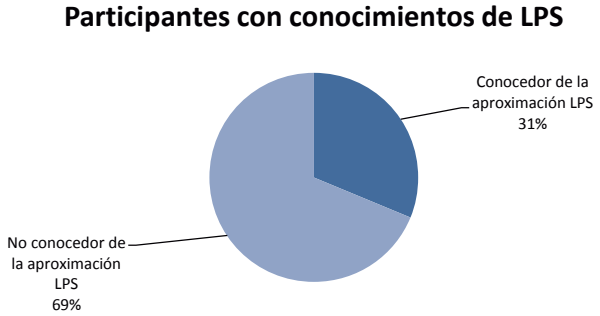


Figura 8.7 Distribución de los participantes atendiendo al factor *Perfil*

Con el fin de descartar la existencia de diferencias significativas entre las poblaciones de participantes con conocimientos de LPS (*Perfil=0*) y sin conocimiento de LPS (*Perfil=1*) o entre las poblaciones de participantes de Brasil (*Sesión=BR)* o de España (*Sesión=ES*), se han llevado a cabo las pruebas estadísticas correspondientes. Dado que el número de observaciones N es menor que 50, para cada variable se realizó el test de Shapiro-Wilk con el fin de comprobar si las variables se distribuían normalmente. La Tabla 8.4 muestra los resultados del test de Shapiro-Wilk para las variables dependientes objetivas.

Tabla 8.4 Resultados de las pruebas de normalidad

	Sesión		*Perfil*	
Variable	***BR***	***ES***	***0***	***1***
Efectividad_RNFs	0,001	0,005	0	0,021
Eficiencia_RNFs	**0,481**	**0,735**	**0,642**	**0,739**
AC_Correctos	*	*	0	0
RNFs_Correctos	0,004	0,004	0	0
Características_Correctas	0,004	*	0	0
Características_Validas	0,004	*	0	0
ModeloCVL_Correcto	0,004	*	0	0
DuracionTotal	**0,726**	**0,365**	0,23	0,541

* La variable se mantiene constante en la población por lo que se descarta la prueba de normalidad

Para aquellas variables normalmente distribuidas ($p \geq 0.05$), se analizó la significancia mediante la prueba t-test para dos muestras independientes con el factor *Perfil* y *Sesión* (España y Brasil) o la prueba de Mann-Whitney para dos muestras independiente con el factor *Perfil* y *Sesión* (España y Brasil) para las variables que no se distribuyen normalmente ($p < 0.05$). No se observaron diferencias significativas (*significancia>0.05*) entre las poblaciones *conocedores* y *no-conocedores* ni entre las poblaciones (España y Brasil), excepto para la variable *Eficiencia_RNFs*[22], por lo que en el resto de la sección analizará los datos de forma conjunta (excepto en el caso de la variable *Eficiencia_RNFs*).

Tabla 8.5 Resultado de las pruebas de significancia de las diferencias entre poblaciones

	Sesión	***Perfil***
Efectividad_RNFs	0,345	0,0364
Eficiencia_RNFs	**0,01***	**0,043***
AC_Correctos	1	1
RNFs_Correctos	0,573	1
Características_Correctas	0,142	1
Características_Validas	0,142	1
ModeloCVL_Correcto	0,142	1
DuracionTotal	0,65*	1

* Resultados de la prueba *t* para dos muestras independientes

La Tabla 8.6 muestra los resultados globales para las variables dependientes objetivas. Las variables que siguen una distribución normal (*Duración* con un p=0,187 en el test de Shapiro-Wilk) y *Eficiencia_NFRs* (cuyos valores p del test de Shapiro-Wilk se muestran en la Tabla 8.4, dado que en este caso sí que se demostró como estadísticamente significativa la influencia de la sesión y el

[22] Los tiempos se midieron a partir de los datos introducidos en el cuestionario online. Dado que algunos participantes experimentaron problemas de acceso durante la realización del ejercicio (afectando sobre todo a los campos relacionados con la duración de las tareas) la duración total, así como la variable de eficiencia, sólo tienen carácter orientativo.

perfil) se describen mediante la media y la desviación típica, mientras que aquellas no distribuidas normalmente (con p =0 en el test de Shapiro-Wilk) se describen mediante la mediana y varianza.

Tabla 8.6 Estadísticos descriptivos para las variables dependientes objetivas

Variable		*Mediana*	*Varianza*
Efectividad_RNFs		3	1,1341
AC_Correctos*		1	0
RNFs_Correctos		1	0,181
Características_Correctas		1	0,426
Características_Validas		1	0,181
ModeloCVL_Correcto		1	1

*AC_Correctos es constante para la población

Variable		*Media*	*Desv. Est*
Eficiencia_RNFs	***ES***	0,982	0,03
	BR	0,301	0,03
	0	0,846	0,045
	1	0,409	0,026
Duración		102,71	34,048

La Figura 8.8 y la Figura 8.9 muestran los diagramas de frecuencias para las distintas variables cuya distribución no se ajusta a una normal. La Figura 8.10 muestra los diagramas de caja para las variables distribuidas normalmente.

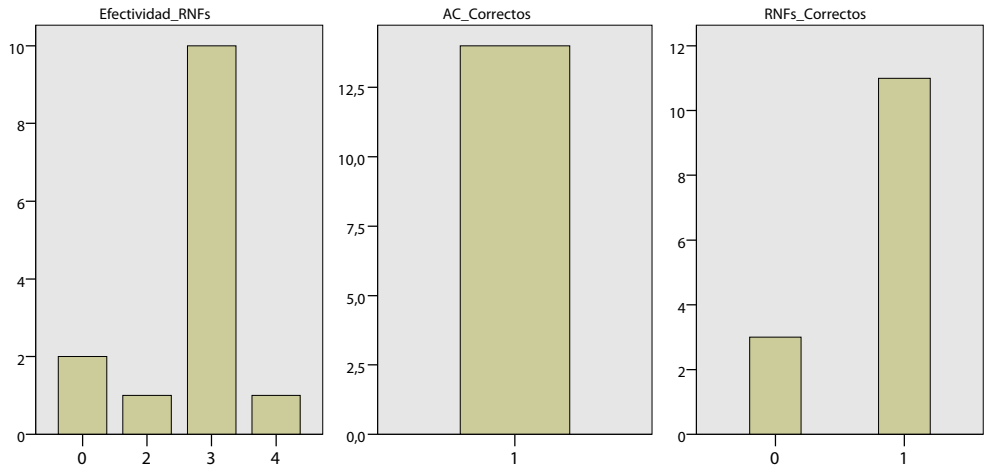


Figura 8.8 Diagramas de frecuencias para las variables *Efectividad_RNFs*, *AC_Correctos* y *RNFs_Correctos*

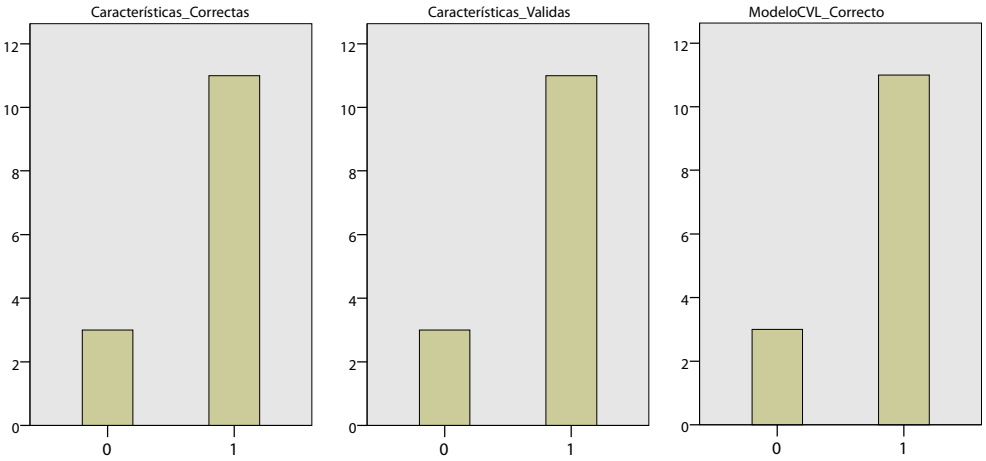


Figura 8.9 Diagramas de frecuencias para las variables Caracteristicas_Correctas, Caracteristicas_Validas y ModeloCVL_Correcto

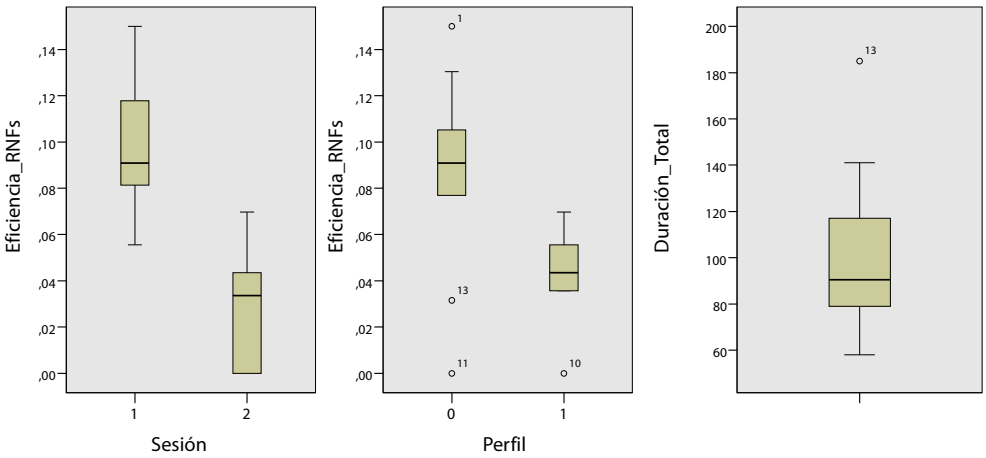


Figura 8.10 Diagramas de caja para las variables *Eficiencia_RNFs* por *Sesión* y *Perfil* y de la variable *Duración_Total*

8.4.2 Análisis Cuantitativo

Al igual que en el caso de las variables objetivas, el primer análisis a llevar a cabo es un test para comprobar si hay diferencias significativas asociadas al factor *Perfil* o a la *Sesión*. Para seleccionar la prueba a aplicar, y dado de que el número de muestras N es menor que 50, primero ejecutamos el test de Shapiro-Wilk con el fin de analizar si las variables se distribuyen normalmente.

La Tabla 8.7 muestra el resultado de dicha prueba para las variables subjetivas por *Sesión* y *Perfil*.

Tabla 8.7 Resultados de las pruebas de normalidad de las variables subjetivas por sesión y perfil

	Sesión		*Perfil*	
Variable	***BR***	***ES***	***0***	***1***
FUP	0,844	0,855	0,829	0,272
UP	**0,02**	0,205	0,381	0,159
IU	0,093	0,06	0,289	0,826

El resultado de la prueba de Shapiro-Wilk nos permite seleccionar las pruebas a aplicar en cada caso, con el fin de descartar diferencias entre las poblaciones. Por un lado, cuando los datos se ajustan a la distribución normal ($p \geq 0,05$) se aplicó el test *t* paramétrico unilateral para muestras independientes (Juristo y Moreno 2001). Por otro lado, en el caso de variables que no se ajustan a una distribución normal, se aplicó la prueba no paramétrica de Mann-Whitney (Conover 1998). La Tabla 8.8 muestra el resultado de dichas pruebas, lo que nos permite descartar la existencia de diferencias significativas entre las variables para las diferentes poblaciones, por lo que el análisis para contrastar las variables se va a llevar a cabo de manera conjunta.

Tabla 8.8 Resultado de las pruebas de significancia de las diferencias entre poblaciones

Variable	***Sesión***	***Perfil***
FUP	0,274	0,918
UP	0,081*	0,893
IU	0,064	0,566

* Resultados de la prueba *t* para dos muestras independientes

La Figura 8.11 muestra los diagramas de caja para las distintas variables subjetivas. Estos diagramas muestran que, bajo las condiciones del estudio de caso, el método es percibido como fácil de usar, útil y los participantes muestran una cierta intención de emplearlo en el futuro.

La Tabla 8.9 muestra un resumen de los resultados globales para las variables subjetivas. Se han utilizado la media y la desviaciones estándar como estadísticos descriptivos para las variables subjetivas cualitativas FUP, UP y IU. La media aritmética de las escala Likert de cada una de las respuestas al cuestionario adoptada para la medición de las variables subjetivas, ha sido también considerada como una escala de intervalo (Carifio y Perla 2007).

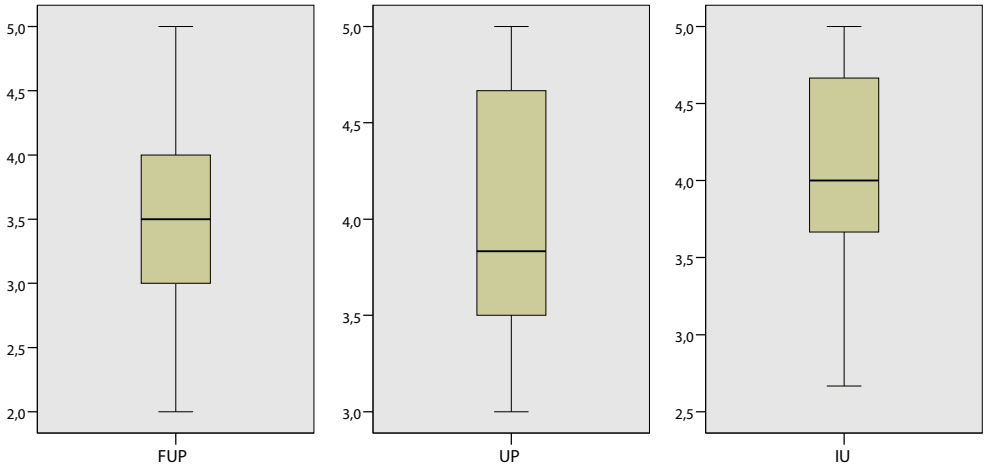


Figura 8.11 Diagramas de caja de las variables subjetivas FUP, UP e IU

Tabla 8.9 Resumen de los resultados para las variables subjetivas

Variable	*Media*	*Desv. Est.*
FUP	3,48	0,823
UP	4	0,65
IU	3,95	0,702

Una vez descartadas diferencias debidas al factor *Perfil* o *Sesión*, se aplicaron pruebas estadísticas para contrastar las hipótesis, comparando si la media de respuestas a las preguntas relacionadas con una variable, era significativamente mayor que el valor Likert neutro. En nuestro caso, las escalas ordinales variaron de 0 a 5 y el valor neutro corresponde al valor 3. Para variables que se distribuyen normalmente, se aplica la prueba *t* para una muestra. Se ejecutó de nuevo el test de Shapiro-Wilk, esta vez sobre una única población (N=14). La Tabla 8.10 muestra el resultado del dicho test, del que se desprende que las variables se distribuyen normalmente, así como los resultados de la prueba *t* para contrastar las hipótesis H1, H2 y H3. Estos resultados nos permiten rechazar las hipótesis nulas $H1_0$, $H2_0$, $H3_0$ y la aceptación de sus correspondientes hipótesis alternativas, lo que significa que los participantes perciben el método de derivación como fácil de usar, útil y que tienen cierta intención de emplearlo en el futuro.

Tabla 8.10 Resultado de las pruebas de significancia de la hipótesis H1, H2 y H3

Variable	*p Shapiro-Wilk*	*t-test*
FUP	0,927	0,047
UP	0,16	0,000
IU	0,377	0,000

8.5 Amenazas a la validez

En esta sección se describen las principales amenazas a la validez del estudio de caso, al considerar el esquema de amenazas a la validez propuesto por Yin (2009) similar al esquema que se emplea tradicionalmente en los experimentos controlados en la ingeniería del software (Runeson y Höst 2008, Wohlin et al. 2012, Runeson et al. 2012).

8.5.1 Validez de constructo

La principal amenaza a la validez del constructo en los estudios de caso, es reflejar hasta qué punto las métricas que se han estudiado representan el objetivo que el investigador tenía en mente. En este caso, la principal amenaza es la fiabilidad del cuestionario en relación con las tres hipótesis del estudio de caso. A este respecto cabe destacar que, durante la ejecución de la primera sesión, se detectó que la pregunta 12 del cuestionario estaba redactada incorrectamente, por lo que fue excluida de todos los análisis posteriores (véase Apéndice A.5). Para analizar la fiabilidad del cuestionario se ha llevado a cabo un análisis del Alfa de Cronbach para cada conjunto de preguntas relacionadas con cada variable subjetiva. Al llevar a cabo este análisis, se detectó que la pregunta 13 no explicaba el mismo constructo que el resto de variables del conjunto de la variable IU, por lo que fue asimismo descartada del análisis[23]. La Tabla 8.11 muestra los resultados del test de fiabilidad, donde puede observarse que los valores son mayores que el umbral mínimo de aceptación α=0.70 (Maxwell 2002).

Tabla 8.11 Resultados del análisis de fiabilidad del cuestionario

Variable	*Alfa de Cronbach*
FUP	0,726
UP	0,828
IU	0,714

8.5.2 Validez interna

Las amenazas a la validez interna son relevantes a la hora de examinar relaciones de causalidad (Runeson y Höst 2008). En el estudio de caso hemos

[23] Este hecho fue detectado por el análisis de Alfa de Cronbach, utilizando el descriptivo adicional de simulación de la eliminación de un elemento.

tratado de analizar, mediante pruebas estadísticas, si los valores de las distintas variables se podían ver afectados por la experiencia o la procedencia de los participantes, concluyendo que dichos factores no impactaban en las variables en estudio. Además, hemos tratado de mitigar el posible efecto de *sesgo de autor* llevando a cabo dos sesiones: la primera de ellas dirigida por uno de los autores del método y la segunda dirigida por un investigador externo, sin que se observen diferencias significativas entre ambas poblaciones.

8.5.3 Validez externa

Este aspecto de la validez se refiere a la extensibilidad de los resultados (Runeson y Höst 2008). En este caso hemos tratado de proponer un estudio de caso de un dominio relevante y con una complejidad suficiente, teniendo en cuenta las limitaciones de tiempo de las sesiones. Se ha tratado, además, de integrar en el estudio a profesionales e investigadores en el área, así como participantes sin antecedentes en el desarrollo de líneas de producto, con el fin de aumentar la extensibilidad de los resultados. Teniendo en cuenta los resultados, en el futuro, será necesario llevar a cabo replicaciones de este estudio de caso en grupos con mayor número de participantes y con mayor homogeneidad entre ellos.

8.5.4 Fiabilidad

Este aspecto de la validez está relacionado con la posibilidad de que los datos y el análisis sean dependientes de los investigadores que los han llevado a cabo. En nuestro caso, para la explicación de las variables estudiadas, se ha empleado un conjunto de preguntas cerradas y las agregaciones de las mismas, en caso necesario, ha sido descrita en detalle. Además, todos y cada uno de los artefactos están disponibles para su descarga en http://users.dsic.upv.es/~jagonzalez/CarCarSPL/.

8.6 Lecciones aprendidas

La aplicación de la fase de derivación de QuaDAI en un contexto realista[24], sobre un ejemplo de una relativa complejidad y tanto por profesionales como por investigadores del área del desarrollo de LPS nos ha permitido adquirir experiencias de las que es posible aprender ciertas lecciones.

Por un lado nos ha permitido analizar la aplicabilidad de la propuesta al modelar todo el proceso completo empleando la infraestructura de soporte al multimodelo. Además nos ha permitido detectar posibles variantes que inicialmente no habían sido contempladas, como en el caso de las transformaciones para obtener el modelo de resolución CVL, donde determinados puntos de variabilidad podrían depender de resoluciones previas de otros puntos de variabilidad. Aunque en el conjunto de transformaciones descrito en la Sección 5.2.2.1 esta opción no se contempla, puede ser resuelto incluyendo en las precondiciones de las reglas las cláusulas necesarias.

Por otro lado nos ha permitido obtener realimentación de los participantes, muchos de sus comentarios hacían hincapié en la dificultad de establecer las prioridades de los atributos de calidad. Este es uno de los puntos de mejora de la propuesta, por un lado mejorando la descripción del proceso en la definición del método y por otro buscando medios alternativos a estos pesos para reflejar las prioridades en cuanto a los atributos de calidad. Otra de las sugerencias de los participantes es el establecimiento de relaciones entre los RNFs de manera que al definir un RNF como *extensión* o *refinamiento* de un RNF existente, la selección del primero automáticamente implique la selección del segundo, tarea que en la actualidad realiza, manualmente el ingeniero de aplicación.

Además, hemos podido constatar que la usabilidad de la infraestructura de soporte al multimodelo es otra de las carencias. Si bien es cierto que los participantes apreciaron las funcionalidades de validación de consistencia y derivación automática de la arquitectura, la navegabilidad de los modelos y la realimentación al usuario, al emplear representaciones en árbol de los modelos, ha sido reportado como problemático por la gran mayoría de los participantes que rellenaron las preguntas abiertas del cuestionario.

[24] A pesar de que el estudio de caso se ha llevado a cabo en entornos académicos y de manera *off-line* (Wohlin et al. 2012), se han recreado los artefactos con la suficiente complejidad y reflejando la problemática propia de las tareas a llevar a cabo, por lo que consideramos que se trata, con ciertas limitaciones, de un contexto realista.

Por último, la aplicación en un caso práctico nos ha hecho ver que el proceso de configuración, en el que primero se seleccionan características y posteriormente se seleccionan requisitos no-funcionales y atributos de calidad (véase Sección 5.1.1) en determinados casos puede ser extremadamente complejo. En el estudio de caso hemos descrito un proceso para configurar el producto en el que primero se establecen las restricciones (prioridades de atributos de calidad y requisitos no-funcionales) y posteriormente se seleccionan las características. No se trata de guiar la selección de características a partir de criterios de calidad, sino más bien restringir el espacio de búsqueda para no seleccionar una configuración que posteriormente, al introducir las restricciones, resulte no valida. En el estudio de caso, el número de configuraciones validas era de 240, pero una vez introducidas las restricciones, únicamente 16 cumplían con los requisitos no-funcionales y las prioridades de atributos de calidad, por lo que comenzar seleccionando características podría ser ineficiente. Somos conscientes de que el artefacto de primer orden en la configuración es la característica, pero llevando a cabo el proceso de configuración de este modo, introduciendo las restricciones de calidad antes de seleccionar las características, es posible, de una manera sencilla, detectar que no es posible satisfacer la totalidad de los requisitos. Esto puede dar lugar a nuevas negociaciones con el cliente con el fin de relajar los requisitos no-funcionales (en la medida de lo posible) y que la funcionalidad demandada por el cliente en forma de características quede cubierta.

8.7 Conclusiones

En este capítulo se han presentado los resultados de la validación empírica con el objetivo de evaluar la facilidad de uso percibida, utilidad percibida e intención de uso futura de los participantes cuando emplean la fase de derivación de QuaDAI sobre un estudio de caso concreto: CarCarSPL.

Los resultados del análisis cualitativo nos demostraron que, en general, los participantes fueron capaces de configurar y obtener una arquitectura de producto que cumple los requisitos tanto funcionales como no-funcionales mediante la aplicación del método. Del análisis cuantitativo se desprende que los participantes aprecian la fase de derivación de arquitecturas del método QuaDAI como fácil de usar, útil y muestran intención de usarlo en el futuro y que los valores obtenidos tienen significancia estadística. Los valores medios de las variables son 3,48 para la *facilidad de uso percibida*, 4 para la *utilidad percibida* y 3.95 para la *intención de uso*. El valor de facilidad de uso, más bajo que el resto, se debe al hecho de que, a pesar de que se hizo hincapié en que las preguntas del

cuestionario hacían referencia al método y no a la infraestructura de soporte del multimodelo, es inevitable que los participantes reflejen la facilidad de uso de la herramienta en las respuestas de dicho cuestionario.

Desde el punto de vista investigador, el estudio de caso nos ha permitido, analizar la aplicabilidad real del proceso de derivación, modelar un caso real y analizar el grado de completitud y corrección de las soluciones aportadas por el método y por último, pero no menos importante, obtener realimentación de los participantes que aplicaron el método, cuyos comentarios nos permitirán mejorar en el futuro tanto el método como la infraestructura de soporte al multimodelo.

Desde el punto de vista práctico, este es un primer estudio con un número muy reducido de participantes, con perfiles heterogéneos y que solo arroja resultados preliminares sobre la utilidad de la fase de derivación del método QuaDAI. En el futuro será necesario replicar este estudio de caso en grupos más homogéneos y con mayor número de participantes para analizar posibles interacciones de factores como la experiencia.

Capítulo 9

Una Familia de Experimentos para la Validación del Proceso de Evaluación y Mejora de Arquitecturas Software

Este capítulo presenta la validación de la fase de evaluación y mejora de QuaDAI mediante una familia de experimentos en las que se compara la efectividad, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso. La estructura del capítulo es la siguiente:

En la Sección 9.1 se presenta el método de evaluación de arquitecturas software ATAM con el que se va a comparar nuestra propuesta.

En la Sección 9.2 se da la visión general de la familia de experimentos en la que se analiza el rendimiento real y percibido de QuaDAI en comparación con ATAM.

En la Sección 9.3 se describe el diseño de cada uno de los cinco estudios que componen la familia de experimentos.

En la Sección 9.4 se presentan los resultados de cada uno de los experimentos.

En la Sección 9.5 se presentan los resultados de la familia de experimentos junto con el meta-análisis.

En la Sección 9.6 se lleva a cabo un análisis pormenorizado de las amenazas a la validez.

Y por último, en la Sección 9.7 se presentan las conclusiones y observaciones finales.

9.1 Métodos de evaluación de arquitecturas comparados

En la familia de experimentos se han comparado dos métodos de evaluación arquitectónica: las fases de evaluación y transformación de nuestra propuesta (QuaDAI) y el método ATAM (*Architecture TradeOff Analysis Method*) propuesto por Kazman et al. (2000). Tanto ATAM como QuaDAI pueden ser clasificados como métodos de evaluación temprana, si atendemos al marco de clasificación propuesto por Roy y Graham (2008). QuaDAI se centra en la evaluación y mejora de arquitecturas de producto una vez éstas se han derivado de la arquitectura de la línea de productos. Aunque ATAM fue inicialmente definido para evaluar arquitecturas software de propósito general, éste puede ser empleado en entornos de desarrollo de líneas de producto software tanto para la evaluación de la arquitectura de la línea de producto, como para la evaluación de arquitecturas de producto en distintas etapas del desarrollo (diseño conceptual, antes de la codificación, durante o tras el desarrollo) (Clements y Northrop 2002), tal como se demuestra en varios informes de experiencias, por ejemplo en los trabajos de Ferber et al. (2001) o Barbacci et al. (2003).

El método ATAM es un método de evaluación de arquitecturas desarrollado por el *Software Engineering Institute* a finales de la década de los 90. El propósito de ATAM es evaluar las consecuencias de las decisiones arquitectónicas en relación a determinados atributos de calidad (Kazman et al. 2000). ATAM ayuda a prever cómo un determinado atributo de interés, puede verse afectado por una decisión de diseño arquitectónico. Los atributos de calidad de interés son clarificados mediante el análisis de los escenarios de las partes interesadas en términos de estímulos y respuestas. Por último, ATAM ayuda a definir cuáles son los enfoques arquitectónicos que pueden afectar a los atributos de calidad de interés. ATAM hace uso de árboles de utilidad para traducir los objetivos de negocio de un determinado sistema en escenarios concretos de atributos de calidad. Los árboles de utilidad son una estructura jerárquica en la que se especifica la utilidad de un sistema en términos de los atributos de calidad que se dividen a su vez en requisitos y escenarios.

Los principales objetivos de ATAM son obtener y refinar los objetivos de calidad de la arquitectura, obtener y perfeccionar las decisiones de diseño arquitectónico y evaluar las decisiones de diseño arquitectónico, con el fin de

determinar si éstos permiten satisfacer los requisitos de calidad descritos en el árbol de utilidad.

ATAM consta de nueve pasos que pueden ser separados en cuatro fases:

i) *Presentación:* que consiste en la presentación del método, de los objetivos de negocio y de la arquitectura a evaluar.

ii) *Investigación y análisis*: que consiste en la identificación de los enfoques arquitectónicos, la generación del árbol de utilidad y el análisis de los enfoques arquitectónicos basado en los escenarios de alta prioridad identificados en el árbol de utilidad.

iii) T*esteo:* en el que se lleva a cabo un *brainstorming* y se priorizan los escenarios descritos en el árbol de utilidad, se analizan los enfoques arquitectónicos teniendo en cuenta los escenarios de alta prioridad y la definición de los enfoques arquitectónicos a aplicar, los riesgos y no riesgos, puntos de sensibilidad y puntos de *tradeoff*.

iv) *Reporte:* que consiste en presentar los resultados de ATAM.

La Figura 9.1 muestra un resumen de estas fases y pasos junto con los principales artefactos generados en cada paso.

Por último, las salidas de ATAM son: i) una descripción priorizada de los requisitos de atributos de calidad; ii) una correspondencia de los enfoques arquitectónicos en relación con los atributos de calidad; iii) un catálogo de los enfoques arquitectónicos identificados y aplicados; iv) un catálogo describiendo los riesgos y no-riesgos identificados; v) un análisis de las cuestiones relativas a los atributos de calidad; y vi) puntos sensibles y puntos de relaciones de *tradeoff* (Kazman et al. 2000; Clements et al. 2002).

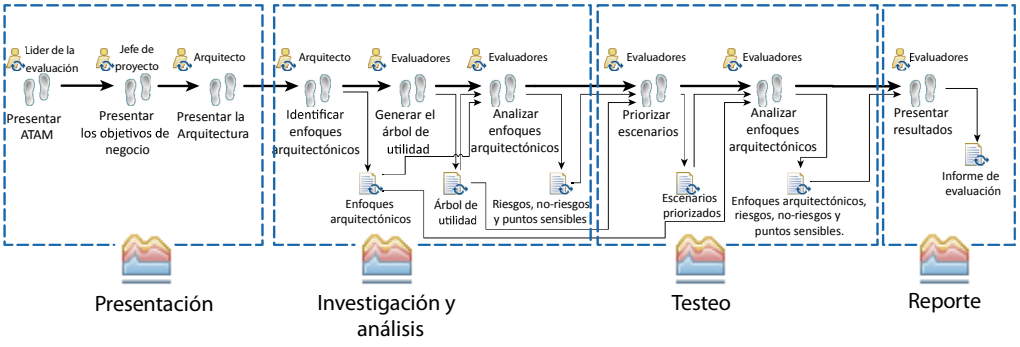


Figura 9.1 Resumen de las fases y pasos de ATAM

Se ha seleccionado ATAM como referencia por las razones siguientes:

- Se trata de un método de evaluación de arquitecturas software ampliamente difundido (Mårtensson 2006).
- Ha sido extensamente aplicado y validado (Roy y Graham 2008) en entornos industriales, en trabajos como el de Reijonen et al. (2010), Boucké et al. (2006) o Barbacci et al. (2003), y en el ámbito académico en trabajos como el de Svahnberg y Mårtensson (2007).
- Se llevan a cabo análisis *tradeoff* entre atributos de calidad y decisiones de diseño (Kazman et al. 2000).
- Puede ser empleado para llevar a cabo actividades de reingeniería de la arquitectura software y actividades para la extracción y aplicación de estilos y patrones arquitectónicos (Roy y Graham 2008).
- Se emplea el análisis cuantitativo y métricas para evaluar los distintos atributos de calidad (Roy y Graham 2008).

9.2 Descripción general de la familia de experimentos

En el campo de la ingeniera del software, como en cualquier otra disciplina técnica, existe la necesidad de emplear estudios empíricos para crear, mejorar o evaluar los procesos, métodos y herramientas empleados para el desarrollo, mantenimiento y aseguramiento de la calidad del software (Basili et al. 1986; Basili 1996; Dzidek et al. 2008; Colosimo et al. 2009).

Un estudio empírico es generalmente un acto u operación que permite descubrir algo que no se conoce o para poner a prueba una hipótesis (Basili 1993). Las estrategias de investigación incluyen experimentos controlados, estudios cualitativos, encuestas y análisis de archivos (Wohlin et al. 2012). Sin embargo, la replicación de estos estudios es necesaria para lograr una mayor validez en los resultados (Shull et al. 2008; Kitchenham 2008). El siguiente paso es la extensión del concepto de replicación a *"familia de experimentos"* (Basili et al. 1999). Una familia se compone de varios experimentos similares que persiguen el mismo objetivo: construir el conocimiento necesario para extraer conclusiones significativas.

En esta sección, se presenta la familia de experimentos llevados a cabo para validar empíricamente las etapas de evaluación y mejora de QuaDAI. La metodología adoptada es una extensión de los cinco pasos propuestos por

Ciolkowski et al. (2002), en la que el quinto paso, el análisis de datos de la familia, ha sido ampliado con el meta-análisis. Cada experimento se diseñó de acuerdo con el proceso experimental propuesto por Wohlin et al. (2012).

9.2.1 Paso 1: Preparación de los experimentos

El objetivo de los experimentos, de acuerdo con el paradigma *Goal-Question-Metric* (Basili et al. 1994), es **analizar** QuaDAI y ATAM **con el propósito** de compararlos **con respecto a** su eficacia, eficiencia, facilidad de manejo, utilidad e intención de uso para obtener arquitecturas software que cumplen con un determinado conjunto de requisitos de calidad **desde el punto de vista** de evaluadores de arquitecturas software noveles.

9.2.2 Paso 2: Definición del contexto

El contexto de la familia de experimentos es la evaluación de dos arquitecturas software llevadas a cabo por evaluadores noveles. Dicho contexto viene definido por i) las arquitecturas software a ser evaluadas; ii) los métodos de evaluación y iii) la selección de participantes.

9.2.2.1 Arquitecturas software evaluadas

Las arquitecturas de software a ser evaluadas en la familia de experimentos son la arquitectura software de un Sistema Antibloqueo de Frenos (sistema ABS) de un sistema de control de automóviles y la arquitectura software de la aplicación Savi (http://goo.gl/1Q49O), que es una aplicación móvil para notificaciones de emergencia.

La arquitectura del sistema ABS, representada a través de la vista componente-conector, expresada en AADL, fue seleccionada como objeto experimental O1 (un extracto de la descripción de la arquitectura se muestra en el Apéndice B.1.1), y la arquitectura de Savi, representada a través de la vista de despliegue, fue seleccionado como objeto de experimental O2. También se seleccionaron dos conjuntos de cuatro patrones arquitectónicos que se pueden aplicar para mejorar los niveles de los atributos de calidad de interés en cada una de las arquitecturas de software.

Las tareas experimentales incluyen la evaluación de estos atributos de calidad por medio de dos métricas en cada objeto experimental antes y después de aplicar los métodos de evaluación de arquitecturas software. La Tabla 9.1 muestra los detalles de los objetos experimentales utilizados en los

experimentos individuales. Tanto las tareas experimentales y el material experimental van a ser descritos en detalle en la Sección 9.2.3.

Tabla 9.1 Detalles de los objetos experimentales

Objeto Experimental	*Vista arquitectónica*	*RNF*	*Métricas*		*Patrones Arquitectónicos*
			Fiabilidad	*Rendimiento*	
O1: Sistema ABS	Componente-conector	Fiabilidad y rendimiento	Probabilidad de fallo	Tiempo de latencia	Triple redundante, Redundante homogénea, Watchdog y Sanity vheck (Douglass 2002)
O2: Savi	Vista de despliegue	Fiabilidad y rendimiento	Tiempo en servicio	Capacidad de carga	Balanceador de carga, Cluster simétrico, Cluster asimétrico, Cluster libre de fallos (Microsoft MSDN 2003)

9.2.2.2 Métodos de evaluación de arquitecturas software

Tal como se describe en la sección 9.1, en esta familia de experimentos se comparan los métodos ATAM y QuaDAI. En concreto nos centramos en las actividades de QuaDAI que tienen lugar después de obtener la arquitectura de producto: *Evaluación de la Arquitectura de Producto* y *Transformación de la Arquitectura de Producto*. Estas actividades están relacionadas con la evaluación y la mejora de arquitecturas de producto, que se alinean con el propósito principal de ATAM.

Antes de la ejecución del experimento, los diseñadores del experimento ejecutaron las tareas asociadas al rol de experto del dominio para el método QuaDAI, que incluye i) la selección de los patrones arquitectónicos para cada dominio, ii) la selección de los atributos de calidad y las métricas que miden cada atributo, iii) la ejecución del proceso de *tradeoff* entre los patrones arquitectónicos y atributos de calidad y iv) el almacenamiento de los resultados del *tradeoff* en el multimodelo.

Las actividades del método ATAM, seleccionadas para ser incluidas como tareas experimentales, fueron: el análisis de enfoques arquitectónicos de la fase de investigación y análisis, la priorización de escenarios y la segunda ejecución del análisis de enfoques arquitectónicos que tiene lugar en la fase de testeo (ver Figura 9.1).

Antes de la ejecución del experimento, s autores ejecutaron las tareas asociadas al rol de papel de arquitecto para la identificación de los enfoques arquitectónicos y el primer paso realizado por el rol de evaluador consistente en la generación del árbol de utilidad.

9.2.2.3 Selección de participantes

El contexto de esta familia de experimentos es la evaluación de la calidad de arquitecturas de software desde el punto de vista de los evaluadores de arquitectura software noveles.

Aunque los evaluadores de arquitecturas software con experiencia aumentan el valor de la evaluación (Clements et al. 2002), nos centramos en el perfil de los evaluadores principiantes, ya que nuestra intención es proporcionar un método que ayude en la realización de evaluaciones a evaluadores con menos experiencia, mediante la reutilización de conocimiento experto del dominio. Partiendo de estas premisas, se identificaron los siguientes grupos de participantes con el fin de facilitar la generalización de los resultados:

- Estudiantes de último curso de Ingeniería Informática de la Universitat Politècnica de València. Asistían a la asignatura de "Ingeniería del Software Avanzada" desde Septiembre de 2012 a Enero de 2013. Recibieron ocho horas lectivas sobre arquitecturas software y evaluaciones arquitectónicas.

- Estudiantes de máster, inscritos en el máster de Ingeniería del Software, Métodos Formales y Sistemas de Información Informática de la Universitat Politècnica de València. Asistían al curso de "Calidad de Sistemas de Información Web" desde Febrero de 2013 a Junio de 2013, curso especializado en control de calidad y que incluye más de ocho horas de contenidos teóricos acerca de arquitecturas software y evaluaciones arquitectónicas.

- Estudiantes de tercer curso del Grado en Informática de la Universitat Politècnica de València. Asistían a la asignatura de "Calidad del Software" desde Febrero de 2013 a Junio de 2013, curso especializado en control de calidad y que incluye más de ocho horas de contenidos teóricos acerca de arquitecturas software y evaluaciones arquitectónicas.

- Estudiantes de máster en la Universidad Nacional de Asunción (Paraguay). Asistían a la asignatura de "Calidad del Software" desde Julio de 2013 a Agosto de 2013, curso especializado en control de calidad y que incluye más de ocho horas de contenidos teóricos acerca de arquitecturas software y evaluaciones arquitectónicas. En este caso se trata de un máster profesional, y los estudiantes contaban con experiencia laboral previa, aunque ninguno de ellos había llevado a cabo evaluaciones de arquitecturas software con anterioridad.

- Estudiantes de Ingeniería Informática en la Università degli Studi della Basilicata (Italia). Asistían a la asignatura de "Ingeniería de Software" desde Marzo de 2013 a Junio de 2013. Uno de los principales temas de este curso es el modelado de sistemas orientados a objetos con UML. Los participantes también tienen experiencia en programación orientada a objetos y tecnología Web.

Se han seleccionado perfiles de estudiante de último año y estudiantes de máster ya que se ha demostrado que, en determinadas condiciones, no hay gran diferencia entre este tipo de estudiantes, considerados como la próxima generación de profesionales (Kitchenham et al. 2002), y los profesionales (Basili et al. 1999; Höst et al. 2000).

No se clasificó a los participantes en función de su experiencia, ya que ninguno de los participantes tenía formación o experiencia previa en la realización de evaluaciones arquitectónicas. Los estudiantes fueron asignados a los grupos experimentales al azar.

9.2.3 Paso 3: Tareas experimentales y material

Las tareas experimentales fueron estructuradas para permitir la comparación entre los métodos, partiendo de una arquitectura software a evaluar, un conjunto de requisitos no-funcionales (los cuales están documentados de forma diferente dependiendo del método a aplicar) y un conjunto de patrones. Cada tarea se descompone en las actividades del método que ayudan a lograr el propósito de la tarea. Después de aplicar cada método, los participantes rellenan un cuestionario post-experimento con las preguntas subjetivas acerca del método.

9.2.3.1 Tareas experimentales: ATAM

Las tareas experimentales llevadas a cabo por los participantes en la aplicación de ATAM incluyen dos procesos de medición, el análisis de los enfoques arquitectónicos (tanto de la fase de investigación y análisis como de la fase de pruebas) y las actividades de priorización de escenarios (ver Sección 9.1). La arquitectura del sistema, los objetivos de negocio, los enfoques arquitectónicos a considerar y el árbol de utilidad del sistema se proporcionan como entrada, fruto de las actividades previas realizadas por los autores (ver Sección 9.2.2.2). El experimento consiste en tres tareas experimentales que en el caso de ATAM se estructuran de la siguiente manera:

- La primera tarea experimental consiste en una primera medición de la arquitectura para comprobar el cumplimiento de los RNF descritos en el árbol de utilidad. Durante esta actividad, los participantes tuvieron también que examinar la documentación de las métricas a aplicar.
- La segunda tarea experimental consiste en tres actividades ATAM: i) el primer análisis de enfoques arquitectónicos, donde los participantes analizan cómo los enfoques arquitectónicos identificados soportan los escenarios y los atributos de calidad descritos en el árbol de utilidad; ii) la priorización de los escenarios, donde los participantes asignan prioridades a los escenarios del árbol de utilidad y; iii) el segundo análisis de los enfoques arquitectónicos, donde los participantes seleccionan el patrón arquitectónico a ser aplicado.
- Por último, la tercera tarea experimental consiste en la medición final de la arquitectura modificada después de la aplicación de patrones, con el fin de comprobar el cumplimiento de los RNF descritos en el árbol de utilidad.

9.2.3.2 Tareas experimentales: QuaDAI

Las tareas experimentales llevadas a cabo por los participantes en la aplicación de QuaDAI incluyen dos ejecuciones de la actividad de evaluación de la arquitectura de producto y la actividad de transformación de la arquitectura de producto. La arquitectura del sistema, los RNFs del sistema y el multimodelo que contiene los resultados del *tradeoff* entre transformaciones arquitectónicas y atributos de calidad se proporcionan como entrada, fruto de las actividades previas realizadas por los diseñadores del experimento (ver Sección 9.2.2.2). El experimento consiste en tres tareas experimentales, que en el caso de QuaDAI se estructuran de la siguiente manera:

- La primera tarea experimental consiste en una primera medición de la arquitectura para comprobar el cumplimiento de los RNF mediante la tarea de evaluación de la arquitectura de producto. Durante esta actividad, los participantes tuvieron también que examinar la documentación de los parámetros que han de aplicarse.
- La segunda tarea experimental consiste en la actividad transformación de la arquitectura de producto, donde los participantes tuvieron que decidir e introducir la importancia relativa de los atributos de calidad como pesos que van de 0 a 1. El archivo Excel que contiene el

multimodelo y simula el proceso de decisión retorna el patrón seleccionado en base a esa información.

- Por último, la tercera tarea experimental consiste en la medición final de la arquitectura modificada después de la aplicación de patrones mediante la tarea de evaluación de la arquitectura de producto, con el fin de comprobar el cumplimiento de los RNF.

9.2.3.3 Material Experimental

El material experimental se compone del conjunto de documentos necesarios para apoyar las tareas experimentales, las sesiones de entrenamiento y el cuestionario post-experimento.

Los materiales de entrenamiento incluyen: i) un conjunto de diapositivas que contienen la introducción a las arquitecturas software, patrones arquitectónicos, y a la evaluación de arquitecturas software; ii) un conjunto de diapositivas que describen el método QuaDAI, con un ejemplo de su aplicación que introduce también el uso de los archivos Excel; iii) un conjunto de diapositivas que describen el método ATAM, también con un ejemplo de su aplicación.

La documentación de las tareas experimentales incluye:

- Dos apéndices que contiene una explicación detallada de cada método de evaluación (QuaDAI y ATAM).
- Cuatro tipos de boletines que cubren las cuatro combinaciones posibles de métodos de evaluación y objetos experimentales (QuaDAI-O1, QuaDAI-O2, ATAM-O1, ATAM-O2). El propósito de estos boletines es: i) describir las tareas experimentales a realizar, ii) describir los sistemas, la arquitectura de cada sistema y los RNFs que se deben cumplir, y iii) recoger los datos de cada tarea experimental. El Apéndice B.1.1 contiene un extracto de la descripción de la arquitectura contenida en el boletín.
- Dos apéndices (O1 y O2) que contienen la descripción de los patrones arquitectónicos a ser aplicados. La descripción de cada patrón viene especificada empleando una plantilla en la que se muestra su nombre, el contexto en el que el patrón se puede aplicar, la descripción del problema a resolver, la estructura del patrón y las consecuencias en términos de ventajas y desventajas. El Apéndice B.1.3 contiene dos ejemplos de estos patrones arquitectónicos.

- Dos apéndices (O1 y O2) que contiene la descripción de las métricas que miden RNF del sistema (Tanto para ATAM y para QuaDAI). El Apéndice B.1.4 contiene un ejemplo de esas métricas.
- Dos apéndices (O1 y O2) que contiene la descripción de la arquitectura después de la aplicación de cada patrón (tanto para ATAM como para QuaDAI). El Apéndice B.1.5 contiene un ejemplo de la arquitectura resultante después de la aplicación de un patrón.
- Dos archivos de Excel (O1 y O2) que automatizan el cálculo de la aplicación de las diferentes métricas (tanto para ATAM como para QuaDAI). Debido a la complejidad de los cálculos necesarios, se automatizó parcialmente el cálculo de las métricas. Los participantes tienen que rellenar los datos requeridos por cada métrica (que se proporcionan en el boletín) y obtienen el resultado final a evaluar.
- Dos archivos de Excel (O1 y O2), que incluyen la selección entre alternativas basada en la importancia relativa de cada atributo de calidad introducida por el sujeto, para ser utilizado durante la aplicación del método QuaDAI.

El cuestionario post-experimento contiene un conjunto de preguntas cerradas con la que los participantes pueden expresar su opinión respecto a la facilidad de uso, utilidad e intención de emplear ese método en el futuro. Las preguntas cerradas incluidas en el cuestionario se pueden encontrar en el Apéndice B.2. El orden de las preguntas fue alterado de forma aleatoria con el fin de evitar el sesgo de respuesta sistemática y se reformularon algunas preguntas para convertirlas en enunciados negativos en la parte izquierda con el fin de evitar así las respuestas monótonas (Hu y Chau 1999). Se incluyeron también dos preguntas abiertas con el fin de recabar la opinión de los participantes acerca de los cambios a realizar para mejorar los métodos y cuáles serían las razones que les moverían a usar el método en el futuro.

Todos los documentos fueron creados inicialmente en español, ya que este era el idioma nativo de los participantes en los cuatro primeros experimentos, y posteriormente traducido al inglés. Todo el material (incluyendo las tareas experimentales y las diapositivas de entrenamiento) está disponible para su descarga en http://www. dsic.upv.es/~jagonzalez/tesis/instrumentacion.html.

9.2.4 Paso 4: Experimentos Individuales

La Figura 9.2 resume la familia de experimentos, en la que se muestran en cada rectángulo los detalles de cada experimento.

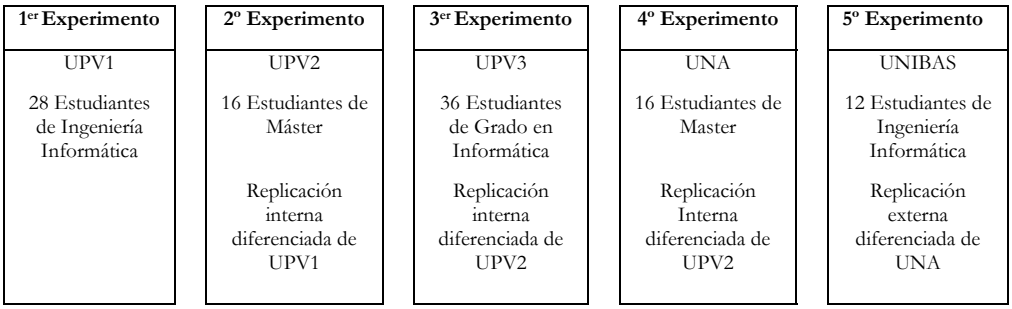


Figura 9.2 Resumen de la familia de experimentos

El experimento original (UPV1) se replicó con el fin de obtener mayores evidencias a los resultados obtenidos en el experimento original y para la verificación de las cuestiones pendientes. El segundo, tercer y cuarto experimentos (UPV2, UPV3 y UNA) fueron repeticiones internas (llevadas a cabo por los autores del método). El segundo experimento (UPV2) es una réplica diferenciada del experimento original, mientras que el tercer y cuarto experimento (UPV3 y UNA) son replicaciones diferenciadas del segundo experimento (UPV2) realizadas en diferentes entornos. El segundo y tercer experimento tuvieron lugar en la Universitat Politècnica de València, mientras que el cuarto tuvo lugar en la Universidad Nacional de Asunción en Paraguay.

El quinto experimento (UNIBAS) era una réplica externa (llevada a cabo por un investigador ajeno a la definición del método) del cuarto experimento (UNA) realizado en la Università degli Studi della Basilicata en Italia a fin de verificar los resultados obtenidos en los tres experimentos anteriores y para evitar el posible sesgo de autoría que pueden estar presentes en los estudios anteriores.

9.2.5 Paso 5: Análisis de los datos de la familia de experimentos y meta-análisis

Los resultados de cada uno de los experimentos individuales fueron recogidos utilizando los boletines y el cuestionario post-experimento y posteriormente analizadas. También se llevó a cabo un meta-análisis con el fin de agregar y llevar a cabo un análisis consolidado de los resultados, ya que las condiciones experimentales fueron muy similares en cada uno de los experimentos. Este análisis (que se detalla en la sección 9.5.2), nos ha permitido obtener resultados más sólidos y extraer conclusiones más generales de cada experimento individual.

9.3 Diseño de los experimentos individuales

En esta sección se van a describir las características de cada uno de los experimentos que forman parte de la familia de experimentos.

9.3.1 Experimento original (UPV1)

En esta subsección se describen los detalles relativos al experimento original. Para evitar redundancias, solo se discutirán algunas aclaraciones del experimento original en relación a la información presentada en la sección 9.2.

9.3.1.1 Planificación

Contexto del experimento: se emplearon los objetos experimentales (O1 y O2) descritos en la sección 9.2.3.3 y se aplicaron los métodos de evaluación arquitectónica descritos en la sección 9.1. En este experimento se seleccionaron 31 estudiantes de Ingeniería Informática como participantes. Todos ellos asistían al curso "Tecnología de Software Avanzada", cuyo perfil se describe en la sección 9.2.2.3.

Selección de variables: Las **variables independientes** en esta familia de experimentos son el uso de cada uno de los métodos de evaluación arquitectónica, con dos valores nominales: ATAM y QuaDAI.

Se han considerado dos **variables dependientes objetivas:**

- *Eficacia* del método, que se calcula como función de la Distancia Euclídea entre los valores de los RNF obtenidos por la arquitectura

evaluada por el sujeto y el valor óptimo de los RNF que se puede alcanzar con los patrones arquitectónicos propuestos.

- *Eficiencia*, que se calcula como el ratio entre la eficacia y el tiempo total invertido al aplicar cada método.

La eficiencia se calcula aplicando la formula (1) a la distancia euclídea normalizada. La normalización de la distancia euclídea se calcula aplicando la formula (2) a la distancia euclídea obtenida aplicando la formula (3) y retorna un valor entre 0 y 1. Es necesario normalizar la distancia euclídea para evitar el efecto de las escalas de las métricas que evalúan cada uno de los RNFs. La función Optimo que aparecen en las formulas (1) y (2) retorna los valores óptimo para cada uno de los RNFs para un objeto experimental dado. La función Max que aparece en la formula (2) retorna la máxima distancia D observada para un determinado objeto experimental.

$$Eficiencia\ (p) = 1 - Norm\left(D\left(p, Optimo(Objeto)\right)\right) \tag{1}$$

$$Norm\left(D\left(p, Optimo(Objeto)\right)\right) = \frac{D\left(p, Optimo(Objeto)\right)}{Max(Objeto)} \tag{2}$$

$$D(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{3}$$

En el estudio también se han considerado tres **variables dependientes subjetivas** basadas en el modelo de aceptación tecnológica (*Technology Acceptance Model TAM)* (Davis 1989), que es uno de los modelos teóricos para el análisis de la aceptación y del comportamiento de uso más ampliamente aceptados, y además cuenta con soporte empírico en múltiples validaciones y replicaciones (Venkatesh 2000). En concreto las variables dependientes subjetivas son:

- *Facilidad de Uso Percibida (FUP),* se refiere al grado en el que los evaluadores creen que el aprendizaje de un método en particular podrá llevarse a cabo sin esfuerzo.

- *Utilidad Percibida (UP),* se refiere al grado en que los evaluadores consideran que el uso de un método especifico aumentará su rendimiento dentro de un contexto organizacional.
- *Intención de Uso (IU)*, se refiere a la medida en que un determinado evaluador tiene la intención de utilizar en el futuro un método particular. Esta última variable representa un juicio perceptual de la eficacia del método, es decir, si realmente es rentable. Esta variable es comúnmente empleada para predecir la probabilidad de aceptación de un determinado método en la práctica.

Estas tres variables subjetivas se midieron mediante un cuestionario Likert con un conjunto de 13 preguntas cerradas: 3 preguntas para la facilidad de uso percibida, 6 preguntas para la utilidad percibida y 4 para la intención de uso. Las respuestas fueron codificadas empleando una escala Likert de 5 puntos, con el formato de preguntas con respuestas opuestas. Cada pregunta contiene dos estados opuestos que representan los valores máximo y mínimo (5 y 1), considerándose el valor 3 como la percepción neutra. El participante selecciona el valor en la escala que más se adecua a su percepción u opinión. El valor agregado de cada una de las variables se calcula como la media aritmética de las respuestas a las preguntas asociadas a cada una de las variables subjetivas.

Formulación de hipótesis: se han formulado cinco hipótesis nulas, enunciadas como unilaterales, dado que esperamos constatar que el comportamiento de QuaDAI sea superior al de ATAM en cada una de las variables dependientes estudiadas. El enunciado de las hipótesis nulas y sus correspondientes hipótesis alternativas es el siguiente:

- **$H1_0$:** No hay diferencias significativas entre la eficacia de QuaDAI y ATAM.
- **$H1_a$:** QuaDAI es significativamente más eficaz que ATAM.
- **$H2_0$:** No hay diferencia significativa entre la eficiencia de QuaDAI y ATAM.
- **$H2_a$:** QuaDAI es significativamente más eficiente que ATAM.
- **$H3_0$:** No hay diferencia significativa entre la facilidad de uso percibida por los evaluadores que aplican QuaDAI y los que aplican ATAM.
- **$H3_a$:** QuaDAI se percibe como más fácil de usar que ATAM.
- **$H4_0$:** No hay diferencia significativa entre la utilidad percibida de QuaDAI y ATAM.
- **$H4_a$:** QuaDAI se percibe como más útil que ATAM.

- **$H5_0$:** No hay diferencia significativa entre la intención de uso de QuaDAI y ATAM.
- **$H5_a$:** los participantes perciben como más probable utilizar QuaDAI en el futuro que ATAM.

Diseño Experimental: El experimento se planificó como un diseño equilibrado *entre-participantes* con un *efecto de confusión*, lo que significa que los participantes aplican ambos métodos con ambos objetos experimentales en diferente orden. Se seleccionó este diseño dado que permite bloquear el efecto aprendizaje sobre los resultados. Se establecieron cuatro grupos (cada grupo aplicaba un método distinto sobre un objeto experimental distinto). Los participantes fueron asignados a los grupos al azar.

La Tabla 9.2 muestra el diseño experimental utilizado en los experimentos. El diseño experimental entre-participantes pretende minimizar el impacto del efecto de aprendizaje sobre los resultados, ya que ningún participante repite tratamiento (método) ni objeto experimental durante la ejecución del experimento. También es necesario controlar otros factores que podrían tener efecto sobre los resultados, como puede ser la complejidad de los objetos experimentales. La compresión de las arquitecturas software a evaluar, de los RNFs, de las métricas que evalúan los RNFs y los patrones arquitectónicos podría afectar al resultado final. Para mitigar la influencia de este factor, se seleccionaron dos sistemas software representativos con RNFs, métricas y patrones arquitectónicos con una complejidad razonable y similar para ambos objetos experimentales. Además, la complejidad de los patrones y métricas seleccionados era adecuada para que se aplicasen en la franja de tiempo disponible para la ejecución de los experimentos.

Tabla 9.2 Diseño experimental

	Grupos (tamaño de la muestra = 4n participantes)			
	G1 (n participantes)	***G2 (n participantes)***	***G3 (n participantes)***	***G4 (n participantes)***
1ª sesión	Aplica ATAM en O1	Aplica ATAM en O2	Aplica QuaDAI en O1	Aplica QuaDAI en O2
2ª sesión	Aplica QuaDAI en O2	Aplica QuaDAI en O1	Aplica ATAM en O2	Aplica ATAM en O1

Instrumentación: se utilizaron los documentos presentados en la sección 9.2.3.3 para soportar las tareas experimentales (4 documentos de captura de datos, 4 apéndices, 1 cuestionario y 3 archivos Excel) y el material de entrenamiento (3 conjuntos de diapositivas).

9.3.1.2 Preparación y ejecución

Esta sección describe la operación experimental que incluye la preparación, la ejecución, el registro y la validación de datos.

Con respecto a la preparación, el experimento se planificó para ser llevado a cabo en tres sesiones (la Tabla 9.3 muestra los detalles de cada sesión). En la primera sesión los participantes recibieron un entrenamiento completo de los métodos de evaluación que se aplicarían durante el experimento, así como de las tareas a realizar en su ejecución. En las dos sesiones siguientes los participantes recibieron una versión reducida del entrenamiento con una visión general de los métodos a aplicar y las tareas que habían de ejecutar. Se estableció una franja de 90 minutos para la aplicación de cada uno de los métodos, aunque se permitió que los participantes continuasen con el experimento aun cuando pasasen de esos 90 minutos, con el fin de eliminar el posible *efecto techo* (Sjøberg et al. 2003).

Tabla 9.3 Planificación del primer experimento

1ª Sesión (120 Minutos)	Entrenamiento: Evaluación de arquitecturas software usando ATAM y QuaDAI			
2ª Sesión (120 Minutos)	Entrenamiento corto: Evaluación de arquitecturas software usando ATAM y QuaDAI			
	ATAM en O1	ATAM en O2	QuaDAI en O1	QuaDAI en O2
	Cuestionario ATAM		Cuestionario QuaDAI	
3 Sesión (120 Minutos)	Entrenamiento corto: Evaluación de arquitecturas software usando ATAM y QuaDAI			
	QuaDAI en O2	QuaDAI en O1	ATAM en O2	ATAM en O1
	Cuestionario QuaDAI		Cuestionario ATAM	

Con respecto a la ejecución, el experimento tuvo lugar en una única habitación, y no se permitía la interacción entre los participantes. Las preguntas que surgieron durante la sesión fueron aclaradas individualmente por los propios instructores durante la sesión.

Con respecto a la validación de los datos, se verificó que uno de los participantes no había completado la segunda sesión y por lo tanto fue necesario eliminar los datos de este participante del análisis. Dado que se contaba finalmente con un total de 30 participantes, fue necesario descartar los datos de dos participantes (seleccionados al azar) con el fin de mantener el diseño equilibrado tal como se muestra en la Tabla 9.2, con un total de 28 participantes distribuidos en cuatro grupos de 7 participantes cada uno.

9.3.2 Segundo experimento (UPV2)

Este experimento (primera replicación) difiere del experimento original en tres aspectos:

- Selección de participantes: Se seleccionaron 19 estudiantes de máster como participantes, todos ellos asistían al curso "Calidad de Sistemas de Información Web", cuyo perfil se describe en la Sección 9.2.2.3.
- Nivel de un RNF: Se modificó el nivel de uno de los RNF en el objeto experimental O2, ya que era más fácil encontrar la mejor solución en el caso del objeto O2 (en el estudio original todos los participantes seleccionaron el mejor patrón cuando evaluaron la arquitectura del objeto O2) en comparación al objeto O1 experimental (tan sólo el 71% de los participantes seleccionó el mejor patrón, independientemente del método).
- Preguntas de control: Se incluyó un conjunto de preguntas de control al material experimental con el fin de analizar la comprensión de los patrones y las métricas que se estaban aplicando. Estas preguntas ayudaron a los participantes a centrarse en la comprensión de los patrones y las métricas, y nos permiten evaluar el grado en que los participantes comprenden el problema. Estas preguntas no influyen en la ejecución del experimento ni en sus resultados, su propósito es ayudarnos a controlar la comprensión de los patrones y las métricas.

Con respecto a la preparación del experimento, se planeó para ser llevado a cabo siguiendo el esquema mostrado en la Tabla 9.3. Al igual que en el experimento original, también se llevó a cabo en una única estancia y no se permitía la interacción entre participantes. Con respecto a la validación de datos, con el fin de mantener el diseño equilibrado, se tuvieron que descartar los datos de tres participantes (seleccionados al azar), contando finalmente con 16 participantes distribuidos en cuatro grupos de 4 participantes en cada grupo.

9.3.3 Tercer experimento (UPV3)

El tercer experimento (segunda replicación) es una replicación del segundo experimento UPV2. La principal diferencia con respecto a UPV2 fue la selección de participantes. Se seleccionaron 40 estudiantes de grado en informática como participantes, todos ellos asistían al curso “Calidad del software”, cuyo perfil se describe en la sección 9.2.2.3.

Con respecto a la preparación y ejecución del experimento, no hubo diferencias con respecto a UPV2, dado que se siguió la misma planificación en las tres sesiones. Con respecto a la validación de los datos, se verificó que tres participantes no completaron la segunda sesión y por lo tanto, fue necesario eliminar su primer ejercicio. Dado que se disponía de 37 participantes distribuidos en cuatro grupos, fue necesario descartar un participante adicional (seleccionado al azar), contando finalmente con 36 participantes distribuidos en cuatro grupos de 9 participantes en cada grupo.

9.3.4 Cuarto experimento (UNA)

El cuarto experimentos (tercera replicación) es una replicación del segundo experimento UPV2. La principal diferencia con respecto a UPV2 fue la selección de participantes. Se seleccionaron 22 estudiantes de máster en informática como participantes, todos ellos asistían al curso "Calidad del Software", y cuyo perfil se describe en la sección 9.2.2.3.

Respecto a la preparación y ejecución del experimento, se siguió el mismo esquema que en los experimentos UPV2 y UPV3. Con respecto a la validación de los datos, se verificó que un sujeto había realizado las dos sesiones aplicando el mismo método, violando el diseño entre-sujetos, que un sujeto no anotó las horas de inicio y fin de las tareas, por lo que no se podía calcular la eficiencia y que otros dos sujetos no habían rellenado parte del cuestionario en uno de los ejercicios y por lo tanto fue necesario no considerar sus ejercicios. Dado que se disponía de 18 participantes distribuidos en cuatro grupos, fue necesario descartar dos participantes adicionales (seleccionado al azar), contando finalmente con 16 participantes distribuidos en cuatro grupos de 4 participantes en cada grupo.

9.3.5 Quinto experimento (UNIBAS)

El quinto experimento (cuarta replicación) es una réplica externa del cuarto experimento (UNA). Este experimento se diferencia de los tres experimentos previos en tres aspectos fundamentales:

- *Selección de participantes:* Se seleccionaron 12 estudiantes de tercer curso de Ingeniería Informática como participantes, todos ellos asistían al curso "Ingeniería del Software", cuyo perfil se describe en la sección 9.2.2.3.

- *Tareas experimentales:* también se incluyó un tercer RNF y una nueva métrica, con el fin de hacer la solución del *tradeoff* más compleja y menos evidente. El nuevo RNF también fue incluido en el cálculo de la eficacia siguiendo las expresiones descritas en la Sección 9.3.1.1.
- *Material experimental:* El material fue traducido del castellano al inglés, que fue el idioma en el que tuvo lugar esta replicación. Esto podría suponer una amenaza a la validez externa y de constructo, puesto que el inglés no es la lengua materna de los participantes. En este caso, todos los estudiantes de esta universidad han de superar un examen de inglés antes de matricularse en el segundo curso del programa de licenciatura, por lo que se mitiga parcialmente las amenazas a la validez, dado que el nivel de inglés de los participantes es prácticamente homogéneo. Además, por primera vez los participantes utilizaron la versión electrónica del material en lugar de las versiones impresas de los documentos y el cuestionario post-experimento se llevó a cabo utilizando un soporte online para la realización de encuestas.

En cuanto a la preparación y ejecución, hubo pequeñas diferencias con respecto a los experimentos previos. Pocos días antes de las sesiones experimentales, los participantes asistieron a tres sesiones de entrenamiento (180 minutos en total). En la primera sesión de 60 minutos, se presentaron los conceptos de evaluación de arquitecturas software. Los métodos ATAM y QuaDAI fueron introducidas en las dos últimas sesiones de entrenamiento con una duración aproximada de 60 minutos cada una. El esquema experimental utilizado fue el mismo que de los otros experimentos (ver Tabla 9.3).

9.3.6 Documentación y comunicación

Cuestiones como la documentación (Shull et al. 2004) y la comunicación entre el personal que lleva a cabo los experimentos (Vegas et al. 2006), pueden tener una influencia crucial en el éxito de la replicación de experimentos. Las deficiencias en la documentación y en los paquetes experimentales son una de las mayores fuentes de problemas en la disciplina que dificultan el empleo de las replicaciones. Como posible solución, los diseñadores del experimento proponen mejorar los paquetes de laboratorio y el uso de herramientas de intercambio de información.

Con respecto a la documentación, los diseñadores del experimento de los tres primeros experimentos tradujeron todo el material al inglés (inicialmente redactados en castellano). El material traducido incluía el cuestionario post-

experimento, los anexos que documentan las métricas, los patrones, las arquitecturas resultantes y las hojas de cálculo Excel que automatizan la aplicación de las métricas. También se incluyó un conjunto adicional de diapositivas al material de entrenamiento, en el que se explica el procedimiento de ejecución del experimento.

En el documento, se discutían las razones que justificaban las decisiones de diseño adoptadas en el experimento original, destacando toda la información útil para reproducir las condiciones experimentales. Al investigador involucrado en las replicaciones externas también se le proporcionaron las publicaciones anteriores sobre el experimento original y la primera réplica.

Los grupos de investigadores que ejecutaron los experimentos intercambiaron el material de entrenamiento para reproducir el mismo escenario experimental utilizado en los experimentos UPV1, UPV2, UPV3 y UNA. Aunque la documentación es un factor clave para ser capaz de llevar a cabo una replicación, la comunicación entre los experimentadores es aún más importante (Vegas et al. 2006).

La interacción entre los grupos de investigadores que ejecutaron los experimentos se llevó a cabo principalmente por correo electrónico y en ocasiones también se utilizaron herramientas de mensajería instantánea. El intercambio de documentos se llevó a cabo mediante el uso de herramientas de compartición de archivos en la nube.

9.4 Análisis de los resultados

En esta sección se discuten los resultados de cada uno de los experimentos individuales, mediante el análisis cuantitativo de acuerdo con las hipótesis planteadas.

Todos los test estadísticos que se presentan en esta sección fueron llevados a cabo utilizando SPSS v20 con un nivel de significancia estadística $\alpha = 0.05$. Para el análisis se utilizaron estadísticos descriptivos, diagramas de cajas, diagramas de densidad y pruebas estadísticas con el fin de analizar los datos obtenidos de cada experimento individual.

En particular, dado que el tamaño de muestra era menor de 50, ha sido necesaria la aplicación de la prueba de Shapiro-Wilk para comprobar si los datos se distribuyen según una distribución normal. El resultado de la prueba de Shapiro-Wilk nos permite seleccionar las pruebas a aplicar en cada caso con el fin de verificar las hipótesis. Por un lado, cuando los datos se ajustan a la

distribución normal ($p \geq 0{,}05$) se aplicó el test *t* paramétrico unilateral para muestras independientes (Juristo y Moreno 2001). Por el contrario, en el caso de variables que no se ajustan a una distribución normal, se aplicó la prueba no paramétrica de Mann-Whitney (Conover 1998). Las variables subjetivas se analizaron por separado para cada método, comparando si la media de respuestas a las preguntas relacionadas con una variable era significativamente mayor que el valor Likert neutro. En nuestro caso, las escalas ordinales variaron de 0 a 5 y el valor neutro corresponde al valor 3.

Se seleccionó este conjunto de pruebas ya que se tratan de un conjunto de tests estadísticos robustos, sensibles, ampliamente aceptados y aplicados en el ámbito de la comunidad de ingeniería del software (Maxwell 2002).

Como es habitual, en todas las pruebas decidimos aceptar una probabilidad del 5% de cometer un *error tipo I* (rechazar la hipótesis nula cuando realmente es cierta) (Wohlin et al. 2012).

La Tabla 9.4 muestra un resumen de los resultados globales de las evaluaciones arquitectónicas realizadas en cada experimento individual.

Tabla 9.4 Resumen de los resultados de los experimentos individuales

		UPV1 (N=56)		*UPV2 (N=32)*		*UPV3 (N=72)*		*UNA (N=32)*		*UNIBAS (N=24)*	
		Media	*Desv. Est.*	*Media*	*Desv. Est.*	*Media*	*Desv. Est.*	*Media*	*Desv. Est.*	*Media*	*Desv. Est.*
Eficacia	***QuaDAI***	**0,684**	0,361	**0,797**	0,209	**0,708**	0,351	**0,848**	0,202	**0,631**	0,277
	ATAM	0,626	0,388	0,461	0,448	0,462	0,404	0,586	0,441	0,297	0,285
Eficiencia	***QuaDAI***	**0,029**	0,013	**0,023**	0,007	**0,025**	0,015	**0,022**	0,014	**0,011**	0,006
	ATAM	0,019	0,018	0,014	0,015	0,015	0,014	0,015	0,009	0,006	0,007
Duración	***QuaDAI***	**25,360**	7,258	**34,630**	7,974	**30,110**	8,671	**45,060**	17,571	**69,830**	23,486
	ATAM	31,110	9,154	39,190	11,356	33,030	0,032	53,310	31,455	73,170	29,735
Facilidad de uso percibida	***QuaDAI***	**3,980**	0,876	**4,312**	0,714	**3,590**	0,659	**3,813**	0,843	**2,667**	1,054
	ATAM	3,500	0,816	4,020	0,714	3,590	0,658	3,479	1,040	2,444	0,729
Utilidad percibida	***QuaDAI***	**3,804**	0,832	**4,167**	0,860	**4,070**	0,498	**4,427**	0,490	**3,569**	0,329
	ATAM	3,723	0,730	3,927	0,672	3,850	0,528	3,761	0,852	3,458	0,342
Intención de uso	***QuaDAI***	**3,654**	0,698	**4,063**	0,710	**3,935**	0,631	**4,210**	0,529	**3,458**	0,673
	ATAM	3,547	0,844	3,906	0,831	3,740	0,638	3,520	1,018	3,417	0,685

Se han utilizado la media y la desviaciones estándar como estadísticos descriptivos para las variables subjetivas cualitativas FUP, UP y IU. La media aritmética de las escala Likert de cada una de las respuestas al cuestionario adoptada para la medición de las variables subjetivas, ha sido también considerada como una escala de intervalo (Carifio y Perla 2007). Las celdas resaltadas en negrita, muestran los mejores valores para cada una de los estadísticos. Los resultados globales nos han permitido concluir que QuaDAI

ha logrado el mejor rendimiento de los participantes en prácticamente la totalidad de las estadísticas analizadas.

9.4.1 Eficacia

La Figura 9.3 muestra los diagramas de caja de la variable *eficacia*, por participante y método por cada experimento de la familia. Esos diagramas muestran que, bajo las condiciones de los experimentos, QuaDAI es relativamente más eficaz que ATAM en la evaluación de arquitecturas software.

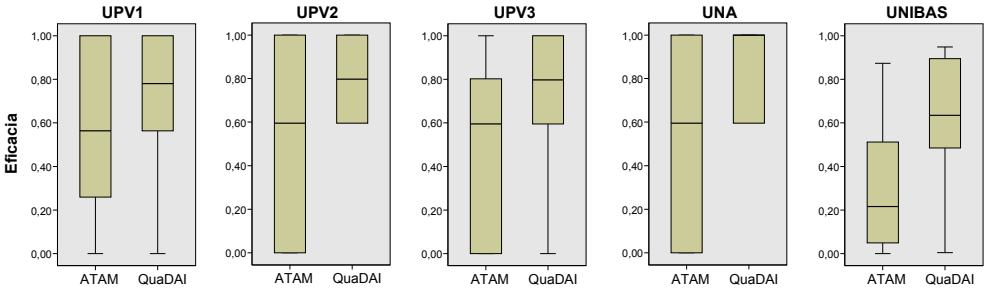


Figura 9.3 Diagramas de caja de la variable eficacia

Para comprobar la significancia estadística de estos resultados, se llevó a cabo el test no paramétrico de Mann-Whitney para verificar la hipótesis H1 dado que para los casos UPV1, UPV2, UPV3 y UNA los resultados del test de Shapiro-Wilk evidencian que, en esos casos, los datos no se ajustan a una distribución normal, tal como muestran las celdas en negrita en la Tabla 9.5 ($p < 0.05$). En el caso de UNIBAS se aplicó el test *t* unilateral dado que los datos se ajustan a una distribución normal.

Tabla 9.5 Resultados de las pruebas de normalidad de la variable eficacia

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
p de Shapiro-Wilk	*QuaDAI*	**0.000**	**0.000**	**0.000**	**0.000**	0.126
	ATAM	**0.000**	**0.001**	**0.000**	**0.001**	0.117

La Tabla 9.6 muestra el resultado de estas pruebas para cada uno de los experimentos. Estos resultados nos permiten rechazar la hipótesis nula $H1_0$ y aceptar su hipótesis alternativa en los experimentos UPV2, UPV3, UNA y UNIBAS, lo que significa que la eficacia de los participantes al aplicar QuaDAI es significativamente mayor que la eficacia de los participantes al aplicar ATAM.

Tabla 9.6 Resultado de las pruebas de significancia de la hipótesis H1

Experimento	*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
Significancia	0.906*	**0.036***	**0.003***	**0.043***	**0.008**

* Resultados del test no-paramétrico de Mann-Whitney

9.4.2 Eficiencia

La Figura 9.4 muestra los diagramas de caja para la variable *eficiencia* por participante y método para cada experimento de la familia. Esos diagramas muestran que, bajo las condiciones de estos experimentos, QuaDAI es relativamente más eficiente que ATAM en la evaluación de las arquitecturas software.

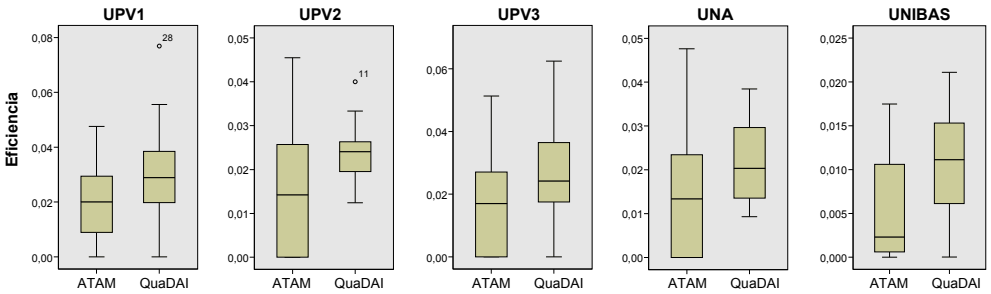


Figura 9.4 Diagramas de caja de la variable eficiencia

Para comprobar la significancia estadística de estos resultados, se llevó a cabo en primer lugar el test no paramétrico de Mann-Whitney para verificar la hipótesis H2 dado que para los casos UPV2, UPV3 y UNIBAS, los resultados del test de Shapiro-Wilk evidencian que los datos en esos experimentos no se ajustan a una distribución normal tal como muestran las celdas en negrita en la Tabla 9.7 ($p < 0.05$). En el caso de UPV1 y UNA se aplicó el test *t* unilateral dado que los datos se ajustan a una distribución normal para ambos métodos.

Tabla 9.7 Resultados de las pruebas de normalidad de la variable eficiencia

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
p de Shapiro-Wilk	*QuaDAI*	0.379	**0.016**	**0.000**	0.249	0.988
	ATAM	0.362	0.487	0.267	0.071	**0.001**

La Tabla 9.8 muestra el resultado de estas pruebas para cada uno de los experimentos. Estos resultados nos permiten rechazar la hipótesis nula $H2_0$ y

aceptar su hipótesis alternativa en los experimentos UPV1, UPV2, UPV3 y UNIBAS (significancia < 0.05), lo que significa que la eficiencia de los participantes al aplicar QuaDAI es significativamente mayor que la eficacia de los participantes al aplicar ATAM.

Tabla 9.8 Resultado de las pruebas de significancia de la hipótesis H2

Experimento	*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
Significancia	**0.030**	**0.043***	**0.000***	0.164	**0.045***

* Resultados del test no-paramétrico de Mann-Whitney

9.4.3 Facilidad de uso percibida

La Figura 9.5 muestra los diagramas de densidad para la variable *facilidad de uso percibida* por método para cada uno de los experimentos. La media de cada población se muestra a través de dos líneas verticales en cada diagrama de densidad. De estos diagramas se puede extraer que en todos los casos que QuaDAI fue percibido por los participantes como más fácil de usar respecto a ATAM, excepto en el caso del experimento UPV3 en que ambos métodos fueron percibidos como igualmente fáciles de usar.

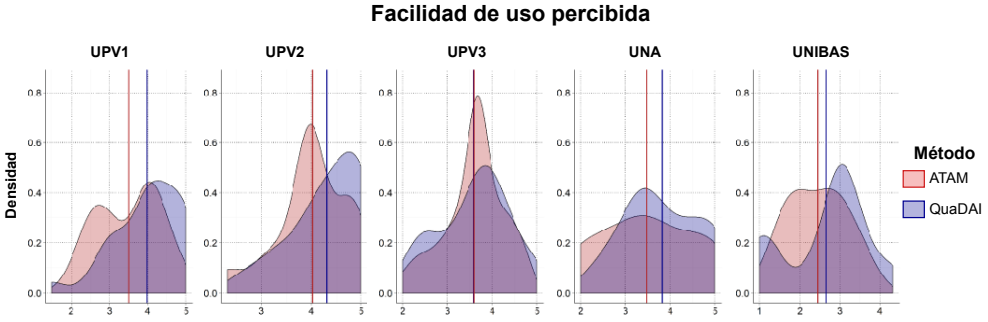


Figura 9.5 Diagramas de densidad de la variable facilidad de uso percibida

Para comprobar la significancia estadística de estos resultados, se realizó el test de Wilcoxon para una muestra contra el valor de prueba v = 3 para cada método por separado con el fin de verificar la hipótesis H3 para los casos UPV1/QuaDAI, UPV1/ATAM, UPV2/QuaDAI y UPV3/ATAM dado que los resultados del test de Shapiro-Wilk evidencian que esas variable no se ajustaba a una distribución normal, tal como muestran las celdas en negrita en la Tabla 9.9 ($p < 0.05$). Por el contrario, se aplicó el test *t* unilateral para una muestra con un valor de prueba v=3 con el fin de verificar H3 para los casos

UPV2/ATAM, UPV3/QuaDAI, UNA/QuaDAI, UNA/ATAM, UNIBAS/QuaDAI y UNIBAS/ATAM dado que se ajustaban a una distribución normal.

Tabla 9.9 Resultados de las pruebas de normalidad de la variable facilidad de uso percibida

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
p de Shapiro-Wilk	*QuaDAI*	**0.014**	**0.026**	0,159	0.335	0.098
	ATAM	**0.024**	0.097	**0,046**	0.199	0.837

La Tabla 9.10 muestra el resultado de las pruebas anteriormente descritas para cada uno de los experimentos y métodos. Por lo tanto, estos resultados apoyan el rechazo de la hipótesis nula $H3_0$ en UPV1, UPV2, UPV3 y UNA (p-valor <0,05) y la aceptación de la hipótesis alternativa, lo que significa que QuaDAI es percibido por los participantes como más fácil de usar que ATAM.

Tabla 9.10 Resultado de las pruebas de significancia de la hipótesis H3

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
Significancia	*QuaDAI*	**0.000***	**0.001***	**0,000**	**0.002**	0.297
	ATAM	**0.003***	**0.000**	**0,000***	**0.043**	**0.023**

* Resultados del test de Wilcoxon para una muestra contra el valor de prueba v=3

9.4.4 Utilidad percibida

La Figura 9.6 muestra los diagramas de densidad para la variable utilidad percibida por método para cada uno de los experimentos. La media de cada población se muestra a través de dos líneas verticales en cada diagrama de densidad. De estos diagramas se puede extraer que en todos los casos QuaDAI fue percibido por los participantes como más útil que ATAM, en el contexto de los experimentos.

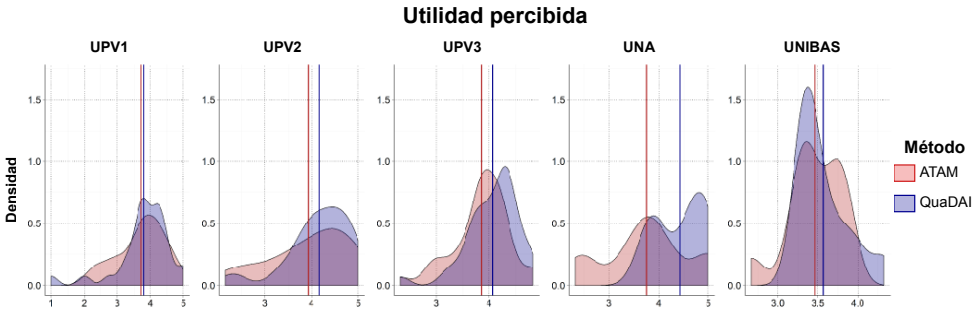


Figura 9.6 Diagramas de densidad de la variable utilidad percibida

Para comprobar la significancia estadística de estos resultados, se realizó el test de Wilcoxon para una muestra contra el valor de prueba v = 3 para cada método por separado con el fin de verificar la hipótesis H4 para los casos UPV1/QuaDAI, UPV2/QuaDAI y UPV3/QuaDAI, UNA/QuaDAI y UNIBAS/QuaDAI dado que los resultados del test de Shapiro-Wilk evidencian que esas variables no se ajustaban a una distribución normal, tal como muestran las celdas en negrita en la Tabla 9.11 ($p < 0.05$). Por el contrario, se aplicó el test *t* unilateral para una muestra con un valor de prueba v=3 con el fin de verificar H3 para los casos UPV1/ATAM, UPV2/ATAM y UPV3/ATAM, UNA/ATAM y UNIBAS/ATAM dado que se ajustaban a una distribución normal.

Tabla 9.11 Resultados de las pruebas de normalidad de la variable utilidad percibida

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
p de Shapiro-Wilk	*QuaDAI*	**0.001**	**0.069**	**0.008**	**0.028**	0.005
	ATAM	0.204	0.169	0.053	0.252	0.103

La Tabla 9.12 muestra el resultado de las pruebas anteriormente descritas para cada uno de los experimentos y métodos. Por lo tanto, estos resultados apoyan el rechazo de la hipótesis nula $H4_0$ en todos los casos (p-valor < 0,05) y la aceptación de la hipótesis alternativa, lo que significa que QuaDAI es percibido por los participantes como más útil que ATAM, en el contexto del experimento.

Tabla 9.12 Resultado de las pruebas de significancia de la hipótesis H4

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
Significancia	*QuaDAI*	**0.000***	**0.001***	**0,001***	**0.000***	**0.002***
	ATAM	**0.000**	**0.001**	**0,001**	**0.003**	**0.001**

* Resultados del test de Wilcoxon para una muestra contra el valor de prueba v=3

9.4.5 Intención de uso

La Figura 9.7 muestra los diagramas de densidad para la variable intención de uso de cada método para cada uno de los experimentos. La media de cada población se muestra a través de dos líneas verticales en cada diagrama de densidad. De estos diagramas se puede extraer que en todos los casos los participantes percibían más probable utilizar en el futuro QuaDAI que ATAM.

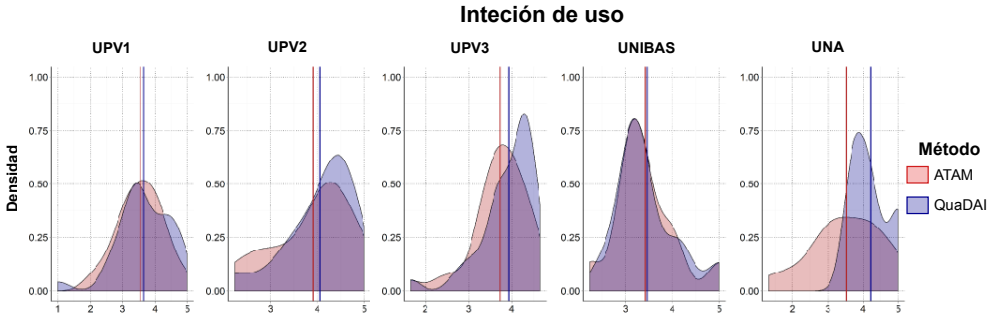


Figura 9.7 Diagramas de densidad de la variable intención de uso

Para comprobar la significancia estadística de estos resultados, se realizó el test de Wilcoxon para una muestra contra el valor de prueba v = 3 para cada método por separado con el fin de verificar la hipótesis H5 para los casos UPV1/QuaDAI, UPV2/QuaDAI y UPV3/QuaDAI, UNA/QuaDAI, UNIBAS/QuaDAI y UNIBAS/ATAM dado que los resultados del test de Shapiro-Wilk evidencian que esas variable no se ajustaba a una distribución normal, tal como muestran las celdas en negrita en la Tabla 9.13 ($p < 0.05$). Para el resto de casos (UPV1/ATAM, UPV2/ATAM, UPV3/ATAM y UNA/ATAM) se aplicó el test *t* unilateral para una muestra con un valor de prueba v = 3 con el fin de verificar H3 dado que los datos se ajustaban a una distribución normal.

Tabla 9.13 Resultados de las pruebas de normalidad de la variable intención de uso

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
p de Shapiro-Wilk	*QuaDAI*	**0.024**	**0.022**	**0.000**	**0.005**	**0.000**
	ATAM	0.894	0.120	0.053	0.628	**0.005**

La Tabla 9.14 muestra el resultado de las pruebas anteriormente descritas para cada uno de los experimentos y métodos. Por lo tanto, estos resultados apoyan el rechazo de la hipótesis nula $H5_0$ en los experimentos UPV1, UPV2, UPV3 y UNA (p-valor <0,05) y la aceptación de la hipótesis alternativa, lo que significa que los participantes perciben que tienen más intención de usar QuaDAI que ATAM en un futuro.

Tabla 9.14 Resultado de las pruebas de significancia de la hipótesis H5

Experimento		*UPV1*	*UPV2*	*UPV3*	*UNA*	*UNIBAS*
Significancia	*QuaDAI*	**0.000***	**0.001***	**0,000***	**0.000***	**0.003***
	ATAM	**0.000**	**0.001**	**0,000**	**0.029**	0.059*

* Resultados del test de Wilcoxon para una muestra contra el valor de prueba v=3

9.5 Análisis consolidado de datos

En esta sección se presenta un resumen de los resultados obtenidos con un análisis de los resultados en el contexto de la familia de experimentos, seguido de un meta-análisis en el que se estudia la agregación de los datos de los resultados empíricos obtenidos en los experimentos individuales.

9.5.1 Resumen de los resultados

Una vez llevados a cabo los experimentos, se realizó un análisis global de los resultados para determinar si se había alcanzado el objetivo principal de nuestra familia de experimentos. También se analizaron todos los resultados de los experimentos individuales en busca de diferencias. La Tabla 9.15 muestra un resumen de los resultados obtenidos en cada experimento individual.

El principal resultado es que en todos los experimentos, los participantes obtuvieron los mejores resultados al aplicar QuaDAI y que todas las hipótesis alternativas fueron aceptadas por lo menos en tres experimentos. En el contexto de esta familia de experimentos, QuaDAI resultó más eficaz y eficiente que ATAM. Además, los participantes también estaban más satisfechos tras aplicar QuaDAI, y encontraron que es más fácil de usar, más útil y es más probable que lo utilicen, en comparación con ATAM, en un contexto de desarrollo de líneas de producto software dirigido por modelos.

Tabla 9.15 Resumen de los resultados de la familia de experimentos

Experimento	*Tipo de Participantes*	*Número de participantes*	*Hipótesis rechazadas*
UPV1	Estudiantes de último curso de ingeniería informática	28	$H2_0$, $H3_0$, $H4_0$, $H5_0$
UPV2	Estudiantes de Máster	16	$H1_0$, $H2_0$, $H3_0$, $H4_0$, $H5_0$
UPV3	Estudiantes de tercer curso de grado en informática	36	$H1_0$, $H2_0$, $H3_0$, $H4_0$, $H5_0$
UNA	Estudiantes de Máster	16	$H1_0$, $H3_0$, $H4_0$, $H5_0$
UNIBAS	Estudiantes de tercer curso de ingeniería informática	12	$H1_0$, $H2_0$, $H4_0$

Con respecto a la variable eficacia (ver Tabla 9.4 y Figura 9.3), se detectó que, tras la aplicación de QuaDAI, los participantes obtenían arquitecturas software en los que los RNFs se acercaban más a los valores requeridos. Sin embargo, en el caso del experimento UPV1, este resultado no alcanzó significancia

estadística. Este hecho puede ser debido a que en dicho experimento era mucho más sencillo dar con la solución óptima para el objeto experimental O2 que para el objeto experimental O1 y esto podría haber afectado a la significancia estadística del resultado (solamente hubo diferencias entre ambos métodos únicamente a nivel del objeto experimental O1 y esto no fue suficiente para alcanzar la significancia estadística). Además puede observarse que en el caso de UNIBAS las diferencias son aún más acusadas, lo cual puede deberse a la adición de un tercer RNF que complica aún más la decisión de que patrón aplicar.

Con respecto a la variable eficiencia (ver Tabla 9.4 y Figura 9.4), también se detectó que los participantes obtenían arquitecturas software en los que los RNFs se acercaban más a los valores requeridos más rápidamente (QuaDAI es entre un 4,56 y un 18,48% más eficiente que ATAM). Esto puede estar relacionado con el hecho de que al aplicar ATAM, los participantes han de considerar todos los patrones arquitectónicos y cómo estos afectan a los RNFs de interés, mientras que por el contrario, al aplicar QuaDAI tienen únicamente que lidiar con la importancia relativa de los atributos de calidad, y la decisión de qué patrón aplicar, es tomada automáticamente en base a dicha importancia relativa. En el caso del experimento UNA, las diferencias no fueron estadísticamente significativas, fruto de la fuerte dispersión que aparece en la variable duración (ver Tabla 9.4) que se combina con la variable eficiencia para el cálculo de la eficiencia.

En cuanto a la facilidad de uso percibida (ver Tabla 9.4 y Figura 9.5), se constató que QuaDAI logra unas puntuaciones medias de 3.980, 4.312, 3.590, 3.813 y 2.667 en la escala Likert de 5 puntos mientras que ATAM logra puntuaciones medias de 3.500, 4.020, 3.590, 3.479 y 2.444. Estos valores pueden ser considerados como buenos resultados para ambos métodos, dado que, excepto en el experimento UNIBAS, están por encima del valor neutro establecido en 3 puntos, y, además, para el caso específico de UNIBAS/QuaDAI, esos valores no eran estadísticamente significativos. También observamos que hay una ligera diferencia entre los resultados de estudiantes de máster (que perciben ambos métodos como más fáciles de usar) en comparación con los estudiantes de pregrado. Esto podría deberse al mayor nivel de experiencia y madurez de los estudiantes de master, que son capaces de apreciar más claramente las dificultades del proceso de evaluación de arquitecturas software.

Con respecto a la variable utilidad percibida, se constató que QuaDAI logra una puntuación media de 3.804, 4.167, 4.070, 4.427 y 3.569 en la escala Likert

de cinco puntos. ATAM logra una puntuación media de 3.723, 3.927, 3.850, 3.761 y 3.458, lo que significa que QuaDAI es percibido por los participantes como más útil que ATAM en este contexto. También es importante destacar que se trata de resultados positivos para ambos métodos, ya que todos los valores medios estaban por encima del valor neutro de la escala Likert establecido a 3 puntos. De nuevo observamos que hay una ligera diferencia entre los resultados de estudiantes de máster (que perciben QuaDAI métodos como más útil en el contexto de los experimentos) en comparación con los estudiantes de pregrado.

Con respecto a la variable intención de uso, se constató que QuaDAI logra una puntuación media 3.654, 4.063, 3.935, 4.210 y 3.458. ATAM logra una puntuación media de 3.547, 3.906, 3.740, 3.520 y 3.417, lo que significa que los participantes perciben como más probable utilizar QuaDAI en el futuro que ATAM. Así mismo, es importante destacar que se trata de resultados positivos para ambos métodos, ya que todos los valores medios estaban por encima del valor neutro de la escala Likert establecido a 3 puntos. En el caso del experimento UNIBAS, estos resultados no alcanzaron la significancia estadística.

En resumen, los resultados apoyan la hipótesis de que QuaDAI tendría un mejor rendimiento que ATAM en el contexto especificado. De acuerdo con los resultados discutidos anteriormente, podemos concluir que QuaDAI puede ser considerado como un enfoque prometedor para llevar a cabo evaluaciones arquitectónicas en un contexto de desarrollo de líneas de producto dirigido por modelos. También se obtuvo información sobre la forma de mejorar el enfoque (por ejemplo, las dificultades que encuentran algunos participantes para asignar la importancia relativa en el conjunto de RNFs o de la posible reutilización de los resultados de la medición para seleccionar entre las transformaciones arquitectónicas). La ejecución de una familia de experimentos (incluyendo réplicas), en lugar de un único experimento nos proporcionó una mayor evidencia de la validez externa, y por lo tanto de la generalización de los resultados del estudio. Cada réplica proporciona una prueba más de la confirmación de las hipótesis. Así pues, podemos concluir que se ha logrado el objetivo general de la validación empírica.

9.5.2 Meta-análisis

Aunque se han propuesto distintos métodos para la agregación e interpretación de resultados obtenidos a partir de estudios empíricos relacionados entre sí

(Glass et al. 1981; Hedges y Olkin 1985; Hunter et al. 1982; Mullen y Rosenthal 1985; Rosenthal 1986; Sutton et al. 2001), hemos empleado el meta-análisis dado que nos permite extraer conclusiones más generales. El meta-análisis es un conjunto de técnicas estadísticas para combinar las magnitudes de los efectos de distintos experimentos interrelacionados con el fin de obtener el efecto global de un factor. La estimación de la magnitud del efecto se puede utilizar con posterioridad para comparar distintos estudios con el fin de evaluar el impacto promedio en todos los estudios de una variable independiente sobre una variable dependiente. Dado que las mediciones pueden provenir de diferentes configuraciones que pueden ser no homogéneas, se debe obtener una métrica estandarizada para cada experimento. Esas métricas deben combinarse para estimar la magnitud del efecto global de un factor. En nuestro estudio, hemos considerado el método de evaluación de arquitecturas software como el principal factor en la familia de experimentos.

El meta-análisis se llevó a cabo mediante la herramienta Meta5.3 (Schwarzer 1987; Schwarzer 1988) aplicando el método de Hunter-Schmidt (Hunter et al. 1982). Puesto que todas las variables en estudio, en al menos un experimento, no se ajustaban a la distribución normal hemos empleado el coeficiente de correlación biserial *r*. El coeficiente de correlación biserial *r* se utiliza cuando queremos conocer la correlación existente entre dos variables, de las cuales una de ella expresa información dicotómica, como es en nuestro caso el método de evaluación aplicado, y la otra variable se mide, al menos, mediante una escala de intervalos. Las correlaciones son la forma más sencilla y difundida de medir la magnitud de un efecto, ya que describen la dirección y potencia de la relación entre dos variables en un rango de -1.0 a 1.0 (Schwarzer 1987). El coeficiente de correlación biserial *r* para cada experimento se calcula aplicando la fórmula (4) (Mullen y Rosenthal 1985), donde *Z* es el cuartil (z-score) del valor *p* obtenido mediante las pruebas de Mann-Whitney/Wilcoxon/prueba *t* y *N* es el tamaño muestral en cada caso.

Puesto que en la familia de experimentos cada uno de ellos tiene un tamaño muestral distinto, la mejor manera de estimar el tamaño del efecto para cada uno experimento, es calcular el coeficiente de correlación $\bar{r}$ ponderado por el número de observaciones en cada estudio (Hunter et al. 1982). El coeficiente de correlación $\bar{r}$ ponderado asigna mayor peso a los estudios con mayor número de observaciones *N* y puede calcularse aplicando la fórmula (5). Para estudios en el ámbito de la ingeniería del software, la magnitud de un efecto medido a partir del coeficiente de correlación biserial *r* (o $\bar{r}$) se clasifica de la

siguiente manera (Kampenes et al. 2007): pequeño (0 - 0.193), medio (0.194 - 0.456) o grande (por encima de 0.456).

El método de Hunter-Schmidt también permite llevar a cabo una prueba de homogeneidad entre estudios mediante un test de significancia chi-cuadrado. Esta prueba se calcula aplicando la fórmula (6) (Hunter et al. 1982) que da como resultado el chi-cuadrado con *k*-1 grados de libertad ${\chi_{k-1}}^2$, para una familia de *k* experimentos y donde la varianza ponderada entre estudios ${S_r}^2$ se calcula aplicando la fórmula (7) (Hunter et al. 1982). Si la prueba chi-cuadrado resulta no significativa, se constata una fuerte evidencia de que no hay una variación real entre los distintos estudios, pero si por el contrario la prueba chi-cuadrado resulta significativa, la variación entre estudios puede sea insignificante en magnitud.

La Tabla 9.16 muestra un resumen de los resultados del meta-análisis según el método de Hunter-Schmidt. La magnitud del efecto global se ha expresado en términos del coeficiente de correlación biserial medio ponderado $\bar{r}$, también se muestra la significancia estadística de dicho efecto y la homogeneidad de los estudios respecto a dicho efecto. En todos los casos el efecto fue muy significativo ($p<0.001$) y las distintas poblaciones resultaron homogéneas en los casos de la eficacia y la eficiencia y heterogéneo para el resto de variables (facilidad de uso percibida, utilidad percibida e intención de uso). El tamaño obtenido fue medio para la eficacia y la eficiencia y grande para las variables dependientes subjetivas (facilidad de uso percibida, utilidad percibida e intención de uso).

$$r = \sqrt{\frac{Z^2}{N}} \tag{4}$$

$$\bar{r} = \frac{\sum r_i * N_i}{\sum N_i} \tag{5}$$

$${\chi_{k-1}}^2 = \frac{\sum N_i}{(1-\bar{r}^2)^2} * {S_r}^2 \tag{6}$$

$$S_r{}^2 = \frac{\sum(N_i(r_i - \bar{r})^2)}{\sum N_i} \quad (7)$$

Tabla 9.16 Magnitud del efecto y test de homogeneidad

Variable		*N*	$\bar{r}$	*Significancia*	*Test de homogeneidad*		
					chi-square	*g.l.*	*Resultado (valor p)*
Eficiencia		216	Medio (0.288)	Si (p<0.001)	6,764	4	Homogéneo (p=0.122)
Efectividad		216	Medio (0.316)	Si (p<0.001)	0,547	4	Homogéneo (p=0.968)
Facilidad de uso percibida	***QuaDAI***	108	Grande (0.735)	Si (p<0.001)	57,968	4	Heterogéneo (p<0.001)
	ATAM	108	Grande (0.529)	Si (p<0.001)	48,729	4	Heterogéneo (p<0.001)
Utilidad percibida	***QuaDAI***	108	Grande (0.948)	Si (p<0.001)	11,167	4	Heterogéneo (p=0.025)
	ATAM	108	Grande (0.891)	Si (p<0.001)	24,169	4	Heterogéneo (p<0.001)
Intención de uso	***QuaDAI***	108	Grande (0.939)	Si (p<0.001)	55,3237	4	Heterogéneo (p<0.001)
	ATAM	108	Grande (0.831)	Si (p<0.001)	21,815	4	Heterogéneo (p<0.001)

Además se ha llevado un análisis de clusters aplicando el método de análisis de clusters (Mullen y Rosenthal 1985) para analizar el tamaño del efecto de cada experimento individual, que se muestran en la Tabla 9.17. Para cada experimento, se indica el tamaño efecto de la población a través de la media no ponderada *r*. Este efecto también ha sido clasificado como pequeño (P), mediano (M) y grande (G) y sigue la misma clasificación para los tamaños de los efectos aplicados al tamaño del efecto ponderado de la Tabla 9.16.

Tabla 9.17 Magnitud del efecto de los experimentos individuales

Experimento	*Eficacia*	*Eficiencia*	*Facilidad de uso percibida*		*Utilidad percibida*		*Intención de uso*	
			QuaDAI	*ATAM*	*QuaDAI*	*ATAM*	*QuaDAI*	*ATAM*
UPV1	P (0.014)	M (0.268)	G (0.962)	M (0.437)	G (1.0000)	G (0.8079)	G (0.8880)	G (0.7684)
UPV2	M (0.368)	M (0.359)	G (1.000)	G (1.000)	G (1.0000)	G (0.8011)	G (1.0000)	G (0.8273)
UPV3	M (0.350)	M (0.323)	G (0.658)	G (0.850)	G (1.0000)	G (1.0000)	G (1.0000)	G (1.0000)
UNA	M (0.380)	M (0.274)	G (0.895)	M (0.404)	G (1.0000)	G (0.8015)	G (1.0000)	G (0.5885)
UNIBAS	G (0.510)	M (0.407)	P (-0.134)	G (-0.680)	G (1.0000)	G (1.0000)	M (0.7965)	M (0.7967)

A pesar de que el primer experimento contribuyó a los resultados globales del meta-análisis, en menor medida dichos resultados presentan un efecto positivo significativo, por lo que podemos rechazar la hipótesis nula que se formularon

para cada variable dependiente (es decir, "no hay diferencias significativas entre QuaDAI y ATAM"). Por tanto, el meta-análisis refuerza todas las hipótesis alternativas, con resultados prometedores en cuanto a resultado de QuaDAI.

9.6 Amenazas a la validez

En esta sección, se explican los principales problemas que pueden poner en peligro la validez de los experimentos, al considerar los cuatro tipos de amenazas que proponen Cook y Campbell (1979).

9.6.1 Validez interna

Las amenazas a la validez interna son relevantes en aquellos estudios que intentan establecer relaciones causales. En nuestro caso, las principales amenazas a la validez interna de la familia de experimentos son:

- *Efecto aprendizaje:* el impacto que este efecto pudiese tener en los resultados de los experimentos, fue mitigado mediante la definición de dos objetos experimentales, asegurándonos de que cada participante aplicaba cada método sobre un objeto diferente y considerando todas las combinaciones posibles de orden de métodos y objetos experimentales siguiendo un diseño experimental entre-sujetos con el que se pretende minimizar el impacto de este efecto.
- *Experiencia de los participantes:* otra de las amenazas a la validez interna, son las posibles diferencias en la experiencia previa de los participantes. En el caso de esta familia de experimentos no habían tales diferencias, dado que ninguno de los participantes en los diferentes experimentos tenía experiencia previa en evaluaciones arquitectónicas. Este hecho fue confirmado preguntando a los participantes acerca de su experiencia previa en este tipo de evaluaciones. Esta es la razón de proporcionar sesiones de entrenamiento de ambos métodos antes de cada experimento, ya que se pretendía dotar y equilibrar el conocimiento del participante en los métodos de evaluación arquitectónica de acuerdo con su perfil de evaluador novel.
- *Intercambio de información:* el intercambio de información es otra de las amenazas a la validez interna que puede alterar el resultado de los distintos experimentos. En nuestro caso este intercambio se reduce al mínimo, por un lado gracias al uso de objetos experimentales diferentes

y por el otro, por el seguimiento a los participantes mientras estos realizaban las tareas. Sin embargo, dado que el experimento fue diseñado para ser llevado a cabo en más de una sesión, los participantes podrían intercambiar información durante el tiempo entre sesiones. Para paliar esta situación se solicita que los participantes devuelvan el material experimental al final cada una de las sesiones.

- *Sesgo de autor:* el hecho de que los experimentos sean conducidos por los propios autores podría influir en el resultado de los mismos. En el caso de los experimentos realizados en España (UPV1, UPV2 y UPV3) y Paraguay (UNA), las sesiones de entrenamiento fueron llevadas a cabo por diferentes autores del método. En estos experimentos se trató de mitigar esta influencia no dando a conocer la autoría del método QuaDAI. Sin embargo, en el experimento UNIBAS esta amenaza fue mitigada totalmente, puesto que el entrenamiento y el experimento fueron llevados a cabo por un investigador externo.
- *Comprensibilidad de los materiales:* este riesgo fue mitigado redactando los materiales para los cuatro primeros experimentos (UPV1, UPV2, UPV3 y UNA) en la lengua materna de los participantes. Sin embargo, para el experimento UNIBAS los materiales fueron traducidos al inglés y no a la lengua materna de los participantes (italiano), lo que podría haber influido en los resultados finales de esta réplica. Finalmente, en todos los experimentos de la familia, se resolvieron todas las dudas y malentendidos acerca de los materiales experimentales conforme apariencia durante las sesiones experimentales.

9.6.2 Validez Externa

La Validez externa se refiere a la verdad aproximada de las conclusiones que implican generalizaciones en diferentes contextos. La principal amenaza a la validez externa es la representatividad de los resultados, que puede verse afectada por el diseño de la evaluación, la selección de participantes y el tamaño y complejidad de las tareas experimentales, cuyos detalles se describen a continuación:

- *Diseño de la evaluación:* el diseño de la evaluación podría restringir la generalización de los resultados debido a la clase de modelos arquitectónicos y atributos de calidad a evaluar. Tratamos de minimizar este riesgo seleccionando dos arquitecturas con un tamaño y

complejidad similares, pero de dos dominios representativos (sistemas de automoción y aplicaciones móviles). Con respecto a los RNFs, tratamos de seleccionar dos RNFs distintos y representativos (fiabilidad y rendimiento) y dos métricas distintas para cada RNF (tiempo de latencia y tolerancia a fallos para cada el primero objeto experimental; y carga de trabajo y porcentaje de actividad para el segundo objeto experimental). Con respecto a los patrones arquitectónicos, los expertos del dominio seleccionaron cuatro patrones diferentes para cada objeto experimental ampliamente difundidos, pero al mismo tiempo específicos, para los dominios de interés de cada uno de los objetos experimentales.

- *Experiencia de los participantes:* Con respecto a la experiencia de los participantes, los experimentos se llevaron a cabo con estudiantes (en los experimentos UPV1, UPV2, UPV3 y UNIBAS) y profesionales (en el caso del experimento UNA), todos ellos sin experiencia previa en evaluaciones arquitectónicas y que únicamente recibieron formación limitada sobre los métodos de evaluación. Sin embargo, en el caso de los estudiantes, se trata estudiantes de últimos cursos de ingeniería, y tanto estos como los profesionales del experimento UNA, pueden ser considerados como usuarios noveles de los métodos de evaluación de arquitecturas software ya que serán la próxima generación de profesionales (Kitchenham et al. 2002). Así, estos resultados pueden considerarse como representativos para evaluadores noveles.

- *Tamaño y complejidad de las tareas:* Por último, el tamaño y complejidad de las tareas también podría limitar la representatividad de los resultados. Hemos decidido utilizar tareas de un tamaño relativamente pequeño con el fin de que éstas puedan ser aplicadas en unos pocos modelos arquitectónicos representativos ya que un experimento controlado, requiere que los participantes completen las tareas asignadas en un tiempo limitado.

9.6.3 Validez de constructo

La validez de constructo de la familia de experimentos puede haberse visto influenciada por las métricas que se aplicaron en el análisis cuantitativo y por la fiabilidad del cuestionario. Estas amenazas fueron paliadas mediante el uso de métricas que se aplican comúnmente en la evaluación y optimización multi-objetivo de arquitecturas. En particular, la eficacia fue medida empleando la

distancia euclídea, que ha sido comúnmente utilizada para medir la bondad de una solución con respecto a un conjunto de RNFs opuestos en diferentes escenarios, como por ejemplo en los trabajos de Dattorro (2005) o de Taher et al. (2005). Las variables subjetivas, están basadas en el método de aceptación de tecnología (TAM), que es un método bien conocido y empíricamente válido para la evaluación de tecnologías de la información (Davis 1989).

La fiabilidad del cuestionario fue analizada mediante la aplicación del test Alfa de Cronbach (Maxwell 2002). La Tabla 9.18 muestra los resultados del test para cada conjunto de preguntas cerradas destinadas a medir las tres variables dependientes subjetivas (facilidad de uso percibida, utilidad percibida e intención de uso). Excepto en el caso de UNIBAS todos los valores obtenidos fueron mayores que el umbral mínimo aceptable (α=0.70) (Maxwell 2002). Esto puede deberse a que, en el caso de los experimentos UPV1, UPV2, UPV3 y UNA, todos los materiales estaban redactados en castellano, la lengua materna de los participantes, mientras que en el caso de UNIBAS los materiales estaban disponibles únicamente en inglés. En el caso de los cuestionarios, donde los matices de las preguntas son menos evidentes, esta diferencia podría haber tenido mayor influencia.

9.6.4 Validez de las conclusiones

Las principales amenazas a la validez de las conclusiones de la familia de experimentos son la recopilación de datos y la validez de las pruebas estadísticas aplicadas. Con respecto a la recogida de datos, se aplicó el mismo procedimiento en cada experimento individual con el fin de extraer los datos, y se aseguró que cada variable dependiente se calculara mediante la aplicación de la misma fórmula. Con respecto a la validez de las pruebas estadísticas aplicadas, esta amenaza ha sido mitigada mediante la aplicación de un conjunto de pruebas comúnmente aceptadas, empleadas en la comunidad de ingeniería del software empírico (Maxwell 2002).

Tabla 9.18 Análisis del Alfa de Cronbach para las variables del cuestionario post-experimento

Variable dependiente	*Facilidad de uso percibida*	*Utilidad percibida*	*Intención de uso*
UPV1	Aceptable (0.824)	Aceptable (0.870)	Aceptable (0.831)
UPV2	Aceptable (0.890)	Aceptable (0.898)	Aceptable (0.814)
UPV3	Aceptable (0.738)	Aceptable (0.758)	Aceptable (0.731)
UNA	Aceptable (0.854)	Aceptable (0.913)	Aceptable (0.872)
UNIBAS	Aceptable (0.748)	No-aceptable (0.011)	No-aceptable (0.666)

9.7 Conclusiones

En este capítulo se han presentado los resultados de la validación empírica con el objetivo de evaluar la eficacia, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso futura de los participantes cuando emplean QuaDAI en comparación con ATAM, un método de evaluación de arquitecturas ampliamente utilizado.

Los resultados del análisis cuantitativo nos demostraron que, en las condiciones descritas, QuaDAI resulto ser más eficaz y eficiente que ATAM (a pesar de que no se pudo probar que las diferencias fuesen estadísticamente significativas, para la eficacia en el primer experimento y para la eficiencia en el segundo, debido en el primer caso a la diferencia de dificultad en los objetos experimentales observada en el primer experimento y en el segundo debido a la fuerte dispersión en el tiempo empleado por los participantes para ejecutar las tareas experimentales). Además, con respecto a la percepción de los evaluadores, los participantes percibieron QuaDAI como más fácil de usar, más útil y con mayores posibilidades de ser utilizado en el futuro (aunque de nuevo en el cuarto experimento las diferencias en cuanto a la facilidad de uso percibida y la intención de uso no resultaron estadísticamente significativas, tal vez porque en la última replica los materiales no estaban en la lengua materna de los participantes y eso hizo que el cuestionario que medía dichas variables subjetivas resultase no ser fiable para este experimento).

Los resultados fueron apoyados por un meta-análisis llevado a cabo con el fin de agregar los hallazgos empíricos obtenidos en cada experimento individual. Los resultados del meta-análisis permiten concluir que existe un efecto positivo significativo en todas las variables estudiadas asociadas al uso de QuaDAI como método de evaluación de la arquitectura.

Desde el punto de vista investigador, la familia de experimentos ha resultado ser un medio valioso para obtener retroalimentación con la que mejorar QuaDAI (por ejemplo, para mejorar el mecanismo para la asignación de la importancia relativa al conjunto de RNFs o reutilizar los resultados del proceso de medición para seleccionar transformaciones arquitectónicas). Por lo que sabemos, éste es el primer estudio empírico que proporciona evidencia de la utilidad de un método de evaluación de la arquitectura de software para un proceso de desarrollo de software basado en modelos.

Desde un punto de vista práctico, somos conscientes de que este estudio sólo proporciona los resultados preliminares sobre la utilidad de QuaDAI como método de evaluación de arquitecturas software.

Si bien los resultados experimentales proporcionan buenos resultados en cuanto al rendimiento de nuestro método de evaluación para evaluación de arquitecturas software para contextos de desarrollo de líneas de producto software dirigido por modelos, estos resultados deben ser interpretados con cautela, ya que sólo son válidos dentro del contexto establecido en esta familia de experimentos. Ahora es necesario analizar si los mismos resultados se producen con participantes más experimentados y con nuevos objetos experimentales, con un conjunto más amplio y más complejo de patrones y requisitos no-funcionales. Por lo tanto, es necesario llevar cabo nuevos estudios empíricos para probar nuestra propuesta en otros entornos.

Capítulo 10

Conclusiones y Trabajos Futuros

Este capítulo revisa los objetivos de la investigación, los resultados del trabajo y las conclusiones del mismo, al tiempo que analizamos en qué medida los objetivos iniciales han sido alcanzados. Por último, se presentan las contribuciones consecuencia de la investigación en forma de publicaciones, estancias de investigación y las becas y galardones obtenidos, así como las posibles vías de extensión de este trabajo.

10.1 Conclusiones

La aproximación de líneas de producto software emergió como un medio para la mejora de los procesos de desarrollo software, con el objetivo de reducir los costes de desarrollo e incrementar sustancialmente la productividad y la calidad de los productos desarrollados.

En torno al desarrollo de líneas de producto software han emergido distintos métodos que proponen la derivación de arquitecturas de producto, si bien en su mayoría, estos métodos no incorporan los criterios de calidad. Es sorprendente, que siendo dichos criterios de calidad una de las razones de ser de la aproximación del desarrollo de líneas de producto software y un pilar básico en la disciplina, haya sido descuidada en un proceso tan crítico y de tan elevada complejidad. Además, en caso de llevarse a cabo procesos de evaluación de la arquitectura derivada, esta se lleva cabo haciendo uso de

métodos que no han sido definidos teniendo en cuenta las peculiaridades de dicha aproximación.

En esta tesis doctoral se ha propuesto y validado empíricamente un método para la derivación, evaluación y mejora de arquitecturas software en entornos de desarrollo de líneas de producto en el que se aplica el paradigma de desarrollo dirigido por modelos. El grado de cumplimiento de los distintos objetivos propuestos se va a analizar en las siguientes subsecciones.

10.1.1 Análisis de técnicas de derivación de arquitecturas de producto

Con respecto a este objetivo, se ha realizado un análisis de las aproximaciones que abordan la derivación de arquitecturas de producto en entornos de desarrollo de líneas de producto software aplicando el paradigma de desarrollo dirigido por modelos. Las principales conclusiones de dicho análisis son que la integración de atributos de calidad en el proceso de derivación de la arquitectura de producto no ha sido cubierta de manera adecuada en los trabajos de investigación en el área.

Únicamente dos trabajos de entre los 16 analizados tienen en cuenta aspectos de calidad a la hora de derivar la arquitectura. Además en la mayoría de los casos, los métodos y técnicas presentados han sido definidos de forma específica para una determinada vista arquitectónica o un determinado lenguaje de modelado arquitectónico, sin que se puedan encontrar en la literatura propuestas genéricas que integren criterios de calidad en los procesos de derivación de la arquitectura de producto.

La mayoría de las técnicas y métodos definidos no soportan la validación de la consistencia ni en los modelos de configuración que toman como entrada, ni de los modelos arquitectónicos obtenidos. Esto permite la generación de productos que no cumplen las restricciones definidas a nivel arquitectónico y/o a nivel de la variabilidad.

10.1.2 Análisis de los métodos de evaluación de arquitecturas de producto

Con respecto al objetivo de análisis de los métodos de evaluación y mejora de arquitecturas software, se analizaron los métodos de evaluación de arquitecturas definidos para ser aplicados en contextos de desarrollo de líneas de producto software. Las principales conclusiones extraídas de dicho análisis

son que la evaluación de la arquitectura de producto en entornos de desarrollo de líneas de producto siguiendo la aproximación de desarrollo de software dirigido por modelos no se halla suficientemente respaldada por métodos que permitan la evaluación de arquitecturas independientemente de los atributos y naturaleza de las arquitecturas de producto a evaluar.

Además se constató que no hay en la literatura métodos de evaluación de arquitecturas basados en métricas para la evaluación, en tiempo de derivación, de la arquitectura obtenida. En el desarrollo de líneas de producto es posible que haya variabilidad en los valores de los niveles de los atributos de calidad y el uso de métodos de evaluación basados en métricas va a permitir analizar si dichos valores se encuentran dentro de los valores definidos por el alcance o si por el contrario se ha generado un producto cuyas niveles de calidad son inferiores a lo establecido.

El proceso de derivación es una fase crítica y compleja que tiene una alto coste (Griss 2000; Rabiser y Dhungana 2007), y la evaluación de los atributos de calidad en fase temprana, tras la derivación de la arquitectura, va a permitir detectar potenciales problemas, cuya detección, si se demorase más en el tiempo, incidirían en un coste económico y temporal sobre el proyecto mucho mayor.

10.1.3 Definición de un multimodelo que de soporte a las vistas de la línea de productos

Con respecto a este objetivo, hemos definido un multimodelo que da soporte a los distintos puntos de vista de interés y que permite definir relaciones, de distintos tipos y con distintas semánticas, entre elementos de las distintas vistas. Dicho multimodelo va a vertebrar el proceso de derivación, evaluación y mejora de la arquitectura de producto, integrando la vista de calidad como artefacto activo en el proceso de desarrollo. Se han definido seis tipos de relaciones posibles entre elementos de las vistas a partir, algunas de ellas, de relaciones entre modelos descritas en la literatura.

Se definieron cuatro vistas (vista de variabilidad, arquitectónica, de calidad y de transformaciones), para representar el problema de la derivación, evaluación y mejora de arquitecturas y se definieron seis relaciones entre elementos de estas vistas de dos tipos concretos (relaciones de impacto y de descomposición) y se formalizó su semántica para las relaciones implicadas en el proceso de derivación. La adición de una vista de variabilidad arquitectónica en el multimodelo (expresada mediante el estándar CVL para la descripción de

variabilidad) nos permite añadir un nivel de abstracción adicional entre los modelos de variabilidad externa de la línea de productos y los modelos arquitectónicos, permitiendo tener una representación explicita de la variabilidad interna, a nivel arquitectónico, sin tener que alterar ni la representación de la variabilidad externa de la línea de productos con información propia de la arquitectura, ni la representación arquitectónica con información de variabilidad. Además, el uso de CVL como lenguaje de especificación de la variabilidad arquitectónica permite que el método de evaluación y mejora propuesta sea aplicable de forma genérica sobre arquitecturas descritas mediante cualquier lenguaje de descripción arquitectónica basado en MOF.

El multimodelo da soporte a las distintas fases y actividades del método QuaDAI mediante las estructuras que se describen a continuación:

- Las relaciones entre características, requisitos no-funcionales y atributos de calidad nos permiten asistir el proceso de configuración, sugiriendo posibles modificaciones de la configuración en función de las prioridades ya seleccionadas hasta ese momento y validar la consistencia de la configuración seleccionada mediante la operacionalización de la validación de dichas relaciones mediante restricciones OCL.

- Las expresividad de la vista de calidad nos permite describir los requisitos no-funcionales de la línea de productos y del producto en desarrollo; describir como esos requisitos son evaluados; y, tras la evaluación, nos permite comprobar, de forma automática, el grado de cumplimiento de dichos requisitos no-funcionales.

- Las relaciones entre puntos de variación arquitectónica, características, requisitos no-funcionales y atributos de calidad nos permiten conducir el proceso de derivación para seleccionar aquellas variantes de la arquitectura (sobre cualquiera de sus vistas) que mejor se adapten a la configuración seleccionada.

- La propia vista de transformación, y las relaciones entre transformaciones alternativas y atributos de calidad nos permite guiar el proceso de transformación de la arquitectura en aquellos casos en que no se cumplan los requisitos de calidad.

Para dar soporte al multimodelo, se ha definido una arquitectura en dos niveles que hace que la propuesta del uso de multimodelo no se circunscriba

únicamente al conjunto de vistas utilizadas en este trabajo, o su uso en el desarrollo de líneas de producto software, sino que además permite definir multimodelos de forma genérica, entre dos o más modelos, ofreciendo soporte mediante una infraestructura de gestión de multimodelos, para cualquier conjunto de vistas, siempre y cuando sus metamodelos sean compatibles con el estándar MOF.

10.1.4 Definición de un proceso de derivación, evaluación y mejora de arquitecturas de producto

Con respecto a este objetivo, se ha definido QuaDAI, un método integrado de derivación, evaluación y mejora de arquitecturas de producto conducido por transformaciones. Dicho método describe el conjunto de actividades a desarrollar en la fase de ingeniería de aplicación para la obtención, a partir de los requisitos definidos por el ingeniero de aplicación, una versión de la arquitectura que cumple con los requisitos funcionales y no-funcionales. Además se han definido las actividades a llevar a cabo en la fase de ingeniería de dominio para capturar la información de las relaciones entre elementos de las distintas vistas.

La fase de derivación del método cubre la mayor parte de las actividades clave identificadas por Rabiser et al. (2011):

- Preparación para la derivación: Se debe soportar la especificación y traslado de los requisitos del cliente (en el multimodelo mediante la vista de características y de calidad), definición de la configuración base, proyección de los requisitos de cliente (el modelo de configuración empleado no solo permite proyectar los requisitos sino que brinda mecanismos de trazabilidad para los mismos), y dar soporte a la configuración mediante mecanismos de recomendación y validación de la consistencia.

- Configuración y derivación: El método cubre esta actividad clave, con mecanismo de asistencia en la selección de la configuración, validación de consistencia y un proceso de derivación automatizado mediante transformación de modelos.

- Desarrollo adicional y testeo: El método prevé la validación de consistencia de los modelos derivados (mediante la validación de CVL) y la verificación de los requisitos no-funcionales mediante la fase de evaluación y mejora.

La fase de derivación se soporta en el método mediante: i) la definición de un proceso compuesto por el conjunto de actividades, roles y artefactos a ser empleados en cada actividad; ii) la definición del modelo de configuración como proyección de las vista de variabilidad y calidad; iii) la formalización de las relaciones entre las vistas de calidad y variabilidad del multimodelo, que permite validar la consistencia de la configuración seleccionada; y iv) la definición del proceso de transformación que permite obtener el modelo de resolución CVL a partir del modelo de configuración.

La fase de evaluación y mejora esta soportada en el método mediante: i) la definición del proceso; ii) la definición de los artefactos y mecanismos que permiten evaluar el grado de satisfacción de los requisitos no-funcionales tras la medición; y iii) unas directrices de diseño para la definición de transformaciones arquitectónicas en las que la selección entre transformaciones arquitectónicas alternativas se realiza en base a atributos de calidad.

10.1.5 Validación empírica del método propuesto

El método propuesto ha sido validado empíricamente a través de un estudio de caso llevado a cabo en España y Brasil y a través de una familia de cinco experimentos con replicaciones tanto internas como externas que se llevaron a cabo en España, Italia y Paraguay.

La validación del proceso de derivación de arquitecturas de productos se llevó a cabo mediante un estudio de caso, ejecutado en dos sesiones, una en la Universitat Politècnica de València y la otra en la Universidade Federal da Bahía (Brasil) involucrando a 14 sujetos, estudiantes de máster y doctorado y profesionales del desarrollo de software, algunos de ellos con experiencia industrial en la aplicación del enfoque de líneas de productos. En el estudio de caso se evaluó la facilidad de uso percibida, utilidad percibida e intención de uso futura de los participantes cuando emplean QuaDAI en un proceso de derivación de una arquitectura de producto de una línea de productos del dominio automovilístico.

Los resultados nos han permitido constatar que, en general, la aplicación del método permitió a los participantes configurar y obtener una arquitectura de producto que cumpliese con los requisitos tanto funcionales como no-funcionales del producto en desarrollo. Del análisis cuantitativo se desprende que los participantes aprecian la fase de derivación de arquitecturas del método QuaDAI como fácil de usar, útil y muestran intención de usarlo en el futuro.

Desde el punto de vista investigador, el estudio de caso nos ha permitido, analizar la aplicabilidad real del proceso de derivación, modelar un caso real y analizar el grado de completitud y corrección de las soluciones aportadas por el método y por último, pero no menos importante, obtener realimentación de los participantes que aplicaron el método, cuyos comentarios nos permitirán mejorar en el futuro tanto el método como la infraestructura de soporte al multimodelo.

Desde el punto de vista práctico, este es un primer estudio con un número muy reducido de participantes, con perfiles heterogéneos y que solo arroja resultados preliminares sobre la utilidad, facilidad de uso e intención de uso del proceso de derivación del método QuaDAI. En el futuro será necesario replicar este estudio de caso en grupos más homogéneos y con mayor número de participantes para analizar posibles interacciones de factores como la experiencia o la habilidad.

La validación del proceso de evaluación y mejora de arquitecturas propuesto se ha llevado a cabo mediante una familia de cinco experimentos, con replicaciones tanto internas como externas en España, Italia y Paraguay, involucrando a 108 sujetos, estudiantes de grado e ingeniería en informática, másteres en ingeniería del software y de doctorado. En dichos experimentos se compara la eficacia, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso futura de los participantes cuando emplean QuaDAI en comparación con ATAM, un método de evaluación de arquitecturas ampliamente difundido.

El análisis cuantitativo de los datos y el posterior meta-análisis llevado a cabo con el fin de agregar los hallazgos empíricos obtenidos en cada experimento individual nos permiten concluir que existe un efecto positivo significativo en todas las variables estudiadas asociadas al uso de QuaDAI como método de evaluación de la arquitectura en entornos de desarrollo de líneas de producto dirigidos por modelos.

Desde el punto de vista investigador, la familia de experimentos ha resultado ser un medio valioso para obtener realimentación con la que mejorar QuaDAI (por ejemplo, para mejorar el mecanismo para la asignación de la importancia relativa al conjunto de RNFs o reutilizar los resultados del proceso de medición para seleccionar transformaciones arquitectónicas). Por lo que sabemos, éste es el primer estudio empírico que proporciona evidencia de la utilidad de un método de evaluación de la arquitectura de software para un proceso de desarrollo de software basado en modelos.

Desde un punto de vista práctico, somos conscientes de que este estudio sólo proporciona los resultados preliminares sobre la utilidad de QuaDAI como método de evaluación de arquitecturas software, que deberán ser contrastados con futuras replicaciones en otros contextos, dado que es necesario analizar si los mismos resultados se producen con participantes más experimentados y con nuevos objetos experimentales, de mayor complejidad.

10.2 Difusión de resultados

El trabajo descrito en esta tesis han sido publicados en **dos conferencias** internacionales catalogados como **Nivel A** según el ranking **ERA-CORE, tres conferencias** internacionales catalogados como **Nivel B** según el ranking **ERA-CORE** (dos de ellos en la conferencia **MODELS**, la conferencia más relevante en el área de desarrollo dirigido por modelos), una conferencia internacional sin índice en el ranking CORE-ERA, un taller internacional (evento satélite de la conferencia **MODELS**), tres publicaciones en congresos nacionales (uno de ellos fue galardonado con el **premio al tercer mejor artículo de la conferencia SBCARS**), un taller nacional y una publicación en ERCIM-News. Actualmente tenemos en fase de revisión tres publicaciones, dos de ellas en revistas indexadas en el ranking JCR y un capítulo de libro cuya propuesta inicial ya ha sido aceptada. Las siguientes subsecciones describen dichas publicaciones con mayor detalle.

10.2.1 Revistas internacionales con índice de calidad relativo

- Gonzalez-Huerta, J., Insfran, E., Abrahão, S., Scanniello, G. (2013). *"Validating a Model-Driven Software Architecture Evaluation Method for Software Product Lines: A Family of Experiments"* **Information and Software Technology**, Impact Factor 1.507 (JCR 2010) **(Aceptado con cambios).**
- Oliveira, R.P. De, Blanes, D, Gonzalez-Huerta, J., Pastor-Perez, J., Insfran, E., Abrahão, S., Cohen, S., Santana, E. (2013) *"Defining and Validating a Feature-Driven Requirements Engineering Approach"*, **Journal of Universal Computer Science**, Impact Factor 0.762 (JCR 2010) **(Aceptado).**

10.2.2 Actas de congresos internacionales

- González-Huerta, J., Insfran, E., Abrahão, S., McGregor, J.D. (2013). "*Architecture Derivation in Product Line Development through Model Transformations*". En actas de: 22nd International Conference on Information Systems Development **(ISD 2013)**, Seville, Spain. **ERA CORE Tier A.**

- González-Huerta, J., Insfrán, E., Abrahão, S. (2013). "*Defining and Validating a Multimodel Approach for Product Architecture Derivation and Improvement*". En actas de: 16th International Conference on Model-Driven Engineering Languages and Systems, **(MODELS 2013)**, Miami, USA, pp. 388–404. **ERA CORE Tier B.**

- Insfran, E., Abrahão, S., González-Huerta, J., McGregor, J.D. (2012). "*A Multimodeling Approach for Quality-Driven Architecture Derivation*". En actas de: 21st International Conference on Information Systems Development **(ISD 2012)**, Prato, Italy. **ERA CORE Tier A.**

- González-Huerta, J., Insfran, E., Abrahão, S. (2012). "*A Multimodel for Integrating Quality Assesment in Model-Driven Engineering*". En actas de: 8th International Conference on the Quality of Information and Communications Technology **(QuaTIC 2012)**, Lisbon, Portugal.

- Insfran, E., Gonzalez-Huerta, J., Abrahão, S. (2010). "*Design Guidelines for the Development of Quality-Driven Model Transformations*". En actas de: 13th International Conference on Model Driven Engineering Languages and Systems **(MODELS 2010)**, Oslo, Norway. pp. 288–302. **ERA CORE Tier B.**

- Gonzalez-Huerta, J., Blanes, D., Insfran, E., Abrahão, S. (2010). "*Towards an architecture for ensuring product quality in model-driven software development*". En actas de: Proceedings of the 11th International Conference on Product Focused Software **(PROFES 2010)**, Limerick, Ireland. pp. 28–31. **ERA CORE Tier B.**

- Botto, M., Insfran, E, Gonzalez-Huerta, J. (2014). *"Are Model-Driven Techniques Used As A Means To Migrate SOA Applications To Cloud Computing?"* En actas de 10th International Conference on Web Information Systems and Technologies **(WEBIST 2014)**, Barcelona, Spain. **ERA CORE Tier C**.

10.2.3 Actas de talleres internacionales

- González-Huerta, J., Insfran, E., Abrahão, S., McGregor, J.D. (2012). "*Non-functional requirements in model-driven software product line engineering*". En actas de: Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages – **(NFPinDSML 2012)**, Innsbruck, Austria.
- González-Huerta, J., Insfran, E., Abrahão, S. (2010). *"Defining an Architecture for Quality-Driven Model Transformations"*. En actas de 1st International Master Class on Model-Driven Engineering **(Modeling-Wizards 2010)**, Poster + Extended Abstract, Oslo, Noruega.

10.2.4 Actas de congresos nacionales

- González-Huerta, J., Insfrán, E., Abrahão, S. (2013). "*On the Effectiveness, efficiency and Perceived Utility of Architecture Evaluation Methods : A Replication Study*". En actas de: XVIII Spanish Conference on Software Engineering and Databases **(JISBD 2013)**, Madrid, España.
- Oliveira, R.P. De, Insfran, E., Abrahão, S., Gonzalez-Huerta, J., Blanes, D. (2013). "*A Feature-Driven Requirements Engineering Approach for Software Product Lines*". En actas de: 7th Brazilian Symposium on Software Components, Architectures and Reuse **(SBCARS 2013)**, Brasilia, Brazil.
- González-Huerta, J., Abrahão, S., Insfran, E. (2011). "*Un enfoque Multi-modelo para la Introducción Criterios de Calidad en el Desarrollo de Líneas de Producto Software*". En actas de: XVI Jornadas de Ingeniería de Software y Bases de Datos **(JISBD 2011)**, A Coruña, España.

10.2.5 Actas de talleres nacionales

- Gonzalez-Huerta, J., Insfran, E., Abrahão, S. (2010). "*Automatización de la Selección de Transformaciones Alternativas Basada en Atributos de Calidad*". En actas de: VII Taller sobre Desarrollo de Software Dirigido por Modelos **(DSDM 2010)**, Valencia, España. pp. 10–18.

10.2.6 Otras revistas no indexadas

- Abrahão, S., González-Huerta, J., Insfran, E., Ramos, I. (2012). "*Software Evolution in Model-driven Product Line Engineering*". **ERCIM-NEWS**, Vol. 88, Enero 2012, Numero Especial: Evolving Software, pp. 41–43.

10.2.7 Artículos en proceso de revisión

- Gonzalez-Huerta, J., Blanes, D., Insfran, E., Abrahão, S. (2013). *"Assuring Non-Functional Requirements in Software Product Lines Development: A Multimodeling Approach"* Capítulo del Libro "Innovative Quality Assurance Techniques in Software Products Lines", IGI-Global **(En revisión).**
- Gonzalez-Huerta, J., Pereira de Oliveira, R, Blanes, D., Abrahão, S., Insfran, E. (2014) *"A Case Study for the Validation of a Quality-Driven Product Architecture Derivation Method for Model-Driven Software Product Lines"*, 18th International Conference on Evaluation and Assessment in Software Engineering **(EASE2014)** Londres, Reino Unido. **ERA CORE Tier A (En revisión).**

10.2.8 Resumen de publicaciones

La Tabla 10.1 resume las publicaciones derivadas de esta tesis, destacando sus índices de calidad.

Tabla 10.1 Publicaciones de esta tesis doctoral

Publicaciones	*Indexada*	*Número*	*Lugar*
Revistas internacionales	JCR	2	IST, JUCS
Actas de congresos internacionales	CORE A	2	ISD
	CORE B	3	MODELS (x2), PROFES
	CORE C	1	WEBIST
	-	2	QuaTIC, Modelling Wizards
Actas de talleres internacionales	-	1	NFPinDSML
Actas de congresos nacionales	-	3	SBCARS, JISBD(x2)
Actas de talleres nacionales	-	1	DSDM
Otras revistas no indexadas	-	1	ERCIM
	Total	16	

10.3 Estancias de Investigación

- **Estancia pre-doctoral** en el LIACS (Leiden Institute of Advanced Computer Science), Leiden Univeristy, Leiden, Holanda, bajo la supervisión del Prof. Michel Chaudron, de Abril de 2011 a Julio de 2011 (4 meses). Subvención para la movilidad de alumnos en enseñanzas universitarias oficiales de máster para el curso académico 2010-2011.

10.4 Becas y galardones

- **Beca pre-doctoral** (2011-2014) Ayuda de formación de personal investigador, dentro del programa VALi+d, Conselleria d'Educació, Generalitat Valencia, (Ref. ACIF/2011/235).
- **Beca de Movilidad** (Abril 2011-Julio 2011) Subvención para la movilidad de alumnos en enseñanzas universitarias oficiales de máster para el curso académico 2010-2011 (Ref. MAS2010-00381).

- **Galardón al 3er mejor artículo:** "*A Feature-Driven Requirements Engineering Approach for Software Product Lines",* 7th Brazilian Symposium on Software Components, Architectures and Reuse **(SBCARS 2013)**, Brasilia, Brazil.

10.5 Trabajos futuros

Esta tesis no representa el fin de esta línea de investigación. Si bien se han alcanzado muchos objetivos y concluidos muchas de las actividades previstas en este trabajo de investigación, al mismo tiempo se han abierto otras muchas, algunas de ellas en curso y otras que se han de acometer en un futuro próximo y que contribuirán a extender este trabajo de investigación en diferentes aspectos. El principal objetivo a corto y medio plazo se centra en afrontar las tareas pendientes hasta liberar la solución definitiva, de acuerdo con el modelo de investigación y transferencia de tecnología aplicado en esta tesis. En esta tesis se han cubierto las seis primeras fases de la metodología de investigación propuesto por Gorschek et al. (2006), quedando pendiente la validación realista (y las posibles iteraciones a fases anteriores) y el lanzamiento definitivo de la solución que acarreará actividades de transferencia de tecnología a la industria.

Las principales cuestiones que están siendo abordadas en este momento son:

- Implementación del conjunto completo de transformaciones que permiten obtener el modelo de resolución CVL a partir del modelo de configuración.
- Mejora del soporte automatizado a la configuración y de la validación de consistencia: mejorar los mecanismos de recomendación y la validación de consistencia para entidades individuales.
- Mejora de la herramienta con respecto a aspectos de usabilidad, resultado de los comentarios de los participantes en el estudio del caso a las preguntas abiertas respecto a la herramienta.
- Incorporación en la herramienta de funcionalidades de filtrado para la configuración para hacer frente a una de los requisitos de la derivación establecidos por Rabiser et al. (2010).
- Mejorar el método de especificación de impactos y analizar otros métodos de optimización multi-objetivo para seleccionar la mejor alternativa en una decisión.

- Analizar alternativas a la aproximación *Lazy* de validación de consistencia para los modelos arquitectónicos incorporando dicha validación en fase de configuración con el fin de no permitir configuraciones que posteriormente no puedan ser satisfechas desde el punto de vista arquitectónico.
- Realización de estudios empíricos con profesionales de la industria con experiencia en evaluaciones arquitectónicas y/o en el desarrollo de líneas de producto software.

Algunas extensiones inmediatas al trabajo son:

- Análisis de la aplicabilidad de las transformación arquitectónicas en escenarios de reconfiguración dinámica de arquitecturas empleando como artefactos de entrada modelos@runtime (Blair et al. 2009).
- Análisis de la aplicabilidad de QuaDAI para la evaluación y mejora de arquitecturas *cloud*.
- Extensión del multimodelo con la vista de requisitos para líneas de productos, en línea con el trabajo conducido por David Blanes, fruto de la colaboración con el grupo RISE de la Universidade Federal da Bahía (UFBA), Salvador, Brasil, con el objetivo de permitir automatizar o guiar la configuración en base a las especificaciones de requisitos en FeDRE (Oliveira et al. 2013).
- Extensión del multimodelo con una vista de costes para poder expresar relaciones de impacto de las configuraciones sobre el coste, añadiendo una nueva dimensión al proceso.
- Análisis de la aplicabilidad de la gestión de la variabilidad interna de servicios *cloud* mediante CVL para obtención de modelos de coreografía y orquestación con reconfiguración dinámica basada en acuerdos de nivel de servicios (*Service Level Agreement-SLA*).

Capítulo 11

Conclusions and Further Work

This chapter reviews the research objectives, the main findings and the conclusions, while we analyse until what extent the research goals had been attained. We also present the contributions of this work to the research community by means of publications, research stays, scholarships and awards. Finally, we draw the possible extensions and further research related to this work.

11.1 Conclusions

The software product lines approach emerged with the aim of improving software development processes so as to reduce costs and substantially enhance productivity and product quality.

Around the software product line had emerged various product derivation methods, although some of these methods do not properly integrate quality criteria in the derivation process. Is surprising that being the quality one of the main reasons for the adoption of the product line approach, it has been neglected in such a critical and complex process. In addition, in these cases in which the derived product architecture is evaluated after its derivation, this evaluation is carried out by using software architecture evaluation methods that had not been specially defined for being applied in software product line development.

In this PhD thesis we have defined and validated a method for the derivation, evaluation and improvement of product architectures in model-driven software product line development environments. The extent until which each of the objectives proposed has been achieved is analysed in the following subsections.

11.1.1 Analysis of the product architecture derivation methods

With regard to the analysis of the derivation methods, we made an analysis of approaches dealing with the product architecture derivation in model-driven software product line environments. The main finding is that the quality attributes in these processes has not been properly covered on the research works defined in the area.

Only two methods of the sixteen methods analysed addressed quality aspects when deriving the product architecture. In addition, the majority of these approaches were tailored for a specific set of quality attributes or for a specific architectural viewpoint or architectural description languages, and is not possible to find on the literature generic approaches that integrate quality aspects in the product architecture derivation processes.

The majority of the approaches partially cover the consistency validation nor in the configuration models taken as input, nor in the derived architectural models. This allows the generation of products that do not fulfil the architectural and/or variability constraints.

11.1.2 Analysis of the product architecture evaluation methods

With regard to the analysis of the software architecture evaluation and improvement methods, we analyzed the software architecture evaluation methods defined for software product lines. The main findings of this analysis is that the evaluation of product architectures in model-driven software product line development processes is not sufficiently covered by methods that allow the evaluation of product architectures regardless the set of attributes or the nature of the architecture being evaluated.

In addition we observed a lack of metric-based product architecture evaluation methods suitable to be applied at derivation time. In software product line development variability in quality attributes levels is also possible and thus the metric-based evaluation methods to be applied at derivation time will allow us to analyse whether the measured values for a specific configuration are within the limits established for the product line or they do not.

The derivation process is a critical and complex time consuming critical task (Griss 2000; Rabiser et al. 2011) and the evaluation of quality attributes after the derivation allows us to detect potential problems, the detection of which, if delayed, will impact on the cost and on the schedule of the project in a much larger manner.

11.1.3 Definition of the multimodel with product-line viewpoint support

Tackling this objective we have defined a multimodel that supports different viewpoints and that allows defining relations of various types and with diverse semantics, between different views. This multimodel is going to enable the process of derivation, evaluation and architectural improvements, integrating the quality viewpoint as an active artifact in the development process. We have defined six relations types between the elements of the different viewpoints based on the types of relations between models present in the literature.

Four viewpoints have been defined (variability viewpoint, architectural viewpoint, quality viewpoint and transformation viewpoint) to represent the problem of derivation, evaluation and architectural improvement; additionally, we defined six relationships between elements of these viewpoints of two concrete types (impact relations and decomposition relations) and formalized their semantic for the relations involved in the derivation process. The addition of the architectural variability viewpoint to the multimodel (expressed by means of the CVL standard for variability description) allows us to express a new level of abstraction between the models of external variability of the product line and architectural models, leading to an explicit representation of the internal variability on the architectural level, without the need to modify, neither the representation of the external variability of the product line with the proper information of the architecture, nor with the architectural representation with the variability information. Additionally the use of CVL as the specification language of the architectural variability enables us the generalization of the evaluation method to a point when it is applicable in a generic form to any architecture described with any MOF-based language.

The multimodel supports all the phases and activities of the QuaDAI method thorough the following structures:

- The relations between characteristic, non-functional requirements and quality attributes assist the configuration process, recommending the possible modifications of the configuration in the order of preselected priorities and validate the consistency of the configuration through the

operationaliziation of the validation of the given relations by means of OCL restrictions.

- The expressiveness of the quality view enable us to describe the non-functional requirements of the product line and the developed products; describe a way in which those requirements are evaluated and, consequently, validate automatically the degree of compatibility.
- The relations between the points of architectural variation points, features, non-functional requirements and quality attributes allow us to drive the derivation process to elect the most suitable variants of the architecture (within any of the architectural viewpoints).
- The transformation viewpoint itself and the relations between alternative transformation and the quality attributes allow us to guide the transformation process in cases when the non-functional requirement is violated.

To enable the support for the multimodel we defined a two-level architecture that serves as an example of the use of the multimodel that goes beyond the set of the viewpoints used in this work or its application in the software product line development; it allows the definition of multimodels in a generic manner, between two or more models, giving support through the multimodel infrastructure for every set of viewpoints, given the metamodel is compatible with MOF.

11.1.4 Definition of the product architecture derivation, evaluation and improvement process

This objective was achieved through the definition of QuaDAI, an integrated, transformation based, derivation, evaluation and architecture improvement method. The method describes the set of activities to be performed in the software engineering phase to obtain, from the predefined requirements, an architecture consistent with the functional and non-functional requirements. Additionally we defined the activities to perform in the domain engineering phase to encompass the information about the relations between the elements of different views.

The method's derivation phase covers the greater portion of the key activities, as identified by (Rabiser et al. 2011):

- The preparation for the derivation: the method supports the client requirement specification (in the multimodel through the variability and quality viewpoints), the definition of the base configuration, the client requirements mapping (the chosen configuration model not only enables us to map the requirements, but it also facilitates traceability mechanisms) and to support the product configuration by means of recommendation mechanism and consistency validation.
- Configuration and derivation: the method covers this key activity with the configuration selection assistance, consistency validation and an automatic derivation process.
- Additional development and testing: The method integrates the consistency validation of the derivate models (through CVL validation) and the non-functional requirements verification though the evaluation and improvement phase.

The derivation phase is supported in this method by means of: i) the process definition, which consists of the activities set, roles and artifacts to be used in each activity; ii) the definition of the configuration model as the projection of the quality and variability viewpoint; iii) the formalization of the relations between quality and variability viewpoints of the multimodel, that allows the consistency validation of the chosen configuration; and iv) the definition of the transformation process that allows us to obtain the CVL resolution model, based on the configuration model.

The evaluation and improvement phase is supported in the method by means of: i) the process definition; ii) the artifact definition ii) the definition of an evaluation mechanism for analyzing the fulfillment degree of each non-functional requirement; and iii) the design guidelines for the architectonical transformation definition in which the choice between the alternative architectonical transformations is realized based on the quality attributes.

11.1.5 Empirical validation of the proposed method

The proposed method has been empirically validated through a case study carried out on Spain and on Brazil, and through a family of five experiments with internal and external replications, carried out in Spain, Italy and Paraguay.

We have empirically validated the product architecture derivation process through a case study, carried out in two sessions, one at Universitat Politècnica de València and the second at Universidade Federal da Bahia (Brazil), which

involved 14 participants, Master and PhD students and software development practitioners, some of them with previous background in software product lines on the industry. In the case study we evaluated the perceived ease of use, perceived utility and intention to use of the participants when they apply QuaDAI in a product architecture derivation process of a automotive product line.

The qualitative analysis has shown that, in general, the application of the method allowed the participants to obtain a product architecture that meets both the functional and non-functional requirement of the product under development. The quantitative analysis has shown that the participants perceive the method as easy to use, useful and that they have also the intention to use the method in the future.

From a research point of view, the case study has allowed us to analyze the feasibility of the method in a real derivation process, to model a real case and to analyze the completeness and correctness of the solutions proposed, and last but not least, to obtain feedback from the participants who applied the method, whose comments will help us improving the method and the infrastructure which supports the multimodel.

From a practical point of view, this is the first study, with a reduced number of participants, with heterogeneous profiles which only yields preliminary results with regard to the perceived ease of use, perceived usefulness and intention to use of the process. More replications with bigger and homogenous groups are required so as to analyze possible interactions of factors (i.e., the subject's experience or ability).

We have empirically validated the proposed process for the derivation and improvement of product architectures through a family of five experiments, with replications, both internal and external, in Spain, Italy and Paraguay. They involved 108 subjects, grade level students of computer science, master level students of software engineering and PhD students. In those experiments we compared the effectiveness, the efficiency, the perceived ease of use, the perceived usefulness and the intention to use of the participants that employed QuaDAI, in comparison with ATAM, which is a widely used software architecture evaluation method.

The quantitative data analysis and the posterior meta-analysis, performed with the objective of aggregating the empirical findings, have shown that there exists a positive and significantly important effect in all the variables studied, associated with the use of QuaDAI as the architecture evaluation method in the development of model-driven product lines.

From the research point of view, the experiment family has proven to be an important mean of feedback on the question of improvements to QuaDAI (for instance the improvement of the assignment of the relative importance of the RNF set, or reuse the results of the measurement process to select architectural transformations). As far as we know, this is the first empirical study that shows any evidence of the utility of the software architecture evaluation method for a model-driven software development process.

From a practical point of view, we are aware that this study only provides some preliminary results on the subject of the utility of QuaDAI as the software architecture evaluation. They must be contrasted with any future replication in various contexts, given the need to confirm if the same results are produced by more experienced participants and with new experimental objectives, of higher complexity.

11.2 Related publications

The work described in this thesis had been published in **two ERA-CORE Tier A conferences**, **three ERA-CORE Tier B conferences** (**2 in** the **MODELS** conference, the most relevant conference in the model-drive software development community)**, one ERA-CORE Tier C conference,** one IEEE international conference out of the ERA-CORE (QuaTIC), one international workshop (a **MODELS conference satellite event**), three national conference (one of them awarded with **the third best paper award in the SBCARS conference**), one national workshop and one publication on the ERCIM-News pair review divulgation journal. Nowadays we have also three ongoing papers, two of them sent to JCR journals, which are booth under review and a book chapter whose proposal has been accepted. The following subsections provide details about these publications:

11.2.1 Refereed International Indexed Journals (JCR)

- Gonzalez-Huerta, J., Insfran, E., Abrahão, S., Scanniello, G. (2013). *"Validating a Model-Driven Software Architecture Evaluation Method for Software Product Lines: A Family of Experiments"* **Information and Software Technology**, Impact Factor 1.507 (JCR 2010) **(Accepted with minor revisions).**

- Oliveira, R.P. De, Blanes, D, Gonzalez-Huerta, J., Insfran, E., Abrahão, S., Cohen, S., Santana, E. (2013) *"Defining and Validating a Feature-Driven*

Requirements Engineering Approach", **Journal of Universal Computer Science**, Impact Factor 0.762 (JCR 2010) **(Accepted).**

11.2.2 Refereed International Conferences

- González-Huerta, J., Insfran, E., Abrahão, S., McGregor, J.D. (2013). "*Architecture Derivation in Product Line Development through Model Transformations*". In proceedings of 22nd International Conference on Information Systems Development **(ISD 2013)**, Seville, Spain. **ERA CORE Tier A.**
- González-Huerta, J., Insfrán, E., Abrahão, S. (2013). "*Defining and Validating a Multimodel Approach for Product Architecture Derivation and Improvement*". In proceedings of 16th International Conference on Model-Driven Engineering Languajes and Systems, **(MODELS 2013)**, Miami, USA, pp. 388–404. **ERA CORE Tier B.**
- Insfran, E., Abrahão, S., González-Huerta, J., McGregor, J.D. (2012). "*A Multimodeling Approach for Quality-Driven Architecture Derivation*". In proceedings of 21st International Conference on Information Systems Development **(ISD 2012)**, Prato, Italy. **ERA CORE Tier A.**
- González-Huerta, J., Insfran, E., Abrahão, S. (2012). "*A Multimodel for Integrating Quality Assesment in Model-Driven Engineering*". In proceedings of 8th International Conference on the Quality of Information and Communications Technology **(QuaTIC 2012)**, Lisbon, Portugal.
- Insfran, E., Gonzalez-Huerta, J., Abrahão, S. (2010). "*Design Guidelines for the Development of Quality-Driven Model Transformations*". In proceedings of 13th International Conference on Model Driven Engineering Languages and Systems **(MODELS 2010)**, Oslo, Norway. pp. 288–302. **ERA CORE Tier B.**
- Gonzalez-Huerta, J., Blanes, D., Insfran, E., Abrahão, S. (2010). "*Towards an architecture for ensuring product quality in model-driven software development*". In proceedings of 11th International Conference on Product Focused Software **(PROFES 2010)**, Limerick, Ireland. pp. 28–31. **ERA CORE Tier B.**

- González-Huerta, J., Insfran, E., Abrahão, S. (2010). *"Defining an Architecture for Quality-Driven Model Transformations"*. En actas de 1st International Master Class on Model-Driven Engineering **(Modeling-Wizards 2010)**, Poster + Extended Abstract, Oslo, Noruega.
- Botto, M., Insfran, E, Gonzalez-Huerta, J. (2014). *"Are Model-Driven Techniques Used As A Means To Migrate SOA Applications To Cloud Computing?"* In proceedings of 10th International Conference on Web Information Systems and Technologies **(WEBIST 2014)**, Barcelona, Spain. **ERA CORE Tier C**.

11.2.3 Refereed International Workshops

- González-Huerta, J., Insfran, E., Abrahão, S., McGregor, J.D. (2012). "*Non-functional requirements in model-driven software product line engineering*". In Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages – **(NFPinDSML 2012)**, Innsbruck, Austria.
- González-Huerta, J., Insfran, E., Abrahão, S. (2010). *"Defining an Architecture for Quality-Driven Model Transformations"*. In proceedings of 1st International Master Class on Model-Driven Engineering **(Modeling-Wizards 2010)**, Poster + Extended Abstract, Oslo, Norway.

11.2.4 Refereed National Conferences

- González-Huerta, J., Insfrán, E., Abrahão, S. (2013). "*On the Effectiveness, efficiency and Perceived Utility of Architecture Evaluation Methods : A Replication Study*". In proceedings of XVIII Spanish Conference on Software Engineering and Databases **(JISBD 2013)**, Madrid, España.
- Oliveira, R.P. De, Insfran, E., Abrahão, S., Gonzalez-Huerta, J., Blanes, D. (2013). "*A Feature-Driven Requirements Engineering Approach for Software Product Lines*". In proceedings of 7th Brazilian Symposium on Software Components, Architectures and Reuse **(SBCARS 2013)**, Brasilia, Brazil.

- González-Huerta, J., Abrahão, S., Insfran, E. (2011). "*Un enfoque Multi-modelo para la Introducción Criterios de Calidad en el Desarrollo de Líneas de Producto Software*". In proceedings of XVI Jornadas de Ingeniería de Software y Bases de Datos **(JISBD 2011)**, A Coruña, España.

11.2.5 Refereed National Workshops

- Gonzalez-Huerta, J., Insfran, E., Abrahão, S. (2010). "*Automatización de la Selección de Transformaciones Alternativas Basada en Atributos de Calidad*". In proceedings of VII Taller sobre Desarrollo de Software Dirigido por Modelos **(DSDM 2010)**, Valencia, España. pp. 10–18.

11.2.6 Other Journals

- Abrahão, S., González-Huerta, J., Insfran, E., Ramos, I. (2012). "*Software Evolution in Model-driven Product Line Engineering*". **ERCIM-NEWS**, Vol. 88, January 2012, Special Issue: Evolving Software, pp. 41–43.

11.2.7 Ongoing papers

- Gonzalez-Huerta, J., Blanes., D., Insfran, E., Abrahão, S (2013). *"Assuring Non-Functional Requirements in Software Product Lines Development: A Multimodeling Approach"* Capitulo del Libro Innovative Quality Assurance Techniques in Software Products Line, IGI-Global **(Under review).**
- Gonzalez-Huerta, J., Pereira de Oliveira, R, Blanes, D., Abrahão, S., Insfran, E. (2014) *"A Case Study for the Validation of a Quality-Driven Product Architecture Derivation Method for Model-Driven Software Product Lines"*, 18th International Conference on Evaluation and Assessment in Software Engineering **(EASE2014)** Londres, Reino Unido. **ERA CORE Tier A (Under review).**

11.2.8 Summary and quality of the publications

This section summarizes all the publications by highlighting their evidences of quality.

Publications	Indexed	Number	Venue
International Journal	JCR	2	IST, JUCS
Refereed international conferences	CORE A	2	ISD
	CORE B	3	MODELS (x2), PROFES
	CORE C	1	WEBIST
	-	2	QuaTIC, Modeling Wizards
Refereed international workshops	-	1	NFPinDSML
Refereed national conferences	-	3	SBCARS, JISBD(x2)
Refereed national workshops	-	1	DSDM
Other journals	-	1	ERCIM
	Total	16	

11.3 Research stays

- **Pre-doctoral stay** at LIACS (Leiden Institute of Advanced Computer Science), Leiden University, Leiden, the Netherlands, under the supervision of Michel Chaudron, from April 2011 to July 2011 (4 months). Granted by the Spanish Ministry of Science and Innovation (under the program "Becas para la movilidad de alumnos en enseñanzas universitarias oficiales de máster para el curso académico 2010-2011").

11.4 Grants and awards

- **Predoctoral fellowship (**2011-2014) Ayuda de formación de personal investigador, dentro del programa VALi+d, Conselleria d'Educació, Generalitat Valencia, (Ref. ACIF/2011/235).
- **Mobility grant** (April 2011 - July 2011) Subvención para la movilidad de alumnos en enseñanzas universitarias oficiales de máster para el curso académico 2010-2011 (Ref. MAS2010-00381).
- **3rd best paper award:** "*A Feature-Driven Requirements Engineering Approach for Software Product Lines",* 7th Brazilian Symposium on

Software Components, Architectures and Reuse **(SBCARS2013)**, Brasilia, Brazil.

11.5 Further works

This PhD thesis does not represent the end of this research work. Although we have achieved most of the objectives defined and completed the majority of the envisaged tasks, at the same time we have opened many issues, some of them which are now in progress, many other which we will have to undertake in the near future and which will extend this research work in many different aspects. The main objective in the short and medium term focuses on addressing the remaining tasks to release the final solution according to the technology transfer model applied in this PhD thesis. In this thesis we have covered the first six stages of the research methodology proposed by Gorschek et al. (2006). The remaining tasks are the realistic validation (and the possible iterations to earlier phases) and the final release of the solution, which, probably, will entail technology transfer activities associated to its implementation in the industry.

The main issues being addressed at this time are:

- The implementation of the complete set of transformations which will allows us to obtain the CVL resolution model from the configuration model.
- Improvement of the configuration support and consistency validation support: to improve the recommendation and the consistency checking for individual entities.
- Improvement of the tool, with respect to usability aspects, based on the comments of the participants in the case study to the open questions regarding the tool.
- Incorporate of filtering capabilities in the tool for supporting the configuration task so as to meet one of the derivation requirements established by Rabiser et al. (2010).
- Improve the impact specification method, and to analyze other multi-objective optimization methods for selecting the best alternative from a design decision.

- To analyze alternatives for the *Lazy* consistency checking mechanism defined for the CVL resolution models, incorporating this validation to the configuration step so as to restrict those configurations that later would be rejected in the CVL tool.
- To conduct new replications of the controlled experiments with practitioners with experience in software architecture evaluations and/or software product line development.

The extensions to this research work are (among others):

- Analyze the applicability of the architecture transformation approach in the dynamic reconfiguration of software architectures taking as inputs models@runtime (Blair et al. 2009).
- Analyze the applicability of QuaDAI to the evaluation and improvement of *cloud* architectures.
- To extend the multimodel with the requirements viewpoint for software product lines aligned with the work being conducted by David Blanes as product of the collaboration with the RISE research group from the Federal University of Bahia, Salvador de Bahia, Brasil. The objective of this extension is to automate or drive the configuration based on the requirements specifications using the FeDRE approach (Oliveira et al. 2013).
- To extend the multimodel with a costs viewpoint to be able to express impact relationships of the configurations on the cost, adding a new dimension to the problem.
- To analyze the applicability of the approach for the internal variability management of *cloud* services in order to obtain choreography and orchestration models with dynamic reconfiguration, based on service level agreements.

Apéndice A: Case Study Materials

A.1 Case Study Booklet

A.1.1 Problem Description

The CarCarSPL is Software Company which develops control systems for the automotive domain. They integrate sensors and computation nodes from different vendors and on top of that, they build their vehicle control system software. For managing the huge possible number of configurations, they are applying the software product line approach.

Now, one of their customers, BWM, a multinational automotive company is opening new market niches and has started to modify one of their cars, their top-class W5 station-wagon to become an ambulance (called W5-Ambu).

The Application Requirements Engineer has captured the customer requirements, and now we should configure the "best" possible vehicle control system.

A.1.2 System Requirements

The first point about the requirements is that they do not have, initially, budget constraints, the Control System's cost is going to be added to their production costs, so what they need is a configuration that fulfills their requirements.

The system should integrate the highest possible number of security and multimedia systems but, at least to have a Control System that has ABS, traction control and the stability control, the GPS-Navigator and the cruise control are also desirable. The description of the different features can be found in Annex I.

They have also a lot of non-functional requirements. Our customer plans to sell their ambulances to EU countries, and there is an EU council regulation which establishes that ambulances should pass some safety tests and their control system should be considered as "Safety Critical System". This implies that the Vehicle Control System should have minimum levels of reliability, and some constraints are also defined for the system's performance.

The reliability characteristic in our systems is broken down in:

- **Fault Tolerance:** the degree to which a system, product or component operates as intended despite the presence of hardware or software faults.
- **Maturity:** the degree to which a system meets needs for reliability under normal operation.

We measure the fault tolerance with, among others, the probability of failure of each system[25] which is extracted from the architecture.

The probability of failure of our systems usually is below 0.00006 but in this specific case the requirements state that the probability of failure should be below 0.00004.

For achieving the levels of reliability to meet the safety critical system consideration, some non-functional requirements are defined related to the reliability of the system. Safety Critical Systems should also have a maximum level of CPU utilization so as to assure that, in every single situation, the systems are able to compute a response in less than a specific time (a.k.a Latency time) that also is critical for safety critical embedded systems. In addition, since the system is going to be deployed in a new set of distributed high-performance set of embedded computation nodes, the memory needs to be also reduced. The performance in our systems is broken down in:

- **Time behavior:** the response and processing times and throughput rates of a system when performing its function, under stated conditions in relation to an established benchmark.
- **Resource Utilization:** the amounts and types of resources used when the software performs its function under stated conditions in relation to an established benchmark.

We measure the time behavior of our system among others with the latency time. Latency time measures the time elapsed between the reception of an input event and the computation of the results. The resource utilization is measured by means of the CPU utilization and the memory consumption.

[25] We will use the term system to refer also to the system (ABS, Traction Control etc.) that integrates the Vehicle Control System.

In this specific case, the systems should have a latency time that goes below the common specific level established for the products of the Vehicle Control System which is 55 μs. our customer needs, to accomplish the regulations, the systems to be below 50 μs.

The Resource Utilization is also critical. EU regulations establish that a safety critical system should have a CPU utilization of less than 80%. The threshold for our vehicle control systems SPL establishes a maximum CPU utilization of 85% and all the possible combinations are below the 85% level. However the most CPU-intensive features will make the vehicle control system as a whole to be consuming more than 80%, so probably some features will need to be discarded. You have also to consider that the CPU-Utilization can be improved by upgrading the hardware in which the vehicle control system is deployed but some other requirements cannot be achieved in such a simple way (e.g., fault tolerance).

The last quality characteristic is a domain specific quality characteristic for the automotive industry, which are the different levels of safety that a car offers to the passengers. In the European Union there are different levels of Safety and are certified through the EuroNCAP safety level. CarCar only develops security systems for EuroNCAP level 4 and EuroNCAP level 5. The different levels of EuroNCAP are obtained by the crash tests results and the number and reliability of the safety systems available in the car. In this specific case the requirement specification specifies that EuroNCAP level 4 is mandatory, but it would be also desirable to achieve the EuroNCAP level 5.

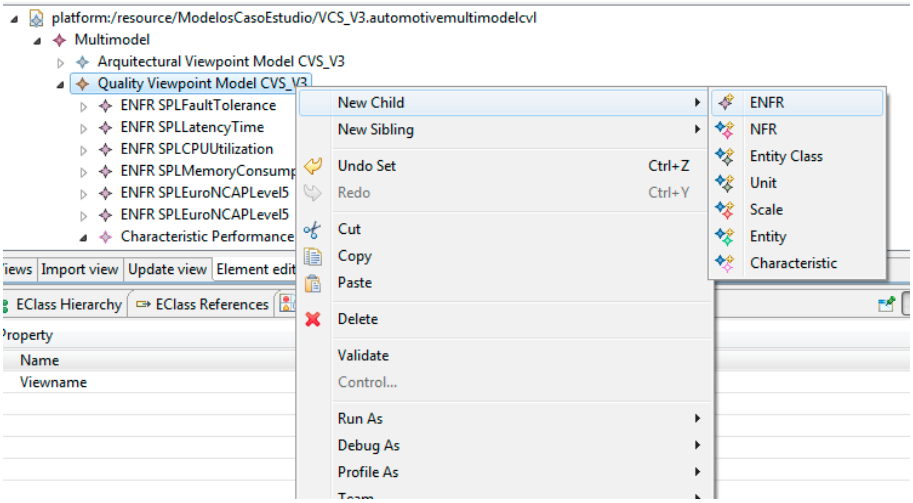
A.1.3 Experimental Tasks

Task 1: Product NFRs definition

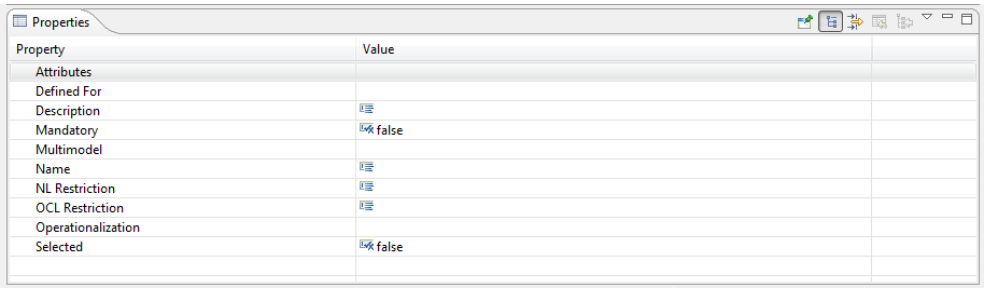
Define the Non-Functional requirements that are specific for the product under development. For doing that you should consider the non- SPL functional requirements aforementioned and define new ones when required[26]. Remember that, if you define a new NFR that goes further the definition of a SPL's please check both when you configure the product. You have to introduce a new ENFR (Extended NFRs defined in the multimodel) in the QuaDAI tool, following the instructions described below:

[26] Please, do not change the NFR definition, if you need additional requirements, define the ones you need.

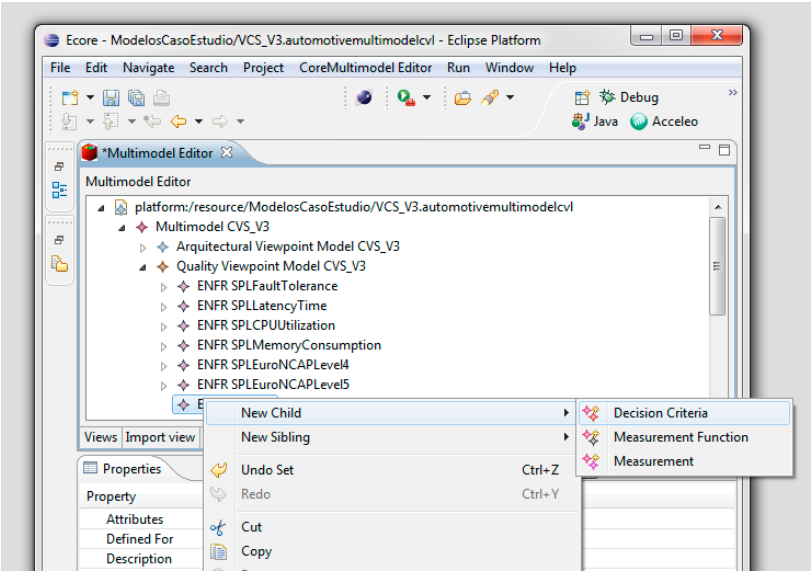
i) Selectthe Quality Viewpoint Model, and in the contextual menu that appears when right-clicking, select ENFR.



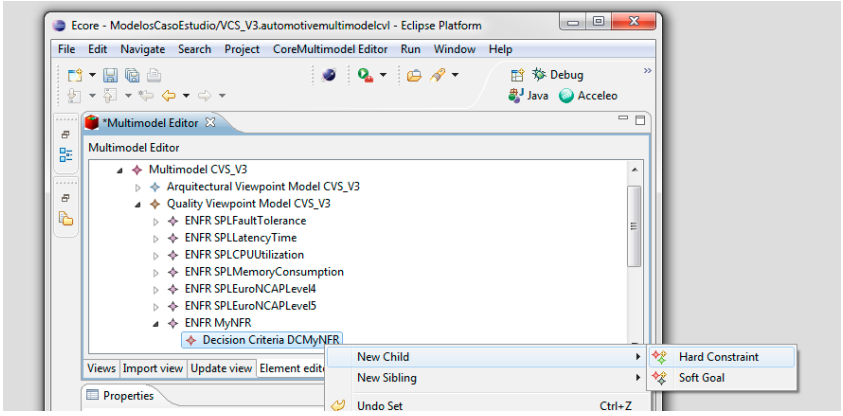
ii) Fill in the data, the Attributes, Description, Name and NL Restriction. Be sure that the multimodel field shows the value Multimodel and the name of our multimodel. Otherwise please in the drop-down menu of the field select the proper value. The mandatory field is not required and the selected field should remain false until the configuration task.



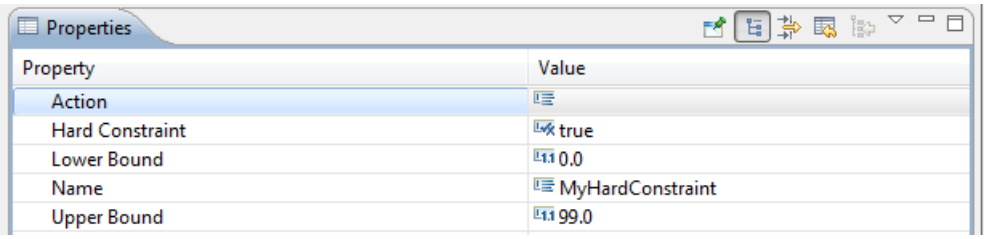
iii) The next step is to define the decision criteria that in which we will define the thresholds for our NFR. For the decision criteria the only thing I should enter is the name field.



iv) Now we should define the NFR's threshold as a hard constraint (if it is something that the product should accomplish) or a soft goal (if it is a desirable property) as shown.



iv.a. <u>Hard Constraints:</u> If we are defining a hard constraint we have to define the name, the Lower and Upper Bound for defining the range of acceptable values. For example the following threshold describes a NFR whose values are allowed to range from 0 to 99.



iv.b. Softgoals: If we are defining a soft goal, we have to define the name, the orientation (positive for the-higher-the-better NFRs and negative for the-lower-the-better ones). For example the following threshold a positive threshold with no minimum value.

Once you have defined the NFRs in the multimodel, please return to the web browser and fill in the questionnaire with the information related to the task 1. You should introduce the product specific NFRs you have defined here.

Task 2: Configuration

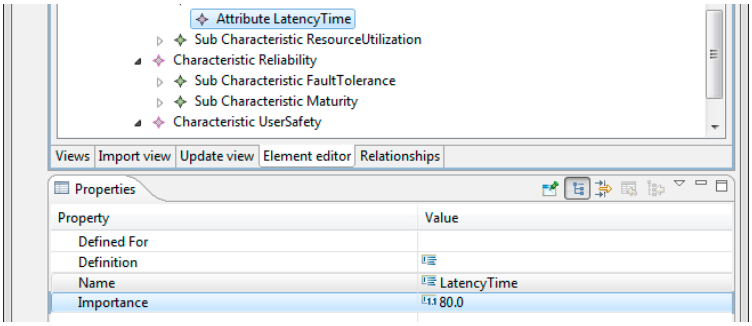
The next step is to define the configuration of the product. You have to select the set of features, the set of NFRs of the W5-Ambu and the prioritization of the quality attributes you want to take into account for solving the eventual design decisions that would appear. The details on how each of these processes should be performed are described below. The natural process could be firstly select the features, then validate the consistency of that features configuration. You will define a valid feature configuration from the point of view of the variability viewpoint. Next you will select the NFRs and probably obtain a configuration that is not valid from the point of view of quality and variability (there are some features that realize non-functional requirements and, if a NFR is selected, and you will need to reconfigure and again and again. Finally you will introduce the quality attributes priority and you will add a third constraint that will make you to re-do the previous steps.

The recommended workflow is as follows:

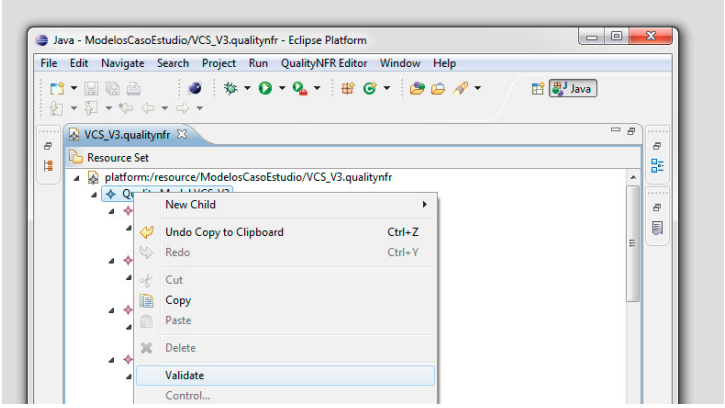
- **Task 2.1:** First, establish the priority of the quality attributes, by introducing a value ranging from 0 to 1. This value expresses the fraction of 1 that you assign to each attribute (1 for critical 0 for trivial). If a quality attribute is not important (or less important) keep it at 0 level. You have to take into account also that you should leave some

degrees of freedom for i) these quality attributes that are impacted negatively by other prioritized quality attributes or ii) these quality attributes that, whereas having certain importance, do not reflect constraints or requirements on the product (the maps of impacts between attributes is described in annex II).

These priorities are introduced in the Tool on the Importance Field of the EAttribute[27].



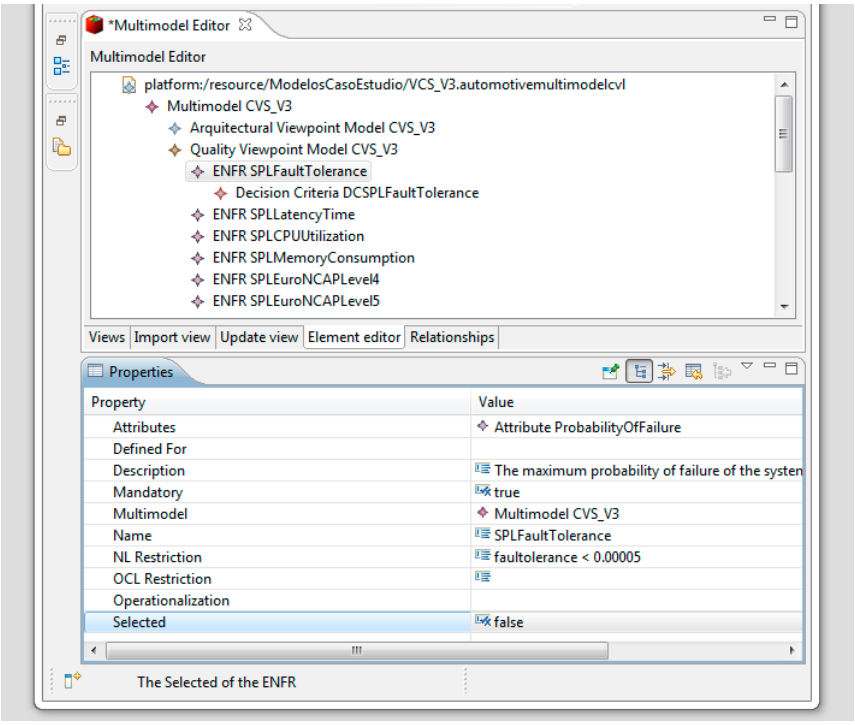
Each time you need to validate whether your priorities are consistent with the impacts described in annex II or not, you can validate the configuration through the validation menu item in the contextual menu.



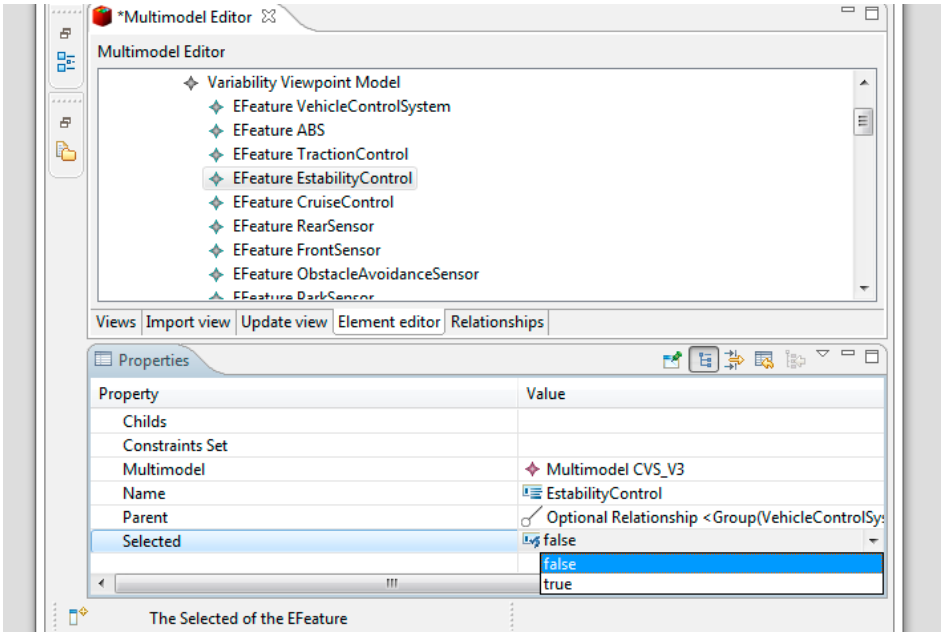
- **Task 2.2:** Second, select the SPL and the product specific NFRs the product has to fulfill. Remember, if a product specific NFRs restricts a

[27] This is only an example from other configuration; please use your own values.

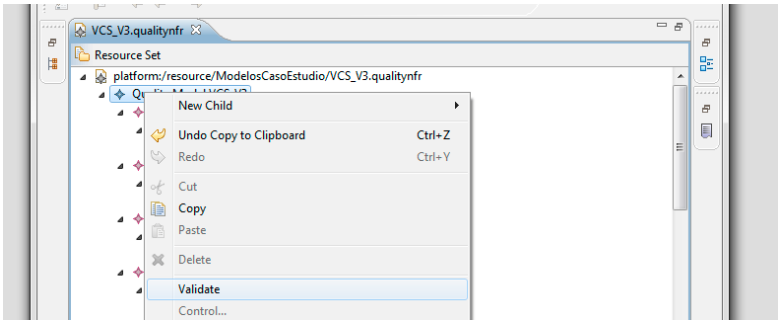
SPL's NFR, please select both (this will help because the relationships with the features are, at the moment, defined with the SPL's NFR not with the product-specific NFRs).



- **Task 2.3:** Finally, select the features you want the product to have by checking them in the tool.



You will have to, validate the consistency as a whole during the process by using the validate option of the contextual menu over an entity in the multimodel, to check that the features that you select meet also the restrictions defined among quality attributes and non-functional requirements through the validation menu item in the contextual menu. These relationships are described in the annex III.

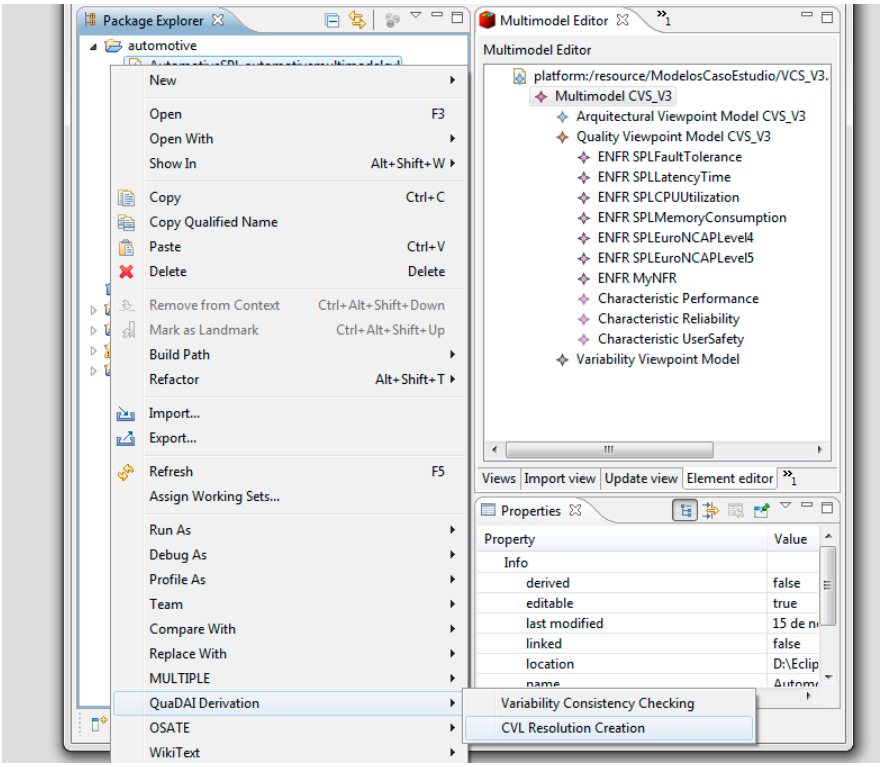


We are aware that the Features are the first-class citizens in SPL and the idea is to select the maximum number of features (functionality), but we first define the restrictions so as to simplify the process. You first prune the search space and then decide which features you want to select. Our CarCarSPL's feature model can produce (without NFR and Quality attributes restrictions) 305 valid

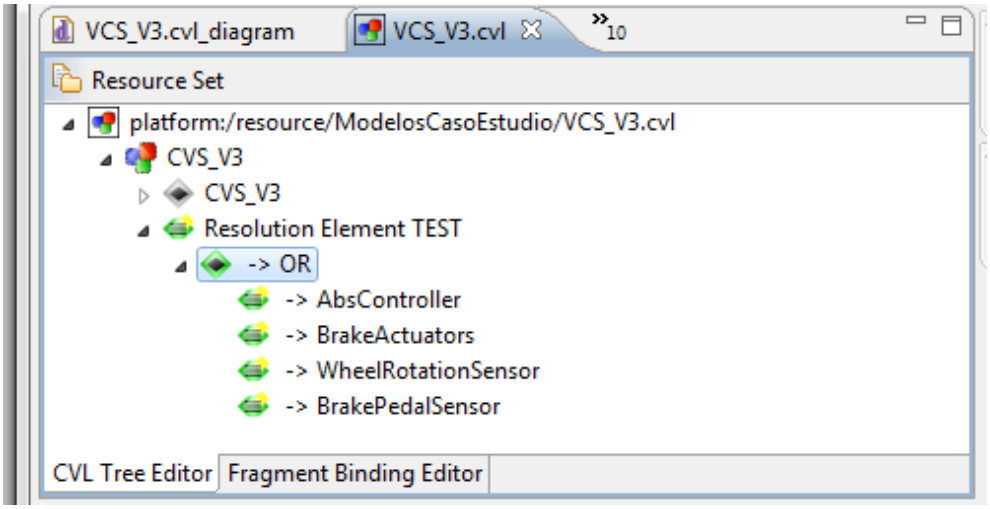
products, and thus, it is better to first constrain the problem space and then select the ones that best fit the functional requirements.

Task 3: Derivation

Once you have obtained the product configuration that fit the customer requirements, what we do is to derive the CVL resolution model as an intermediate step before obtaining the final architecture. What we simply do is to right-click on the project explorer and to select the CVL Resolution Creation. This will ask were to put the CVL file, and once introduced, it generates the CVL Resolution Model.



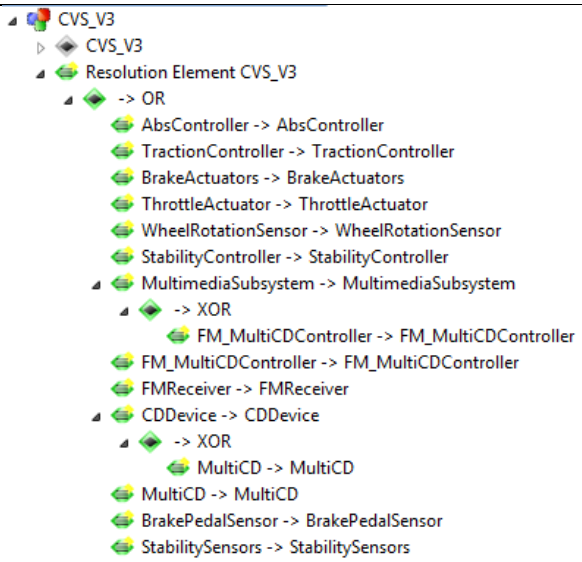
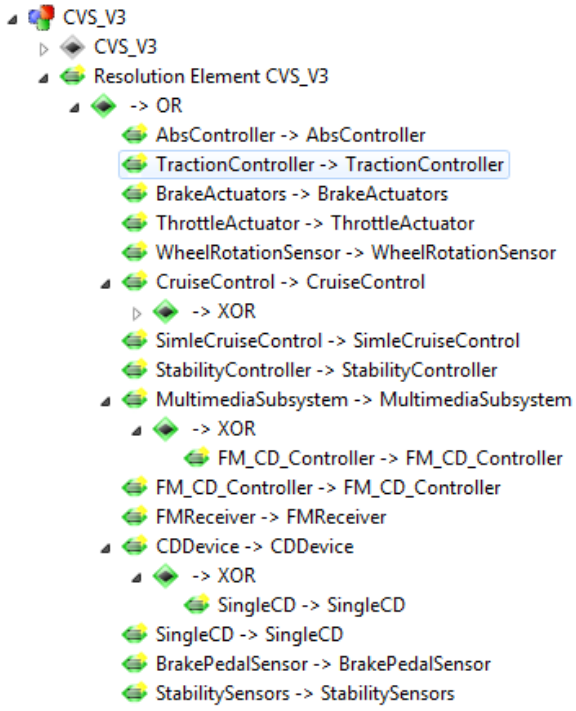
Next we show a test case in which we have selected only the ABS Feature and, with the relationships among features and variability points in the architectural view we can, by applying model transformation techniques of obtaining the CVL Resolution Model.



This is an important result because CVL allows to, with the resolution model, obtain the final models of the product architecture (in general CVL works with any kind of model).

Now you have to introduce in the online questionnaire the results obtained:

Resolution	***Items***
Configuration 1	CVS_V3 CVS_V3 Resolution Element CVS_V3 -> OR AbsController -> AbsController TractionController -> TractionController BrakeActuators -> BrakeActuators ThrottleActuator -> ThrottleActuator WheelRotationSensor -> WheelRotationSensor StabilityController -> StabilityController MultimediaSubsystem -> MultimediaSubsystem -> XOR FM_CD_Controller -> FM_CD_Controller FM_CD_Controller -> FM_CD_Controller FMReceiver -> FMReceiver CDDevice -> CDDevice -> XOR SingleCD -> SingleCD SingleCD -> SingleCD BrakePedalSensor -> BrakePedalSensor StabilitySensors -> StabilitySensors

Resolution	*Items*
Configuration 2	
Configuration 3	

Resolution	*Items*
Configuration 4	CVS_V3 CVS_V3 Resolution Element CVS_V3 -> OR AbsController -> AbsController TractionController -> TractionController BrakeActuators -> BrakeActuators ThrottleActuator -> ThrottleActuator WheelRotationSensor -> WheelRotationSensor CruiseControl -> CruiseControl -> XOR SimleCruiseControl -> SimleCruiseControl SimleCruiseControl -> SimleCruiseControl StabilityController -> StabilityController MultimediaSubsystem -> MultimediaSubsystem -> XOR FM_MultiCDController -> FM_MultiCDController FM_MultiCDController -> FM_MultiCDController FMReceiver -> FMReceiver CDDevice -> CDDevice -> XOR MultiCD -> MultiCD MultiCD -> MultiCD BrakePedalSensor -> BrakePedalSensor StabilitySensors -> StabilitySensors
Configuration 5	Configuration 1 + GPSCell -> GPSCell
Configuration 6	Configuration 2 + BluetoothModule -> BluetoothModule
Configuration 7	Configuration 3 + iSafeDialer -> iSafeDialer
Configuration 8	Configuration 4 +
None of the aforementioned	(In any other case)

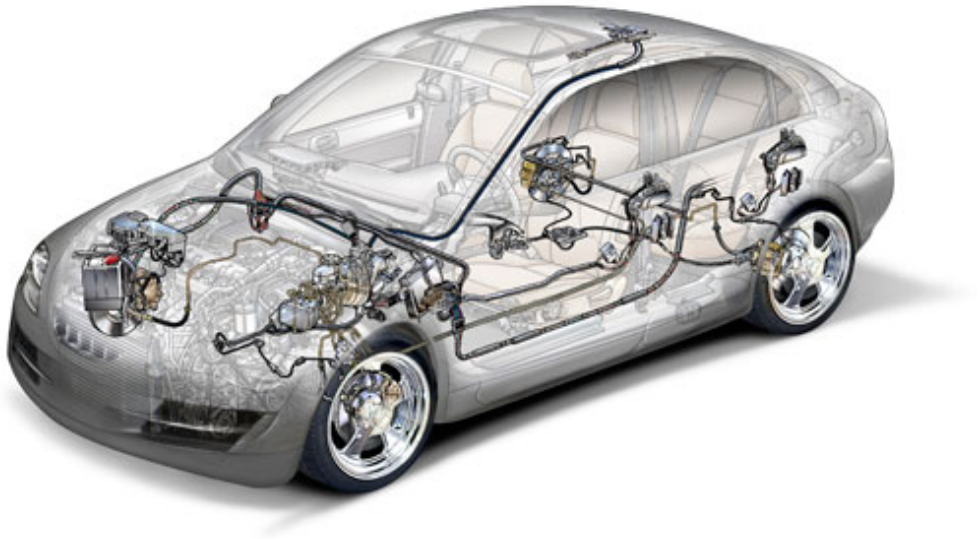
The last thing to do is to fulfill the method survey. Please take into account that the survey is aimed to evaluate the method not the tool. We know that the tool has a lot of things that can be improved and you are invited to provide us feedback through the question 23 of the survey.

A.2 Feature model and feature description

This Section describes the different features of the products that integrate the VehicleControlSystem SPL:

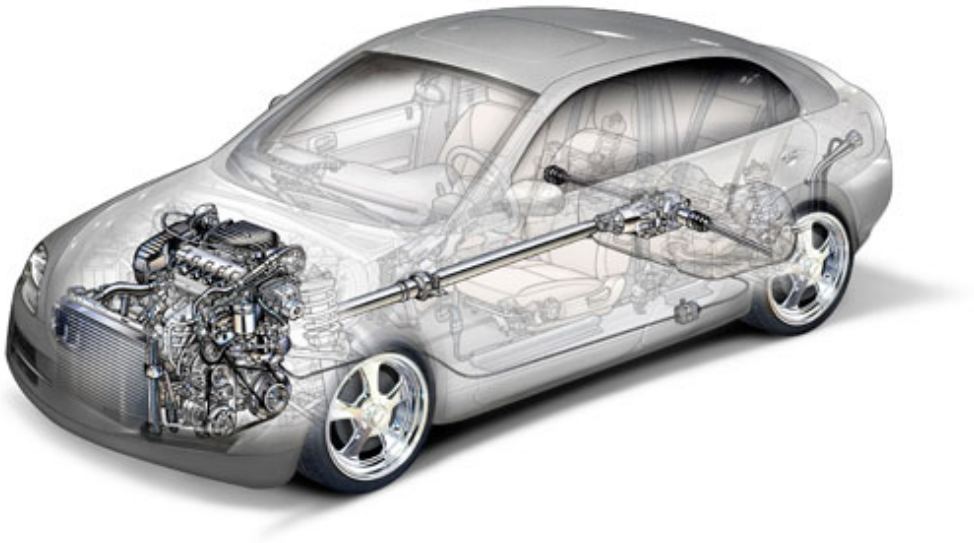
- **Antilock Braking System (ABS):** The Goal of the Antilock Braking System (ABS) is to ensure that maximum braking force is transmitted to all four wheels of the vehicle, even under adverse conditions such as skidding on rain, snow or ice. Antilock braking system works by sensing slippage at the wheels during braking, through a wheel

rotating sensor, and adjusting brake pressure to ensure maximum contact between tires and road, through the electronic brake actuators. In the most basic version, wheel rotation sensors from all four wheels are used as input and the output is the brake valve on each brake line.



- **Traction Control System (TCS):** the Goal of the Traction Control System (TCS) is to avoid wheels to slip while accelerating. TCS deals with the front to back loss of tire to road friction during acceleration. The traction control system uses the data from the rotation sensor of each wheel of the vehicle, compares the rotation data with the speed to detect slipping wheels, and compensates these slipping wheels by reducing the speed. This is achieved either by applying individual braking force to the slipping wheel or by reducing the power of the engine via the throttle control to ensure maximum contact between the road surface and the tires, even under less - than ideal road conditions, such as ice or snow.

- **Stability Control System (SCS):** The goal of the Stability Control System (SCS) is to keep the vehicle going in the direction in which the driver is steering the car. To achieve this, the stability control system applies the brake to one wheel (or passes the torque to the opposite one) to help steer the car in the correct direction. The SCS differs from the TCS in what both systems prevent. A Traction Control System acts on a vehicle's traction wheels to prevent unwanted wheel spin under acceleration. The SCS, on the other hand, goes one step further by detecting when a driver has lost some steering control over the car's trajectory, and then automatically stabilizes the vehicle to help the driver regain the control over the vehicle.

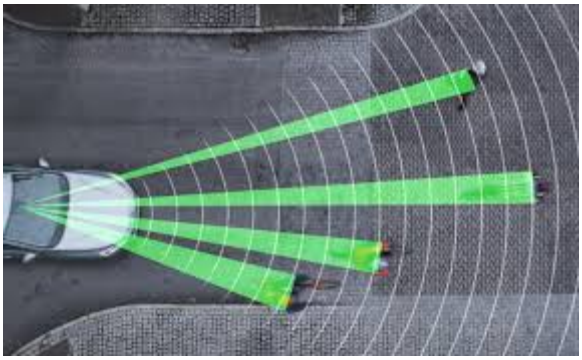


- **Cruise control system (CC):** The goal of the cruise control system (CC) is to maintain a constant speed as determined by the driver. The system is in effect between some minimum and maximum speeds

(e.g., 40 Km/h to 120 MPH). The cruise control system maintains the vehicle speed at the predetermined value (target value) by storing the speed of the wheel rotation when the speed value is set and attempts to keep the throttle actuator at a position to maintain the vehicle speed at the target value. As the road inclination changes, the vehicle speed changes, and the throttle position should change to maintain the vehicle speed. The control system observes the speed difference between the current speed and the target value and either decreases or increases the throttle actuator position to counteract the speed differential. The algorithm to accomplish this is called the control law. Depending on which other sensors are available (due to the selection of other features) the functionality given by the cruise control may change, we have three possible configurations:

- Basic Cruise Control: The goal of the Basic Cruise Control is to maintain a constant vehicle velocity as determined by the driver (target speed).
- Adaptive Cruise Control: The goal of the Adaptive Cruise Control System is to extend the Basic Cruise Control and provide a new function of constant distance cruise control that maintains the distance to a target vehicle traveling in front of the vehicle. In order to ensure constant distance cruise control, this system includes obstacle avoidance sensor that yields the following information: the distance from the equipped vehicle to the target vehicle and the relative speed between these two vehicles.

- The goal of the Fully Adaptive Cruise Control System is to extend the Adaptive Cruise Control, by adding a camera that captures the behavior of vehicles and objects in front of the system, and extends the effective range of the radar sensor and is capable to stop the vehicle in case of imminent collision.

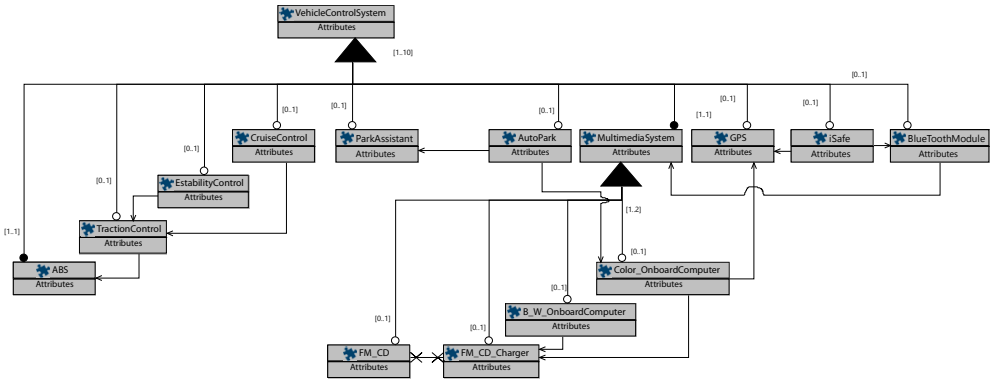


- **Park Assistant:** The goal of the Park assistant system is to inform the driver about the presence of an obstacle during the parking maneuvers. The system is monitoring a set of park sensors and obstacle avoidance sensors (on the front and on the rear part of the car) and, when a sensor measures a distance less than a certain threshold, the system assumes the presence of an obstacle and starts emitting a beep and a lighting signal. As the distance with the obstacle decreases, the warnings grow in intensity and frequency.
- **Auto Parking:** The goal of the auto-parking system is to automatically execute the parking maneuvers for the driver. The systems uses a set of proximity sensors and obstacle avoidance sensor so as to measure the environment, and set of cameras for identifying, through artificial vision techniques, a suitable place for parking the car. The system controls the steering, the brakes and the throttle to conduct the parking maneuvers.

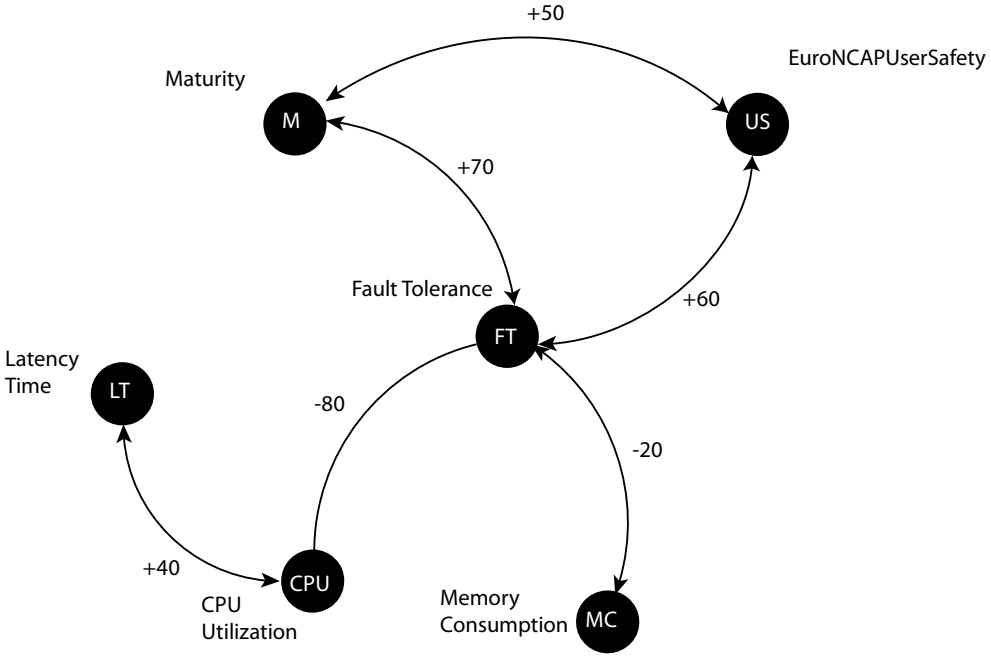
- **Multimedia System:** The Multimedia System encapsulates a set of features, described below:
  - **FM_CD:** the goal of the FM_CD system is to control the simplest stereo system, which is composed of a FM-Radio and a single-CD unit. The system allows tuning, storing and retrieving radio stations, to select the CD-track to play, to pause and to stop the system.
  - **FM_CD_Charger:** the goal of the system is to control the high-end stereo system, which is composed of a FM-Radio and a six cd charger unit. The system allows tuning, storing and retrieving radio stations, tuning, storing and retrieving radio stations sound equalizations, selecting the CD disk and CD-track to play, to pause and to stop the system.
  - **B_W_OnboardComputer:** the BW_Onboard_System allows configuring other systems of the vehicle. In addition, it shows statistics about the car performance and about eventual maintenance requirements. It has a Black and White screen and a set of buttons with which the user can interact with the system.
  - **ColorOnboardComputer:** the Colour_OnboardComputer allows configuring other systems of the vehicle. In addition, shows statistics about the car performance and about eventual maintenance requirements. It has a full -color high-resolution screen and a set of buttons with which the user can interact with the system. If the VehicleControlSystem is selected, the onboard computer will include the GPS navigation capability.
- **GPS:** The GPS allows t calculate the position, trajectory and speed based on the GPS information. The system integrates a GPS Module with which the system is able to calculate the car's position. The ColourOnboardComputer will show the navigation information in these cases in which the GPS feature is selected.
- **Bluetooth:** The Bluetooth module allows to connect to a mobile phone and to make/receive calls. The system integrates a GPS module and is able to interact with the driver through a set of buttons, a microphone and the sound system (FM_CD or FM_CD_Charger).

- **iSafe:** The iSafe system is a safety system that is able, in case of accident to send the GPS position and to call and communicate with the emergency services. The system is able to read the position from the GPS information and it uses the Bluetooth module to communicate with the emergency services.

Vehicle Control System SPL Feature Model:

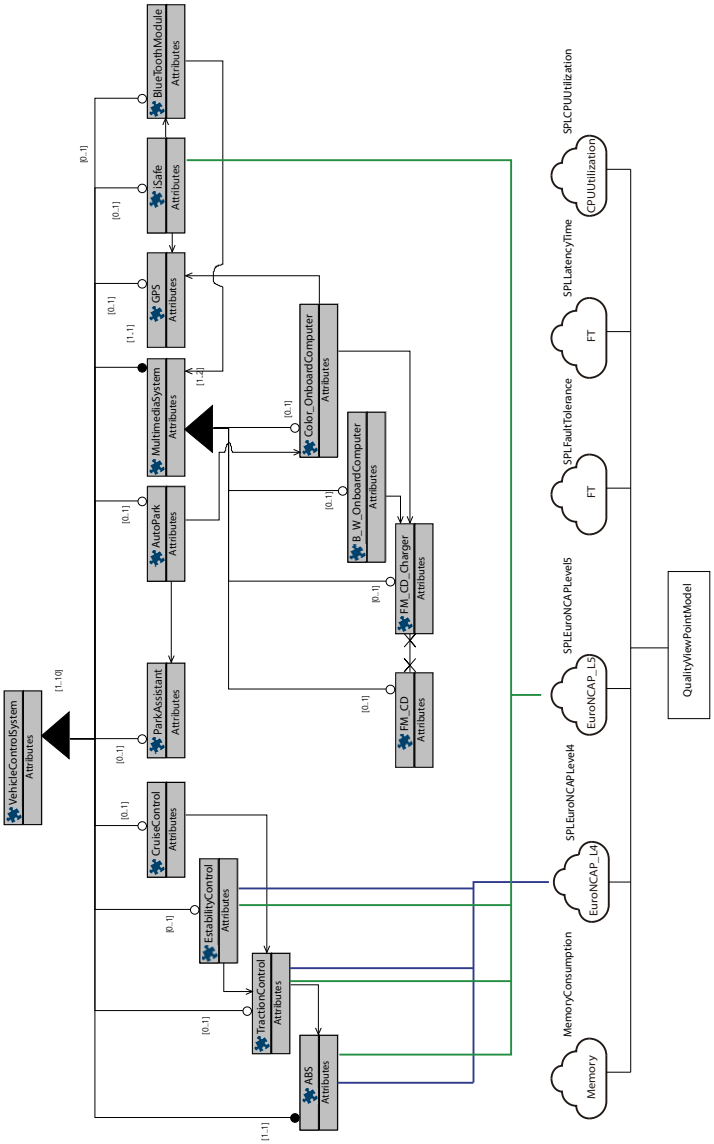


A.3 Impacts among quality attributes

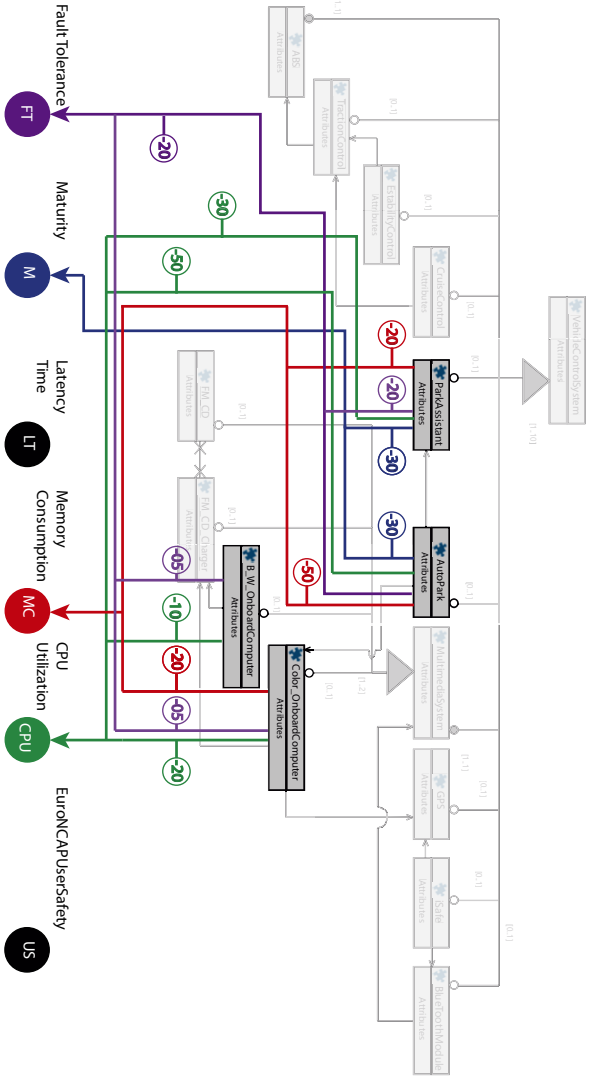


A.4 Relationships among elements on different viewpoints

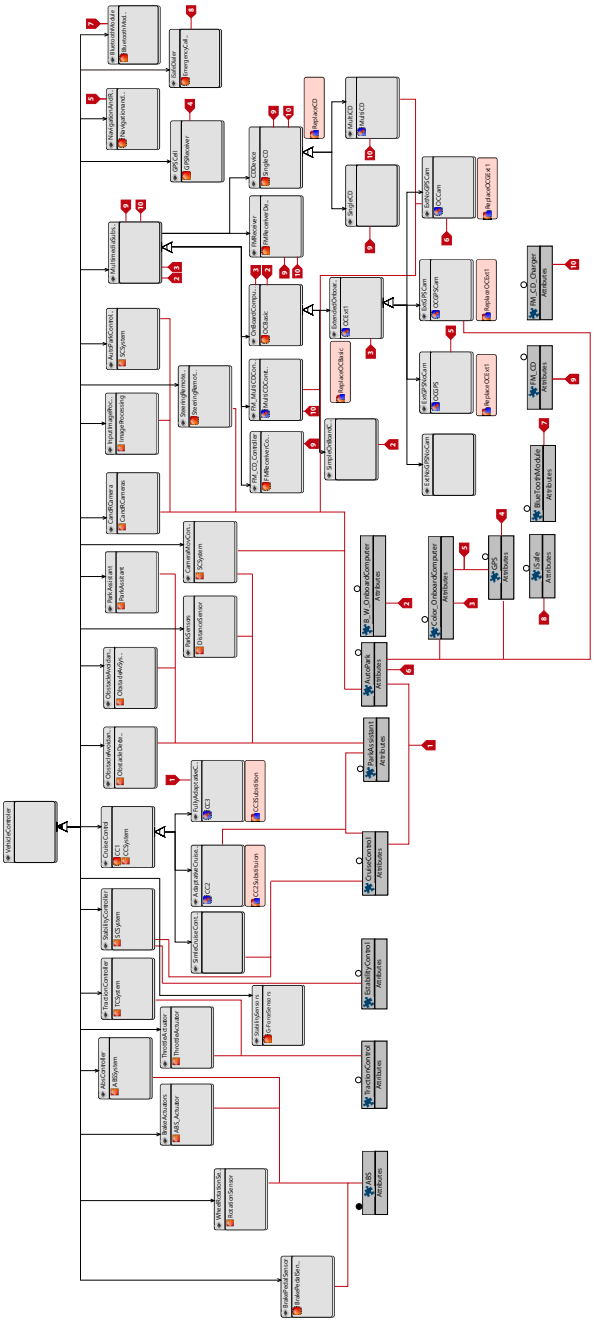
Realization Relationships among Features and Non-Functional Requirements



Impact Relationships among Features and Quality-Attributes



Realization Relationships among Architectural Variability Points and Features



A.5 Subjective Questionnaire

Survey about the use QuaDAI as a Product Configuration and Product Architecture Derivation Method

For each question, please select one option, by crossing the circle that is closest to your opinion.

Please read carefully each statement before answering.

1. The product configuration and architecture derivation method is complex and difficult to follow. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

2. I believe this method would reduce the time and effort required for the product configuration and the product architecture derivation. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

3. In general, the product configuration and product architecture derivation method is difficult to be understood. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

4. I believe that in general, the product configuration and architecture derivation method is useful. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

5. The product configuration and architecture derivation method is difficult to be learnt. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

6. I believe this software configuration and derivation method is useful to obtain product architectures taking into account both functional and quality requirements. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

7. If I would need to use a product architecture configuration and derivation in the future I believe I will take this method into account. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

8. I believe this method lacks of the required mechanisms for configuring and deriving product architectures from the product line architecture. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

9. In general, I believe this method does not efficiently support the product configuration software architecture derivation. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

10. The use of this method will improve my performance on configuring products and deriving product architectures. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

11. I believe it would be easy to become skilled using this method.*

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

12. I have the intention to use this method in the future.*

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

13. I would not recommend using this software architecture derivation method.*

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

14. The time for performing the tasks was appropriate. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

15. The tasks to be performed were not clearly defined. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

16. I have no problem to understand the architecture of the system to be derived. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

17. It was difficult to understand the features and NFRs to be selected. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

18. The metrics to be applied were simple and easy to understand. *

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

19. I have found the exercised useful.*

Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Totally agree

20. Do you have any suggestion on how to make this software method more easy to use?

_____
_____
_____

21. What are the reasons that will make user or not this method in the future?

_____
_____
_____

22. Please write any other comment or suggestion you want to do related to the product configuration and product architecture derivation method in the space below:

_____
_____
_____

23. Please write any other comment or suggestion you want to do related to the tool.

_____
_____
_____

A.6 Leveling Questionnaire

1. Regarding my profile: *

Select the option that best fits to your profile (note: consider the options in order and select the first one that fits to you).

Mark only one oval.

- ◯ I have been involved in software development teams applying the Software Product Line approach.
- ◯ I am a researcher working on topics related to Software Product Line Development.
- ◯ I know what Product Lines are but I have never participated in a software project applying SPL development.
- ◯ I have never heard about Software Product Lines.

2. How many months of experience do you have in SPL? *

Enter a number.

_____

3. Have you applied the SPL approach in building software? *

Mark only one oval.

- ◯ Yes, but only in the research domain.
- ◯ Yes, but only in the industry domain.
- ◯ Yes, both research and industry domain.
- ◯ No.

4. A Software Product Line (SPL) is a: *

Mark only one oval.

- Set of software intensive systems sharing a common managed set of features developed from a common set of core assets in a prescribed way.
- Set of products built from a platform of components.
- I am not sure about that.

5. What is a feature? *

Mark only one oval.

- A feature is a user-visible characteristic of a system or software product and which are usually organized in feature models.
- A feature is everything that can vary through a SPL.
- I am not sure about that.

6. In SPL Development ... *

Mark only one oval.

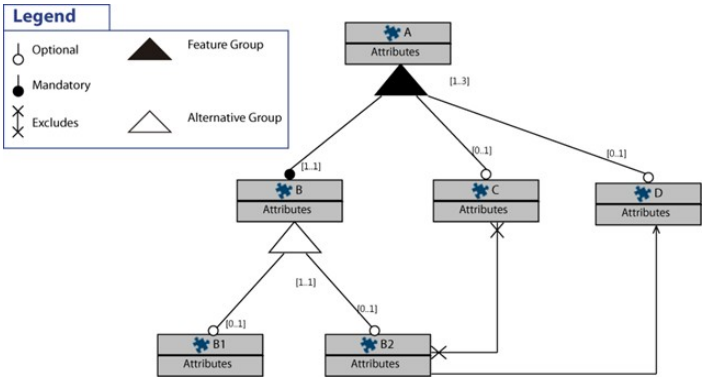
- ... there are two software architectures: i) the Product Line Architecture (PLA) with should include the variation mechanisms so as to cover all the possible products within the SPL and ii) the Product Architecture (PA) that is derived from the PLA by exercising these variation mechanisms in order to permit the achievement the product specific requirements of the product.
- ... Software Architectures should be built from the scratch for every single product.
- ... the software architecture concepts are not applicable.

7. In SPL development, a configuration is: *

Mark only one oval.

- A combination of features and/or non-functional requirements that define a product and which conforms to the restrictions defined on a features model and/or SPL non-functional requirements.
- A set of functionalities the define a product.
- The concept of configuration is not applicable to SPL development.

8. In this Feature Model ...



Mark only one oval.

- The {A, B, B2, C} is an invalid configuration.
- The {A, B, B1, C, D} is an invalid configuration.
- The {A, B, B2, D} is an invalid configuration.

Apéndice B. Material experimental

Este apéndice presenta un extracto de los diferentes materiales experimentales. El apéndice B.1 presenta un extracto de los boletines ATAM y QuaDAI para el objeto experimental O1. Se ha decidido mostrar únicamente el material experimental de uno de los objetos experimentales, dado que de este modo se facilitará la comprensión de las tareas experimentales. Los materiales del segundo objeto experimental (O2) están disponibles para su descarga en:

http://www. dsic.upv.es/~jagonzalez/tesis/instrumentacion.html.

Finalmente el Apéndice A.2 presenta el cuestionario post-experimental utilizado para medir las variables subjetivas.

B.1 Ejemplos de la arquitectura software, patrones y métricas

B.1.1. Arquitectura software

La primera arquitectura software a evaluar pertenece a un sistema de frenos antibloqueo (ABS). El objetivo de un sistema ABS es el control de los actuadores de freno de un automóvil. El sistema controla el sensor de pedal de freno y activa el actuador del freno tan pronto como el conductor presiona el pedal. Además, con el fin de evitar el deslizamiento de las ruedas cuando se activa el actuador de freno, el sistema está supervisando cuatro sensores de rotación, uno en cada rueda. Cada sensor envía señales mientras la rueda está girando. Si el sistema detecta deslizamiento de las ruedas, por la ausencia de pulsos, se desactivará el actuador de freno, que se activa de nuevo después de un pequeño lapso de tiempo.

La Figura A.1 muestra la arquitectura del sistema (ABS) expresada mediante la sintaxis grafica de AADL. A la izquierda se encuentran los sensores de entrada, dentro del sistema ABS se puede ver los componentes software de procesamiento y control y, finalmente en el lado derecho se encuentran los actuadores. En este caso, aparte de los sensores y actuadores, también se han considerado las señales de la consola de cabina. La consola de cabina incluye los interruptores para activar o desactivar los diferentes sistemas y las señales que indican al usuario que el ABS está activado.

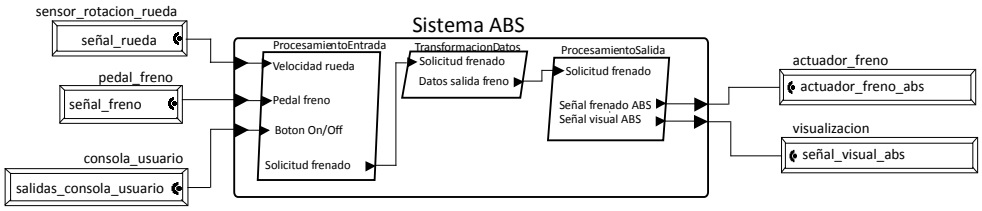


Figura B.1 Arquitectura del sistema ABS

B.1.2. Árbol de utilidad de ATAM

El árbol de utilidad del sistema permite expresar los factores de calidad en los que se descompone la "utilidad" del sistema. La Figura 2 presenta el árbol de utilidad de la arquitectura para el sistema ABS: en el nivel 1 se puede observar los atributos de calidad fiabilidad, rendimiento y consumo de recursos; en el nivel 2 los tres requisitos no funciónales y finalmente en el nivel 3 los escenarios que describen dichos requisitos**.**

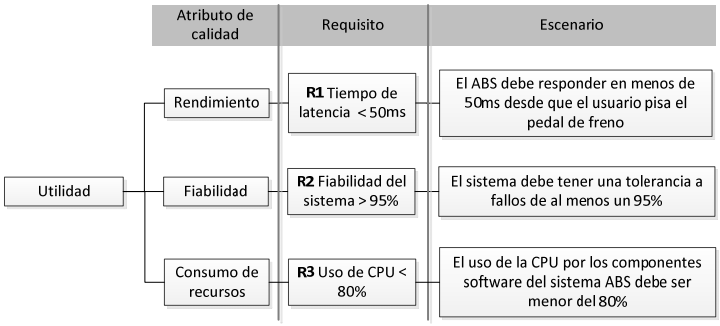


Figura B.2 Árbol de utilidad del sistema ABS

B.1.3. Patrones Arquitectónicos

A continuación, en las Tabla A.1 y A.2, se presentan dos ejemplos de los patrones arquitectónicos empleados en el experimento, el patrón de *Triple Redundancia Modular* y el patrón *Watchdog* (Douglass 2002). Ambos se han documentado empleando la misma plantilla para documentar los patrones que se utilizó en el material experimental, en la que se describen los patrones en términos de contexto de aplicación, problema a resolver, estructura del patrón y consecuencias tras su aplicación.

Tabla B.1 Plantilla del patrón triple redundancia modular

Patrón	Triple Redundancia Modular
Contexto	El patrón de Triple Redundancia Modular (Triple Redundancy Pattern) o TMR es un patrón usado para mejora la fiabilidad y seguridad de los sistemas donde no hay estado libre de fallos. El Patrón TMR es un patrón que ofrece un número de canales impar (tres) que se ejecutan de forma concurrente en paralelo, cada uno verificando los resultados del resto. Se comparan los resultados de los distintos canales, y en caso de discrepancia, se aplica la política de *mayoría-de-dos-tercios-gana* (el resultado con dos votos gana).
Problema	El problema que trata de resolver el patrón de Triple Redundancia Modular es básicamente dar protección contra fallos aleatorios, con la restricción adicional de que en caso de fallo el dato de entrada no debe perderse y no debemos consumir tiempo adicional en computar la respuesta en caso de fallo.
Estructura del patrón	El patrón tiene una estructura replicada que consiste en tres canales operando en paralelo, tal como se muestra en la figura A.3. Cada canal individual contiene los componentes que procesan los datos de entrada en una serie de pasos de computación. Los canales no cotejan los resultados con el resto de canales en puntos estratégicos, los canales operan completamente en paralelo, y solamente al final se comparan los resultados. El comparador implementa la política *winner-take-all* por lo que los dos canales que producen el resultado considerado correcto ganarán.
Consecuencias	El patrón triple redundante modular solo puede detectar fallos aleatorios. Dado que los canales son homogéneos, por definición un fallo sistemático aparecerá en todos los canales. Dado que los canales están ejecutando en paralelo el sensor está replicado o al menos los tres canales son capaces de adquirir el dato, por lo que no hay pérdidas de datos en caso de fallo y no se necesita recalcular la salida. Este patrón añade el tiempo de computación del comparador afectando al comportamiento del sistema en el caso general. Otra desventaja del patrón es el alto coste de replicación. El patrón TMR es muy común en aplicaciones en las que los requisitos de fiabilidad son muy altos y compensan el coste de replicación.

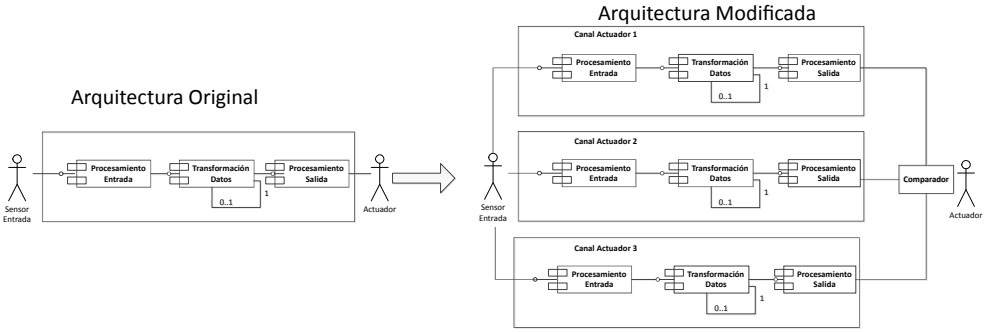


Figura B.3 Estructura del patrón triple redundancia modular

Tabla B.2 Plantilla del patrón watchdog

Patrón	Watchdog (perro guardián)
Contexto	El patrón watchdog es un patrón muy ligero que provee de una cobertura mínima frente a fallos. El patrón watchdog únicamente comprueba que los cálculos internos se están llevando a cabo satisfactoriamente. Esto significa que la cobertura frente a fallos es mínima, y un amplio rango de fallos no será detectado.
Problema	Los sistemas de tiempo real son aquellos en que se puede predecir la respuesta en el tiempo. En estos sistemas las salidas tienen un deadline en el que tienen que ser aplicadas. Si el cálculo acaba más allá del deadline el resultado puede considerarse irrelevante o incorrecto. Si la salida llega demasiado tarde, el sistema no podrá ser controlado, se dice entonces que el sistema está en la región inestable.
Estructura del patrón	La estructura del patrón se muestra en la Figura A.4, en la que se puede apreciar su simplicidad. El canal principal trabaja de manera independiente al watchdog, y le envía "pruebas de vida" cada cierto tiempo al watchdog. A esto se le denomina *"stroking the watchdog"* (acariciar al guardián). El watchdog emplea la periodicidad del *stroking* para detectar si se ha producido un fallo. Muchos watchdog únicamente comprueban que el *stroke* se produce cada cierto lapso de tiempo y no se preocupan si el *stroke* llega demasiado pronto, por el contrario otros aseguran que el *stroke* llega exactamente en el instante previsto.
Consecuencias	El patrón watchdog es un patrón extremadamente ligero que raramente se utiliza solo en sistemas críticos. Es extremadamente bueno identificando fallos en la línea de tiempo, especialmente cuando una base de tiempo independiente guía al watchdog. Puede ser utilizado para detectar deadlocks en el canal principal. Dado que su cobertura es tan baja su efecto sobre la fiabilidad es prácticamente inapreciable y raramente se utiliza solo.

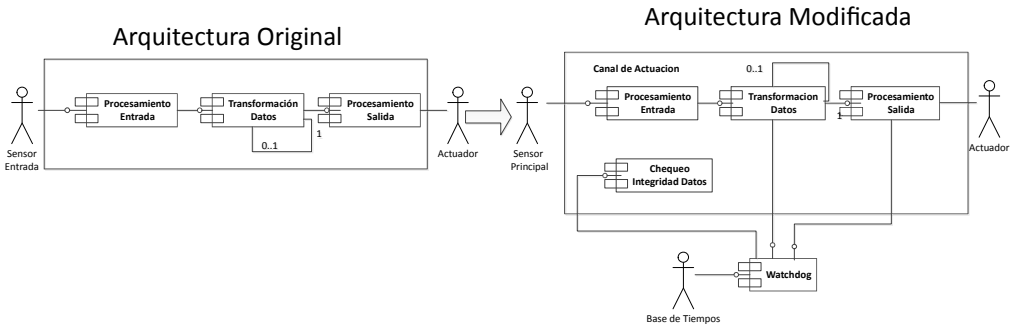


Figura B.4 Estructura del patrón watchdog

B.1.4. **Métricas**

En la Tabla A.3 se muestra la documentación de la métrica *Tiempo de Latencia* con la que evaluar el RNF de rendimiento. En dicha tabla muestra la información empleando la plantilla para la documentación de las métricas que se utilizó en el material experimental, en la que se describen las métricas en términos de su descripción, su fórmula de cálculo, su interpretación y un ejemplo de su aplicación.

Tabla B.3 Plantilla para la métrica Tiempo de Latencia

Métrica	***Tiempo de Latencia***
Descripción	La métrica tiempo de latencia se define como el tiempo transcurrido entre la recepción de un evento de entrada y la generación de la salida correspondiente.
Calculo	Para el cálculo se han de considerar los flujos de datos y eventos que siguen a la recepción del evento de entrada en un sensor hasta que el estado del actuador cambia. Solo se considerara el tiempo de latencia del caso óptimo. $$T_{Latencia} = \sum_{Componenetes\ en\ el\ flujo} T_{Laten\ \square\square}(Componente)$$
Interpretación	La métrica tiempo de latencia da como resultado un valor positivo. Cuanto más cercano a 0 mejor es el valor. Puede medirse empleando distintas unidades de tiempo (ms, seg, etc.).
Ejemplo	La Figura A5 muestra el flujo considerado para el cálculo del tiempo de latencia en esa arquitectura en el mejor de los casos. Así, en ese caso particular, el cálculo del tiempo de latencia se calculará según la expresión: $$T_{Latencia} = T(Sensor) + T(Procesamiento) + T(TransformacionDatos) + T(ProcesamientoSalida) + T(Actuador)$$

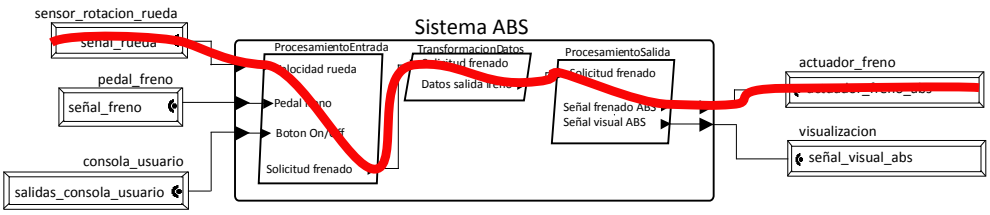


Figura B.5 Flujo para el cálculo del tiempo de latencia del sistema

B.1.5. **Arquitectura resultante tras la aplicación de los patrones**

La Figura A.6 muestra un ejemplo de cómo se documenta el resultado de las transformaciones tras la aplicación de patrones. Concretamente la Figura A.6 muestra el resultado de aplicar el patrón triple redundancia modular a la arquitectura original.

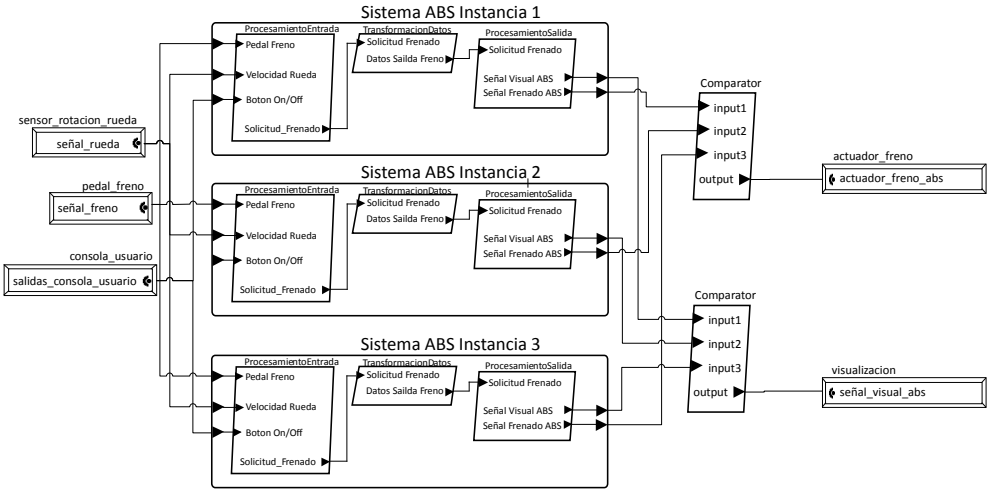


Figura B.6 Arquitectura resultante tras la aplicación del patrón triple redundancia modular

B.2 Cuestionario de evaluación de las variables subjetivas

La Tabla A.3 muestra el cuestionario post-experimento con el que se han medido las variables subjetivas facilidad de uso percibida (FUP), utilidad percibida (UP) e intención de uso (IU). En la numeración de las preguntas se ha incluido la variable a la que cada una de ellas hace referencia.

Tabla B.3 Cuestionario post-experimento para evaluar las variables subjetivas

Pregunta	*Afirmación positiva (5 puntos)*	*Afirmación negativa (1 punto)*
FUP1	El método de Evaluación de arquitecturas software es simple y fácil de seguir.	El método de Evaluación de arquitecturas software es complejo y difícil de seguir.
FUP2	De manera general, el método evaluación de arquitecturas software es fácil de entender.	De manera general, el método de evaluación de arquitecturas software es difícil de entender.
FUP3	El método de evaluación de arquitecturas software es fácil de aprender.	El método evaluación de arquitecturas software es difícil de aprender.
IU1	Si tuviera que utilizar un método de evaluación de arquitecturas en el futuro creo que tendría en cuenta este método.	Si tuviera que utilizar un método de evaluación de arquitecturas en el futuro creo que NO tendría en cuenta este método.
IU2	Pienso que sería fácil ser hábil usando este método.	Pienso que sería difícil ser hábil usando este método.
IU3	Tengo la intención de utilizar este método en el futuro.	NO tengo la intención de utilizar este método en el futuro.
IU4	Recomendaría el uso de este método de evaluación de arquitecturas software.	No recomendaría el uso de este método de evaluación de arquitecturas software.
UP1	Creo que este método reduciría el tiempo y el esfuerzo requerido para evaluar arquitecturas software.	Creo que este método aumentaría el tiempo y esfuerzo requerido para evaluar arquitecturas software.
UP2	De manera general, considero que el método de evaluación de arquitecturas software es útil.	De manera general, considero que el método evaluación de arquitecturas software NO es útil.
UP3	Creo que este método de evaluación de arquitecturas software es útil para obtener arquitecturas que cumplen los requisitos de calidad.	Creo que este método evaluación de arquitecturas software NO es útil para obtener arquitecturas que cumplen los requisitos de calidad.
UP4	Pienso que este método tiene los mecanismos necesarios para detectar las mejoras a realizar sobre las arquitecturas software.	Pienso que este método CARECE de los mecanismos necesarios para detectar las mejoras a realizar sobre las arquitecturas software.
UP5	De manera general, pienso que este método proporciona una manera eficaz de una manera eficaz de evaluar arquitecturas software.	De manera general, pienso que este método NO proporciona una manera eficaz de evaluar arquitecturas software.
UP6	El uso de este método mejoraría mi rendimiento en la evaluación de arquitecturas software.	El uso de este método NO mejoraría mi rendimiento en la evaluación de arquitecturas software.

Bibliografía

Ali Babar, M. (2008). "*Assessment of a framework for designing and evaluating security sensitive architecture*". En actas de: 12th International Conference on Evaluation and Assessment in Software Engineering, Bari, Italy.

Ali Babar, M., Gorton, I. (2004). "*Comparison of Scenario-Based Software Architecture Evaluation Methods*". En actas de: 11th Asia-Pacific Software Engineering Conference, Busan, Korea. pp. 600–607.

Ali Babar, M., Kitchenham, B. (2007)(a). "*Assessment of a Framework for Comparing Software Architecture Analysis Methods*". En actas de: 11th International Conference on Evaluation and Assessment in Software Engineering, Keele, England. pp. 12–20.

Ali Babar, M., Kitchenham, B. (2007)(b). "*The Impact of Group Size on Software Architecture Evaluation: A Controlled Experiment*". En actas de: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, Spain. pp. 420–429.

Ali Babar, M., Kitchenham, B., Jeffery, R. (2007). "*Comparing distributed and face-to-face meetings for software architecture evaluation: A controlled experiment*". Empirical Software Engineering, Vol. 13, Num. 1, pp. 39–62.

Ali Babar, M., Lago, P., Deursen, A. (2011). "*Empirical research in software architecture: opportunities, challenges, and approaches*". Empirical Software Engineering, Vol. 16, Num. 5, pp. 539–543.

Ali Babar, M., Winkler, D., Biffl, S. (2007). "*Evaluating the Usefulness and Ease of Use of a Groupware Tool for the Software Architecture Evaluation Process*". En actas de: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, Spain. pp. 430–439.

Ali Babar, M., Zhu, L., Jeffery, R. (2004). "*A framework for classifying and comparing software architecture evaluation methods*". En actas de: 2004 Australian Software Engineering Conference, Melbourne, Australia. pp. 309–318.

Alonso, A., García-Valls, M., Puente, J. (1998). "*Assessment of Timing Properties of Family Products*". En actas de: F. Lindened. ESPRIT-ATES Workshop, Las palmas de Gran Canaria, Spain. pp. 161–169.

Alves, E., Junior, D.O., Gimenes, I.M.S. (2008). "*A Metric Suite to Support Software Product Line Architecture Evaluation*". En actas de: XXIV Conferencia Latinoamericana de Informática, Santa Fé, Argentina. pp. 489–498.

Ameller, D., Ayala, C., Cabot, J., Franch, X. (2013). "*Non-functional Requirements in Architectural Decision Making*". IEEE Software, Vol. 30, Num. 2, pp. 61–67.

Antonio, E.A., Ferrari, F.C., Fabbri, S.C.P.F. (2012). "*A Systematic Mapping of Architectures for Embedded Software*". En actas de: 2012 Second Brazilian Conference on Critical Embedded Systems, Sao Paulo, Campinas, Brazil. pp. 18–23.

Asikainen, T., Soininen, T., Männistö, T. (2003). "*A Koala-based approach for modelling and deploying configurable software product families*". En actas de: 5th International Workshop on Product-Family Engineering, Sienna, Italy. pp. 225–249.

Atkinson, C., Bayer, J., Muthig, D. (2000). "*Component-based product line development: the KobrA approach*". En actas de: 1st Internation Conference on Software Product Lines, Denver, Colorado. pp. 289–309.

Atkinson, C., Gerbig, R., Kennel, B. (2012). "*On-the-Fly Emendation of Multi-level Models*". En actas de: 8th European Conference on Modelling Foundations and Applications, Kgs. Lyngby, Denmark. pp. 194–209.

Atkinson, C., Gerbig, R., Tunjic, C. (2013). "*A multi-level modeling environment for SUM-based software engineering*". Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling - VAO '13, New York, New York, USA pp. 1–9.

Bachmann, F., Bass, L. (2001). "*Managing variability in software architectures*". En actas de: Symposium on Software Reusability: Putting Software Reuse in Context, Toronto, Ontario, Canada. pp. 126–132.

Bagheri, E., Noia, T. Di, Ragone, A., Gasevic, D. (2010). "*Configuring Software Product Line Feature Models based on Stakeholders ' Soft and Hard Requirements*". En actas de: 14th Software Product Line Conference, Jeju Island, South Korea. pp. 16–31.

Barbacci, M., Clements, P., Lattanze, A., Northrop, L., Wood, W. (2003). "*Using the Architecture Tradeoff Analysis Method SM (ATAM SM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study*". CMU/SEI-2003-TN-012, Software Engineering Institute, Carnegie Mellon University.

Barbero, M., Fabro, M. Del, Bézivin, J. (2007). "*Traceability and provenance issues in global model management*". En actas de: 3rd European Conference on Model Driven Architecture® Foundations and Applications, Haifa, Israel. pp. 47–55.

Barkmeyer, E.J., Feeney, A.B., Denno, P., Flater, D.W., Libes, D.E., Steves, M.P., Wallace, E.K. (2003). "*Concepts for Automating Systems Integration*". NISTIR 6928, U.S. Department of Commerce.

Basili, V. (1993). "*The experimental paradigm in software engineering*". En actas de: International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions, Wadern, Merzig-Wadern, Germany. pp. 3–12.

Basili, V. (1996). "*The role of experimentation in software engineering: past, current, and future*". En actas de: 18th international conference on Software engineering, Berlin, Germany. pp. 442–449.

Basili, V., Shull, F., Lanubile, F. (1999). "*Building knowledge through families of experiments*". IEEE Transactions on Software Engineering, Vol. 25, Num. 4, pp. 456–473.

Basili, V.R., Caldiera, G., Rombach, H.D. (1994). "*The goal question metric approach*". Encyclopedia of Software Engineering. pp. 1–10.

Basili, V.R., Member, S., Rombach, H.D. (1988). "*The TAME Project: Towards Improvement-Oriented Software Environments*". IEEE Transactions on Software Engineering, Vol. 14, Num. 6, pp. 758–773.

Basili, V.R., Selby, R.W., Hutchens, D.H. (1986). "*Experimentation in software engineering*". IEEE Transactions on Software Engineering, Vol. SE-12, Num. 7, pp. 733–743.

Bass, L., Clements, P., Kazman, R. (1998). "*Software Architecture In Practice*". Addison-Wesley.

Bass, L., Clements, P., Kazman, R. (2003). "*Software Architecture in Practice. 2nd Edition*". Addison-Wesley.

Bayer, J., Flege, O., Gacek, C. (2000). "*Creating product line architectures*". En actas de: International Workshop on Software Architectures for Product Families, Las palmas de Gran Canaria, Spain. pp. 210–216.

Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.-M. (1999). "*PuLSE: a methodology to develop software product lines*". En actas de: 1999 symposium on Software reusability, Los Angeles, California, USA. pp. 122–131.

Bayer, J., Gacek, C. (2000). "*PuLSE-I: Deriving instances from a product line infrastructure*". En actas de: 7th International Conference onnEngineering of Computer Based Systems, London, UK. pp. 237–245.

Benavides, D., Trinidad, P., Ruiz-Cortez, A. (2005). "*Automated reasoning on feature models*". En actas de: 17th International Conference on Advanced Information Systems Engineering, Porto, Portugal. pp. 491–503.

Benbasat, I., Goldstein, D.K., Mead, M. (1987). "*The Case Research Strategy in Studies of Information Systems Case Research: Definition*". MIS Quaterly, Vol. 11, Num. 3, September.

Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H. (2004). "*Architecture-level modifiability analysis (ALMA)*". Journal of Systems and Software, Vol. 69, Num. 1-2, pp. 129–147.

Bergey, J., Fisher, M., Jones, L., Kazman, R. (1999). "*Software Architecture Evaluation with ATAM in the DoD System Acquisition Context*". CMU/SEI-99-TN-012, Software Engineering Institute, Carnegie Mellon University.

Berntsson, L., Blom, H., Chen, D., Cuenot, P. (2008). "*EAST-ADL 2.0 Specification*". ATESST Consortium.

Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P. (2005). "*Modeling in the Large and Modeling in the Small*". En actas de: Model Driven Architecture, European MDA Workshops: Foundations and Applications, Amsterdam, The Netherlands. pp. 33–46.

Bézivin, J., Jouault, F., Valduriez, P. (2004). "*On the Need for Megamodels*". En actas de: OOPSLA/GPCE Workshop on Best Practices for Model-Driven Software Development, Nashville, Tenesse, USA. pp. 1–9.

Blair, G., Bencomo, N., France, R.B. (2009). "*Models@ run.time*". Computer, Vol. 42, Num. 10, pp. 22–27.

Bosch, J. (2000). "*Design and Use of Software Architecture - Adopting and evolving a product-line approach*". Pearson Education.

Botterweck, G., Lee, K., Thiel, S. (2009). "*Automating Product Derivation in Software Product Line Engineering*". En actas de: Software Engineering, Kaiserslautern, Germany. pp. 177–182.

Boucké, N., Weyns, D., Schelfthout, K., Holvoet, T. (2006). "*Applying the ATAM to an Architecture for Decentralized Control of a Transportation System*". En actas de: Second International Conference on Quality of Software Architectures, Västerås, Sweden. pp. 181–199.

Brambilla, M., Cabot, J., Wimmer, M. (2012). "*Model-Driven Software Engineering in Practice Synthesis Lectures on Software Engineering*". Morgan & Claypool Publishers.

Breivold, H.P., Crnkovic, I., Larsson, M. (2012). "*A systematic review of software architecture evolution research*". Information and Software Technology, Vol. 54, Num. 1, pp. 16–40.

Brun, C., Pierantonio, A. (2008). "*Model differences in the eclipse modeling framework*". UPGRADE, The European Journal for the Informatics Professional, Vol. 2, Num. IX, pp. 29–34.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). "*Pattern-Oriented Software Architecture Volume 1: A System of Patterns*". John Wiley & Sons: Chichester.

Cabello, M.E. (2008). "*Baseline-Oriented Modeling: una Aproximación Mda Basada en Líneas de Productos Software para el Desarrollo de Aplicaciones*". PhD Thesis, Departamento de Sistemas Informáticos y Computación, Universitat Poltècnica de València.

Cabello, M.E., Ramos, I., Gomez, A., Limon, R. (2009). "*Baseline-Oriented Modeling: An MDA Approach Based on Software Product Lines for the Expert Systems Development*". En actas de: First Asian Conference on Intelligent Information and Database Systems, Dong hoi, Quang binh, Vietnam. pp. 208–213.

Carifio, J., Perla, R. (2007). "*Ten common misunderstandings, misconceptions, persistent myths and urban legends about Likert scales and Likert response formats and their antidotes*". Journal of Social Sciences, Num. 3, pp. 106–116.

Chen, L., Ali Babar, M., Ali, N. (2009). "*Variability management in software product lines: a systematic review*". En actas de: 13th International Software Product Line Conference, San Francisco, USA. pp. 81–90.

Ciolkowski, M., Shull, F., Biffl, S. (2002). "*A family of experiments to investigate the influence of context on the effect of inspection techniques*". En actas de: 6th International Conference on Empirical Assesment in Software Engineering, pp. 48–60.

Clauß, M. (2001). "*Modeling variability with UML*". En actas de: Young Researchers Workshop, Generative and Component-Based SW Engineering, Messe Erfurt, Erfurt, Germany.

Clements, P., Garlan, D., Bass, L., Stafford, J. (2011). "*Documenting software architectures: views and beyond*" 2nd Editio. Pearson Education.

Clements, P., Kazman, R., Klein, M. (2002). "*Evaluating software architectures: methods and case studies.*". Addison-Wesley Professional.

Clements, P., Northrop, L. (2001). "*Software Product Lines: Practices and Patterns*". Addison-Wesley Professional.

Clements, P.C., Northrop, L.M. (2002). "*Salion , Inc .: A Software Product Line Case Study*". CMU/SEI-2002-TR-038, ESC-TR-2002-038, Software Engineering Institute, Carnegie Mellon University.

Colosimo, M., Lucia, A., Scanniello, G., Tortora, G. (2009). "*Evaluating legacy system migration technologies through empirical studies*". Information and Software Technology, Vol. 51, Num. 2, pp. 433–447.

Conover, W. (1998). "*Practical nonparametric statistics*". John Wiley & Sons: New York, USA.

Cook, T., Campbell, D. (1979). "*Quasi-experimentation: Design & analysis issues for field settings*". Houghton Mifflin Company.

Crnkovic, I., Larsson, M., Preiss, O. (2004). "*Concerning Predictability in Dependable Component- Based Systems: Classification of Quality Attributes*". En actas de: ICSE 2004 Workshops on Software Architectures for Dependable Systems, Edinburgh, Scotland, UK. pp. 257–278.

Cuenot, P., Frey, P., Johansson, R. (2011). "*11 The EAST-ADL Architecture Description Language for Automotive Embedded Software*". Model-Based Engineering of Embedded Real-Time Systems, LNCS 6100. pp. 297–307.

Czarnecki, K., Antkiewicz, M. (2005). "*Mapping features to models: A template approach based on superimposed variants*". En actas de: 4th International Conference on Generative Programming and Component Engineering, Tallin, Estonia. pp. 423–437.

Czarnecki, K., Kim, C.H.P. (2005). "*Cardinality-Based Feature Modeling and Constraints: A Progress Report*". University Ave.WestWaterloo, ON N2L 3G1, Canada: University of Waterloo.

Dajsuren, Y., Brand, M. Van Den, Box, P.O., Huisman, R. (2012). "*Automotive ADLs: A Study on Enforcing Consistency Through Multiple Architectural Levels Categories and Subject Descriptors*". En actas de: Eighth International ACM Sigsoft Conference on the Quality of Software Architectures, Bertinoro, Italy. pp. 71–80.

Dantzig, G. (1951). "*Application of the simplex method to a transportation problem*". Activity analysis of production and allocation, Vol. 13, pp. 359–373.

Dattorro, J. (2005). "*Convex optimization and Euclidean distance geometry*". Meboo Publishing.

Davis, F. (1989). "*Perceived usefulness, perceived ease of use, and user acceptance of information technology*". MIS quarterly.

Deelstra, S., Sinnema, M., Bosch, J. (2005). "*Product derivation in software product families: a case study*". Journal of Systems and Software, Vol. 74, Num. 2, pp. 173–194.

Dershowitz, N., Jouannaud, J. (1990). "*Rewrite systems*". Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics. pp. 243–320.

Dolan, T.J. (2001). "*Architecture assessment of information-system families: a practical perspective*". PhD Thesis, Technische Universiteit Eindhoven.

Douglass, B.P. (2002). "*Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*". Addison Wesley.

Duran-Limon, H.A., Castillo-Barrera, F.E., Lopez-Herrejon, R.E. (2011). "*Towards an ontology-based approach for deriving product architectures*". En actas de: 15th International Software Product Line Conference, Munich, Germany. p. Volume 2, Article 29.

Dyba, T., Kitchenham, B.A., Jorgensen, M. (2005). "*Evidence-based software engineering for practitioners*". IEEE Software, Vol. 22, Num. 1, pp. 58–65.

Dzidek, W.J., Arisholm, E., Briand, L. (2008). "*A realistic empirical evaluation of the costs and benefits of UML in software maintenance*". IEEE Transactions on Software Engineering, Vol. 34, Num. 3, pp. 407–432.

Eclipse (2013)(a). "*Eclipse Modeling Framework Project (EMF)*". http://www.eclipse.org/modeling/emf/.

Eclipse (2013)(b). "*Eclipse OCL*". http://projects.eclipse.org/projects/modeling.mdt.ocl.

Eclipse (2013)(c). "*Eclipse Public License*".

Etemaadi, R., Lind, K., Heldal, R., Chaudron, M.R. V (2013). "*Quality-driven optimization of system architecture: Industrial case study on an automotive sub-system*". The Journal of Systems & Software, Vol. 86, Num. 10, pp. 2559–2573.

Etxeberria, L. (2008). "*Evaluación de atributos de calidad en líneas de productos software de forma efectiva en costes*". PhD Thesis, Departamento de Electonica e Informatica, Modragon Unibesitatea.

Etxeberria, L., Sagardui, G. (2005). "*Product-Line Architecture: New Issues for Evaluation*". En actas de: 9th Software Product Lines Conference, Rennes, France. pp. 174–185.

Etxeberria, L., Sagardui, G. (2008). "*Variability Driven Quality Evaluation in Software Product Lines*". En actas de: 2008 12th International Software Product Line Conference, Limerick, Ireland. pp. 243–252.

Etxeberria, L., Sagardui, G., Belategi, L. (2008). "*Quality aware Software Product Line Engineering*". Journal of the Brazilian Computer Society, Vol. 14, Num. 1, pp. 57–69.

Falessi, D., Ali Babar, M., Cantone, G., Kruchten, P. (2009)(a). "*Applying empirical software engineering to software architecture: challenges and lessons learned*". Empirical Software Engineering, Vol. 15, Num. 3, pp. 250–276.

Falessi, D., Ali Babar, M., Cantone, G., Kruchten, P. (2009)(b). "*Applying empirical software engineering to software architecture: challenges and lessons learned*". Empirical Software Engineering, Vol. 15, Num. 3, pp. 250–276.

Falessi, D., Cantone, G., Kazman, R., Kruchten, P. (2011). "*Decision-making techniques for software architecture design*". ACM Computing Surveys, Vol. 43, Num. 4, pp. 1–28.

Falessi, D., Cantone, G., Kruchten, P. (2008). "*Value-Based Design Decision Rationale Documentation: Principles and Empirical Feasibility Study*". Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), pp. 189–198.

Falessi, D., Capilla, R., Carlos, J., Cantone, G. (2008). "*A Value-Based Approach for Documenting Design Decisions Rationale: A Replicated Experiment*". En actas de: 3rd International Workshop on Sharing and Reusing Architectural Knowledge, Leipzig, Germany. pp. 63–69.

Favre, J.-M. (2004)(a). "*Foundations of Model (Driven) (Reverse) Engineering: Models -- Episode I: Stories of the Fidus Papyrus and of the Solarus*". En actas de: Dagsthul Seminar on Model Driven Reverse Engineering, Wadern, Merzig-Wadern, Germany.

Favre, J.-M. (2005). "*Megamodelling and Etymology*". En actas de: Dagstuhl Seminar on Transformation Techniques in Software Engineering, Wadern, Merzig-Wadern, Germany.

Favre, J.-M. (2004)(b). "*Towards a Basic Theory to Model Model Driven Engineering*". En actas de: 3rd International Workshop on Software Model Engineering, Lisbon, Portugal.

Feiler, P. (2007)(a). "*Flow Latency Analysis with the Architecture Analysis and Design Language (AADL)*". CMU/SEI-2007-TN-010, Software Engineering Institute, Carnegie Mellon University.

Feiler, P. (2007)(b). "*Modeling of System Families*". CMU/SEI-2007-TN-047, Software Engineering Institute, Carnegie Mellon University.

Feiler, P.H., Gluch, D.P. (2013). "*Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*". Addison Wesley.

Feiler, P.H., Lewis, B.A., Vestal, S., History, A. (2006). "*The SAE Architecture Analysis & Design Language (AADL) a Standard for Engineering Performance Critical Systems*". pp. 1206–1211.

Ferber, S., Heidl, P., Lutz, P. (2001). "*Reviewing Product Line Architectures: Experience Report of ATAM in an Automotive Context*". En actas de: 4th International Workshop on Software Product-Family Engineering, pp. 364–382.

Fernandez, A., Abrahão, S., Insfran, E. (2011). "*A Web Usability Evaluation Process for Model-Driven Web Development*". En actas de: 23rd International Conference on Advanced Information Systems Engineering, London, UK. pp. 108–122.

Fleurey, F., Haugen, Ø., Møller-Pedersen, B. (2009). "*A generic language and tool for variability modeling*". SINTEF, Oslo, .

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). "*Design Patterns. Elements of Reusable Object-Oriented oftware*". Addison Wesley.

Gannod, G., Lutz, R. (2000). "*An approach to architectural analysis of product lines*". En actas de: 22nd International Conference on Software Engineering, Limerick, Ireland. pp. 548–557.

Ghezzi, C., Sharifloo, A.M. (2011). "*Verifying Non-functional Properties of Software Product Lines: Towards an Efficient Approach Using Parametric Model Checking*". En actas de: 15th International Software Product Line Conference, Munich, Germany. pp. 170–174.

Glass, G., McGaw, B., Smith, M. (1981). "*Meta-analysis in social research*". Sage Publications.

Glinz, M. (2007). "*On Non-Functional Requirements*". En actas de: 15th IEEE International Requirements Engineering Conference (RE 2007), Delhi, India. pp. 21–26.

Golden, E., John, B.E. (2005). "*The Value of a Usability-Supporting Architectural Pattern in Software Architecture Design: A Controlled Experiment Categories and Subject Descriptors*". En actas de: 27th International Conference on Software Engineering, ST Louis, MI, USA. pp. 460–469.

Gomaa, H. (2004). "*Designing software product lines with UML*". Addison-Wesley Object Technology Series: Boston, ME, USA.

Gomaa, H., Shin, M.E. (2007). "*Automated Software Product Line Engineering*". En actas de: 40th Annual Hawaii International Conference on System Science, Hawaii, USA. pp. 1–10.

Gómez, A. (2012). "*Model Driven Software Product Line Engineering: System Variability View and Process*". PhD Thesis, Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València.

Gómez, A., Boronat Moll, A., Carsí Cubel, J.Á. (2006). "*Soporte Gráfico para Trazabilidad en una Herramienta de Gestión de Modelos*". Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València.

Gómez, A., Ramos, I. (2010). "*Cardinality-Based Feature Modeling and Model-Driven Engineering : Fitting them Together*". En actas de: International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria. pp. 61–68.

González-Baixauli, B., Sampaio do Prado Leite, J.C., Mylopoulos, J. (2004). "*Visual Variability Analysis for Goal Models Julio Cesar Sampaio do Prado Leite*". En actas de: 12th IEEE International Requirements Engineering Conference, Kyoto, Japan. pp. 198–207.

Gorschek, T., Wohlin, C., Garre, P., Larsson, S. (2006). "*A Model for Technology Transfer in Practice*". IEEE Software, Vol. 23, Num. 6, pp. 88–95.

Graaf, B., Dijk, H. van, Deursen, A. van (2005). "*Evaluating an Embedded Software Reference Architecture—Industrial Experience Report—*". En actas de: 9th European Conference on Software Maintenance and Reengineering, Manchester, United Kingdom.

Griss, M. (2000). "*Implementing product-line features with component reuse*". En actas de: 6th International Conference on Software Reuse, Vienna, Austria. pp. 137–152.

Guana, V., Correal, D. (2013). "*Improving software product line configuration: A quality attribute-driven approach*". Information and Software Technology, Vol. 55, Num. 3, pp. 541–562.

Guana, V., Correal, D. (2011). "*Variability quality evaluation on component-based software product lines*". En actas de: 15th Software Product Line Conference, New York, USA. p. Article No. 19.

Guessi, M., Box, P.O., Carlos, S., Oquendo, F. (2012). "*Architectural Description of Embedded Systems : A Systematic Review*". En actas de: 3rd international ACM SIGSOFT symposium on Architecting Critical Systems, Bertinoro, Italy. pp. 31–40.

Van Gurp, J., Bosch, J. (2002). "*Design erosion: problems and causes*". Journal of Systems and Software, Vol. 61, Num. 2, pp. 105–119.

Haugen, Ø., Moller-Pedersen, B., Olsen, G.K., Svendsen, A., Fleurey, F., Zhang, X. (2010). "*Consolidated CVL language and tool*". MoSiS Project, D.2.1.4., SINTEF, Univeristy of Oslo.

Hebig, R., Seibel, A., Giese, H. (2012). "*On the Unification of Megamodels*". Electronic Communications of the EASST, Vol. 42.

Hedges, L., Olkin, I. (1985). "*Statistical Methods for Meta-Analysis*". Academia Press.

Heidenreich, F., Kopcsek, J., Wende, C. (2008). "*FeatureMapper: mapping features to models*". En actas de: Companion of the 30th international conference on Software engineering, Vancouver, Canada. pp. 943–944.

Her, J.S., Kim, J.H., Oh, S.H., Rhew, S.Y., Kim, S.D. (2007). "*A framework for evaluating reusability of core asset in product line engineering*". Information and Software Technology, Vol. 49, Num. 7, pp. 740–760.

Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C. (2004). "*A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0*". http:/resources/tutorials/ProtegeOWLTutorial.pdf.

Höst, M., Regnell, B., Wohlin, C. (2000). "*Using students as subjects—a comparative study of students and professionals in lead-time impact assessment*". Empirical Software Engineering, Vol. 5, pp. 201–214.

Hu, P., Chau, P. (1999). "*Examining the technology acceptance model using physician acceptance of telemedicine technology*". Journal of Management Information Systems, Vol. 16, Num. 2, pp. 91–112.

Huet, G. (1980). "*Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems*". Journal of the ACM, Vol. 27, Num. 4, pp. 797–821.

Hunter, J.E., Schmidt, F.L., Jackson, G.B. (1982). "*Meta-Analysis. Cumulating research findings across studies*". Sage Publications.

ISA Research Group (2011). "*Fama Tool Suite*". http://www.isa.us.es/fama/.

ISO (2011). "*ISO / IEC / IEEE 42010:2011 Systems and software engineering*".

ISO (2005). "*ISO/IEC 25000:2005. Software Engineering. Software product Quality Requirements and Evaluation SQuaRE*".

Janota, M., Botterweck, G. (2008). "*Formal approach to integrating feature and architecture models*". En actas de: 11th Conference on Fundamenteal Approaches to Software Engineering, Budapest, Hungary. pp. 31–45.

Jarzabek, S., Yang, B., Yoeun, S. (2006). "*Addressing quality attributes in domain analysis for product lines*". IEE Proceedings - Software, Vol. 153, Num. 2, pp. 61–73.

Jazayeri, M., Ran, A.C.M., van der Linden, F., Ran, A. (2000). "*Software Architecture for Product Families: Principles and Practice*". Addison-Wesley.

Juristo, N., Moreno, A. (2001). "*Basics of Software Engineering Experimentation*". Kluwer Academic Publishers.

Kampenes, V.B., Dybå, T., Hannay, J.E., Sjøberg, D.I.K. (2007). "*A systematic review of effect size in software engineering experiments*". Information and Software Technology, Vol. 49, Num. 11-12, pp. 1073–1086.

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S. (1990). "*Feature-Oriented Domain Analysis (FODA) Feasibility Study*". CMU/SEI-90-TR-21 ESD-90-TR-222, Software Engineering Institute, Carnegie Melon University.

Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M. (1998). "*FORM: A feature-;oriented reuse method with domain-;specific reference architectures*". Annals of Software Engineering, Vol. 5, Num. 1, pp. 143–168.

Kazman, R., Bass, L., Abowd, G., Webb, M. (1994). "*SAAM: a method for analyzing the properties of software architectures*". En actas de: 16th International Conference on Software Engineering, Sorrento, Italy. pp. 81–90.

Kazman, R., Klein, M., Clements, P. (2000). "*ATAM: Method for architecture evaluation*". CMU/SEI-2000-TR-004, ESC-TR-2000-004, Software Engineering Institute, Carnegie Mellon University.

Khachan, C. (2012). "*Un Framework para el Análisis Automático de Líneas de Producto Software*". Master Thesis, Master en Ingeniería del Software, Metodos Formales y Sistemas de Información, Univeristat Politècnica de València.

Kim, T.K.T., Ko, I.Y.K.I.Y., Kang, S.W.K.S.W., Lee, D.H.L.D.H. (2008). "*Extending ATAM to assess product line architecture*". 2008 8th IEEE International Conference on Computer and Information Technology, Vol. 2, pp. 790–797.

Kitchenham, B. (2008). "*The role of replications in empirical software engineering—a word of warning*". Empirical Software Engineering, Vol. 13, Num. 2, pp. 219–221.

Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J. (2002). "*Preliminary guidelines for empirical research in software engineering*". IEEE Transactions on Software Engineering, Vol. 28, Num. 8, pp. 721–734.

Kleppe, A., Warmer, J., Bast, W. (2003). "*MDA Explained: The Model Driven Architecture: Practice and Promise (Addison-Wesley Object Technology)*". Addison Wesley.

Kruchten, P. (1995). "*Architectural Blueprints — The " 4 + 1 " View Model of Software Architecture*". IEEE Software, Vol. 12, Num. November, pp. 42–50.

Kruchten, P. (1999). "*The Rational Unified Process, an Introduction*". Addison Wesley: Boston.

Krueger, C. (2002). "*Eliminating the Adoption Barrier*". IEEE Software, Vol. 19, Num. August, pp. 29–31.

Kühne, T. (2005). "*What is a Model?*". En actas de: Dagstuhl Seminar on Language Engineering for Model-Driven Software Development, Wadern, Merzig-Wadern, Germany.

Lankhorst, M., Proper, H., Jonkers, H. (2009). "*The architecture of the archimate language*". Enterprise, Enterprise, Business-Process and Information Systems Modeling, pp. 367–380.

Lewis, B.A., Feiler, P., Sokolski, O., Hugues, J. (2012). "*OSATE TOOL*". https://wiki.sei.cmu.edu/aadl/index.php/OSATE_1510.

Li, R., Etemaadi, R., Emmerich, M.T.M., Chaudron, M.R. V. (2011). "*An evolutionary multiobjective optimization approach to component-based software architecture design*". En actas de: 2011 IEEE Congress of Evolutionary Computation, New Orleans, USA. pp. 432–439.

Li, Y., Zhu, Y., Ma, C., Xu, M. (2011). "*A Method for Constructing Fault Trees from AADL Models*". En actas de: 8th International Conference on Autonomic and Trusted Computing, Banff, Canada. pp. 243–258.

Van der Linden, F., Schmid, K., Rommes, E. (2007). "*Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, 2007*". Springer Berlin Heidelberg.

Lindsay, R., Ehrenberg, A. (1993). "*The design of replicated studies*". The American Statistician, Vol. 47, Num. 3, pp. 217–228.

Lindvall, M., Tvedt, R.T., Costa, P. (2003). "*An Empirically-Based Process for Software Architecture Evaluation*". Empirical Software Engineering, Vol. 8, Num. 1, pp. 83–108.

Loughran, N., Sánchez, P., Gámez, N., Garcia, A., Fuentes, L., Kovacevic, J. (2007). "*Survey on State-of-the-Art in Product Line Architecture Design*". AMPLE E.C. Project Deliverable D.2.1.

Loughran, N., Sánchez, P., Garcia, A., Fuentes, L. (2008). "*Language support for managing variability in architectural models*". En actas de: 7th International Symposium on Software Composition, Budapest, Hungary. pp. 36–51.

Maccari, A. (2002). "*Experiences in assessing product family software architecture for evolution*". En actas de: 24th International Conference on Software Engineering, Orlando, Florida. pp. 585–592.

Martens, A., Koziolek, H., Prechelt, L., Reussner, R. (2010). "*From monolithic to component-based performance evaluation of software architectures*". Empirical Software Engineering, Vol. 16, Num. 5, pp. 587–622.

Mårtensson, F. (2006). "*Software Architecture Quality Evaluation Approaches in an Industrial Context*". PhD Thesis, Department of Systems and Software Engineering, Blekinge Institute of Technology.

Matinlassi, M., Niemelä, E., Dobrica, L. (2002). "*Quality-driven architecture design and quality analysis method*". VTT Publications 456,VTT Technical Research Centre of Finland, Oulu, Findland.

Maxwell, K. (2002). "*Applied statistics for software managers*". Prentice Hall.

McCabe, T.J. (1976). "*A Complexity Measure*". IEEE Transactions on Software Engineering, Vol. SE-2, Num. 4, pp. 308–320.

McGregor, J.D. (2010). "*Attached Processes*". Journal of Object Technology, Vol. 9, Num. 2, pp. 7–16.

Mens, T., Czarnecki, K., Gorp, P. Van (2005). "*A taxonomy of model transformation*". En actas de: International Workshop on Graph and Model Transformation, Tallin, Estonia. pp. 1–10.

Microsoft MSDN (2003). "*Performance and Reliability Patterns. Versión 1.1.0*". http://msdn.microsoft.com/en-us/library/ff648802.aspx.

Montagud, S. (2009). "*Un Método para la Evaluación de la Calidad de Líneas de Productos Software basado en SQuaRE*". Tesis de Master, Master en Ingeniería del Software, Metodos Formales y Sistemas de Información, Univesitat Politécnica de Valéncia.

Montagud, S., Abrahão, S. (2009). "*Gathering current knowledge about quality evaluation in software product lines*". En actas de: 13th Software Product Line Conference, San Francisco, USA. pp. 91–100.

Montagud, S., Abrahão, S., Insfran, E. (2011). "*A systematic review of quality attributes and measures for software product lines*". Software Quality Journal, Vol. 20, Num. 3-4, pp. 425–486.

Mora, B., García, F., Ruiz, F., Piattini, M. (2008). "*Software Measurement by Using QVT Transformations in an MDA Context*". En actas de: 10th International Conference on Enterprise Information Systems, Barcelona, Spain. pp. 117–124.

Mullen, B., Rosenthal, R. (1985). "*Basic Meta-Analysis: Procedures and Programs*". L. Earlbaum Associates.

Nelder, J., Mead, R. (1965). "*A simplex method for function minimization*". The computer journal, Vol. 7, Num. 4, pp. 308–313.

Niemelä, E. (2005). "*Architecture centric software family engineering*". En actas de: Product Family Engineering Seminar, Oulu, Finland. pp. 1–24.

Niemelä, E., Immonen, A. (2007). "*Capturing quality requirements of product family architecture*". Information and Software Technology, Vol. 49, Num. 11-12, pp. 1107–1120.

O'Leary, P., de Almeida, E.S., Richardson, I. (2012). "*The Pro-PD Process Model for Product Derivation within software product lines*". Information and Software Technology, Vol. 54, Num. 9, pp. 1014–1028.

Object Management Group (2012)(a). "*Common Variability Language (CVL) OMG Revised Submission*".

Object Management Group (2003). "*MDA Guide Version 1.0.1*".

Object Management Group (2008)(a). "*Meta Object Facility (MOF) 2.0 Query / View / Transformation Specification*".

Object Management Group (2006). "*Meta Object Facility (MOF) Core Specification*".

Object Management Group (2008)(b). "*Software & Systems Process Engineering Meta-Model Specification (SPEM) v2.0*".

Object Management Group (2012)(b). "*Systems Modeling Language (OMG SysML) v1.3*".

Oliveira, R.P. De, Insfran, E., Abrahão, S., Gonzalez-huerta, J., Blanes, D. (2013). "*A Feature-Driven Requirements Engineering Approach for Software Product Lines*". En actas de: 7th Brazilian Symposium on Software Components, Architectures and Reuse, Brasilia, Brazil.

Olumofin, F.G., Misic, V.B. (2007). "*A holistic architecture assessment method for software product lines*". Information and Software Technology, Vol. 49, Num. 4, pp. 309–323.

Ommering, R. van, van der Linden, F., Kramer, J., Magee, J. (2000). "*Model for Consumer Electronics Software A component-oriented approach is an ideal way to handle the diversity of*". IEEE Computer, Vol. 33, Num. 3, pp. 78–85.

OSGI Alliance (2012). "*Osgi 5.0 Specification*".

Perez, J. (2006). "*PRISMA: Aspect-Oriented Software Architectures*". PhD Thesis, Departamento de Sistemas Informaticos y Computacion, Universitat Politècnica de València.

Perovich, D., Bastarrica, M.C., Rojas, C. (2009). "*Model-Driven approach to Software Architecture design*". En actas de: 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge, Vancouver, Canada. pp. 1–8.

Perovich, D., Rossel, P.O., Bastarrica, M.C. (2009). "*Feature model to product architectures: Applying MDE to Software Product Lines*". En actas de: 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, Cambridge, UK. pp. 201–210.

Perrouin, G., Klein, J., Guelfi, N., Jézéquel, J.M. (2008). "*Reconciling automation and flexibility in product derivation*". En actas de: 12th Software Product Line Conference, Limerick, Ireland. pp. 339–348.

Pohl, K., Böckle, G., van der Linden, F. (2005). "*Software product line engineering*". Springer: Berlin.

Qureshi, N., Usman, M., Ikram, N. (2013). "*Evidence in Software Architecture, a Systematic Literature Review*". En actas de: 17th International Conference on Evaluation and Assessment in Software Engineering, Porto de Galinhas, Brazi. pp. 97–106.

Rabiser, R., Dhungana, D. (2007). "*Integrated Support for Product Configuration and Requirements Engineering in Product Derivation*". En actas de: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007), Lübeck, Germany. pp. 219–228.

Rabiser, R., Grünbacher, P., Dhungana, D. (2010). "*Requirements for product derivation support: Results from a systematic literature review and an expert survey*". Information and Software Technology, Vol. 52, Num. 3, pp. 324–346.

Rabiser, R., O'Leary, P., Richardson, I. (2011). "*Key activities for product derivation in software product lines*". Journal of Systems and Software, Vol. 84, Num. 2, pp. 285–300.

Rahman, A. (2004). "*Metrics for the structural assessment of product line architecture*". Master Thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden.

Reijonen, V., Koskinen, J., Haikala, I. (2010). "*Experiences from Scenario-Based Architecture Evaluations with ATAM*". En actas de: 4th International Conference on Software Architecture, Copenhagen, Denmark. pp. 214–229.

Reis, S., Metzger, A., Pohl, K. (2006). "*A reuse technique for performance testing of software product lines*". En actas de: 3rd International Workshop on Software Product Line Testing, Baltimore, USA.

Riebisch, M., Böllert, K., Streitferdt, D., Philippow, I. (2002). "*Extending Feature Diagrams with UML Multiplicities*". En actas de: 6th World Conference on Integrated Design & Process Technology, Pasadena, CA, USA. pp. 1–7.

Riva, C., Rosso, C. Del (2003). "*Experiences with software product family evolution*". En actas de: 6th International Workshop on Principles of Software Evolution, Helsinki, Finland.

Robertson, S., Robertson, J. (1999). "*Mastering the requirements process*". Addison Wesley.

Robson, C. (2002). "*Real world research: A resource for social scientists and practitioners-researchers*" 2nd Editio. Blackwell Publishers Ltd.: Oxford, UK.

Roos-Frantz, F., Benavides, D., Ruiz-Cortés, A., Heuer, A., Lauenroth, K. (2011). "*Quality-aware analysis in product line engineering with the orthogonal variability model*". Software Quality Journal, Vol. 20, Num. 3-4, pp. 519–565.

Rosenthal, R. (1986). "*Meta-analytic procedures for social research*". Sage Publications.

Roshandel, R., Banerjee, S., Cheung, L., Medvidovic, N., Golubchik, L. (2006). "*Estimating software component reliability by leveraging architectural models*". En actas de: Proceeding of the 28th international conference on Software engineering - ICSE '06, New York, New York, USA. p. 853.

Roy, B., Graham, T.C.N. (2008). "*Methods for Evaluating Software Architecture: A Survey*". School of Computing, Queen's University at Kingston, Ontario, Canada.

Ruiz, F., Verdugo, J. (2008). "*Guía de Uso de SPEM 2 con EPF Composer*". Grupo Alarcos, Escuela Superior de Informatica, Universidad de Castilla la Mancha, Ciudad Real, Spain.

Runeson, P., Höst, M. (2008). "*Guidelines for conducting and reporting case study research in software engineering*". Empirical Software Engineering, Vol. 14, Num. 2, pp. 131–164.

Runeson, P., Host, M., Rainer, A., Regnell, B. (2012). "*Case Study Research in Software Engineering: Guidelines and Examples*". John Wiley & Sons.

Saaty, T. (2008). "*Decision making with the analytic hierarchy process*". International Journal of Services Sciences, Vol. 1, Num. 1, pp. 83–98.

Salay, R., Mylopoulos, J., Easterbrook, S. (2008). "*Managing Models through Macromodeling*". En actas de: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, L'Aquila, Italy. pp. 447–450.

Sánchez, P., Gámez, N., Fuentes, L., Loughran, N., Garcia, A. (2007). "*A Metamodel for Designing Software Architectures of Aspect-Oriented Software Product Lines*". AMPLE Project, Deliverable D2.2.

Sánchez, P., Loughran, N., Fuentes, L., Garcia, A. (2008). "*Engineering languages for specifying product-derivation processes in software product lines*". En actas de: Software Languaje Engineering, Toulousse, France. pp. 188–208.

Schaefer, I., Worret, A., Poetzsch-Heffter, A. (2009). "*A model-based framework for automated product derivation*". En actas de: 1st International Workshop on Model-driven Approaches in Software Product Line Engineering, San Francisco, California, USA. pp. 14–21.

Schwarzer, R. (1987). "*Meta-analysis programs*". http://userpage.fu-berlin.de/~helath/meta_e.htm.

Schwarzer, R. (1988). "*Meta-analysis programs*". Behavior Research Methods, Instruments, & Computers, Vol. 20, Num. 3, pp. 338–338.

Seaman, C.B. (1999). "*Qualitative methods in empirical studies of software engineering*". IEEE Transactions on Software Engineering, Vol. 25, Num. 4, pp. 557–572.

Seibel, A., Neumann, S., Giese, H. (2009). "*Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance*". Software & Systems Modeling, Vol. 9, Num. 4, pp. 493–528.

Seidewitz, E. (2003). "*What models mean*". IEEE Software, Vol. 20, Num. 5, pp. 26–32.

Shaw, M., Garlan, D. (1996). "*Software Architecture: Perspectives on an Emerging Discipline*". Prentice Hall, Upper Saddle River: New Jersey, USA.

Shiraishi, S. (2010). "*An AADL-based approach to variability modeling of automotive control systems*". En actas de: 13h Model Driven Engineering Languages and Systems, Oslo, Norway. pp. 346–360.

Shiraishi, S. (2013). "*Qualitative Comparison of ADL-Based Approaches to Real-World Automotive System Development*". Journal of Information Processing, Vol. 21, Num. 1, pp. 34–45.

Shull, F., Carver, J., Vegas, S., Juristo, N. (2008). "*The role of replications in empirical software engineering*". Empirical Software Engineering, Vol. 13, pp. 211–218.

Shull, F., Mendoncça, M.G., Basili, V., Carver, J., Maldonado, J.C., Fabbri, S., Travassos, G.H., Ferreira, M.C. (2004). "*Knowledge-Sharing Issues in Experimental Software Engineering*". Empirical Software Engineering, Vol. 9, Num. 1/2, pp. 111–137.

Siegmund, N., Kuhlemann, M., Rosenmüller, M., Kästner, C., Saake, G. (2008). "*Integrated Product Line Model for Semi-Automated Product Derivation Using Non-Functional Properties.*". En actas de: Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems VAMOS 2008, Essen, Germany. pp. 25–32.

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G. (2011). "*SPL Conqueror: Toward optimization of non-functional properties in software product lines*". Software Quality Journal, Vol. 20, Num. 3-4, pp. 487–517.

Sinnema, M., Deelstra, S., Hoekstra, P. (2006). "*The COVAMOF Derivation Process*". En actas de: M. Morisioed. 9th International Conference on Software Reuse, Turin, Italy. pp. 102–114.

Sjøberg, D., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Vokác, M. (2003). "*Challenges and Recommendations When Increasing the Realism of Controlled Software Engineering Experiments*". Empirical Methods and Studies in Software Engineering, Experiences from ESERNET. pp. 24–38.

Sjøberg, D., Hannay, J., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.-K., Rekdal, A.C. (2005). "*A survey of controlled experiments in software engineering*". IEEE Transactions on Software Engineering, Vol. 31, Num. 9, pp. 733–753.

Soltani, S., Asadi, M., Gašević, D. (2012). "*Automated planning for feature model configuration based on functional and non-functional requirements*". En actas de: 16th Software Product Line Conference, Salvador de Bahia, Brazil. pp. 56–65.

Stahl, T., Völter, M., Bettin, J., Haase, A., Helsen, S. (2006). "*Model-Driven Software Development: Technology, Engineering, Management*". Wiley.

Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., Ralisback, J. (2002). "*Fault tree handbook with aerospace applications*". NASA Office of Safety and Mission Assurance, Washington DC, USA.

Sutton, A.J., Abrams, K.R., Jones, R.D., Sheldon, A.T., Song, F. (2001). "*Methods for meta-analysis in medical research*". John-Wiley & Sons.

Svahnberg, M., Mårtensson, F. (2007). "*Six years of evaluating software architectures in student projects*". Journal of Systems and Software, Vol. 80, Num. 11, pp. 1893–1901.

Taher, L., El Khatib, H., Basha, R. (2005). "*A framework and QoS matchmaking algorithm for dynamic web services selection*". En actas de: 2nd International Conference on Innovations in Information Technology, Dubai, UAE.

Tawhid, R., Petriu, D.C. (2011)(a). "*Automatic Derivation of a Product Performance Model from a Software Product Line Model*". En actas de: 15th International Software Product Line Conference, Munich, Germany. pp. 80–89.

Tawhid, R., Petriu, D.C. (2011)(b). "*Product Model Derivation by Model Transformation in Software Product Lines*". En actas de: 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, Newport Beach, Canada. pp. 72–79.

Thiel, S., Hein, A. (2002). "*Systematic integration of variability into product line architecture design*". En actas de: 2nd Software Product Line Conference, London, UK. pp. 130–153.

Trendowicz, A., Punter, T. (2003). "*Quality Modeling for Software Product Lines*". En actas de: 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Darmstadt, Germany. pp. 7–10.

Vegas, S., Juristo, N., Moreno, A., Solari, M., Letelier, P. (2006). "*Analysis of the influence of communication between researchers on experiment replication*". En actas de: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering - ISESE '06, New York, New York, USA. p. 28.

Venkatesh, V. (2000). "*Determinants of perceived ease of use: Integrating control, intrinsic motivation, and emotion into the technology acceptance model*". Information systems research, Vol. 11, Num. 4, pp. 342–365.

Wang, W., Chen, M. (1999). "*An Architecture-Based Software Reliability Model Component-Based Reliability Modeling Style-based Reliability Modeling*". En actas de: 1999 Pacific Rim International Simposium on Dependable Computing, Hong Kong, China. pp. 143–150.

Williams, L., Smith, C. (2002). "*PASA SM: a method for the performance assessment of software architectures*". Workshop on Software and Performance, Rome, Italy pp. 179–189.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2012). "*Experimentation in Software Engineering*". Springer Heidelberg.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2000). "*Experimentation in software engineering: An introduction*". Kluwer Academic Publishers.

Yin, R. (2009). "*Case study research: Design and methods*" Sage.

Zelkowitz, M. V., Wallace, D.R. (1998). "*Experimental models for validating technology*". Computer, Vol. 31, Num. 5, pp. 23–31.

Zhang, G., Ye, H., Lin, Y. (2013). "*Quality attribute modeling and quality aware product configuration in software product lines*". Software Quality Journal, Vol. 21, Num. 1.

Zhang, G., Ye, H., Lin, Y. (2010). "*Quality Attributes Assessment for Feature-Based Product Configuration in Software Product Line*". En actas de: 2010 Asia Pacific Software Engineering Conference, Sydney, Australia. pp. 137–146.

Zhang, T., Deng, L., Wu, J., Zhou, Q., Ma, C. (2008). "*Some Metrics for Accessing Quality of Product Line Architecture*". En actas de: 2008 International Conference on Computer Science and Software Engineering, Richmond Hill, Ontario, Canada. pp. 500–503.

Zhang, X., Haugen, Ø., Moller-Pedersen, B. (2011). "*Model Comparison to Synthesize a Model-Driven Software Product Line*". En actas de: 2011 15th International Software Product Line Conference, Munich, Germany. pp. 90–99.

Ziadi, T., Jézéquel, J. (2006). "*Software product line engineering with the UML: Deriving products*". En actas de: 10th Software Product Lines Conference, Baltimore, Maryland, USA. pp. 557–588.

Después de escalar una montaña muy alta, descubrimos que hay muchas otras montañas por escalar.

Nelson Mandela (1918-2013)