Una Aproximación de Ingeniería de Requisitos para Líneas de Productos Software Basada en una Estrategia de Desarrollo Dirigido por Modelos

David Blanes Domínguez

Directores:

Dr. Emilio Insfrán Pelozo | Dr. Javier González Huerta

Departamento de Sistemas Informáticos y Computación

5 de Febrero 2016

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

Una Aproximación de Ingeniería de Requisitos para Líneas de Productos Software Basada en una Estrategia de Desarrollo Dirigido por Modelos

David Blanes Domínguez

Directores:

Dr. Emilio Insfrán Pelozo

Dr. Javier González Huerta

Valencia, 5 de febrero de 2016

Tesis Doctoral

Este trabajo ha sido financiado por el Ministerio de Educación, Cultura y Deporte bajo el programa Formación de Personal Universitario con número de referencia AP2009-4635 y el proyecto MULTIPLE (TIN 2009-13838).

Título:	Una Aproximación de Ingeniería de Requisitos para Líneas de Productos Software Basada en una Estrategia de Desarrollo Dirigido por Modelos
Presentado por:	David Blanes Domínguez
Directores:	Dr. Emilio Insfrán Pelozo Dr. Javier González Huerta
Institución:	Universitat Politècnica de València (UPV)
Departamento:	Sistemas Informáticos y Computación (DSIC)
Programa de doctorado:	Doctor en Informática
Fecha de envío:	21 de diciembre de 2015
Fecha de defensa:	5 de febrero de 2016
Revisores externos:	Dra. Cristina Cachero Castro (Universidad de Alicante) Dra. Marcela Genero Bocco (Universidad de Castilla-La Mancha) Dr. Jaime Gómez Ortega (Universidad de Alicante)
Tribunal:	Presidenta: Dra. Marcela Genero Bocco (Universidad de Castilla-La Mancha) Secretario: Dr. Santiago Escobar (Universitat Politècnica de València) Vocal: Dra. María Valeria de Castro (Universidad Rey Juan Carlos)

Agradecimientos

El camino de realización de una tesis doctoral resulta arduo, no siendo posible recorrerlo solo. A continuación se citan a todas las personas que contribuyeron de alguna forma en este trabajo.

A Emilio por su confianza, su apoyo y su infinita paciencia durante este tiempo.

A Javier por todo trabajo realizado y sus contribuciones aportadas a esta tesis.

A Raphael por su colaboración en la definición del método, su colaboración con la validación del trabajo y sus réplicas de los cuasi-experimentos en Brasil.

A Silvia por su colaboración en la validación de este trabajo.

A Abel por todos sus consejos y ayuda sobre la plataforma Eclipse.

A Isidro por sus consejos durante las reuniones del proyecto MULTIPLE.

A los compañeros del grupo ISSI, por todos los momentos que hicieron ameno todo este tiempo y por su inestimable colaboración en los cuasi-experimentos (Adrián, Alex, Fernando, José Antonio, Kamil, Patricia, Miguel Ángel, Priscila, Sonia, Vicente).

A mis padres, mi hermana, familia y amigos por todo el apoyo moral durante todos estos años.

Finalmente a todos los que contribuyeron de alguna forma en este trabajo.

Resumen

La actividad de Ingeniería de Requisitos (IR) resulta crucial dentro de la ingeniería del software. Un fallo durante la definición de los requisitos de un sistema puede provocar sobrecostes durante todo el proceso de desarrollo de un producto e incidir negativamente en el tiempo de desarrollo. Este problema se acentúa aún más en el paradigma de desarrollo de Líneas de Producto Software (LPS) debido a que la definición y especificación de los requisitos deben de tratar con una nueva dimensión: la variabilidad de los requisitos. Esta variabilidad de los requisitos de la LPS se específica durante el proceso de ingeniería del dominio, donde se definen los puntos de variabilidad que permiten diferenciar qué requisitos serán comunes en todos los productos de la LPS y cuáles serán variables. Estos puntos de variabilidad se resuelven durante el proceso de ingeniería de la aplicación para obtener los requisitos de un producto específico, en la actividad llamada derivación de requisitos.

Otro paradigma ampliamente aplicado en las LPS es el Desarrollo de Software Dirigido por Modelos (DSDM). En el DSDM se utilizan los modelos como principal entidad durante el proceso de desarrollo. Cada modelo trata un aspecto del sistema en diferentes niveles de abstracción. El DSDM se basa en una estrategia de reutilización de software mediante el empleo de modelos y transformaciones. El DSDM resulta interesante para establecer la trazabilidad entre los requisitos de la LPS y la variabilidad de los requisitos y derivar, mediante técnicas de transformación de modelos, los requisitos del producto.

Sin embargo a pesar de que el DSDM puede reducir costes de producción, gracias al aumento de la automatización y las posibilidades de reutilización de software a nivel de modelos, muchas de las aproximaciones de IR para LPS que se encuentran en la literatura presentan algunas debilidades. Muchas de las aproximaciones actuales representan la información de la variabilidad de los requisitos exclusivamente en el mismo modelo de requisitos, perjudicando la legibilidad de los requisitos y comprometiendo la consistencia con el modelo de características de la LPS. Por otra parte durante la ingeniería de la aplicación, las aproximaciones de IR para LPS normalmente se limitan a derivar los requisitos del producto a partir de los requisitos de la LPS, pero no explicitaban cómo representar requisitos que no existían previamente en la LPS: los requisitos delta. Este hecho incide negativamente en la flexibilidad de las aproximaciones.

El objetivo de esta tesis doctoral es definir y validar una aproximación de IR en el contexto de LPS que soporte la definición y especificación de los requisitos de una LPS, permitiendo derivar a partir de ellos los requisitos de un producto

haciendo uso del paradigma de DSDM, y soportando además la definición y especificación los requisitos delta.

En este contexto, se ha definido un proceso llamado FeDRE que cubre los procesos de ingeniería del dominio y la aplicación, dando soporte así mismo a la especificación de requisitos delta. Durante la ingeniería del dominio se utiliza una estrategia de definición y especificación de los requisitos a partir del modelo de características. Durante la ingeniería de la aplicación se cubre la derivación de los requisitos y la validación para comprobar que satisfacen las necesidades del cliente. En el caso de que fuera necesario se permiten modelar los requisitos delta, evitando que los requisitos del producto estén limitados a una mera combinación de requisitos de la LPS.

Por otra parte se ha definido una aproximación tecnológica basada en una estrategia de DSDM. Durante la ingeniería del dominio se representan en un multimodelo las vistas de variabilidad de la LPS y la de requisitos, estableciendo relaciones de trazabilidad entre ellas. Durante la ingeniería de la aplicación se define una configuración del producto y se derivan, mediante una estrategia basada en transformaciones de modelos, los requisitos del producto a desarrollar.

El proceso propuesto en FeDRE se ha validado mediante dos cuasi-experimentos. El primer cuasi-experimento modelaba los requisitos de una LPS de aplicaciones móviles para la notificación de emergencias con el objetivo de validar las guías de la ingeniería del dominio de FeDRE. Los resultados mostraron que FeDRE fue percibido por los sujetos como fácil de usar y útil para especificar los requisitos de una LPS. En el segundo cuasi-experimento se validaron los requisitos de una LPS centrada en el comercio electrónico con el objetivo de comprobar si cubrían las necesidades del cliente para un producto en concreto. En el caso de que no lo hicieran, los participantes especificaron los requisitos delta aplicando las guías de FeDRE. La mayor parte de los sujetos fueron capaces de identificar correctamente qué necesidades estaba cubiertas y qué requisitos debían de añadirse como requisitos delta.

Esta tesis doctoral contribuye al campo de desarrollo de LPS proveyendo de un proceso y una aproximación tecnológica, automatizada y genérica para la definición y especificación de requisitos en entornos LPS.

Abstract

The Requirements Engineering (RE) activity is crucial in software engineering. A failure when defining the requirements of a system could increase both the costs of the entire product development process and the development time. This problem is even more critical in the Software Product Lines (SPL) development paradigm, since the definition and specification of requirements must deal with a new dimension: requirements variability. Requirements variability is specified during the domain engineering process, in which variability points are defined so as to distinguish which requirements will be common to all the SPL products, and which will be variable and defined for one specific product. These variability points are resolved during an application engineering activity called requirements derivation in order to obtain the requirements for a specific product.

Another paradigm that is widely applied in SPL Development is Model-Driven Software Development (MDSD). In MDSD, models are used as the principal entity during the development process. Each model represents an aspect of the system at a different level of abstraction. The MDSD is based on a software reuse strategy in which models and transformations are employed. MDSD is interesting as regards establishing the traceability among SPL requirements and their variability and deriving the product requirements by using model transformation techniques.

Despite the fact that MDSD can reduce production costs by facilitating the automation and increasing the software reuse at the model level, most of the RE approaches for SPL found in literature have some weaknesses. Many of the current approaches represent the variability information in the requirements models, thus reducing the requirements' readability and compromising their consistency with the characteristics of the feature model. Moreover, the RE approaches for SPL used during application engineering are normally limited to deriving the product requirements from the SPL requirements and do not indicate how to represent non-existent requirements in the SPL: the delta requirements. This has an undesirable effect on the flexibility of the approaches.

The aim of this thesis is to define and validate an RE approach in the context of SPL that will support the definition and specification of the requirements of an SPL, allowing to derive the product requirements from them using the MDSD paradigm, and also supporting the definition and specification of delta requirements.

In this context, we have defined a process called FEDRE, which covers engineering processes and the application domain, and likewise supports the

specification of delta requirements. During domain engineering, strategy definition and requirements specification obtained from model features are used. During application engineering, the derivation and validation of product requirements ensure that requirements meet customer needs. The necessary delta requirements could be specified, when they will be required, with the aim to prevent that product requirements are limited to a mere combination of LPS requirements.

Moreover, a technological approach based on a MDSD strategy was defined. During domain engineering, the variability of the SPL and the requirements variability are represented as multi-model views, and traceability relationships are established among them. During application engineering, the product configuration is defined and the requirements of the product to be built are derived using a strategy based on model transformations.

The process proposed in FEDRE has been validated using two quasi-experiments. In the first quasi-experiment, the requirements of a mobile application LPS for emergency notification were specified with the aim of validating the FEDRE domain engineering guidelines. According to the results, the participants perceived FeDRE to be easy to use and useful as regards specifying the requirements for an LPS. In the second quasi-experiment, the requirements of an e-commerce SPL were validated with the aim of verifying whether the customer needs for a particular product where covered. In the case of their not being covered, the participants specified the delta requirements by applying FEDRE guidelines. Most of the subjects were able to correctly identify what needs were covered and which requirements had to be added as delta requirements.

This dissertation contributes to the field of development of LPS by providing a process and technology, along with an automated and generic approach with which to define and specify requirements in SPL environments.

Resum

L'activitat d'Enginyeria de Requeriments (ER) és crucial dins de l'enginyeria del programari. Un error durant la definició dels requeriments d'un sistema pot provocar sobrecostos durant tot el procés de desenvolupament d'un producte i incidir negativament en el temps de desenvolupament. Aquest problema s'accentua encara més en el paradigma de desenvolupament de Línies de Producte Software (LPS) a causa de que la definició i especificació dels requeriments han de treballar amb una nova dimensió: la variabilitat dels requeriments. Aquesta variabilitat dels requeriments de l'LPS s'especifica durant el procés d'enginyeria del domini, on es defineixen els punts de variabilitat que permeten diferenciar quins requeriments seran comuns en tots els productes de l'LPS i quins seran variables. Aquests punts de variabilitat es resolen durant el procés d'enginyeria de l'aplicació per obtenir els requeriments d'un producte específic, en l'activitat anomenada derivació de requeriments.

Un altre paradigma àmpliament aplicat a les LPS és el Desenvolupament de Programari Dirigit per Models (DSDM). Al DSDM s'utilitzen els models com a principal entitat durant el procés de desenvolupament. Cada model tracta un aspecte del sistema en diferents nivells d'abstracció. El DSDM es basa en una estratègia de reutilització de programari mitjançant l'ús de models i transformacions. El DSDM resulta interessant per a establir la traçabilitat entre els requeriments de l'LPS i la variabilitat dels requeriments i derivar, mitjançant tècniques de transformació de models, els requeriments del producte.

No obstant això, malgrat que el DSDM pot reduir costos de producció, gràcies a l'augment de l'automatització i les possibilitats de reutilització de programari a nivell de moldels, moltes de les aproximacions d'ER per a LPS que es troben a la literatura presenten algunes debilitats. Moltes de les aproximacions actuals representen la informació de la variabilitat dels requeriments exclusivament en el mateix model de requeriments, perjudicant la llegibilitat dels requeriments i comprometent la consistència amb el model de característiques de l'LPS. D'altra banda, durant l'enginyeria de l'aplicació, les aproximacions d'ER per a LPS normalment es limiten a derivar els requeriments del producte a partir dels de la LPS, però no expliciten com representar requeriments que no existien prèviament a l'LPS: els requeriments delta. Aquest fet incideix negativament en la flexibilitat de les aproximacions.

L'objectiu d'aquesta tesi doctoral és definir i validar una aproximació d'ER en el context de LPS que done suport a la definició i especificació dels requeriments d'una LPS, permetent derivar a partir d'ells els requeriments d'un producte fent

ús del paradigma de DSDM i donant suport a més a la definició i especificació dels requeriments delta.

En aquest context, s'ha definit un procés anomenat FeDRE que cobreix els processos d'enginyeria del domini i de l'aplicació, donant suport així mateix a l'especificació de requeriments delta. Durant l'enginyeria del domini s'utilitza una estratègia de definició i especificació dels requeriments a partir del model de característiques. Durant l'enginyeria de l'aplicació es cobreix la derivació dels requeriments i la validació per comprovar que satisfan les necessitats del client. En el cas que fora necessari es permetrà modelar els requeriments delta, evitant que els requeriments del producte estiguen limitats a una mera combinació de requeriments de l'LPS.

D'altra banda s'ha definit una aproximació tecnològica basada en una estratègia de DSDM. Durant l'enginyeria del domini es representen en un multimodel les vistes de variabilitat de l'LPS i la de requeriments, establint relacions de traçabilitat entre elles. Durant l'enginyeria de l'aplicació es defineix una configuració del producte i es deriven, mitjançant una estratègia basada en transformacions de models, els requeriments del producte a desenvolupar.

El procés proposat en FeDRE s'ha validat mitjançant dos quasi-experiments. El primer quasi-experiment modelava els requeriments d'una LPS d'aplicacions mòbils per a la notificació d'emergències amb l'objectiu de validar les guies de l'enginyeria del domini de FeDRE. Els resultats mostren que FeDRE va ser percebut pels subjectes com fàcil d'utilitzar i útil per especificar els requeriments d'una LPS. En el segon quasi-experiment es van validar els requeriments d'una LPS centrada en el comerç electrònic amb l'objectiu de comprovar si cobrien les necessitats del client per a un producte en concret. En el cas que no ho feren, els participants especificaren els requeriments delta aplicant les guies de FeDRE. La major part dels subjectes van ser capaços d'identificar correctament quines necessitats estaven cobertes i quins requeriments havien d'afegir com a requeriments delta.

Aquesta tesi doctoral contribueix al camp del desenvolupament de LPS proveint d'un procés i d'una aproximació tecnològica, automatitzada i genèrica per a la definició i especificació de requeriments en entorns LPS.

Palabras Clave

Key Words: Requirements Engineering, Software Product Lines, Model-Driven Software Development.

Palabras Clave: Ingeniería de Requisitos, Líneas de Productos Software, Desarrollo de Software Dirigido por Modelos.

Paraules Clau: Enginyeria de Requeriments, Línies de Producte de Programari, Desenvolupament Dirigit per Models.

Contenido

Índice de Figuras

Índice de Tablas

Índice de Listados

Acrónimos

BVR	Base Variability Resolution
CVL	Common Variability Language
DSDM	Desarrollo de Software Dirigido por Modelos
EMF	Eclipse Modeling Framework
IR	Ingeniería de Requisitos
LED	Lenguaje Específico de Dominio
LPS	Líneas de Producto Software
MDA	Model Driven Architecture
MOF	Meta Object Facility
OMG	Object Management Group
QVT	Query View Transformation
UML	Unified Modeling Language
URL	Uniform Resource Locator
VSpec	Variability Specification
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

Capítulo 1. Introducción

En este capítulo se contextualiza el trabajo de investigación de esta tesis. A continuación se presentan: el problema a resolver, los objetivos y las metas planteadas y por último se describe la metodología de investigación.

La sección 1.1 introduce la problemática asociada a la ingeniería de requisitos en el contexto de desarrollo de líneas de producto software.

La sección 1.2 presenta brevemente la derivación de los requisitos del producto mediante una estrategia de desarrollo dirigido por modelos.

La sección 1.3 describe el planteamiento del problema que se aborda en el presente trabajo.

La sección 1.4 contiene los objetivos, las metas derivadas del mismo, y las hipótesis que se pretenden contrastar en el presente trabajo.

La sección 1.5 describe la metodología de investigación empleada en esta tesis.

La sección 1.6 describe la estructura del documento.

1.1 La ingeniería de requisitos para líneas de producto software

De acuerdo con Zave (1997), la Ingeniería de Requisitos (IR) es la disciplina para desarrollar una especificación [de requisitos] completa, consistente y no ambigua donde se describen las funciones que realizará el sistema y que servirá como base para realizar acuerdos comunes entre todas las partes involucradas. Otros autores como Loucopoulos y Karakostas (1995), definen la IR como el proceso sistemático de desarrollar requerimientos a través de un proceso iterativo y cooperativo de análisis del problema, documentando las observaciones resultantes en una variedad de formatos de representación y chequeando la precisión del entendimiento ganado. Este análisis del problema produce como resultado final la especificación de los requisitos del sistema a construir.

La especificación de los requisitos resulta crucial en el desarrollo de un sistema software puesto que un error en dicha fase afectará negativamente tanto en el tiempo como en los costes de desarrollo del producto. Pressman (2005) en la misma línea evidencia la importancia de la IR en el proceso de desarrollo de software: "el problema más difícil de corregir, relacionado con los requisitos, es que no se descubran a tiempo aquellos que son relevantes para el proyecto". Históricamente la IR para dar solución a estos desafíos, ha tenido que enfrentarse a diversas dificultades como la diversidad de fuentes para identificar los requisitos, la dificultar de expresar los requisitos, la cantidad de requisitos, el no empleo de un vocabulario común entre desarrolladores, etc.

La IR resulta aún más compleja en el desarrollo de Líneas de Producto Software (LPS) puesto que se añade una dimensión nueva que se debe contemplar: la variabilidad. Aunque no existe una definición homogénea de lo que es un LPS, existen algunas definiciones propuestas en la comunidad como la propuesta por Clements y Northrop: "es un conjunto de sistemas de software que comparten un conjunto común y gestionado de aspectos que satisfacen las necesidades específicas de un segmento de mercado o misión y que son desarrollados a partir de un conjunto común de activos fundamentales [de software] de una manera prescrita" (Clements y Northrop 2007).

Concretamente en la IR de LPS aparecen tres tipos de requisitos: comunes, opcionales y variables. La IR tradicional debe adaptarse en este contexto para proveer de los mecanismos necesarios para la definición y especificación de esta variabilidad a nivel de requisitos. Por otra parte la IR en del desarrollo de LPS, a diferencia de la IR tradicional, se centra en dos grandes procesos (Pohl y otros 2005):

- *Ingeniería de requisitos del dominio*. En este proceso se definen la especificación de los requisitos comunes en la familia de productos. La principal novedad de la IR del dominio respecto a la IR tradicional reside en la aparición de variabilidad a nivel de requisitos. Esto implica de que algunos requisitos serán comunes en toda la familia de productos y otros serán variables.
- *Ingeniería de requisitos de la aplicación*. El objetivo durante este proceso es definir y especificar los requisitos para un producto en particular y al mismo tiempo reusar los artefactos de la ingeniera del dominio de requisitos en la medida de lo posible.

1.2 Desarrollo de software dirigido por modelos en LPS

Durante la IR de la aplicación algunas aproximaciones de la literatura han aplicado una estrategia de Desarrollo de Software Dirigido por Modelos (DSDM) en los últimos años. En el DSDM se utilizan los modelos como principal elemento durante el proceso de desarrollo. Cada modelo trata un aspecto del sistema, que puede ser especificado a un nivel más elevado de abstracción y de forma independiente de la tecnología utilizada. Esta elevación del nivel de abstracción nos permite un mayor desacoplamiento respecto a las plataformas tecnológicas, beneficiando la mantenibilidad y evolución de los sistemas. Otra de las ventajas de la aplicación de DSDM es que favorece la reutilización del software de forma independiente de la tecnología subyacente.

Esta filosofía de reutilización de software utilizada en el DSDM coincide con la empleada en la producción de productos en una LPS. A partir de estos dos paradigmas surgió el término de DSDM para LPS utilizando primeramente en el trabajo de Czarnecki y Antkiewicz (2005). Esta combinación permite aprovechar las ventajas de la abstracción del software del DSDM y la gestión de la variabilidad de las LPS. Para crear una LPS aplicando una estrategia de DSDM, tiene que representarse la variabilidad en cada nivel de abstracción en los modelos creados durante la ingeniería del dominio. Además para aplicar una estrategia DSDM es preciso definir los puntos de variabilidad en la especificación de requisitos. Eso se complica cuando se deben representar dicha variabilidad en múltiples artefactos software.

Durante el proceso de la ingeniería de la aplicación, EL DSDM puede utilizarse para resolver los puntos de variabilidad de una especificación de requisitos para la familia del dominio con el objetivo de derivar de forma automatizada la especificación de requisitos del producto.

1.3 Planteamiento del problema

La IR para LPS tiene como reto dar solución a la problemática planteada durante los dos procesos de una LPS (Pohl y otros 2005): la ingeniería de requisitos del dominio y la ingeniería de requisitos del producto.

Durante la *ingeniería de requisitos del dominio* se definen y especifican los requisitos de la familia de productos mediante diversos artefactos que forman la especificación de requisitos (ej. modelo de casos de uso, modelo de variabilidad, etc.); siendo preciso definir de una manera adecuada los artefactos para representar los requisitos y su variabilidad. Sin embargo muchas de las aproximaciones actuales de la literatura incluyen dentro de los artefactos que representan los requisitos de la LPS información de la variabilidad, como por ejemplo modelos de requisitos tradiciones como casos de uso (Moon y otros 2006). Esta estrategia plantea algunas limitaciones debido a que la existencia de un gran número de requisitos y características puede provocar que los requisitos sean difíciles de entender, mantener y sean propensos a inconsistencias (Alférez y otros 2009). En otros casos se extraen modelos de características del modelo a partir de los modelos de requisitos (Mussbacher y otros 2012). Sin embargo los desarrolladores de LPS y los expertos del dominio están más familiarizados con los conceptos de característica y modelado de variabilidad (Oliveira y otros 2014). Además el considerar la identificación de variabilidad a nivel de requisitos hace que se omita toda la planificación estratégica de la LPS. Estos hechos hacen que la utilización de técnicas de requisitos para especificar la variabilidad de la LPS puede resultar no ser totalmente adecuada.

Una alternativa a estas aproximaciones es utilizar modelos de características, los cuales resultan adecuados para expresar la variabilidad de la LPS. La utilización de un modelo de características mejora a las aproximaciones que meramente expresan la variabilidad en artefactos de requisitos. Sin embargo situar la información de variabilidad únicamente en los modelos de características hace difícilmente trazable esta variabilidad a otros artefactos de requisitos (ej. casos de uso), siendo complejo expresar dependencias entre diferentes artefactos de requisitos y sus variantes (Bühne y otros 2004). De acuerdo con Pohl y otros (2005) modelar la variabilidad utilizando únicamente el modelo de características tiene diversas desventajas: i) la información de variabilidad (del modelo de características) se propaga a diferentes modelos (ej. modelos de requisitos, modelos de diseño) siendo casi imposible mantener dicha información consistente, ii) es difícil determinar qué información de la variabilidad en los requisitos está influenciada por la información de variabilidad de otras etapas del proceso de desarrollo (diseño, pruebas, etc.), iii) los modelos de desarrollo que

son complejos, pueden sobrecargarse añadiendo información de variabilidad (ej. modelos de casos de uso, modelos de actividad), iv) los conceptos usados para definir la variabilidad difieren entre diferentes tipos de modelos de desarrollo (ej. de requisitos, de diseño), y como consecuencia, la variabilidad definida en diferentes modelos no se integra bien en el esquema general de la variabilidad, y v) la definición de la información de variabilidad en un modelo de desarrollo único puede llevar a definiciones ambiguas de la variabilidad contenida en los diferentes artefactos de desarrollo. Por estos motivos es preciso definir un modelo con la información de la variabilidad de requisitos en un modelo separado.

De acuerdo con Käkölä y Dueñas (2007) el otro proceso principal en el desarrollo de LPS, la *ingeniería de requisitos de la aplicación*, ha sido tradicionalmente descuidado en detrimento de la IR del dominio y como consecuencia no todas las propuestas de IR para LPS soportan este proceso. Para dar soporte a la IR de la aplicación es necesario proveer los mecanismos necesarios para resolver los puntos de variabilidad. Esta resolución de la variabilidad puede facilitarse con soporte automatizado, sin embargo existe una falta de herramientas en las aproximaciones de IR para LPS (Alves y otros 2010). Para dar una solución a esta problemática algunas propuestas de la literatura han aplicado soluciones basadas en DSDM, con la ventaja de automatizar total o parcialmente el proceso de derivación de los requisitos del producto. Sin embargo a pesar de la ventaja que nos proporciona una estrategia de DSDM, la simple derivación de los requisitos del producto no es suficiente. La especificación de requisitos del producto puede no siempre satisfacer las necesidades del cliente, dado que no siempre una combinación de requisitos de la LPS puede satisfacer las necesidades del cliente en un producto (Djebbi y otros 2007). En consecuencia durante la IR de la aplicación es necesario utilizar técnicas de validación de los requisitos del producto para comprobar que efectivamente se satisfacen las necesidades del cliente. En los casos en que los requisitos derivados para el producto no satisfagan totalmente las necesidades del cliente es preciso añadir nuevos requisitos tras la derivación, llamados requisitos delta. A pesar de este hecho en la literatura la mayoría de aproximaciones no indican cómo realizar la definición y especificación de los requisitos delta (Asadi y otros 2012). Para suplir esta carencia es preciso que las aproximaciones de IR para LPS proveen técnicas para especificar los requisitos delta.

Por otra parte existe una problemática general en las aproximaciones que cubren alguno de los dos procesos de IR para LPS. Generalmente las aproximaciones definen sus procesos de forma informal sin suficientes guías para cubrir la IR en LPS (Santana Neiva 2009), limitando la aplicabilidad de estas. Son necesarias

guías en la definición de los procesos dentro de cada aproximación para facilitar la aplicación de estas en entornos industriales.

Otro problema general de las aproximaciones es la falta de validaciones empíricas (Alves y otros 2010). Muchas aproximaciones utilizan el término "caso de uso" de forma errónea para referirse a ejemplos utilizados como pruebas de concepto. Sin embargo estas pruebas de concepto no son validaciones empíricas reales al carecer de un diseño experimental. Este problema es coherente con el hecho de que generalmente las aproximaciones no presentan evaluaciones cuantitativas o cualitativas bien diseñadas (Alves y otros 2010). Con el objetivo de realizar validaciones rigurosas, y suplir esta debilidad en las aproximaciones, únicamente los experimentos constituyen una forma de validación empírica verdadera (Sepúlveda y otros 2015). En consecuencia es preciso realizar experimentos para validar dichas aproximaciones con el objetivo de incrementar la adopción en entornos industriales.

1.4 Objetivos e hipótesis

El objetivo de esta tesis doctoral es definir y validar una aproximación de IR en el contexto de LPS que soporte la definición y especificación de los requisitos de una LPS, permitiendo derivar a partir de ellos los requisitos de un producto haciendo uso del paradigma de DSDM, soportando la definición y especificación de los requisitos delta.

Este objetivo principal puede descomponerse en las siguientes metas:

1. Definición de un proceso de requisitos para definir y especificar los requisitos de una LPS y derivar los requisitos de un producto a partir de los requisitos de una LPS, añadiendo los requisitos delta necesarios. Para cubrir esta meta se definirán dos procesos de IR. En el primero se definen y especifican los requisitos de la LPS. En el segundo proceso se derivan los requisitos, a partir de una configuración de producto, la cual se completa mediante la definición y especificación de los requisitos delta aplicando las guías de la ingeniería de la aplicación.
2. Definición de una aproximación tecnológica para especificar los requisitos de la LPS y derivar, a partir de estos, los requisitos del producto mediante en una estrategia de DSDM. Se definirá un multimodelo para integrar las distintas vistas que representen los requisitos de la LPS. En este contexto se entiende como multimodelo a una representación de los distintos modelos y las relaciones entre ellos que componen la representación de un sistema

software. Por último se implementará una herramienta que soporte dicha aproximación tecnológica basada en el marco tecnológico Eclipse.

3. Validación mediante cuasi-experimentos con el fin de comprobar la facilidad de uso percibida, la utilidad de uso y la intención de uso en el futuro de los dos procesos definidos de IR para LPS. Se realizarán concretamente dos cuasi-experimentos. El primero se utilizará para validar el proceso para definir y especificar los requisitos de la LPS durante la ingeniería del dominio. El segundo se utilizará para validar el proceso para derivar los requisitos del producto, validarlos, y definir y especificar los requisitos delta durante la ingeniería de la aplicación.

Con el cumplimiento del objetivo propuesto se pretenden cubrir las siguientes hipótesis de investigación:

- El uso de una aproximación de IR para LPS adoptando una estrategia orientada a características es una forma efectiva y eficiente para definir y especificar los requisitos de una LPS.
- La derivación de los requisitos de un producto mediante una estrategia de DSDM y la posterior validación y refinamiento provee una forma efectiva y eficiente para satisfacer las necesidades de un cliente para un producto concreto.
- La utilización de un modelo de variabilidad de requisitos desacoplado del modelo de variabilidad de la LPS y la especificación de requisitos, permite especificar la variabilidad de los requisitos a un nivel de granularidad adecuado sin influir negativamente en la legibilidad de la especificación de requisitos de la LPS.

1.5 Metodología de investigación

Se ha utilizado una extensión de la metodología propuesta por Gorschek y otros 2006) donde se incluyen actividades de evaluación y observación tanto en el ámbito académico como en la industria. Este modelo nos propone ocho actividades (ver Figura 1.1) relacionadas en un proceso iterativo donde se formulan soluciones candidatas que se evalúan de una forma empírica con el objetivo de alcanzar una solución realista. A continuación se enumeran las actividades:

1. *Identificación del problema*. Se pretende entender el problema que el socio industrial pretende resolver.

2. *Formulación del problema*. Una vez que el problema está identificado, tras las reuniones necesarias, es necesario acotarlo y definirlo de una manera más precisa. Se deben de especificar los factores contextuales claramente.
3. *Revisión del estado del arte*. Se realiza una revisión de la literatura de forma crítica con el objetivo de identificar hasta qué punto las soluciones actuales están cubiertos y cuáles son los problemas abiertos a resolver con la investigación a desarrollar.
4. *Solución candidata*. Se idean soluciones posibles que serán depuradas en las distintas etapas de refinamiento (etapas 6 y 7).
5. *Entrenamiento*. Esta etapa se realiza de forma incremental. En las primeras fases el entrenamiento se centra en crear el conocimiento necesario para que los profesionales puedan opinar sobre la aplicabilidad de la propuesta. En fases más 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 el contexto de investigación o en una configuración industrial limitada. Pueden realizarse experimentos controlados o casos de estudio con alumnos o profesionales en el ámbito académico. En el ámbito académico podrían realizarse seminarios, talleres prácticos y encuestas.
7. *Validación realista*. Se llevan a cabo casos de estudio en configuraciones industriales. Es esta fase se definirán guías prácticas y se desarrollarán 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 se preparan para el uso en contextos industriales.

Durante este trabajo de investigación se han cubierto las seis primeras etapas de la investigación. En las primeras fases de la tesis doctoral y en el contexto del proyecto de investigación MULTIPLE se identificó el problema de proveer una aproximación de IR en el desarrollo de LPS. Además se realizó un análisis del problema junto con un equipo de investigación de la Universidad Federal de Bahía identificando las necesidades principales de las aproximaciones de IR para LPS actuales en el contexto de la LPS de desarrollo de aplicaciones de notificaciones de emergencia llamada Savi. Una vez diseñada una solución candidata esta se mejoró mediante la etapa de entrenamiento y se realizó una validación inicial, mediante dos cuasi-experimentos, en el contexto de la Universitat Politècnica de València (España) y la Universidad Federal de Bahía (Brasil). En la siguiente subsección se presenta la metodología de

experimentación, basada en cuasi-experimentos que se ha utilizado en el contexto de esta tesis doctoral.

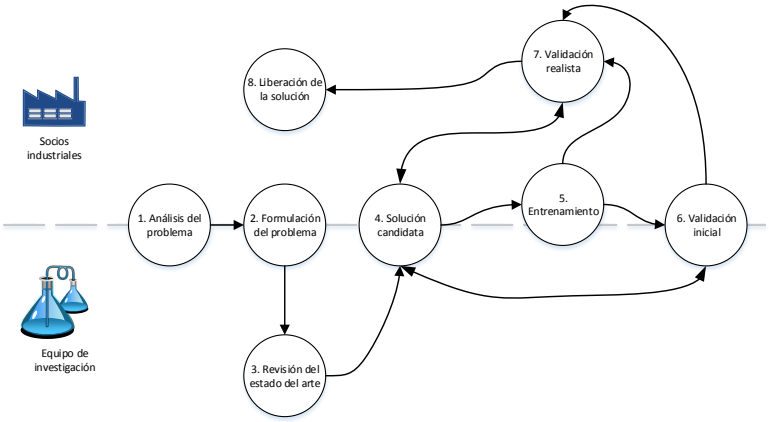


Figura 1.1 Modelo de transferencia tecnológica

1.5.1 Cuasi-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 los modelos, etc. De acuerdo con Campbell y Stanley (1982) los experimentos pueden dividirse en dos categorías: lo que aplicaron una asignación aleatoria de grupos de control, y los que no. En el primer caso se trata de un experimento mientras que el segundo grupo se llama cuasi-experimento. Autores como Laitenberger y Rombach (2003) consideran los cuasi-experimentos como aproximaciones prometedoras para incrementar la cantidad de estudios empíricos en la industria de la ingeniería del software. En este trabajo de tesis se utilizará la modalidad de cuasi-experimento para realizar la validación de los procesos definidos. Sendos cuasi-experimentos se han diseñado de acuerdo a las guías propuestas por Wohlin y otros (2000). Las principales actividades de este proceso pueden observarse en la Figura 1.2.

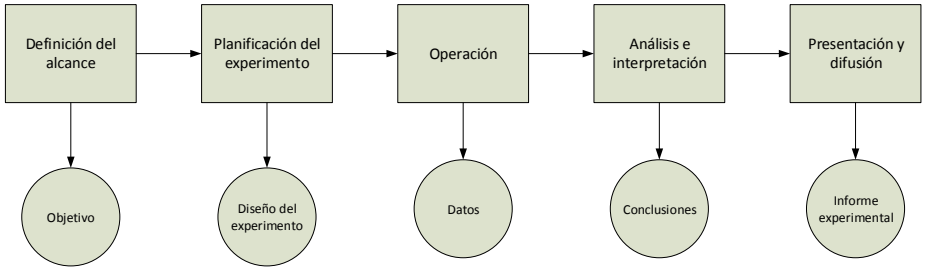


Figura 1.2 Proceso experimental con los artefactos generados es las distintas actividades

La primera actividad es la *definición del alcance*. Aunque inicialmente no estén definidas formalmente las hipótesis del experimento, deberán de definirse en esta actividad junto con los objetivos y las metas. Las metas del experimento se formulan a partir del problema a resolver siguiendo el marco de trabajo llamado en *Goal/Question/Metric* (GQM) (Basili y otros 1998)con el esquema siguiente esquema:

- *Analizar*: qué se analiza, cuál es el objeto de estudio.
- *Con el propósito*: cuál es la intención.
- *Con respecto a su*: cuál es su efecto estudiado.
- *Desde el punto de vista*: qué vista.
- *En el contexto de*: 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 se define en profundidad el contexto, que incluye tanto la componente personal (qué 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 preparará 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 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.

1.6 Estructura de la tesis

En este capítulo se han presentado: las motivaciones de la investigación, los objetivos y metas a resolver, el contexto de investigación y el método de investigación aplicado. El resto de la tesis de organiza en los siguientes capítulos:

- Capítulo 2. Marcos y espacios tecnológicos.

Este capítulo presenta el marco tecnológico en el que se desarrolla esta tesis: ingeniería de requisitos, líneas de producto software, desarrollo de software dirigido por modelos. Adicionalmente se presenta la notación de procesos software SPEM2 y el lenguaje de especificación de variabilidad CVL. Por último se presentará el espacio tecnológico de la plataforma Eclipse.

- Capítulo 3. Estado del arte.

En este capítulo de analizan las propuestas existentes en la literatura en el ámbito de la IR para desarrollo en LPS.

- Capítulo 4. FeDRE: un método de IR para LPS.

En este capítulo se presenta el método FeDRE para la definición y especificación de los requisitos de una LPS y la derivación de los requisitos de los productos a partir de los requisitos de LPS, soportando la validación y la especificación de requisitos delta.

- Capítulo 5. Aproximación tecnológica para la derivación de requisitos del producto.

En este capítulo se presenta la aproximación tecnológica definida para soportar la derivación de requisitos del producto apoyando al proceso FeDRE. Se presentan los metamodelos definidos para dar soporte al multimodelo y las vistas que lo componen. Igualmente se presentan las transformaciones de modelos utilizadas durante la derivación de los requisitos del producto.

- Capítulo 6. Una herramienta para la definición y especificación de los requisitos de una LPS y la derivación de requisitos de producto.

En ese capítulo se presenta la herramienta que da soporte a la aproximación metodológica utilizando Eclipse y el marco de trabajo EMF.

- Capítulo7. Validación empírica de FeDRE.

En este capítulo se presenta la validación empírica de la propuesta mediante dos cuasi-experimentos.

- Capítulo 8. Conclusiones y trabajos futuros.

En este capítulo se presentan las contribuciones 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.

- Apéndice A: Guías de FeDRE.

Este apéndice contiene las guías propuestas en FeDRE durante la ingeniería del dominio y la ingeniería de la aplicación.

- Apéndice B: Material del cuasi-experimento de la ingeniería del dominio.

Este apéndice contiene el material experimental utilizado durante el cuasi-experimento que evalúa la ingeniería del dominio de FeDRE. El contenido está en inglés dado que el cuasi-experimento se realizó utilizando ese idioma.

- Apéndice C: Material del cuasi-experimento de la ingeniería de la aplicación.

Este apéndice contiene el material experimental utilizado durante el cuasi-experimento que evalúa la ingeniería de la aplicación de FeDRE.

Capítulo 2. Marcos y espacios tecnológicos

En este capítulo presentaremos los fundamentos y espacios tecnológicos necesarios para comprender los términos y conceptos de este trabajo.

La sección 2.1 describe los marcos tecnológicos relacionados con este trabajo de tesis: la ingeniería de requisitos, las líneas de producto software, los estándares de la OMG utilizados en el desarrollo de software dirigido por modelos y el estándar de representación de variabilidad CVL.

La sección 2.2 describe el espacio tecnológico que se utilizará durante la implementación de la aproximación tecnológica propuesta en este trabajo de tesis.

La sección 2.3 presenta un breve resumen de los conceptos introducidos durante este capítulo.

2.1 Marcos tecnológicos

Este trabajo de tesis se desarrolla en el ámbito de tres paradigmas principales de desarrollo de software que se presentan durante esta sección: la ingeniería de requisitos, las líneas de producto software y el desarrollo de software dirigido por modelos. Además se presentan el metamodelo SPEM y su lenguaje de definición y procesos software y el lenguaje de modelo de la variabilidad para líneas de producto software CVL.

2.1.1 Ingeniería de requisitos

El éxito de un sistema software depende del grado de entendimiento de las necesidades del usuario y su entorno. Para comprender estas necesidades, la Ingeniería de Requisitos (IR) juega un papel clave en el desarrollo del software. El éxito en la aplicación de técnicas de IR influye directamente en la calidad del producto software desarrollado. No existe una definición homogénea de los que es la IR aunque existen algunas comúnmente utilizadas como la de Zave. Según Zave: "la rama de la ingeniería del software que trata con el establecimiento de los objetivos, funciones y restricciones de los sistemas software. Asimismo, se ocupa de la relación entre estos factores con el objeto de establecer especificaciones precisas" (Zave 1997).

Para entender qué es la IR, es necesario definir el concepto de *requisito*. Existen múltiples definiciones sobre lo que es un requisito de software. De acuerdo con el estándar IEEE 610.12-1990 (IEEE 1990), un requisito puede ser:

a) Una condición o capacidad necesaria para un usuario con el objetivo de resolver un problema o conseguir una meta.

b) Una condición o capacidad que debe reunir o poseer un sistema, o componente de un sistema, para satisfacer un: contrato, estándar, especificación, u otro documento formalmente impuesto.

c) Una representación documentada de una condición o capacidad como las definidas en a) o b).

Los requisitos por otra parte pueden clasificarse en:

- *Funcionales*. Indican características y restricciones sobre la funcionalidad del software.
- *No funcionales*. Los Requisitos No Funcionales (RNF) definen restricciones sobre los requisitos no funcionales como fiabilidad, seguridad, usabilidad, etc.

Por otra parte, una especificación de requisitos del software (SRS) es una descripción completa del comportamiento del sistema a desarrollar. La SRS debería especificar: qué funciones son realizadas, sobre qué datos, cuales son los resultados esperados y para quién.

Sin embargo la IR no se limita únicamente a la representación de un SRS e incluye diversas actividades. A continuación se presentarán brevemente estas actividades utilizado una clasificación de actividades de IR adaptada de la propuesta por (Cheng y Atlee 2007): adquisición de requisitos, definición y especificación de requisitos, análisis de requisitos, validación y verificación, y gestión de requisitos.

2.1.1.1 Adquisición

La *adquisición* (también conocida como elicitación o captura) de requisitos suele ser la primera actividad de la IR. Esta utiliza técnicas que permiten comprender las metas, objetivos y motivaciones para construir el sistema software propuesto. Esta actividad usualmente también incluye el establecimiento de los límites del sistema. Los límites se definen a un alto nivel, incluyendo los límites del entorno operacional del sistema. Existen múltiples técnicas y notaciones propuestas para obtener esta información:

Dentro de la actividad de adquisición de los requisitos del sistema, puede incluirse la identificación de los *stakeholders*. El término *stakeholder*, hace referencia a quienes pueden afectar o son afectados por las actividades de una empresa. Un *stakeholder* puede ser un cliente, desarrollador o un usuario.

Otra técnica utilizada usualmente es la definición de modelos de metas. Una meta denota los objetivos que un sistema debe alcanzar. Identificar las metas de alto nivel tempranamente en el sistema es una tarea crucial en el desarrollo de software. La adquisición orientada a metas, es una actividad donde los requisitos de alto nivel como metas de negocio, se refinan en metas de más bajo nivel como metas técnicas que eventualmente se operacionalizan en un sistema.

2.1.1.2 Definición y especificación de requisitos

Originalmente la clasificación de las actividades de IR (Cheng y Atlee 2007) define la actividad de modelado de requisitos, la cual tiene como objetivo representar la especificación de requisitos en términos de uno o más modelos. No obstante la representación de la SRS mediante modelos no es la única forma de representar la especificación de requisitos de un sistema (ej. lenguaje natural, notaciones gráficas, notaciones matemáticas, etc.). Por este motivo durante este trabajo de tesis hablará en su lugar de *definición y especificación* de requisitos y no de modelado de requisitos.

A grandes rasgos la actividad de definición y especificación de requisitos tiene como objetivo construir la representación de la SRS. Esta actividad está compuesta por dos actividades. Primeramente la *definición de requisitos* es la actividad que produce una descripción de la funcionalidad y restricciones operacionales del sistema (Sommerville 2011). La intención es comunicar al desarrollador del sistema el *qué* del sistema a construir y sirve como base del contrato con el desarrollador del sistema. Esta descripción puede usar lenguaje natural, diagramas o cualquier notación que sea entendible por el cliente (Sommerville 2011). Por otra parte la *especificación de los requisitos* describe los requisitos del sistema a un nivel de detalle mayor, respecto a la definición de los requisitos, y debe ser consistente con esta. La especificación de los requisitos está orientada al diseñador del software dando una descripción de la funcionalidad y restricciones del sistema más precisa (Sommerville 2011).

Existen diferentes alternativas para definir y representar la SRD (Sommerville 2011):

- *Lenguaje natural*. Los requisitos se escriben utilizando frases en lenguaje natural. Cada frase representa un requisito.
- *Lenguaje natural estructurado*. Los requisitos se escriben en lenguaje natural representado un formulario o plantilla. Cada campo define la información sobre un aspecto del requisito.
- *Lenguaje de descripción del diseño*. Se define un modelo operacional del sistema. Utilizado principalmente para representar interfaces del sistema a construir.
- *Notaciones gráficas*. Se proveen modelos gráficos que pueden complementarse con anotaciones textuales. Por ejemplo diagramas de casos de uso de UML y diagramas de actividad (OMG 2007).
- *Especificaciones matemáticas*. Se utilizan conceptos matemáticos como máquinas de estados finitas.

2.1.1.3 Análisis de requisitos

Esta actividad incluye técnicas para evaluar la calidad de la especificación de requisitos. Algunas analizan errores de mal formación de la especificación, donde se entienden como "errores" factores como la ambigüedad, inconsistencia o la no completitud. Otras técnicas analizan anomalías como interacciones desconocidas entre requisitos, obstáculos posibles para la satisfacción de los requisitos y razonamientos perdidos.

En esta actividad, según (Cheng y Atlee 2007), se pueden utilizar, en otras, las siguientes técnicas: listas de aspectos a comprobar o *checklists*, análisis de consistencia, análisis de conflictos, gestión de riegos o análisis de variabilidad.

2.1.1.4 Validación y verificación de requisitos

La validación de requisitos se encarga de que los modelos y la documentación expresen correctamente las necesidades de los *stakeholders*. La validación suele hacerse típicamente de manera subjetiva, debido a la documentación de los requisitos de manera informal. Este tipo de validaciones requieren que el usuario tenga que participar activamente en el proceso de validación, revisando directamente los artefactos. Este es el caso de la técnica de inspección, utilizada para realizar una validación minuciosa de forma manual de los requisitos (Fagan 2001).

Para los casos en que la especificación de los requisitos se haya hecho de forma formal, se pueden aplicar técnicas de verificación para comprobar que esta satisface las necesidades de los *stakeholders*. En las técnicas de verificación se pueden considerar la *comprobación de modelos*, donde se verifica el comportamiento del modelo de requisitos contra alguna propiedad de lógica temporal durante la verificación de las trazas de ejecución. Otra técnica de verificación es la *satisfacibilidad de modelos*, donde se comprueba si existen instancias validas de los modelos de requisitos y que determinadas operaciones sobre los modelos de requisitos preservan invariantes definidos.

2.1.1.5 Gestión de requisitos

La gestión de requisitos es una actividad que incluye varias técnicas para gestionar los requisitos, incluyendo la evolución de los requisitos a través del tiempo. Un tema de especial interés, es el de las herramientas y técnicas que dan soporte parcial a la tarea de identificar y documentar las relaciones de trazabilidad entre los artefactos de requisitos y los artefactos de diseño. También se incluyen las técnicas para determinar la madurez y estabilidad de los requisitos adquiridos, para que los requisitos que se posible que cambien se puedan aislar.

2.1.2 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 para reducir los costes en la producción de software a gran escala consiste en aplicar el desarrollo de LPS. El desarrollo de software mediante LPS permite obtener los siguientes beneficios (Clements y Northrop 2007): 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 LPS. Se obtienen sustanciales mejoras de la productividad y se reducen los costes cuando se desarrollan los productos a partir de un conjunto de activos existentes de un modo predefinido, en vez de desarrollarlos separados desde cero o de un modo arbitrario.

Una vez presentadas los beneficios potenciales debe definirse qué es una LPS. Aunque no existe una definición estandarizada, algunos autores como Clements y Northrop definen una LPS como:

"*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 2007)

En esta definición se utiliza un concepto clave: la *característica*. Una característica se define como:

"Una unidad lógica de comportamiento que es especificada por un conjunto de requisitos funcionales o de calidad." (Bosch 2000)

Cada producto de una LPS va a estar formado por un conjunto de componentes seleccionados una base común de activos software o *assets*, que serán adaptados a las necesidades del producto gracias a una serie de mecanismos de variación, añadiéndose nuevos componentes cuando estos sean necesarios. La selección de qué características o no se incluirán en el producto nos permitirán decidir qué componentes deberán de incluirse en un producto.

Por otra parte existen dos procesos principales dentro del desarrollo de LPS de acuerdo con Pohl y otros (2005):

- *Ingeniería del dominio*. Se construyen los activos software o *assets* con el objetivo de crear una infraestructura que maximice la reutilización. Este proceso incluye la definición de la variabilidad de la LPS.
- *Ingeniería de la aplicación*. Se incluyen todas las actividades necesarias para desarrollar un producto concreto a partir de la infraestructura construida durante la ingeniería del dominio.

La Figura 2.1 muestra las actividades principales dentro de cada uno de estos procesos.

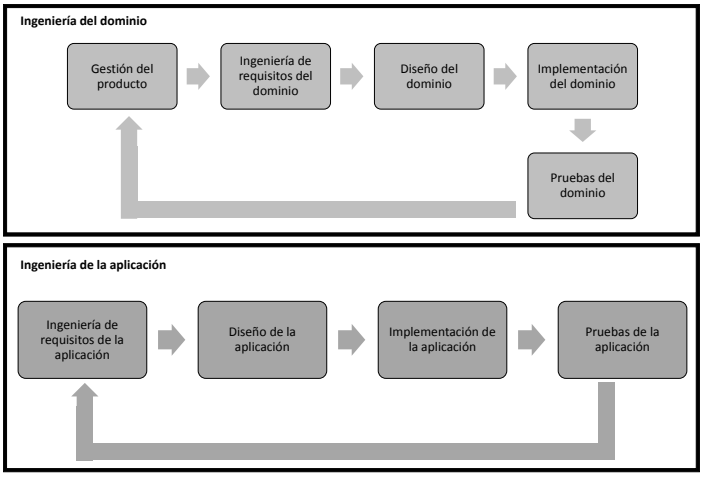


Figura 2.1 Procesos del desarrollo de LPS

A continuación se analizan cada uno de estos procesos en las siguientes subsecciones de acuerdo con el proceso propuesto por Pohl y otros (2005). Además se presentan las estrategias principales en el desarrollo de LPS.

2.1.2.1 Ingeniería de dominio

La ingeniería del dominio es el proceso de la LPS donde se define y materializa la variabilidad de la LPS. En este proceso se deciden qué productos se incluirán en la LPS y se definirán los artefactos reutilizables que contienen la variabilidad deseada. Este proceso se compone de cinco subprocesos: gestión del producto, ingeniería de requisitos del producto, ingeniería del diseño, realización del dominio y pruebas del dominio.

- G*estión del producto*. También se conoce como *scoping* por otros autores (John y Eisenbarth 2009). En este subproceso se definen los productos que se van a cubrir en la LPS y los límites donde se define qué va a entrar en la LPS y qué no. La entrada a este subproceso son los objetivos a más alto nivel de la organización. Como salida de este subproceso se producen, en otros artefactos, el modelo de características. Este modelo determina las características comunes y variables y los productos planificados en la LPS junto con sus fechas de entrega previstas. Adicionalmente en este subproceso se provee una lista de *assets* existentes con el objetivo de establecer qué artefactos pueden usarse para maximizar la reutilización.

  El *scoping* es una actividad donde se determinan los productos que se incluyen en la LPS y los límites que abarcará la LPS. De acuerdo con Bosch (Bosch 2000) la actividad de *scoping* se divide en tres niveles:

  - *Nivel de portfolio*. Determina qué productos y qué características deberían de incluirse en la LPS.
  - *Nivel de dominio*. Determina las áreas funcionales y sub-áreas del dominio de la LPS.
  - *Nivel de asset*. Identifica los *core assets* con los costes y beneficios estimados para cara uno de ellos.

- *Ingeniería de requisitos del dominio.* Incluye todas las actividades necesarias para definir y especificar los requisitos comunes y variables de la LPS. La entrada de este subproceso son el modelo de características, que contiene la variabilidad de la LPS, los productos planificados en la LPS y el mapa de productos, donde se relacionan los productos con las características que implementan. Como salida se obtienen los modelos de requisitos necesarios y la variabilidad de los requisitos que se incluyen en la LPS. En este subproceso se analizan los requisitos para analizaran cuáles serán comunes en todas los productos y cuáles serán variables.

- *Diseño del dominio.* Incluye todas las actividades necesarias para definir la arquitectura referencia de la LPS. La arquitectura referencia provee una estructura de alto nivel que es común en todos los productos de la LPS. La entrada este proceso son los requisitos del dominio y el modelo de variabilidad de la LPS. Como salida se obtiene la arquitectura referencia y un modelo de variabilidad refinado que incluye la variabilidad interna. En el diseño del dominio se incorporan mecanismos de variabilidad en la arquitectura referencia para soportar la variabilidad de la LPS.

- *Implementación del dominio.* Este subproceso cubre el diseño detallado y la implementación de los *assets* de la LPS. Como entrada de este subproceso la arquitectura referencia incluye una lista de *assets* que deben de desarrollarse. La salida produce el diseño detallado y un conjunto de *assets* implementados. La realización del dominio incorpora los mecanismos de configuración necesarios para materializar la variabilidad de la LPS en los *assets* como se defina en la arquitectura de referencia.

- *Pruebas del dominio.* Este subproceso es responsable de la validación y verificación de los *assets*. Este subproceso incluye las prueba sobre todos los componentes contra su especificación (ej. requisitos, arquitectura referencia, artefactos de diseño, etc.). Adicionalmente durante las pruebas del diseño se desarrollan artefactos de prueba para disminuir el esfuerzo necesario durante las pruebas. Como entrada de este subproceso se toman los requisitos del dominio, los componentes y las interfaces del diseño y los *assets*

implementados. Como salida a este subproceso se producen los resultados de las pruebas que se realizan sobre todos los artefactos del dominio.

2.1.2.2 Ingeniería de la aplicación

La ingeniería de la aplicación es el proceso de la LPS donde los productos de una LPS se construyen a partir de la reutilización de artefactos del dominio y se ejerce la variabilidad de la LPS. Este subproceso está compuesto de las siguientes actividades:

- *Ingeniería de requisitos de la aplicación.* En este subproceso se desarrolla la especificación de requisitos del producto basándose en la especificación de requisitos de la LPS. Una actividad crítica en este subproceso es la definición de los requisitos delta[1], o requisitos que no se pueden derivar a partir de los requisitos de la LPS. La entrada a este subproceso son los requisitos del dominio y el mapa de productos con las características correspondientes al producto. La salida de este subproceso son los requisitos para un producto concreto.

- *Diseño de la aplicación.* En este subproceso se utiliza la arquitectura de referencia para instanciar la arquitectura del producto. Se seleccionan y configuran las partes que se requieren de la arquitectura referencia y se incorporan adaptaciones específicas para el producto a desarrollar. La entrada al diseño de aplicación consiste en la arquitectura referencia y los requisitos específicos de la aplicación. Como salida se obtiene la arquitectura de la aplicación. Esta arquitectura se obtiene ejerciendo los puntos de variabilidad (por ejemplo haciendo selecciones específicas en los lugares donde la arquitectura de referencia tiene diferentes variantes).

- *Implementación de la aplicación.* Este subproceso crea el producto en sí. Principalmente se seleccionan y configuran los *assets* y se materializan los *assets* específicos del producto. Como entrada se toma la arquitectura de la aplicación y los *assets* que se utilizarán para reutilizar de la plataforma. La salida consiste en el producto, ya ejecutable, y los artefactos del diseño detallado que se han diseñado.

- *Pruebas de la aplicación.* Este subproceso incluye todas las actividades necesarias para validar y verificar el producto contra su especificación. La

[1] Utilizaremos durante esta tesis el término "*requisito delta*" (Pohl y otros 2005), el cual es más cercano al desarrollo de LPS que el término de "*deltas de requisitos*". Este último se utiliza de una forma mucho más genérica en la IR (Glinz y Heymans 2009).

entrada consiste en los tipos de artefactos del producto que se utilizarán como referencia para las pruebas, la aplicación implementada y los *assets* de prueba utilizados en las pruebas del dominio. Adicionalmente se documenta los defectos detectados en informes de problemas.

2.1.2.3 Particularidades de la IR para LPS

De acuerdo con el *Software Engineering Institute* (SEI) (Clements y Northrop 2007) los requisitos constituyen uno de los principales *core assets* de la LPS. Los requisitos del dominio de la LPS se utilizan para definir productos en la LPS junto con las características y las restricciones en estos productos. Una de las diferencias de estos requisitos respecto a los requisitos tradicionales es que estos contienen una variabilidad asociada. Esto implica que algunos de estos requisitos serán *comunes* y estarán presenten en todos los productos que forman la LPS. Por otra parte otros requisitos pueden ser *variables* y estar presentes en algunos productos pero no en otros.

Al margen de los requisitos comunes o variables, puede darse el caso de que un requisito no se corresponda con ninguna característica de la LPS. Es decir no es posible completar la especificación de requisitos de un producto y satisfacer las necesidades del cliente a partir de los requisitos comunes en la LPS, ni el conjunto de requisitos variables seleccionados para este producto. Esto implica que es necesario añadir nuevos requisitos a la especificación de productos, llamados *requisitos delta* y se utilizan para completar para especificación de requisitos de un producto.

2.1.2.4 Estrategia de desarrollo

Para producir una LPS existen tres diferentes estrategias propuestas en la literatura científica: 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. Esta la más costosa puesto que precisa un inicio de la LPS desde cero con el consiguiente riesgo asociado.

- 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.

2.1.3 El lenguaje SPEM2 para definir procesos software

En este trabajo de tesis se utilizará la notación para definir procesos software *Software & Systems Process Engineering Metamodel Specification* (SPEM 2.0) dado que es un estándar conocido en el campo de la ingeniería del software y utilizado por la OMG. SPEM es un metamodelo para definir modelos de procesos de ingeniería del software. El objetivo es proveer de una solución de compromiso que permita definir estos procesos sin añadir características especiales de un dominio o disciplina y que al mismo tiempo pueda adaptarse a modelos de proceso de diferentes estilos, culturas, niveles de formalidad o diferentes paradigmas.

SPEM 2.0 se basa en tres conceptos básicos (Figura 2.2):

- *Rol*. Quien lleva a cabo una tarea para obtener ciertas entradas (productos de trabajo) y salidas (productos de trabajo).
- *Producto de trabajo*. Representan las entradas necesarias o las salidas que se producen en una tarea.
- *Tarea*. Representan el esfuerzo que se realiza.

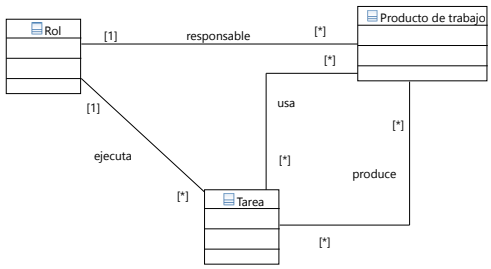


Figura 2.2 Conceptos principales de SPEM 2.0

La Figura 2.2 Conceptos principales de SPEM 2.0 nos muestra a grandes rasgos las relaciones entre estos conceptos claves para representar contenidos de métodos o procesos. Los contenidos de un proceso se muestran utilizando definiciones de productos de trabajo, definiciones de roles y definiciones de tareas. Las guías se definen en la intersección entre el contenido del método y el proceso debido a que un guía puede definirse en ambos. En la parte derecha se muestran los elementos que se utilizan para representar procesos en SPEM 2.0. El elemento principal es la entidad actividad, que puse descomponerse a su vez en otras actividades para definir un flujo de trabajo. Las actividades pueden utilizarse para definir un proceso y para gestionar referencias al contenido del método.

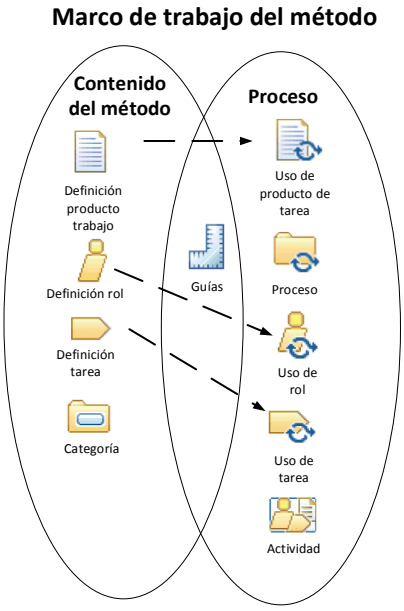


Figura 2.3 Terminología clave en SPEM 2.0

En la Figura 2.3 se describen las primitivas utilizadas para definir un proceso con la notación SPEM 2.0. El uso de SPEM 2.0 provee muchas ventajas para modelar procesos software entre otras:

- Facilita el entendimiento y la comunicación entre los interesados del sistema (*stakeholders*). Se provee un marco de trabajo común en el que los conceptos tiene una definición formal.
- Facilita la reutilización debido a que pueden integrarse otros procesos o partes de otros procesos y patrones.
- Se soporta la utilización de un repositorio donde se puede almacenar todo el contenido del proceso. Los roles responsables pueden acceder a estos procesos.

Tabla 2.1 Primitivas de modelado utilizadas en SPEM 2.0

Icono	Nombre	Descripción
	Definición de rol	Conjunto de habilidades, competencias y responsabilidades de un individuo o grupo
	Definición de tarea	Describe una unidad de trabajo que puede ser asignable y manejable. Identifica el trabajo que se lleva a cabo por los roles. Puede dividirse en varios pasos.
	Definición de producto de trabajo	El producto usado o producido por las tareas. Hay dos tipos de productos: artefactos de naturaleza tangible o artefactos entregables. Pueden asociarse entre ellos con relaciones de agregación, composición o impacto.
	Categoría	Clasifica a elementos como tareas, roles o productos basándose en el criterio establecido por el ingeniero de procesos. Hay diferentes tipos de categorías: grupos de roles, disciplinas o dominios.
	Guías	Proveen información adicional sobre otros elementos. Existen sub-tipos de guías: *assets* reusables, guías o plantillas de documentación.
	Uso de rol	Representa un rol que lleva a cabo una tarea o actividad dentro de un proceso definido.
	Uso de tarea	Representa una tarea en un proceso definido.
	Uso de producto de trabajo	Un producto de trabajo representa una entrada o salida relacionada con una actividad o tarea.
	Actividad	Representa un conjunto de tareas que se ejecutan dentro de un proceso con sus respectivos roles y productos de trabajo.
	Paquete de proceso	Representa un paquete que contiene todos los elementos de un proceso definido.

2.1.4 Desarrollo de software dirigido por modelos

El *Desarrollo de Software Dirigido por Modelos* (DSDM) es una aproximación de desarrollo de sistemas de software basada la utilización de modelos como principal elemento en el proceso de desarrollo. Un modelo permite definir la funcionalidad, estructura y/o comportamiento de un sistema. Los modelos permiten trabajar a un nivel de abstracción más cercano a los conceptos del dominio, en lugar de centrarse en conceptos orientados a plataforma como ocurre con el desarrollo de software tradicional. El objetivo del DSDM es maximizar la productividad y la interoperabilidad entre sistemas, facilitando la reusabilidad y la adaptación del sistema a cambios tecnológicos.

2.1.4.1 La arquitectura dirigida por modelos

En este trabajo se aplicará la propuesta de DSDM propuesta por el Object Management Group (OMG 2009), la *arquitectura dirigida por modelos* o *Model Driven Architecture (MDA)*. MDA se basa en cuatro principios fundamentales (Beydeda y otros 2005):

1. Los modelos se expresan es una notación bien definida que es la clave del entendimiento del sistema para soluciones a nivel de organización.
2. La construcción de los sistemas se puede organizar entorno a un conjunto de modelos imponiendo una serie de transformaciones entre ellos, organizada en un marco de trabajo de capas y transformaciones.
3. Se da soporte para describir modelos en un conjunto de metamodelos que facilita la integración y transformación entre modelos, y es la base de la automatización mediante herramientas.
4. La adopción y aceptación de este enfoque basado en modelos precisa de estándares industriales para proveer accesibilidad a los clientes y fomentar la competitividad entre proveedores.

MDA recomienda el uso de estándares propuestos por la OMG: MOF, UML, CWM, QVT, etc. Durante las siguientes subsecciones se presentarán los estándares empleados en este trabajo de tesis.

2.1.4.2 Modelos en MDA

MDA propone la creación de tres tipos de modelos:

- *Platform Independent Model (PIM).* Un modelo de un sistema que es independiente de la información acerca de la plataforma o tecnología que es usada para implementarlo. Representa el modelo de negocio, la

funcionalidad y el comportamiento del sistema. Un ejemplo de PIM podría ser un sistema representado utilizando la notación UML.

- *Platform Specific Model (PSM).* Un modelo de un sistema que incluye información acerca de la tecnología específica que se usará para su implementación sobre una plataforma específica (por ejemplo un programa en Java). Representa a una implementación concreta del PIM. El PSM puede obtenerse a partir del PIM mediante el uso de transformaciones de modelos.
- *Code Model (CM).* Dado que el PSM es dependiente de plataforma, se puede automatizar la generación de código mediante transformaciones de modelos.

2.1.4.3 Meta-Object Facility

El *Meta-Object Facility* (MOF) es un estándar propuesto por la OMG (2009) para dar soporte a la arquitectura MDA. En este estándar se propone una arquitectura de metamodelado basado en cuatro capas. Si consideramos un modelo como una abstracción de un fenómeno en el mundo real; el metamodelo es una abstracción donde se reflejan las propiedades del modelo. El metamodelado se utiliza básicamente en la utilización de modelos para definir otros modelos.

Respecto a la arquitectura de MOF, se distinguirán las siguientes capas:

- *Nivel M3: Meta-metamodelado.* En esta capa reside el meta-metamodelo, un lenguaje usado para definir los metamodelos del nivel M2. Dentro de OMG, MOF es el lenguaje estándar utilizado en esta capa.
- *Nivel M2: Metamodelo.* Los metamodelos de la capa M2 se utilizan para describir los modelos del nivel M1. Por ejemplo el meta-modelo de UML (OMG 2007) describiría al propio UML.
- *Nivel M1: Modelo.* En este nivel se definen los modelos. Un modelo es una instancia de un metamodelo del nivel M2. Por ejemplo si en el M2 residiera el metamodelo de UML, en el nivel M1 podríamos tener uno de sus modelos: modelo de clases, diagrama de actividad, etc.
- *Nivel M0: Instancias.* En esta capa se describen objetos del mundo real.

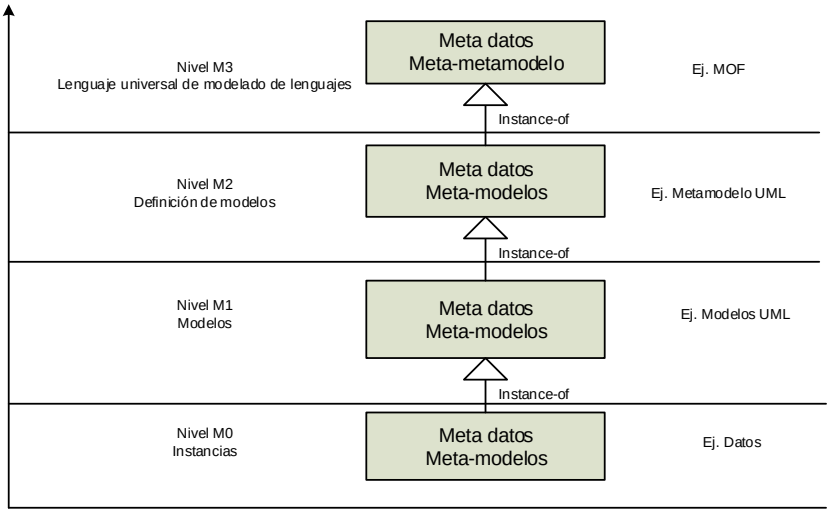


Figura 2.4 Niveles de la arquitectura MOF

MOF se considera una arquitectura de meta-modelado cerrada. Esto quiere decir que el último nivel, el M3, se pueden definir con instancias de elementos de M3. Esto implica que MOF se puede definir a sí mismo.

Además MOF provee los siguientes conceptos para definir un lenguaje:

- *Clases*, las cuales modelan los meta-objetos MOF.
- *Tipos de datos*, modelan la necesidad de datos descriptivos.
- *Asociaciones*, modelan relaciones binarias entre meta-objetos.
- *Paquetes*, modularizan los objetos.

La OMG ha definido dos variantes de MOF:

- *EMOF for Essential MOF*. Subconjunto de MOF, con el objetivo de proveer un marco de trabajo para mapear modelos MOF a implementaciones como JMI o XMI para meta-modelos simples.
- *CMOF for Complete MOF*. El modelo CMOF se utiliza para definir meta-modelos como UML2. Está construido a partir de EMOF y el Core:Constructs de UML2.

2.1.4.4 Object Constraint Language

El *Object Constraint Language (OCL)* es un lenguaje formal que puede utilizarse para definir restricciones o consultas sobre modelos en términos de lógica de predicados (OMG 2012). OCL fue desarrollado por IBM como lenguaje de

modelado de negocio. Surgió a partir del método Syntropy, un lenguaje de modelado de negocios de la *IBM Insurance Division*, desarrollado por Steve Cook y John Daniels.

Usualmente se suele utilizar para definir expresiones sobre modelos UML. Estas expresiones suelen especificar condiciones invariantes que deben cumplirse en el sistema que se está modelando o definir consultas sobre los objetos descritos en un modelo. Cuando se evalúa una expresión OCL, no se producen efectos secundarios (es decir, su evaluación no puede alterar el estado del sistema de ejecución correspondiente).

OCL no solo se puede aplicar para modelos UML, también es posible aplicarlo a meta-modelos UML o MOF, ya que están expresados en UML. Aquí el OCL puede usarse para restringir la semántica del meta-modelo, por ejemplo basándose en estereotipos. En el contexto MDA, OCL puede utilizarse de tres maneras posibles: construir modelos más precisos, en el nivel M1 de MOF, definición de Lenguajes de Modelado y especificar transformaciones.

2.1.5 El lenguaje CVL

El lenguaje *Common Variability Language* (CVL) (Object Management Group 2012), renombrado a *Base Variability Resolution Models (BRV)*[2] en su segunda versión, se propone como un lenguaje de modelado de variabilidad para modelos definidos en un Lenguaje Específico de Dominio (LED). Estos LED para poder ser soportados por CVL tienen que estar definidos bajo MOF. CVL se basa en el principio de modelado de variabilidad ortogonal (OVM) (Pohl y otros 2005), separando la variabilidad del DSL. La Figura 2.5 muestra los modelos principales de la arquitectura CVL:

- *Modelo base*. Un modelo descrito en el LED.
- *El modelo de variabilidad*. Un modelo que define la variabilidad del modelo base.
- *El modelo de resolución*. El modelo que define cómo se resuelve el modelo de variabilidad para crear un nuevo modelo en el modelo base DSL.

[2] Por convención durante esta tesis nos referiremos mediante el acrónimo CVL a su versión revisada en la versión 2.0.

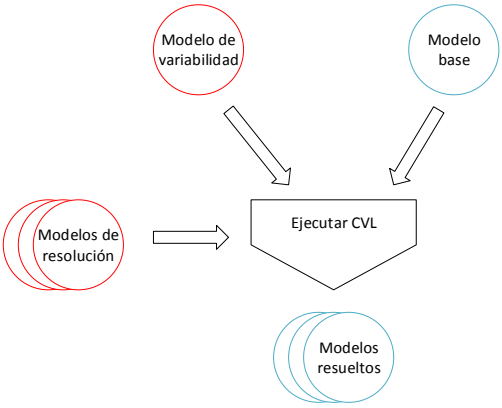


Figura 2.5 Principios de CVL

Un *modelo base* puede tener múltiples modelos de variabilidad y cada modelo de variabilidad puede tener a su vez múltiples modelos de resolución. Con el modelo de variabilidad y el modelo de resolución definidos correctamente el usuario puede ejecutar la transformación de modelo a modelo de CVL para generar un nuevo modelo resuelto que es conforme al modelo base DSL.

El modelo de variabilidad es una evolución de los modelos de características propuestos en *Feature-Oriented Domain Analysis* (FODA) (Kang y otros 1990). Sin embargo el objetivo principal de CVL es proveer una definición completa de aquellos modelos de productos que se pueden generar automáticamente a partir de un modelo de variabilidad CVL, el modelo de resolución y el modelo de realización.

La arquitectura CVL se escribe en la Figura 2.6. Esta está formada por diferentes modelos inter-relacionados. En esta arquitectura podemos observar los siguientes elementos:

- La *abstracción de la variabilidad* provee los constructos necesarios para especificar la variabilidad sin definir las consecuencias concretas en el modelo base, de ahí reside el nombre de abstracto. Esta capa aísla los componentes lógicos de CVL de las partes que manipula el modelo base. El principal constructo de la abstracción de variabilidad es la especificación de variabilidad o *VSpec*, que representa una elección binaria, un valor paramétrico o una especificación de un elemento que puede instanciarse múltiples veces. Otro elemento importante son los *Choice*, los cuales tienen la propiedad de poder organizarse en forma de árbol. Por otra parte las restricciones pueden asociarse con nodos, de los árboles *Choice*, para definir restricciones en resoluciones de variabilidad válidas.

- La *realización de la variabilidad* contiene los *puntos de variabilidad* que representan la trazabilidad entre la variabilidad de la abstracción y el modelo base de tal forma que los productos seleccionados se pueden generar automáticamente. Un punto de variabilidad es una modificación aplicada al modelo base durante el proceso de transformación en un modelo de producto, llamada *materialización*. Esta es la parte de CVL que impacta en el modelo base. Los puntos de variabilidad hacen referencia al modelo base a través de los *manejadores del modelo base*. Los puntos de variabilidad también hacen referencia a los *VSpecs* que definen que noción de variabilidad abstracta de la variabilidad de un punto de variabilidad se está realizando. Estas referencias forman el enlace o *binding* entre la abstracción de variabilidad y la realización de la variabilidad.
- Las *unidades configurables* definen una capa con el objetivo de estructurar e intercambiar módulos. Proveen los constructos necesarios para facilitar la especificación de componentes configurables y reusables. Además facilitan la modularidad a través de jerarquías composicionales, clonando la estructura composicional del modelo base. Las unidades configurables exhiben la *interfaz de variabilidad*, compuesta de *VSpecs* que son configurables.

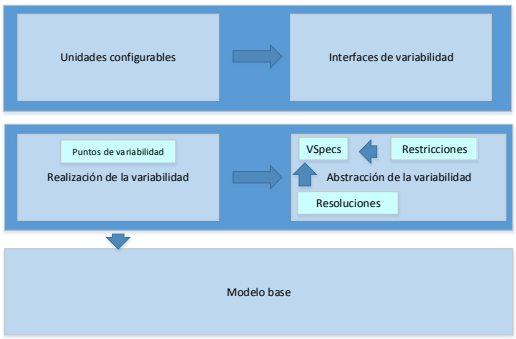


Figura 2.6 Arquitectura CVL

2.1.5.1 Abstracción de variabilidad

La capa de abstracción de variabilidad provee los constructos necesarios para resolver la variabilidad en el nivel abstracto. El concepto central es la *especificación de variabilidad*, que se describe con el acrónimo *VSpec* del inglés *Variability Specification*. Los *VSpec* son técnicamente similares a las características de los modelos de características pero vistas como especificadores de variabilidad abstracta y no tanto como características. Para mostrar esta sutil diferencia imaginemos por ejemplo una característica "GPS" de un móvil inteligente. Utilizando la visión clásica de característica podríamos verla como una

funcionalidad que un móvil puede tener o no dependiendo de la configuración del producto. Sin embargo en la visión de *VSpec* el "GPS" se ve como una elección a resolver positivamente o negativamente y no tanto en términos de funcionalidad del producto.

El *VSpec* es una indicador de variabilidad del modelo base. En CVL el efecto sobre un modelo base se especifica mediante el uso de *enlaces a puntos de variabilidad* a *VSpec*, que hacen referencia al modelo base.

Existen tres tipos diferentes de *VSpec*:

- Elección o *Choice*. Es un VSpec cuya resolución requiere una decisión de términos de sí o no.
- *Variable*. Es un *VSpec* cuya resolución implica proveer un valor para un tipo específico. Este valor se utilizará en el modelo base.
- Clasificador de variabilidad o *Vclassifier*. Un *VClassifier* es un tipo de *VSpec* cuya resolución implica crear instancias y proveer una resolución a los *VSpec* que están contenidos en su sub-árbol. Cada *VClassifier* tiene una multiplicidad de instancia que indica cuantas instancias están permitidas crearse.

2.1.5.2 Realización de variabilidad

Los mecanismos de realización de variabilidad hace posible materializar los productos de la descripción de CVL a partir de transformaciones del modelo base, definido en un lenguaje basado en MOF, en un modelo de producto definido en el mismo lenguaje.

Existen cuatro tipos de puntos de variabilidad que pueden definir modificaciones en el modelo base:

- *Existencia*. Una indicación de que existe un objeto particular, enlace o valor en el modelo base en cuestión.
- *Sustitución*. Una indicación de que un objeto único o un fragmento de un modelo entero puede ser sustituido por otro. Una *sustitución de objeto* implica dos objetos y significa redirigir todos los enlaces en los que se implica en el otro y entonces borra en el que lo compone. Una *sustitución de enlace-final* implica que un enlace final y un objeto que sustituye al que forma su objeto en el otro extremo. Una sustitución de fragmento implica identificar un fragmento de un modelo base a través un elemento frontera o *boundary* donde se puede crear un "hueco" conceptual que puede rellenarse con un *fragmento de reemplazo* de un tipo compatible.

- *Asignación de valor*. Una indicación de que un valor puede asignarse a un slot particular de un objeto de un modelo base.
- *Punto de variación opaco*. Una indicación de que una variabilidad especifica de dominio se asocia con el objeto u objetos donde la variabilidad específica de dominio se especifica explícitamente usando un lenguaje de transformación como QVT.

2.1.5.3 Unidades de configuración

Técnicamente una unidad de configuración es un tipo de punto de variabilidad que referencia a un objeto de un modelo con el objetivo de indicar que el objeto es un contenedor de variabilidad interna. Esta variabilidad interna se especifica a través de puntos de variabilidad obtenidos.

2.1.5.4 Notación de CVL 2 o BVR

A continuación se muestra un ejemplo representando la notación de CVL en su segunda versión y los conceptos principales presentados en las subsecciones anteriores. La Figura 2.7 se muestra la variabilidad en una línea de productos basada en automóviles diésel que tienen el cambio manual o automático. Los automóviles con cambio automático sólo pueden tener tracción total en todas las ruedas y utilizarán la tecnología de 140 caballos de vapor (cv). Por otra parte los automóviles con cambio manual pueden escoger entre la tracción total o tracción trasera. Los automóviles con tracción delantera sólo tienen la modalidad de 110 cv, mientras que los automóviles con tracción total pueden escoger entre los motores de 110 cv en su versión de baja potencia o 140 cv en su versión de alta potencia.

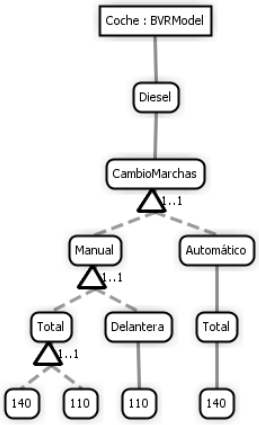


Figura 2.7 Ejemplo de coche diésel en CVL

En esta notación las líneas solidas indican que las *Vspec* hijas nos obligatorias cuando su *VSpec* padre se encuentra en una resolución. Las líneas de puntos indican que la *VSpec* es opcional. El triángulo pequeño indica que con los números asociados la multiplicidad del grupo, concretamente el rango de cuantas *VSpec* hijas pueden y deben escogerse. En el ejemplo se indica que en los coches de tracción total debe de escogerse como mínimo una de las dos *VSpec* hijas 140 o 110 y como máximo una de las dos.

Es posible además imponer restricciones sobre el modelo como se muestra en la Figura 2.8. En el siguiente ejemplo se añade la opción de un asistente de aparcamiento. Para permitir que se active el asistente de aparcamiento se requiere la versión del motor de 140 cv. Formalmente esta restricción implica que todas las ocurrencias de la *Vspec* "140" deben de satisfacer esta restricción.

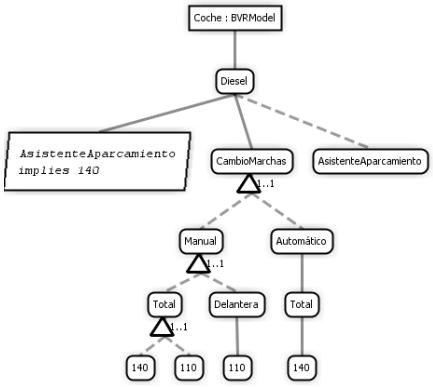


Figura 2.8 Modelo CBR tras añadir el asistente de aparcamiento

Por otra parte también es posible introducir el concepto de *tipo*. En nuestro ejemplo puede encontrarse dos veces el VSpec "total". Pueden representarse estas dos ocurrencias como instancias de un tipo. En la Figura 2.9 puede verse la definición del nuevo tipo de variabilidad "total" que reemplaza a las dos VSpec homónimas. Este tipo se define en la parte superior del árbol con el objetivo de cubrir todas las instancias. La instancia *CambioAutomático:Total* es del tipo *ChoiceOccurrence*, el cual representa una ocurrencia de una instancia del tipo "total". De forma similar la instancia *CambioManual:Total* es del tipo *ChoiceOccurrence*, el cual representa una ocurrencia de una instancia del tipo "total".

Por otra parte también es posible especificar conjuntos de decisiones como *VClassifiers*, los cuales representan decisiones repetidas de forma similar a un árbol *VSpec*. De forma similar a los *ChoiceOccurrences* que *Choices* tipados, los *VClassOccurrences* son *Occurrences* tipados.

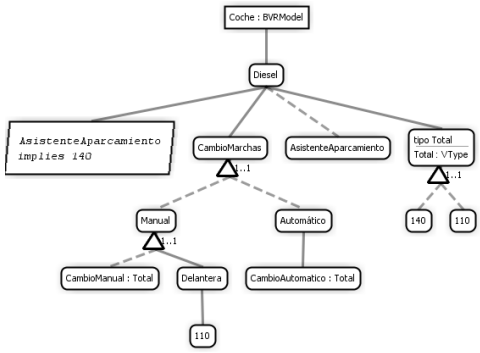


Figura 2.9 Añadiendo el tipo de variabilidad "total"

2.1.5.5 Herramienta de soporte a CVL

El lenguaje CVL contiene una herramienta[3] que le da soporte y la cual se integra en nuestro soporte tecnológico a FeDRE para derivar los requisitos de un producto que se muestra la sección 5.2. Esta versión de la herramienta soporta la versión de CVL propuesta en el proyecto MoSiS y que se utilizó en la propuesta de estándar enviada a la OMG. La herramienta CVL contiene un editor que se utiliza para definir un modelo de variabilidad consistente en los elementos de las capas de abstracción y realización. La herramienta CVL se enfoca en la generación automática de modelos de producto. La capa de realización es muy importante dado que es la que hace posible que el usuario seleccione elementos del modelo base y asociar librerías y definir los cambios en detalle. Una vez que se definido el modelo de variabilidad consistente en las capas de abstracción y realización, la resolución puede definirse seleccionando los elementos de variabilidad. Una vez que se ha definido el modelo de resolución entonces es posible la ejecución del modelo CVL completo de producir los modelos resultantes tras resolver los puntos de variabilidad.

Recientemente se ha propuesto una evolución de CVL llamada *Base Variability Resolution models* (BVR) presentada en (Haugen y Øgård 2014). En esta nueva versión se eliminaron algunos constructos para mejorar la simplicidad y se añadieron constructos nuevos para dotar de más expresividad. BVR está basada en CVL, pero CVL no es un subconjunto de BVR. Esto es debido a que BVR ha eliminado algunos de los mecanismos de CVL que no se emplearon durante las demos industriales. La herramienta (Vasilevskiy y otros 2015) permite la edición de los modelos de variabilidad, resolución, realización y derivación de

[3] Herramienta disponible en la URL:
http://www.omgwiki.org/variability/doku.php/?id=cvl_tool_from_sintef

los productos así como la realización de pruebas y análisis. La herramienta consiste en un conjunto de *plug-in* en la plataforma Eclipse. Durante este trabajo de tesis se hará referencia a los conceptos de CVL haciendo referencia a la versión 2.0 llamada BVR.

2.2 Espacios tecnológicos

En esta sección se presentarán los espacios tecnológicos relacionados con el trabajo de tesis: Eclipse y su marco de trabajo EMF.

2.2.1 Eclipse

La comunidad Eclipse es una comunidad de código abierto, cuyos proyectos se centran en la creación de una plataforma de desarrollo abierta formada por marcos de trabajo extensibles, herramientas para crear, desplegar y gestionar software a lo largo del ciclo de vida.

Uno de sus proyectos más relevantes es la herramienta con el mismo nombre: Eclipse. Eclipse es un proyecto de código abierto, robusto, con múltiples características, con calidad comercial y una plataforma industrial para el desarrollo de herramientas altamente integradas. El *Entorno Desarrollo Integrado* (IDE) utiliza diferentes módulos para enriquecer su funcionalidad, esto supone una ventaja frente a otro tipo de entornos monolíticos donde la funcionalidad no es configurable. Por ejemplo añadiendo el *plug-in* Subversion se podía disfrutar de forma adicional de un sistema de control de versiones integrado en el entorno desarrollo.

En definitiva, la naturaleza esta herramienta la convierte en un IDE abierto y extensible para múltiples propósitos. En este trabajo tendrá especial interés el *Eclipse Modelling Framework* (EMF), un marco tecnológico que permite gestionar modelos y generación de código a partir de modelos descritos en XMI. EMF se describe detalladamente en el punto 2.2.2.

2.2.2 El marco de trabajo EMF

El proyecto *Eclipse Modelling Framework* (EMF) (Steinberg y otros 2008) es un marco de trabajo orientado a la edición de modelos. EMF provee facilidades para generación de código con el objetivo de construir herramientas y otro tipo aplicaciones basadas en modelos de datos estructurados. En EMF se proveen editores para definir metamodelos descrito en un formato llamado Ecore que está basado en *XML*. Estos metamodelos pueden especificarse adicionalmente utilizando Java anotado, documentos XML, o herramientas de modelado como

Rational Rose gracias a que se puede importar a partir de ellas a EMF. Lo más importante de todo, es que EMF provee las bases para establecer interoperabilidad con otras herramientas y aplicaciones basadas en EMF.

Las instancias de los metamodelos definidos en EMF son modelos representados en el formato *Metadata Interchange* (XMI), también basado en XML. Estos modelos instancia del metamodelo definido, pueden editarse fácilmente dado que EMF provee herramientas para producir un conjunto de clases de Java que permiten la edición de dichos modelos. Además se generan un conjunto de clases adaptadoras que permiten vistas y ediciones basadas en comandos del modelo mediante una API (Steinberg y otros 2008).

La *Object Management Group* (OMG) y su estándar *Meta Object Facility* (MOF) están relacionados con EMF. De hecho EMF empezó como una implementación de la especificación MOF madurada a partir de la experiencia obtenida en el desarrollo de herramientas por parte de los desarrolladores de Eclipse. *EMF* puede ser visto como una implementación eficiente en Java de uso conjunto de la API de *MOF*. Sin embargo para evitar confusiones, el metamodelo basado en el núcleo *MOF* de *EMF* se llamará *Ecore* (The Eclipse Foundation 2009).

En la Figura 2.10 se muestra una vista de los componentes del *Ecore*. En la propuesta actual de MOF 2.0 el Essential MOF (EMOF), un subconjunto similar del modelo MOF, se ha separado. Existen pequeñas diferencias, principalmente los nombres, entre Ecore y EMOF; sin embargo, EMF puede leer y escribir señalizaciones de EMOF.

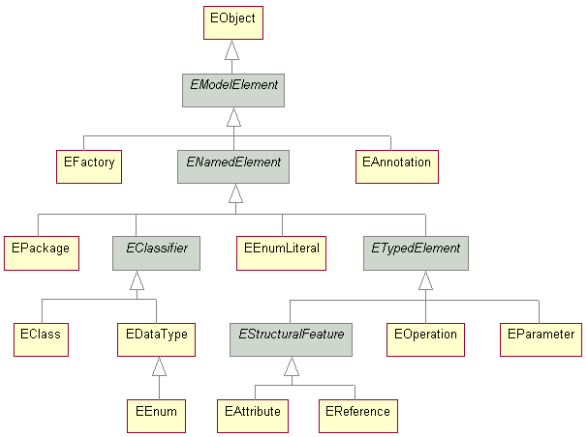


Figura 2.10 Componentes del Ecore[4]

[4] Figura extraída de (The Eclipse Foundation 2009)

2.3 Resumen

En este capítulo se ha proporcionado una breve introducción a los marco tecnológicos que se utilizan en este trabajo de tesis: líneas de producto software, ingeniería de requisitos, desarrollo de software dirigido por modelos y los lenguajes SPEM para la definición de procesos software y CVL para el modelado de la variabilidad en LPS.

Dentro de la sección de marcos tecnológico se han presentado primeramente los conceptos y las actividades principales de la ingeniería de requisitos y sus actividades realizadas: adquisición, definición y especificación, análisis, validación y verificación, y gestión.

Por otra parte se presenta una introducción a las LPS. Un desarrollo de una está formado por dos grandes procesos: la ingeniería de dominio y la ingeniería de la aplicación. En la ingeniería del dominio se construyen un conjunto de *assets* que se utilizan para construir una arquitectura referencia que será común a todos los productos. Por otra parte durante la ingeniería de la aplicación se reutiliza esta arquitectura de referencia para producir el producto software deseado.

Por otra parte se presenta el lenguaje SPEM 2.0 el cual se utiliza en la definición de los procesos de software de este trabajo de tesis. SPEM tiene la ventaja de ser una propuesta de la organización OMG que comprende a muchos de los grandes fabricantes de software.

La OMG también ha realizado la propuesta MDA para cubrir el desarrollo de software dirigido por modelos. La MDA compendia varios estándares, de los cuales se presentan los que están relacionados con este trabajo de tesis: la arquitectura MOF y el lenguaje para expresar restricciones OCL.

Otra propuesta presentada es el lenguaje para representación de la variabilidad CVL. Durante este trabajo de tesis se utiliza la segunda versión de la notación CVL, también llamada BVR. Esta nueva versión incluye una herramienta integrada como un conjunto de *plug-ins* de Eclipse.

Por último dentro de los espacios tecnológicos relacionados de la tesis se presentan los espacios tecnológicos de Eclipse y EMF.

Capítulo 3. Estado del arte

En este capítulo se van a analizar los métodos, técnicas y enfoques existentes para la definición y especificación de los requisitos en el contexto del desarrollo de una línea de productos software. Primeramente se analizan aproximaciones de IR para definir y especificar los requisitos de una LPS. El segundo análisis se enfocará en aproximaciones que soporten la derivación de los requisitos a partir de los requisitos de una LPS.

La sección 3.1 analiza aproximaciones para la definición y especificación de los requisitos de la línea de productos software utilizando técnicas de IR y modelos de características.

La sección 3.2 analiza aproximaciones que cubren la derivación de los requisitos del producto. Además se analizan, en los casos que estén soportados, las técnicas de estas aproximaciones que cubren la definición y especificación de requisitos delta.

La sección 3.3 contiene las conclusiones extraídas del análisis de las propuestas en este capítulo.

3.1 Aproximaciones de IR para definir y especificar requisitos de LPS

La comunidad científica en el área de la IR para LPS ha propuesto diversas aproximaciones para definir y especificar los requisitos de una familia de productos. Algunas aproximaciones extienden técnicas tradicionales de requisitos como casos de uso o metas para modelar la variabilidad. Otro tipo de aproximaciones utilizan técnicas de requisitos tradicionales con modelos de características que capturan la variabilidad de los requisitos. A continuación se analizan estas aproximaciones dependiendo del tipo de notación utilizada para representar los requisitos de una familia de productos software.

3.1.1 Aproximaciones basadas en técnicas IR tradicionales que no expresan la variabilidad con modelos de características

Algunas aproximaciones extienden diferentes modelos de requisitos tales como casos de uso y escenarios con los conceptos de variabilidad sin utilizar explícitamente un modelo de características (Bayer y otros 2000), (Moon y otros 2006).

El enfoque Pulse-CDA (Bayer y Gacek 2000) propuesto por el *Fraunhofer Institute for Experimental Software Engineering*, toma la información del ámbito de aplicación económica o *scoping* (una serie de características del sistema y una definición de los límites de la LPS) y luego produce como salida un *modelo de dominio* (compuesto de un conjunto de *productos de trabajo* que capturan diferentes vistas de dominio) y un *modelo de decisión*. En (Muthig y otros 2004) la técnica de *casos de uso* se utiliza como un producto de trabajo para representar la variabilidad en los casos de uso. Cualquier elemento en el diagrama de casos de uso o en el *escenario textual* puede ser una variante (por ejemplo, un actor o un caso de uso). Los elementos de la variante se adjuntan a las etiquetas de estilo XML para marcar explícitamente variabilidad. Esta solución proporciona al usuario flexibilidad, cubriendo la variabilidad de grano grueso (variabilidad en caso de uso) y grano fino (pasos del escenario textual). Sin embargo, desde nuestro punto de vista, teniendo en cuenta cualquier elemento en el diagrama de casos de uso o en la especificación textual de los casos de uso sea variable podría dar lugar a un gran número de requisitos diferentes para el mismo problema, generando casos de uso ambiguos. En Pulse-CDA el modelo de decisión hace referencia a los elementos variables del modelo de casos de uso y las descripciones de escenarios con el objetivo de instanciar los modelos. Podemos encontrar un ejemplo de instanciación en el ejemplo desarrollador en (Muthig y otros 2004), utilizado como prueba de concepto.

DREAM (Moon y otros 2005) es una aproximación diferente que extiende casos de uso tradicionales añadiéndoles información de la variabilidad. En DREAM el punto de partida son un conjunto de sistemas heredados de los cuales se obtienen los *Requisitos Primitivos* (RP). Los RP tienen un nivel de granularidad menor que un caso de uso y se definen como una transacción que tiene un efecto sobre un actor externo. Estos RP se utilizan como una unidad para representar un requisito identificado. Cada uno de estos RP se representa en una plantilla con la descripción de cada requisito junto su variabilidad. Una vez definidos los RP se construye un diagrama de casos de uso del dominio. A nivel de casos de uso DREAM también considera la variabilidad de requisitos donde se representa utilizando dos estereotipos: *«common»* cuando el caso de uso está siempre presente en todas las configuraciones del producto y *«optional»* cuando el caso de uso está presente en algunas configuraciones de productos. Esta estrategia de variabilidad permite a DREAM soportar variabilidad de grano grueso con los casos de uso, y a nivel fino con la variabilidad a nivel de PR. Sin embargo esta estrategia tiene inconvenientes y al igual que ocurre con la propuesta Pulse-CDA la inclusión de la variabilidad en el mismo modelo de casos de uso puede producir un impacto negativo en la legibilidad y la mantenibilidad de los diagramas. DREAM se presenta –según los autores- como un "caso de estudio" para demostrar la utilidad del método, aunque realmente es un ejemplo utilizado como prueba de concepto.

3.1.2 Aproximaciones de IR para LPS basadas en metas

Otra alternativa existente en la literatura científica para representar los requisitos de la LPS y su variabilidad es el uso de técnicas de IR tradicional como los modelos de metas.

Asadi y otros (2011) utilizan el marco de trabajo i* para representar las interacciones de los usuarios en el modelo de metas y un modelo de características para representar la variabilidad. El proceso de ingeniería del dominio empieza con la representación de los objetivos del producto en el modelo de metas. A partir de las tareas y planes de este modelo de metas se extraen los modelos de características. A partir de estas características se especifica el modelo de características y se establece la trazabilidad desde estas hacia las tareas del modelo de metas. La variabilidad en este caso es de grano grueso, expresándose en términos de características. Sobre la definición del proceso, observamos que se definen los modelos que se utilizan de entrada y salida. La aproximación se aplica en un ejemplo que modela una LPS de sistemas de compra electrónica como prueba de concepto.

Este es el caso de *Aspect-oriented User Requirements Notation* (AoURN) (Mussbacher y otros 2012) donde se proponen cuatro actividades principales de ingeniería de dominio:

i) Construir un modelo de metas de los *stakeholders*.

ii) Construir un modelo de características, en el que las funciones se representan como metatareas etiquetadas con el estereotipo «*feature*».

iii) Construir el modelo de impacto que estable el impacto de las características en el modelo de metas de los *stakeholders*.

iv) Crear el modelo de escenario de las características, en el que las características que no son hojas se describen con más detalle con los *Aspect-oriented Use Case Maps* (AoUCM).

En AoURN la trazabilidad de las características a los requisitos se hace mediante el uso de enlaces desde las tareas estereotipadas en el modelo de función hacia el modelo de escenario de las características AoURN. Sobre la definición del proceso, AoURN define los artefactos de entrada y de salida sólo parcialmente. Respecto a la granularidad de la variabilidad se considera una granularidad gruesa y fina; puesto que la variabilidad es a nivel de característica y meta, y también un soporte a la variabilidad fina con modelos de escenarios. La aproximación se ejemplifica con un sistema de gestión de coches accidentados como prueba de concepto.

3.1.3 Aproximaciones de IR para LPS que utilizan otros modelos UML

Otra alternativa para representar los requisitos de una LPS y su variabilidad es extender notaciones tradicionales de UML para representar los requisitos de una LPS. Esta idea se aplica en *Feature-Oriented Requirements Modeling Language* (FORML) (Shaker y otros 2012). FORML combina el *modelado de características* y *máquinas de estado UML*. FORML descompone los requisitos en un modelo del mundo y una modelo de comportamiento. Una característica en el modelo del mundo se descompone en varios módulos de características en el modelo de comportamiento. Se soporta granularidad de la variabilidad gruesa (mediante características) y fina (mediante la descomposición en diagramas de comportamiento). Esta descomposición permite la modularidad, la cual es una de las mayores contribuciones de la propuesta. FORML no define roles ni guías en el proceso de especificación de requisitos del dominio. Por último la propuesta se aplica a dos ejemplos utilizados como prueba de concepto en los dominios automovilístico y telefónico.

3.1.4 Aproximaciones de IR para LPS basadas en técnicas de requisitos tradicionales y modelos de características

Durante los últimos años se han propuesto múltiples modelos y técnicas para especificar los requisitos en LPS. Algunas aproximaciones combinan modelos de características con las técnicas más tradicionales de IR como casos de uso (Griss y otros 1998), (Eriksson y otros 2005).

FeatuRSEB (Griss y otros 1998) integra las propuestas de FODA (Kang y otros 1990), para representar la variabilidad con modelos de características, y *Reuse-Driven Software Engineering Business* (FeatuSEB), donde el modelo de características se deriva del modelo de casos de uso del dominio. FeatuRSEB soporta una granularidad de la variabilidad de nivel grueso a nivel de característica en el modelo de características. La definición del proceso es parcial, definiéndose dos roles: el ingeniero de sistemas que es responsable de crear el modelo de casos de uso y el ingeniero del dominio que es responsable de crear el modelo de características.

PLUSS (Eriksson y otros 2005) mejora el enfoque FeatuRSEB añadiendo más mecanismos de variabilidad:

i) A nivel de casos de uso.

ii) A nivel de escenarios alternativos.

iii) En el flujo de los acontecimientos desde un escenario alternativo.

iv) Con aspectos transversales (*crosscutting*) que afectan a varios casos de uso.

De este modo PLUSS da un a la variabilidad de grano grueso con las características y la variabilidad de grano fina a nivel de escenario. Ni FeatuRSEB ni PLUSS proponen roles en sus métodos, y se limitan a proporcionar guías parciales para ayudar en la actividad RE. Además, los artefactos de entrada y de salida están definidos tan solo parcialmente. Respecto a la validación FeatuRSEB se ha aplicado en entornos industriales pero no presenta una validación empírica. PLUSS también se ha aplicado en entornos industriales pero sí que ha realizado una validación empírica comparando la utilidad de los modelos generados por PLUSS comparados con los obtenidos aplicando la metodología RUP (Kruchten 1999). El experimento concluye que PLUSS resulta más apropiado que RUP en entornos industriales.

La idea de combinar un modelo de características con la técnica de casos de uso también se utiliza en la aproximación *Variability Modeling Language for Requirements* (VML4RE) en el contexto del proyecto AMPLE (Alférez y otros 2011). Este enfoque presenta dos contribuciones principales. En primer lugar el lenguaje

VML4RE, el cual es un lenguaje específico de dominio que es capaz de componer requisitos con modelos de características; y en segundo lugar, un proceso de requisitos que utiliza:

i) Un *modelo de características* para realizar la identificación de la variabilidad.

ii) Los *casos de uso y diagramas de actividad* para describir los requisitos del dominio de la SPL.

iii) Un *modelo VML4RE* que relaciona el modelo de características con los modelos de requisitos.

Estos tres modelos, junto con la configuración del producto, se toman como entrada por un intérprete para obtener los requisitos del producto. También proporcionan la comprobación de coherencia entre las características y escenarios de casos de uso. VML4RE da un soporte a la variabilidad de grano grueso a nivel de característica y de grano fino al enlazar esta variabilidad a los casos de uso y diagramas de actividad mediante el LED que proponen para componer los requisitos. VML4RE define las entradas y salidas del proceso sin proporcionar guías ni roles. VML4RE no documenta la aplicación de ninguna validación empírica a la propuesta.

De una forma análoga a la propuesta anterior *Modeling Scenario Variability as Crosscutting Mechanisms* (MSVCM) (Bonifácio y Borba 2009) utiliza escenarios. MSVCM se centra en la derivación de los requisitos del producto mediante el uso de los siguientes artefactos: modelo de casos de uso, modelo de características, configuración del producto, y conocimientos de la configuración. Estos artefactos se toman como entrada en un proceso de *weaving* que resuelve las relaciones trasversales entre requisitos y características para producir como resultado el modelo de casos de uso del producto. El *modelo de casos de uso* se compone de casos de uso y casos de uso aspectuales. Los casos de uso tienen una descripción y una lista de escenarios textuales mientras que los casos de uso aspectuales tienen un nombre y una lista de *advices*. Estos *advices* se utilizan para extender el comportamiento de los casos de uso existentes. La granularidad es de grano grueso mediante las características y de grano fino mediante los *advices* a nivel de escenario. Por otra parte el modelo de *conocimiento de la configuración* establece la trazabilidad desde la configuración del producto a transformaciones que se utilizarán en el proceso de *weaving*. Por último MSVCM define las entradas y salidas del proceso pero no provee guías para aplicarlo ni roles. Por último MSVCM realizó un experimento comparándolo con PLUSS. Se utilizaron un conjunto de métricas que median los resultados tras aplicar el experimento, concluyendo que MSVCM tiene mayor nivel de modularidad de las características y un mayor nivel de cohesión de los escenarios respecto a PLUSS.

3.1.5 Discusión

En esta subsección profundizamos en las aproximaciones anteriormente descritas siguiendo los criterios de comparación presentados en la Tabla 3.1 con el fin de analizar las diferentes estrategias para definir y especificar los requisitos de la SPL. El criterio C1 analiza las actividades de SPL relacionadas con la ingeniería del dominio (*scoping*, ingeniería de dominio). En el criterio C2 se analiza qué artefactos se utilizan por las diferentes aproximaciones. En el criterio C3 analizamos el soporte que se da a la variabilidad de requisitos. Un soporte de granularidad gruesa puede cubrir por ejemplo elementos UML como paquetes o casos de uso. Un soporte de granularidad fina cubre la variabilidad de elementos como acciones dentro de un diagrama de actividad que especifica un caso de uso. El criterio C4 analiza "cómo se definió el proceso", donde se incluyen tres sub-criterios: entradas y salidas si el enfoque proporciona pautas, si el enfoque define los roles, y si el enfoque tiene bien definidos. Por último en el criterio C5 se analizó el tipo de realización empírica. Distinguiremos entre las siguientes modalidades: ningún tipo de validación (no), ejemplo utilizado como prueba de concepto (pc), validación empírica en el ámbito académico (va) o validación empírica industrial (vi). Consideraremos como validación académica o industrial en aquellos casos que se haya realizad un experimento o cuasi-experimento.

Tabla 3.1 Criterio de evaluación de las aproximaciones de requisitos que cubren la IR

Criterio	Descripción
C1	Actividades de la LPS cubiertas
C2	Artefactos utilizados
C3	Soporte a la variabilidad
C4	Definición del proceso
C5	Tipo de validación empírica

Los resultados de esta comparativa se muestran en la Tabla 3.2 muestra los resultados de la comparativa de las aproximaciones analizadas. De este análisis podemos concluir lo siguiente:

- En muchos casos hemos encontrado que el *scoping* y la IR del dominio se han considerado como actividades separadas. De acuerdo con John y Eisenbarth (2009) postulan que son necesarias relaciones bien definidas entre los artefactos de las actividades de *scoping* y la IR del dominio con el objetivo de reducir el esfuerzo en la definición de los requisitos.
- En la mayoría de las aproximaciones se soporta granularidad a nivel grueso (mediante características) y fino mediante diferentes notaciones como casos

de uso ((Eriksson y otros 2005), (Griss y otros 1998)) o modelos de metas (Soltani y otros 2012) que se extienden para soportar la variabilidad. Estas aproximaciones representan la variabilidad de requisitos y extraen a partir de ellos los modelos de características. Sin embargo el hecho de representar la variabilidad dentro en la especificación de requisitos puedo hacer que esta sea más difícil de entender y mantener (Alférez y otros 2009).

- Algunas aproximaciones utilizan los modelos de requisitos para identificar la variabilidad de la LPS y producir como salida un modelo de características representado la variabilidad de la LPS. Sin embargo los desarrolladores de LPS y los expertos del dominio están más familiarizados con los conceptos de característica y modelado de variabilidad (Oliveira y otros 2014) con lo que la utilización de requisitos para identificar la variabilidad de la LPS puede no resultar adecuada.

- La mayoría de aproximaciones presentadas no tienen procesos de especificación de requisitos completamente definidos. La mayoría de propuestas no presentan guías para establecer qué pasos tiene que seguir el ingeniero de requisitos para aplicar correctamente el proceso.

- La mayoría de aproximaciones no realizan una validación empírica. Este hecho se evidencia en estudios como el de Alves (Alves y otros 2010). La mayoría de las aproximaciones se presentaban mediante ejemplos que mostraban una prueba de concepto.

Tabla 3.2 Comparación de las aproximaciones de IR que cubren la ingeniería del dominio

Propuesta	C1	C2	C3	C4	C5
Pulse-CDA (Bayer y Gacek 2000)	SC, ID	Modelo análisis dominio, CU	Grueso, fino	Parcial	PC
DREAM (Moon y otros 2005)	ID	Matriz trazabilidad requisitos primitivos, CU	Grueso, fino	Parcial	PC
Goal-driven SPL engineering (Asadi, y otros 2011)	ID	Modelo de metas, MC	Grueso	Parcial	PC
AoURN (Mussbacher y otros 2012)	ID	Modelo de metas de *stakeholders*, MC, modelo de impacto de características, modelo de escenarios de características	Grueso, fino	Parcial	PC
FORM (Kang y otros 1998)	ID	MC, modelo comportamiento	Grueso, fino	Parcial	PC
FeatuRSEB (Griss y otros 1998)	ID	CU, MC	Grueso, fino	Parcial	PC
PLUSS (Eriksson y otros 2005)	ID	CU, MC, casos de cambio	Grueso, fino	Parcial	VI
VML4RE (Alférez y otros 2011)	ID	UC, MC, DA	Grueso, fino	Parcial	PC
MSVCM (Bonifácio y Borba 2009)	ID	UC, MC	Grueso, fino	Parcial	VA

Leyenda:

- SC: *scoping*
- ID: Ingeniería del dominio
- CU: Modelos de casos de uso
- MC: Modelo de características
- DA: Diagramas de actividad
- PC: Prueba de concepto
- VA: Validación empírica académica
- VI: Validación empírica industrial

3.2 Aproximaciones de IR para la derivación de los requisitos del producto y requisitos delta

En esta subsección se analizan diferentes aproximaciones que cubren el proceso de la ingeniería de la aplicación de los requisitos. Históricamente las aproximaciones de IR se han centrado principalmente en la ingeniería del dominio, dando un menor soporte a la ingeniería de la aplicación (Alves y otros 2010). Algunas aproximaciones proveen soporte mediante herramienta para asistir la derivación en un proceso semi-automática ((Kang y otros 1998), (Eriksson y otros 2005), (Djebbi y otros 2007), (Mussbacher y otros 2012)). Por otra parte existen basadas en una estrategia de DSDM para soportar la derivación de los requisitos ((Bragança y Machado 2009), (Alférez y otros 2011), (Czarnecki y Antkiewicz 2005), (Bonifácio y Borba 2009)).

3.2.1 Aproximaciones de IR para LPS basadas en la derivación semi-automática

En el análisis de aproximaciones basadas en la derivación semi-automática encontramos propuestas especifican los requisitos y utilizan un modelo de características para representar su variabilidad (Kang y otros 1998), (Eriksson y otros 2005). Otras propuestas sin embargo no representan la variabilidad de los requisitos (Djebbi y otros 2007) y simplemente soportan un *matching* para seleccionar un conjunto de requisitos del producto a partir de los requisitos de la LPS.

FORM (Kang y otros 1998) es un método de análisis del dominio que se centra en la descripción de la variabilidad de la línea de productos mediante características. FORM extiende el trabajo de *Feature-Oriented Domain Analysis* (FODA) con el objetivo de cubrir el proceso entero de reutilización y no limitarse únicamente a especificar la variabilidad en la fase de la ingeniería del dominio. FORM cubre la ingeniería del dominio mediante un análisis donde se crea, como artefacto de salida, un modelo de características. Este modelo se utiliza en la actividad de análisis de requisitos del usuario donde se crea como artefacto de salida una especificación de la selección de características. Esta derivación de los requisitos no incorpora requisitos delta, limitándose a requisitos ya existentes en la LPS. Respecto a la definición del proceso, FORM define sus entradas y salidas del proceso junto con los roles que participan, sin embargo no provee guías explicitas para aplicar el método. Por último se provee una herramienta que soporta el modelado y la definición de trazabilidad entre el

modelo de características y el resto de artefactos. FORM se presenta únicamente con un ejemplo como prueba de concepto.

PLUSS (Eriksson y otros 2005) es una aproximación basada en FeatuRSEB (Griss y otros 1998) para cubrir la definición y especificación de requisitos de la familia de productos y posteriormente la derivación de los requisitos del producto. PLUSS utiliza casos de uso y escenarios textuales para especificar los requisitos, y un modelo de variabilidad para especificar la variabilidad de la línea de productos. Para derivar los requisitos de un producto específico, se genera un modelo de casos de uso del producto a través de un filtrado en el modelo de características. PLUSS utiliza el concepto de *caso de uso cambiante* para indicar una funcionalidad aprobado, pero todavía no aceptado en la ingeniería del dominio. Los casos de uso cambiantes que son aceptados se agregan al diagrama de casos de uso del dominio. Esta técnica permite que PLUSS modele requisitos delta, definiendo requisitos que no existían en la LPS. PLUSS no define roles y sus guías de aplicación y definición de entradas y salidas en el proceso son parciales. PLUSS documenta soporte de herramienta a partir de una extensión de la herramienta comercial Telelogic DOORS y la herramienta de modelado de UML IBM-Rational Rose. Por último PLUSS se ha validado industrialmente mediante un experimento comparándolo con la metodología RUP.

RED-PL (Djebbi y otros 2007) propone una aproximación para derivar los requisitos a partir de una especificación de requisitos para la familia de productos. RED-PL provee un proceso metodológico que guía la adquisición de los requisitos y realiza la derivación tratándola como una actividad de decisión. Estas decisiones sirven de guía en la en un proceso sistemático e interactivo para derivar los requisitos del producto. Durante la derivación de los requisitos en RED-PL se identifican los requisitos del producto y se realiza un proceso de comparación entre los requisitos adquiridos del producto y los requisitos de la LPS, derivando un conjunto de requisitos del dominio de modo que no cubre la especificación de requisitos delta. No se representa en este caso de manera explícita la variabilidad de los requisitos. Por otra parte la definición del proceso es parcial dado que se definen las entradas y salidas del método únicamente. RED-PL provee una herramienta que implementa la técnica de derivación de forma interactiva. Esta herramienta se basa en el razonador GNU-Prolog. La propuesta se muestra con un ejemplo utilizado como prueba de concepto.

3.2.2 Aproximaciones de IR para LPS basadas en DSDM

Dentro de las aproximaciones que soportan la automatización algunas como TDL (Yu y otros 2014) aún están construyendo el prototipo. Otras propuestas sin embargo como (Bragança y Machado 2009), (Alférez y otros 2011) y (Bonifácio y Borba 2009) proveen prototipos funcionales que permiten derivar los requisitos de forma automática.

La propuesta *Transformation Description Language* (TDL) (Yu y otros 2014) propone una aproximación para la derivación automática de casos de uso a partir de un modelo de características. Se propone un LED para especificar la información de transformación del modelo de características a los diagramas de casos de uso. La granularidad de la variabilidad es gruesa, dado que la variabilidad de requisitos se define a nivel de características. Por otra parte el tipo de variabilidad es positiva al añadirse los requisitos durante la derivación a partir del programa TDL. La definición del proceso es parcial dado que únicamente se definen las entradas y las salidas. Por último la propuesta asegura que tienen una herramienta en fase de desarrollo con el objetivo de dar un soporte automatizado a la derivación a partir de un programa especificado en TDL que sea correcto, siendo actualmente el soporte automático a la derivación conceptual. Dado que la aproximación se centra en la mera derivación de los requisitos, no soporta la definición y especificación los requisitos delta. TDL ejemplifica mediante tres ejemplos aplicados como prueba de concepto: una LPS domótica, una LPS de videojuegos de arcade y una LPS de tiendas de productos en Internet.

Bragança y Machado (2009) proponen una propuesta para derivar los requisitos del producto mediante una aproximación basada en transformaciones de modelos. En esta aproximación durante la ingeniería del dominio propone identificar los requisitos comunes y variables de la LPS y agruparlos en características. A continuación se definen los requisitos relacionados con las características utilizando la técnica de casos de uso. Respectivamente los casos de uso se especifican con diagramas de actividad. La trazabilidad entre las características y los elementos UML se almacena en una tabla de enlaces de trazabilidad. La granularidad para especificar la variabilidad es gruesa (características y casos de uso) y fina (a nivel de diagramas de actividad), sin embargo los diagramas de caso de uso proveen estereotipos para indicar la variabilidad y en consecuencia no separan correctamente los requisitos funcionales y la variabilidad. Durante la derivación de los requisitos se toman como entrada una configuración del producto, el modelo de requisitos del dominio (casos de uso y diagramas de actividad) y los enlaces de trazabilidad. Esta derivación es automática mediante transformaciones de modelos, de modo que soporta el tipo de variabilidad positiva. Esto implica que esta aproximación

no soporta la definición y especificación de requisitos delta dado que sólo soporta la derivación de los requisitos del producto. En la definición del producto para derivar dichos requisitos se definen los artefactos de entrada y salida. Esta aproximación se presenta con un ejemplo de una LPS de librerías, utilizado como prueba de concepto.

Variability Modeling Language for Requirements (VML4RE) (Alférez y otros 2011) extiende la propuesta inicial de Bragança y Machado. VML4RE permite derivar de forma automática los requisitos del producto de forma automática. VML4Re utiliza en la ingeniería del dominio los siguientes artefactos:

i) Un modelo de características para llevar a cabo la identificación de la variabilidad.

ii) Los casos de uso y diagramas de actividad para describir los requisitos de la ingeniería del dominio.

iii) El modelo VML4RE que relaciona el modelo de características con los modelos de requisitos.

La granularidad de la variabilidad es gruesa (a nivel de característica y caso de uso) y fina (variabilidad de diagramas de actividad). Durante la ingeniería de la aplicación se componen los requisitos con el modelo de variabilidad, con un lenguaje específico de dominio que es capaz de componer requisitos con modelos de características. En la ingeniería de la aplicación se utilizan los tres modelos de la ingeniería del dominio, junto con la configuración del producto, los cuales se toman como entrada por un intérprete que deriva los requisitos del producto. El DSL para componer los requisitos permite especificar la variabilidad simultáneamente de forma positiva y negativa. Adicionalmente se provee análisis de consistencia entre características y los escenarios de los casos de uso. Sin embargo no se da soporte a la definición y especificación de requisitos delta dado que meramente se derivan los requisitos del producto. Respecto a la definición del proceso, al igual que la propuesta anterior de Bragança y Machado, se definen los artefactos de entrada y salida, pero no se definen los roles y las guías para aplicar el proceso. VML4RE se aplica en un ejemplo de una LPS domótica, utilizado como prueba de concepto.

Model Templates (Czarnecki y Antkiewicz 2005) propone una solución para derivar los requisitos utilizando el concepto de modelo plantilla. En esta aproximación se utiliza durante la ingeniería del dominio un modelo de características y un modelo de plantillas. Los modelos de plantilla son una extensión de los diagramas de actividad utilizando anotaciones con condiciones de presencia y metaexpresiones. Estas condiciones se utilizan para indicar si un elemento de un modelo se incluirá o no en la especificación del producto dependiendo de la

configuración del producto. La granularidad de la variabilidad será por lo tanto de nivel grueso (características) y de nivel fino mediante las condiciones en los diagramas de actividad. El tipo de variabilidad es negativa dado que sólo se proveen operadores para eliminar requisitos en caso de que no se incluyan en el producto a derivar. Para realizar la derivación de los requisitos se realiza un proceso de configuración manual para crear la configuración del producto a partir del modelo de características. Esta configuración del producto se utiliza en un *proceso de instanciación* que es una transformación modelo-a-modelo donde se evalúan las condiciones de presencia y meta-expresiones para derivar la instancia de la plantilla. Dado que la propuesta se basa en la derivación no permite definir y especificar requisitos delta. Respecto a la definición del proceso, se definen parcialmente las entradas y salidas del proceso, no se definen explícitamente roles y no se especifican guías para aplicar el método. Esta aproximación se aplica a un ejemplo como prueba de concepto.

Modeling Scenario Variability as Crosscutting Mechanisms (MSVCM) (Bonifácio y Borba 2009) está orientado en la derivación de los requisitos del producto. La variabilidad de la línea de producto software se va a representar mediante un modelo de características. Los requisitos se definen mediante *diagramas de caso de uso*, los cuales se especifican mediante escenarios textuales. Las relaciones entre estos dos modelos se expresan en el *modelo de conocimiento*. En la ingeniería del producto se utilizan la configuración del producto y el modelo de conocimiento de la configuración junto con los artefactos de la ingeniería del dominio. Estos artefactos se toman de entrada en el proceso de *weaving* para derivar la especificación de requisitos del producto. Esto implica que la granularidad de la variabilidad es grueso (nivel de características) y fino (variabilidad en el caso de uso aspectual). La variabilidad es positiva dado que se utiliza un enfoque composicional para derivar los requisitos. Por otra parte sólo se soporta la derivación de requisitos, pero no se ha definido un mecanismo para definir y especificar los requisitos delta. Por último en la definición del método se han establecido los modelos de entrada y de salida, aunque no se han definido los roles ni guías para aplicar el método. Como se comentó en el análisis de aproximaciones que cubren la ingeniería del dominio, MSVCM se ha validado mediante un experimento en el ámbito académico.

3.2.3 Discusión

En la subsección 3.2 se han analizado aproximaciones de la literatura con el objetivo de definir y especificar los requisitos de un producto. Se ha definido un criterio de comparación para analizar dichas aproximaciones compuesto de 6 criterios principales que se muestran en la Tabla 3.3. El criterio C1 analiza el soporte a la ingeniería del dominio pudiendo ser: derivación o derivación y refinamiento mediante la definición y especificación de requisitos delta. En el criterio C2 se analizan los artefactos utilizados por las aproximaciones. El criterio C3 analiza la definición del proceso para derivar los requisitos (definición de entradas y salidas, definición de guías para aplicar el proceso y definición de roles en el proceso). El criterio C4 analiza la granularidad variabilidad. Un soporte de granularidad gruesa puede cubrir por ejemplo elementos UML como paquetes o casos de uso. Un soporte de granularidad fina cubre la variabilidad de elementos como acciones dentro de un diagrama de actividad que especifica un caso de uso. El criterio C5 analiza el soporte de variabilidad en los procesos que automatizan la derivación de los requisitos del producto: positivo si se ofrecen operadores para añadir requisitos a la especificación en caso de que se resuelva positivamente para una configuración de producto. Por otra parte se soporte la variabilidad negativa si se soportar operadores como borrar o reemplazar un requisito dependiendo de la configuración del producto. El criterio C6 analiza el soporte con herramienta a la aproximación. Por último en el criterio C7 se analizó el tipo de realización empírica. Distinguiremos entre las siguientes modalidades: ningún tipo de validación (no), ejemplo utilizado como prueba de concepto (pc), validación empírica en el ámbito académico (va) o validación empírica industrial (vi). Consideraremos como validación académica o industrial en aquellos casos que se haya realizad un experimento o cuasi-experimento.

Tabla 3.3 Criterio de evaluación de las aproximaciones de IR que cubren la ingeniería de la aplicación

Criterio	Descripción
C1	Soporte a la ingeniería de la aplicación
C2	Artefactos utilizados
C3	Definición proceso (guías, entradas, salidas)
C4	Granularidad de la variabilidad
C5	Tipo de variabilidad
C6	Soporte con herramienta
C7	Tipo de validación empírica

La Tabla 3.4 muestra los resultados del análisis. Tras este análisis podemos observar lo siguiente:

- La mayoría de las aproximaciones se centran en la derivación de los requisitos del producto. Sin embargo la mayoría de aproximaciones no soportan la definición y especificación de requisitos que no se pueden derivar a partir de la especificación de requisitos de la LPS (requisitos delta).
- La mayoría de procesos que acompañan a las aproximaciones están definidos parcialmente. No se proveen guías para aplicar correctamente los procesos. Se proveen algunos prototipos para definir y especificar los requisitos del producto. Sin embargo sólo la propuesta PLUSS y su herramienta permiten definir y especificar los requisitos del producto y los requisitos delta en su herramienta.
- Al igual que se ha evidenciado en el análisis de propuestas de IR para LPS que cubren la ingeniería del dominio, la mayoría de aproximaciones no han realizado una validación empírica. Muchas aproximaciones utilizan erróneamente el término "estudio de caso" pero realmente son meros ejemplos utilizados como pruebas de contexto. Entre todas las propuesta únicamente una aproximación realizó un experimento en el contexto académico y otra aproximación realizó un experimento en el ámbito industrial.

Tabla 3.4 Comparación de las aproximaciones de IR que cubren la ingeniería de la aplicación

Propuesta	C1	C2	C3	C4	C5	C6	C7
FORM (Kang y otros 1998)	DER	MC, modelo comportamiento	P	Grueso	--	Prototipo	PC
PLUSS (Eriksson y otros 2005)	DER, DEL	MC, CU, casos de cambio	P	Grueso, fino	--	Industrial	VI
RED-PL (Djebbi y otros 2007)	DER	Modelo de requisitos de la LPS, modelo de requisitos de los *stakeholders*	P	--	--	Prototipo	PC
TDL (Yu y otros 2014)	DER	MC, CU, programa TDL (modelo configuración)	P	Grueso, fino	Positiva	No	PC
Bragança y Machado (Bragança y Machado 2009)	DER	MC, CU, DA	P	Grueso	Positiva	Prototipo	PC
VML4RE (Alférez y otros 2011)	DER	MC, UC, DA	P	Grueso, fino	Positiva y negativa	Prototipo	PC
Model Templates (Czarnecki y Antkiewicz 2005)	DER	MC, DA	P	Grueso, fino	Negativa	Prototipo	PC
MSVCM (Bonifácio y Borba 2009)	DER	MC, UC	P	Grueso, fino	Positiva	Prototipo	VA

Leyenda:
ID: Ingeniería del dominio
- MC: Modelo de características
- DER: Derivación
- DEL: Especificación de requisitos delta
- P: Parcial
- CU: Modelos de casos de uso
- MC: Modelo de características
- DA: Diagramas de actividad
- --: No aplicable
- PC: Prueba de concepto
- VA: Validación empírica académica
- VI: Validación empírica industrial

3.3 Conclusiones

Finalmente se presentan las conclusiones obtenidas del análisis del estado del arte. No todas las aproximaciones analizadas cubren los procesos de ingeniería del dominio e ingeniería de la aplicación. Entre las aproximaciones que cubren la ingeniería del dominio se observa que existe una desconexión entre la actividad de *scoping* y la actividad de la ingeniería del dominio. Por otra parte muchas de las aproximaciones se centran meramente en extender técnicas y notaciones tradicionales de lR con el objetivo de representar la variabilidad. Sin embargo estas aproximaciones presentan el problema de centrarse en utilizar técnicas de IR para capturar la variabilidad de la LPS cuando esto sucede realmente anteriormente en la fase de *scoping*, alterando el orden lógico del proceso.

Como resultado del segundo análisis del estado del arte entre las aproximaciones que cubren la IR de la aplicación, encontramos propuestas para derivar los requisitos de la aplicación de una forma guiada y en muchos casos automática gracias a la ayuda o bien de técnicas de DSDM o bien mediante soporte de herramienta. Sin embargo el hecho de seleccionar entre un conjunto de requisitos predefinidos restringe considerablemente la creatividad para tratar con problemas nuevos y reduce el valor añadido de los productos nuevos que se desarrollar (en la LPS) (Djebbi y otros 2007). En consecuencia una estrategia basada en la mera derivación de los requisitos de un producto a partir de los requisitos del dominio debería de complementarse con una estrategia de definición de deltas para completar las necesidades de un cliente. Tan sólo propuestas como PLUSS son capaces de anticipar los cambios y la definición de requisitos delta.

Por otra parte se observan tanto en las aproximaciones que soportan la ingeniería del dominio, como las que soportan la ingeniería de la aplicación una falta de definición de los procesos que las acompañan completa. Normalmente las propuestas definen los modelos de entrada y se salida pero guías con el objetivo de ayudar al ingeniero de requisitos a aplicar correctamente el proceso.

Finalmente tanto en el análisis de aproximaciones que cubren la ingeniería del dominio como de la aplicación se observa que la mayoría de las aproximaciones se limitan a mostrar un ejemplo como prueba de concepto. En general es necesario la aplicación de validaciones empíricas de las aproximaciones de IR en el desarrollo de LPS.

Capítulo 4. FeDRE: un método de IR para LPS

En este capítulo se presenta FeDRE, un método que permite la definición y especificación de los requisitos en el desarrollo de LPS. Se presentarán los artefactos, roles, actividades y tareas propuestos respectivamente en los procesos de ingeniería del dominio e ingeniería de la aplicación. Las distintas fases y actividades que integran el método se especifican utilizando el estándar SPEM (*Software and Systems Process Engineering Metamodel Specification*), el cual fue propuesto por la OMG (2008) para dar soporte a la ingeniería de procesos.

La sección 4.1 presenta la vista global del método. Se describen a grandes rasgos las principales actividades de los dos procesos de FeDRE con el objetivo de tener una visión en conjunto del proceso.

La sección 4.2 presenta los artefactos de la actividad de *scoping* que se utilizan de entrada para el proceso de la ingeniería del dominio de FeDRE.

La sección 4.3 presenta el proceso de ingeniería del dominio con mayor nivel de detalle con el objetivo de mostrar cómo se definen y especifican los requisitos de la LPS a partir de los artefactos de la fase de *scoping*.

La sección 4.4 presenta el proceso de la ingeniería de la aplicación. Se muestra el proceso para derivar un producto y especificar los requisitos delta, si es necesario, para cubrir todas las necesidades del usuario.

La sección 4.5 presenta las conclusiones del capítulo.

4.1 Vista general del proceso

En esta sección se presenta el ejemplo que se utilizará durante el resto del capítulo para ilustrar los conceptos y técnicas de *Feature-Driven Requirements Engineering* (FeDRE). También se presentará los dos procesos que componen FeDRE.

4.1.1 La línea de productos Savi

En esta sección se presenta la LPS Savi, la cual se utilizará durante todo el capítulo para ilustrar los distintos conceptos utilizados en FeDRE. La especificación completa se encuentra en el material experimentan en el Apéndice B. Savi es una línea de productos software para móviles inteligentes. Los productos de esta LPS permiten enviar mensajes de alerta en situaciones de emergencia (accidentes de emergencia, secuestros, necesidad de rescate). Estas notificaciones se envían a la lista de contactos del usuario mediante un mensaje SMS y e-mail.

A continuación presentamos brevemente un producto creado a partir de esta LPS: *Savi Standard*. Este producto provee un sistema de control de acceso para gestionar las cuentas del usuario. La pantalla de inicio de *Savi Standar* es una ventana de identificación que solicita al usuario la introduccion de su nombre de usuario y la contraseña (ver Figura 4.1). En otros productos de la LPS se permite realizar la identificación a través de las redes sociales Facebook y/o Twitter.

Figura 4.1 Pantalla de identificación en Savi Standard

Una vez el usuario se identifica se muestra la pantalla principal. El usuario tiene cuatro opciones:

- *Botón de rescate*. Inicia el seguimiento o *tracking* del usuario.
- *Botón de contacto*. Permite gestionar los contactos.
- *Botón de localización*. El usuario envía un e-mail o SMS para avisar a los contactos que se encuentra perdido.
- *Botón de configuración*. Muestra opciones para configurar la aplicación.

En la pantalla principal nos encontramos el botón de confirmación de la situación de emergencia (ver Figura 4.2). Si el usuario presiona este botón el sistema solicita al usuario que confirme la situación de emergencia. Si el usuario acepta, se inicia el sistema de seguimiento y se cambia la interfaz de usuario al modo seguimiento.

Figura 4.2 Pantalla principal de Savi Standard

Si el usuario confirma el seguimiento se inicia la opción para enviar un e-mail a los contactos del teléfono (ver Figura 4.3). En otros productos de la LP es posible enviar notificaciones de la situación de emergencia utilizando contactos de las redes sociales Twitter o Facebook.

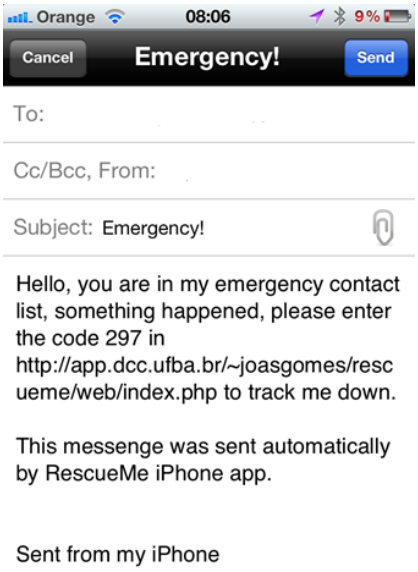


Figura 4.3 Envío de e-mail a los contactos en Savi Standard

Otra de las funcionalidades de *Savi Standard* es la gestión de contactos. Pueden añadirse contactos en la agenda o importarlos desde la agenda de contactos del propio teléfono (Figura 4.4). En otros productos de la LPS que tienen soporte social es posible importar contactos desde Facebook y/o Twitter.

Figura 4.4 Formulario para crear contactos en Savi Standard

Otra funcionalidad en Savi Standard es la función de pérdida. Esta funcionalidad tiene dos alternativas. La primera opción mostrar la ubicación del usuario en el caso de que se encuentre perdido (por ejemplo en una tormenta, un desastre natural, etc.) tal como muestra la Figura 4.5.

Figura 4.5 Pantalla de ubicación del usuario en Savi Standard

La segunda opción es mostrar los números de emergencia. Se pueden añadir números de emergencia locales de servicios de bomberos, policía, etc.

Figura 4.6 Números de emergencia en Savi Standard

Finalmente se permite al usuario configurar diferentes parámetros como el idioma de la aplicación. En la versión Lite sólo se permite como idioma por defecto el inglés. En los productos con idiomas adicionales se permiten seleccionar otros idiomas: español o portugués.

Figura 4.7 Opciones de configuración en Savi Standard

4.1.2 Vista general del método

El método *Feature-Driven Requirements Engineering* (FeDRE) se centra en dar soporte a la definición y especificación de los requisitos en los dos principales procesos del desarrollo LPS: la ingeniería del dominio y la ingeniería de la aplicación. En la Figura 4.8 se muestra una visión global de FeDRE con el flujo de acción y las principales actividades de cada uno de los procesos. Durante esta sección se describirán cada una de estas actividades, los roles que participan y los artefactos que se producen como entrada o salida.

Respeto al proceso de la *ingeniería del dominio*, FeDRE, a diferencia de otros métodos de IR para LPS, se utiliza el modelo de características como artefacto principal para especificar los requisitos de la familia de productos. El objetivo durante la ingeniería del dominio de FeDRE es realizar la especificación de requisitos de forma que se utilicen sistemáticamente las características del dominio de la LPS para identificar y definir los requisitos de la LPS. Se proveen un conjunto de guías, presentadas en el apéndice A.1, que asistirán al ingeniero de requisitos para realizar esta definición y especificación de los requisitos. Estos requisitos se representan mediante diagramas de casos de uso y diagramas de actividad. Una vez se validan los modelos de la especificación de requisitos del dominio el ingeniero de requisitos definirá la variabilidad de los mismos.

Por otra parte durante la *ingeniería de la aplicación* el objetivo es derivar los requisitos para un producto concreto, resolviendo la variabilidad prevista de los requisitos de la familia de productos, a partir de una configuración del producto. Esta especificación derivada se refinará, si fuera necesario, identificando y definiendo los requisitos delta que sean necesarios. Para ello se deben de aplicar las guías de la ingeniería de la aplicación presentadas en el apéndice A.2.

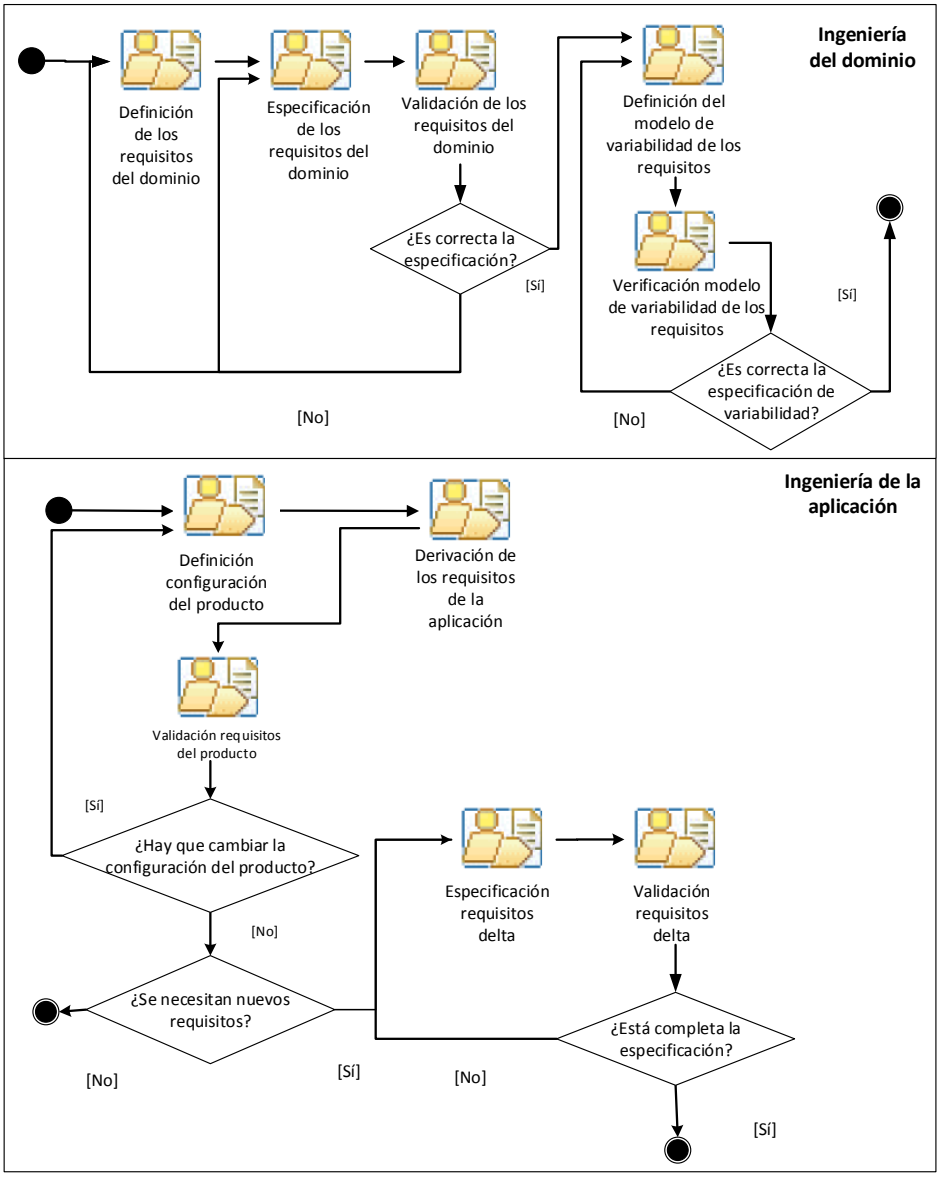


Figura 4.8 Proceso global de FeDRE

4.2 La actividad de *scoping*

Previamente a la presentación de los procesos de ingeniería del dominio y la ingeniería de la aplicación, se presentarán los artefactos que se toman como entrada en FeDRE. Estos artefactos se definen en la actividad *scoping*, donde se definen los límites de la LPS, los productos que estarán incluidos en ella y las características incluidas en cada producto. FeDRE no ofrece soporte específico a la actividad de *scoping*, sin embargo durante la ingeniería del dominio, se emplean artefactos resultantes de la ejecución de esta actividad como punto de partida del proceso. La Figura 4.9 muestra las entradas, salidas y roles que participan en la actividad de *scoping*.

Figura 4.9 Actividad de scoping

En esta actividad participan tres roles principales:

- *Analista del dominio*. Es un experto en la reutilización y participa en el establecimiento de los límites de la LPS (Oliveira y otros 2014).
- *Experto del dominio*. Posee conocimiento técnico del dominio. Este conocimiento se utiliza en el desarrollo de los productos desde un punto de vista técnico (van der Linden y otros 2007).
- *Experto de marketing*. Posee conocimiento de mercado sobre el dominio. Este conocimiento lo utiliza para dar su punto de vista en el desarrollo de los productos (Oliveira y otros 2014).

A continuación se describen cada uno de los artefactos utilizados en la actividad de *scoping*: *assets* existentes, modelo de características, especificación de características y mapa de productos.

4.2.1 Assets existentes

Este artefacto se utilizará de forma opcional dado que dependerá del tipo de estrategia de desarrollo de la LPS (Sección 2.1.2.4). Por ejemplo en una estrategia de desarrollo proactiva no tendrá sentido este artefacto dado que la LPS comienza desde cero. En estrategias de desarrollo extractivas o reactivas los *assets* existentes pueden servir como buen comienzo para desarrollar los componentes en la arquitectura referencia de la LPS. Este artefacto también puede utilizarse como fuente de información para documentar los *assets* existentes se utilizarán los sistemas legados y los productos ya existentes en el dominio. La arquitectura referencia de la LPS que se construirá puede tomar elementos de diseños o productos ya existentes en la compañía. El analista del dominio documentará los *assets* existentes con el objetivo de tener disponible todos los componentes, librerías, marcos tecnológicos, algoritmos, herramientas, componentes y servicios que se pueden utilizar durante todo el proceso de LPS.

4.2.2 Modelo de características

El modelo de características se utiliza para capturar, organizar y visualizar características de una LPS representado propiedades comunes y variables del sistema. De acuerdo con Kang y otros (1998) este modelo identifica características, variaciones y restricciones entre las características de la LPS. El analista del dominio y el experto de marketing identifican características utilizando los *assets* existentes de la LPS como entrada o utilizando la información de los expertos y del experto del mercado. La Tabla 4.1 muestra la leyenda utilizada en el modelo de características. La Figura 4.10 muestra un ejemplo de modelo de características para expresar la variabilidad de la LPS Savi

Tabla 4.1 Leyenda del modelo de variabilidad

Símbolo	Leyenda
●	Característica obligatoria
○	Característica opcional
▲	Grupo alternativo OR (una o más de una característica se pueden seleccionar)
⩓	Grupo alternativo XOR (sólo se puede seleccionar una característica)

4.2.3 Especificación de las características

El analista del dominio es responsable de especificar las características utilizando la plantilla de especificación de características. En esta plantilla se captura la información de las características y a mantener la trazabilidad con todos los artefactos implicados.

Mostramos un ejemplo de esta plantilla en la Tabla 4.2 siguiendo con el ejemplo de la LPS Savi. De acuerdo con la plantilla cada característica tiene un identificador único y un nombre. Para definir el tipo de *variabilidad* de la característica se utiliza la clasificación de Kang y otros (1990): obligatoria, opcional o alternativa. La prioridad de la característica puede ser alta, media o baja. Si la característica requiere o excluye otra característica se debe de especificar el id de la característica relacionada. Si la característica tiene una característica padre se ha de escribir su identificador en el campo respectivo. En tiempo de compilación se puede seleccionar entre tiempo de compilación o tiempo de ejecución de acuerdo con el tiempo en el que se incluirá la característica en el producto (Czarnecki y Eisenecker 2000). El tipo de característica puede ser concreto o abstracto. Por último la descripción es una breve explicación de la característica.

Tabla 4.2 Ejemplo de especificación de la característica control de acceso

[F002] Control de acceso			
Prioridad:	Alta	Variabilidad:	Opcional
Requiere:	-	Excluye:	-
Tiempo de compilación	Compilar	Característica padre:	001 Savi
Característica hija:	F002, F003	Tipo:	Abstracta
Descripción:		El usuario puede crear una cuenta en el sistema utilizando el formulario de registro o importando sus datos desde una cuenta de una red social (Twitter, Facebook). Si el usuario olvida la contraseña entonces puede requerir una nueva. En este caso el sistema envía un e-mail con la nueva contraseña. Una vez el usuario está registrando puede entrar en el sistema introduciendo su nombre de usuario y contraseña. Además podrá modificar su perfil.	

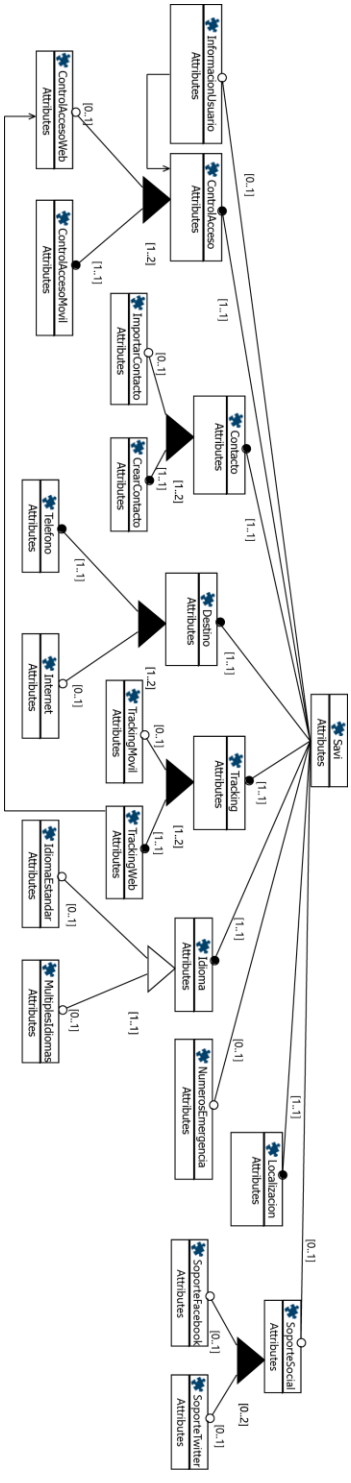


Figura 4.10 Ejemplo de modelo de características

4.2.4 Mapa del producto

El mapa de productos es un artefacto que relaciona las características con los diferentes productos. Un mapa de producto es una tabla que lista las características que implementa un producto en las filas y la lista de productos como columnas. Las celdas de la tabla contienen una cruz en el caso de que un producto contenga una característica.

La Tabla 4.3 muestra un fragmento del mapa de productos de la LPS Savi. En esta LPS se han incluido inicialmente cinco productos:

- *Savi Lite*. Es la versión más simple de todos los productos disponibles. Cubre la funcionalidad básica. Este producto permite enviar notificaciones de emergencia por SMS.
- *Savi Standard*. Este producto añade a Savi Little la opción de importar contactos desde el teléfono y mostrar una lista de números de emergencia.
- *Savi Social*. Este producto añade funcionalidad "social" a la aplicación como: importar contactos o enviar notificaciones de emergencia desde Facebook y Twitter.
- *Savi Pro*. Añade la opción de rastrear la posición del teléfono.
- *Savi Ultimate*. Es la versión más completa de la familia de productos.

Tabla 4.3 Fragmento del mapa del producto para la LPS Savi

Características / productos	Savi Lite	Savi Standard	Savi Social	Savi Pro	Savi Ultimate
Información usuario					X
Control de acceso	X	X	X	X	X
Control de acceso Web					X
Control de acceso móvil	X	X	X	X	X
Contacto	X	X	X	X	X
Añadir contacto	X	X	X	X	X
Importar contacto			X	X	X
Importar móvil		X	X	X	X
Destino	X	X	X	X	X
Teléfono	X	X	X	X	X
Internet			X	X	X
Tracking				X	X
Tracking móvil				X	X
Tracking Web					X
Idioma	X	X	X	X	X
Idioma estándar	X	X	X	X	X
Múltiples idiomas		X	X	X	X
Números de emergencia		X	X	X	X
Localización	X	X	X	X	X
Soporte social			X	X	X
Soporte Facebook			X	X	X
Soporte Twitter			X	X	X

4.3 Ingeniería del dominio

En la ingeniería del dominio se definen y especifican los requisitos de la LPS. Estos requisitos permiten implementar la funcionalidad de las características y productos que se definieron durante la actividad del *scoping*. FeDRE va a soportar el proceso de ingeniera de requisitos mediante un conjunto de guías para que el *ingeniero de requisitos del dominio* pueda identificar de una forma iterativa e incremental los requisitos funcionales de la familia de productos. Estas guías pueden consultarse en el apéndice A.1.

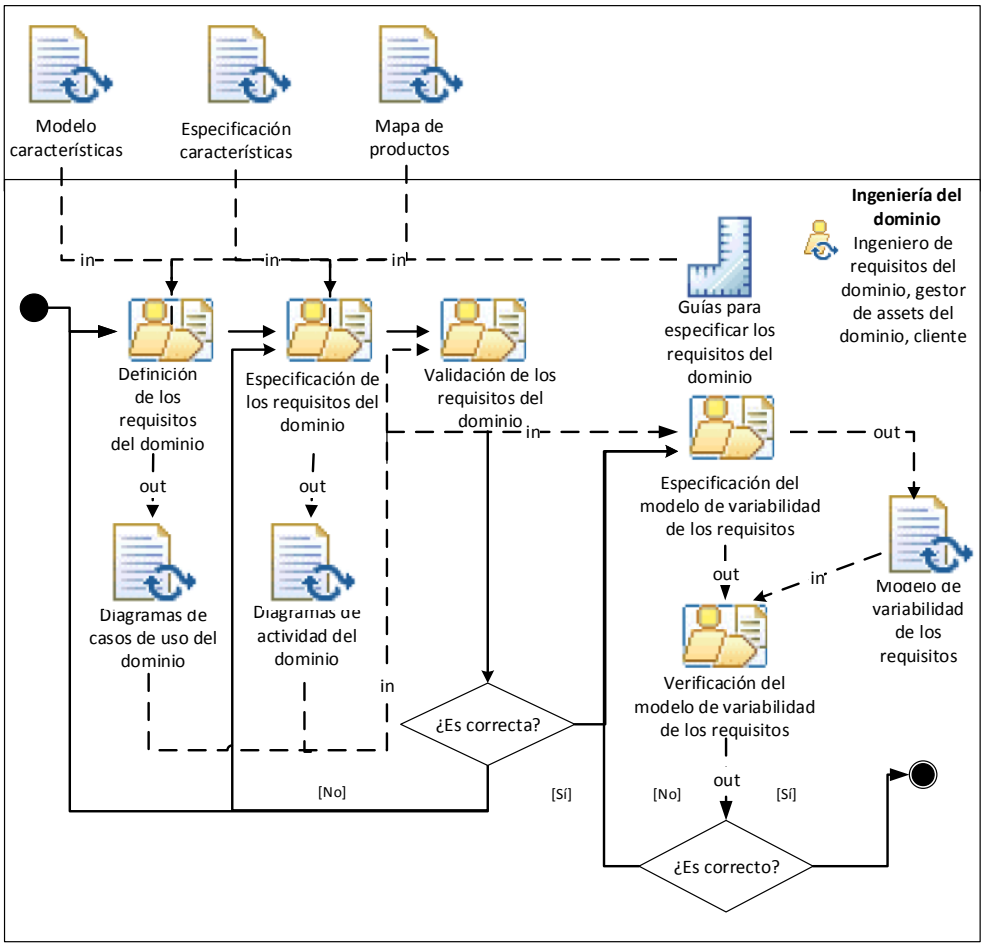


Figura 4.11 Proceso de la ingeniería del dominio

Un proceso tradicional de IR finalizaría en este punto. Sin embargo en una LPS una vez definición y especificación los requisitos del dominio, es preciso especificar la variabilidad. Cuando se identifica o refina un requisito es necesario determinar si ese requisito se va a compartir entre otros productos de la LPS o si por el contrario se va a utilizar únicamente en un producto. Los requisitos compartidos se deben clasificar en comunes, opcionales y variables. Los *comunes* se utilizan en cualquier producto de la LPS, los comunes pueden utilizase en algunos productos y otros no, mientras que los *variables* deben de configurarse o parametrizarse en la especificación de variabilidad de la LPS. Adicionalmente algunos requisitos pueden requerir o excluir otros requisitos o pueden restringirse en algunas configuraciones de productos.

Para llevar a cabo estas actividades van a participar los siguientes roles:

- *Ingeniero de requisitos del dominio*. El ingeniero de requisitos del dominio es responsable del desarrollo y mantenimiento de todos los requisitos que son relevantes en el rango completo de productos (van der Linden y otros 2007). Es responsable de realizar la definición y especificación de requisitos a partir del modelo y la especificación de características.
- *Gestor de productos*. Es el encargado de planear los productos actuales y presentes que forman parte de la LPS (van der Linden y otros 2007). Es el responsable de identificar la variabilidad en los modelos de requisitos y definir un modelo que represente la variabilidad de los mismos. Durante la IR asiste al ingeniero de requisitos para definir la variabilidad de los requisitos.
- *Cliente*. Participa en la validación de la especificación de requisitos con el objetivo de comprobar que satisface los requerimientos de la LPS.

4.3.1 Definición de los requisitos del dominio

La definición de los requisitos de la ingeniería del dominio se lleva a cabo de una manera iterativa e incremental. Los conjuntos de características seleccionadas del modelo de características pueden definirse en unidades de *incremento de la especificación*. Permitiendo seleccionar incrementos a especificar hasta completar la especificación de requisitos. Esta actividad utiliza el modelo de características, la especificación de características y el mapa de productos como artefactos de entrada y produce el glosario, requisitos funcionales y la matriz de trazabilidad. En esta actividad el ingeniero del dominio aplicará las guías del dominio propuestas en el apéndice A.1.

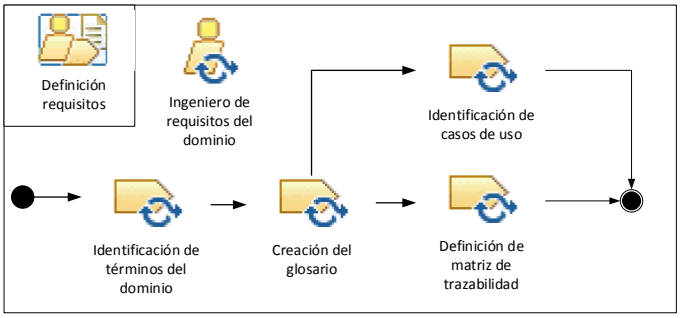


Figura 4.12 Tareas que componen la definición de requisitos del dominio

A continuación se describen cada uno de los artefactos que se producen como salida de esta actividad.

4.3.1.1 Glosario

Un rasgo distintivo del desarrollo LPS es la presencia de múltiples expertos de dominio y desarrolladores. Para facilitar la comunicación entre estos múltiples participantes es necesario tener un vocabulario común para describir los conceptos relevantes del dominio. El glosario describe y explica los principales términos del dominio con el objetivo de proveer un vocabulario común con el objetivo de evitar ambigüedades. Se presenta como una tabla de dos columnas que contiene el nombre del término y su descripción en lenguaje natural. La Tabla 4.4 muestra un fragmento de la LPS Savi.

Tabla 4.4 Fragmento del glosario de la LPS Savi

Término	Descripción
Facebook	Web social. Este servicio permite enviar mensajes privados a sus contactos
Protocolo de alerta común	El Formato de Alerta Común (FAC) es un formado de datos baso en XML para intercambiar alertas y emergencias entre diferentes dispositivos de alerta. El FAC permite que los avisos de emergencia sean interoperables entre diferentes sistemas de notificación de emergencias.
SMS	Acrónimo del inglés *Short Message Service* (SMS). Servicio de telefonía que permite el intercambio de pequeños mensaje de texto.
Twitter	Servicio web de *microblogging* (servicio que permite enviar textos breves). Permite enviar pequeños mensajes de texto a los usuarios suscritos a la cuenta del usuario

4.3.1.2 Diagramas de casos de uso

Los diagramas de caso de uso (OMG 2007) contienen los requisitos identificados (comunes o variables) para la familia de productos que constituyen la LPS. Los *casos de uso* se usan para especificar los requisitos de la LPS (cada requisito funcional se especificar como caso de uso). Para ejemplificar la notación utilizada se utilizarán dos diagramas de casos de uso correspondientes a las características control de acceso (Figura 4.13) y control de acceso móvil (Figura 4.14).

El elemento principal es el *caso de uso*, el cual especifica una secuencia de acciones, incluyendo variantes, que el sistema puede ejecutar y que produce un resultado observable de valor para un particular actor" (OMG 2007).

Otro concepto de los casos de uso es el *actor*, el cual se utiliza para representar un rol jugado por un usuario o cualquier otro sistema que interactúa con el sujeto (OMG 2007). En los casos que el actor representa a un sistema externo se le debe de etiquetar con el estereotipo «*system*». También sería posible añadir

relaciones de generalización entre actores. La Figura 4.14 muestra un ejemplo donde el actor usuario identificado especializa al actor usuario. Por otra parte en el diagrama de casos de uso para el control de acceso móvil encontramos dos ejemplos de sistemas externos marcados con el estereotipo «*system*»: Twitter y Facebook (Figura 4.14).

Por otra parte los casos de uso pueden relacionarse entre ellos con el tipo de relación «*include*». Este tipo de relación se utilizado cuando un caso de uso llama a otro caso de uso. En la Figura 4.13 se muestra una llamada *include* desde el caso de uso recordar contraseña al caso de uso enviar e-mail.

Otro tipo de relación entre casos de uso son los «*extend*». Este tipo de relaciones se utilizan originalmente para indicar que un caso de uso puede entender a otro. El caso de uso extensión se inserta en el caso de uso extendido siempre que se cumpla una condición del caso de uso base. Gráficamente se representa con una flecha de punta abierta con trazo discontinuo desde el caso de uso extensión al caso de uso base etiquetado con la palabra «*extend*». En la Figura 4.14 se aplican sendas relaciones «*extend*» para representar la variabilidad. En este caso el uso base es *crear usuario*. Este caso de uso se ejecuta por defecto creando un usuario introduciendo sus datos. Sin embargo de forma alternativa podría crearse un usuario mediante la vinculación de su cuenta de twitter (caso de uso *vincular Twitter*) o, de forma alternativa, mediante la vinculación de su cuenta de Facebook (caso de uso *vincular Facebook*). Estos dos casos de uso con los casos de uso entendedores y se ejecutarán si se cumpla la condición de que las características *soporte Twitter* y *soporte de acceso Facebook* estén respectivamente incluidas en la configuración de un producto.

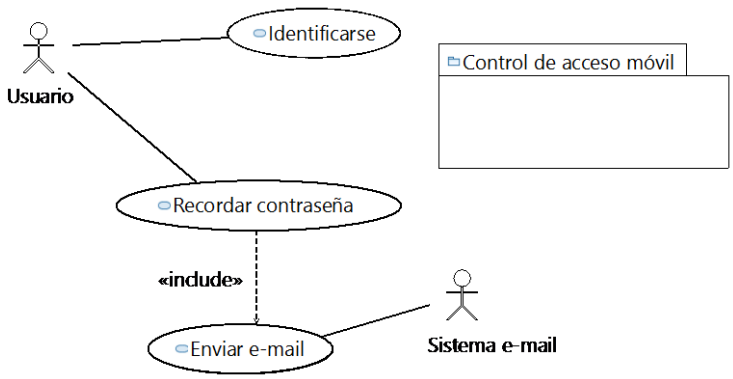


Figura 4.13 Diagrama de casos de uso para la característica control de acceso

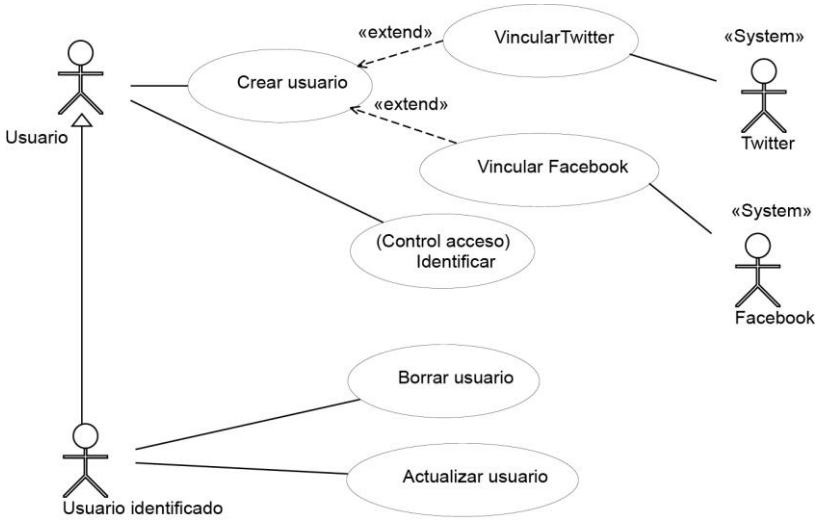


Figura 4.14 Diagrama de caso de uso para la característica control de acceso móvil

4.3.1.3 Matriz de trazabilidad

La matriz de trazabilidad es una matriz que contiene los enlaces entre las características y los requisitos funcionales. Las filas de la matriz muestran las características y las columnas muestran los requisitos funcionales (caso de uso). Esta matriz es útil para ayudar con el mantenimiento de los requisitos. En la Tabla 4.5 se muestra un fragmento de la matriz de trazabilidad para los casos de uso identificados en la subsección anterior.

Tabla 4.5 Fragmento de matriz de trazabilidad para la LPS Savi

Caso de uso / característica	Control de acceso	Control de acceso Web	Control acceso móvil	Soporte Facebook	Soporte Twitter
UC-01 Recordar contraseña	X	X	X		
UC-02 Enviar e-mail	X	X	X		
UC-03 Identificarse	X	X	X		
UC-04 Crear usuario	X		X		
UC-05 Vincular Twitter					X
UC-06 Vincular Facebook				X	

4.3.2 Especificación de los requisitos del dominio

Una vez definidos los requisitos mediante los casos de uso la siguiente actividad es especificar estos requisitos y su variabilidad. De forma similar a la definición de requisitos el ingeniero de requisitos del dominio va a aplicar las guías del dominio para especificar de forma iterativa e incrementa cada uno de los casos de uso. Una vez especificados cada uno de los casos de uso se completa la matriz de trazabilidad.

La representación de la especificación de los requisitos se lleva a cabo con diagramas de actividad UML (OMG 2007), los cuales representan flujos de actividades con decisiones, iteraciones y concurrencia. Se utilizarán diagramas de actividad con particiones. Las particiones dividen nodos y artistas que restringen y muestran una vista de los nodos contenidos (OMG 2007).

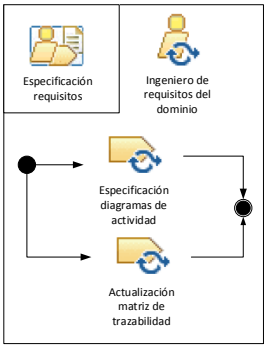


Figura 4.15 Especificación de requisitos del dominio

4.3.2.1 Los diagramas de actividad

La Tabla 4.6 muestra la notación de los elementos de un diagrama de actividad. Un flujo de acción parte de un *nodo inicial* y se conecta mediante diversos conectores llamados *flujos de acción* hasta un *nodo final*. Los *nodos de decisión* representan nodos de control (OMG 2007) donde se debe de elegir entro uno de los flujos de salida. Los nodos de decisión permiten expresas múltiples alternativas que se utilizan para representar la variabilidad.

Tabla 4.6 Notación del diagrama de actividad

Elemento	Representación
Nodo inicial	●
Nodo final	◉
Nodo de decisión / nodo de unión	◊
Acción	▢
Flujo de control	▢→▢

En FeDRE cada actor tiene una *partición de actividad* que representa el flujo de acción. El sistema tendrá siempre una partición de actividad donde se modela la interacción entre diferentes actores y el sistema.

Otro elemento de interés en un diagrama de actividad son los *nodos de decisión*. En un nodo de decisión parten varios flujos de salida dependiendo de una *guarda*. Estas guardas se asocian a una condición. Volviendo al ejemplo de Savi, la Figura 4.17 muestra el diagrama de actividad utilizado para especificar el caso de uso *identificarse*. De acuerdo con este diagrama el flujo de datos comienza en la partición de actividad del usuario. En cada nodo de decisión existen múltiples flujos de acción de salida dependiendo de una guarda. En la Figura 4.17 podemos observar un nodo de decisión con tres guardas. Si se realiza la identificación mediante el móvil, el cual es el caso por defecto, en ese caso el flujo de datos ejecuta la comprobación de credenciales con el móvil. Alternativamente puede realizarse la comprobación en el sistema de las credenciales de Twitter o realizar la comprobación de credenciales en Facebook. Una vez ha finalizado el flujo de acción correspondiente se utiliza un *nodo de unión* para volver a un único flujo de acción.

4.3.3 Validación de los requisitos del dominio

Una vez se han definidos y especificado los requisitos del dominio es necesario validar con el cliente si esta especificación del dominio satisface las necesidades de la LPS y es correcta. Esta actividad toma como entrada el glosario, la especificación de requisitos del dominio (diagramas de caso de uso del dominio y diagramas de actividad). La Figura 4.16 muestra las tareas que forman parte de la actividad de validación.

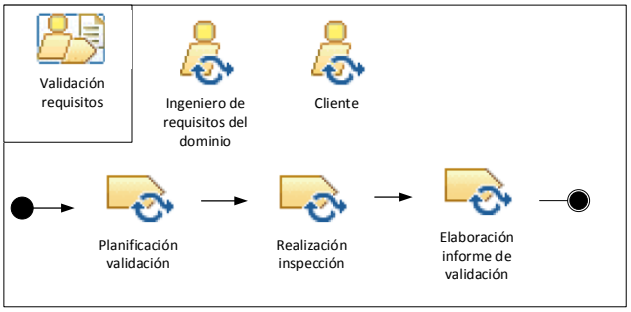


Figura 4.16 Validación de los requisitos del dominio

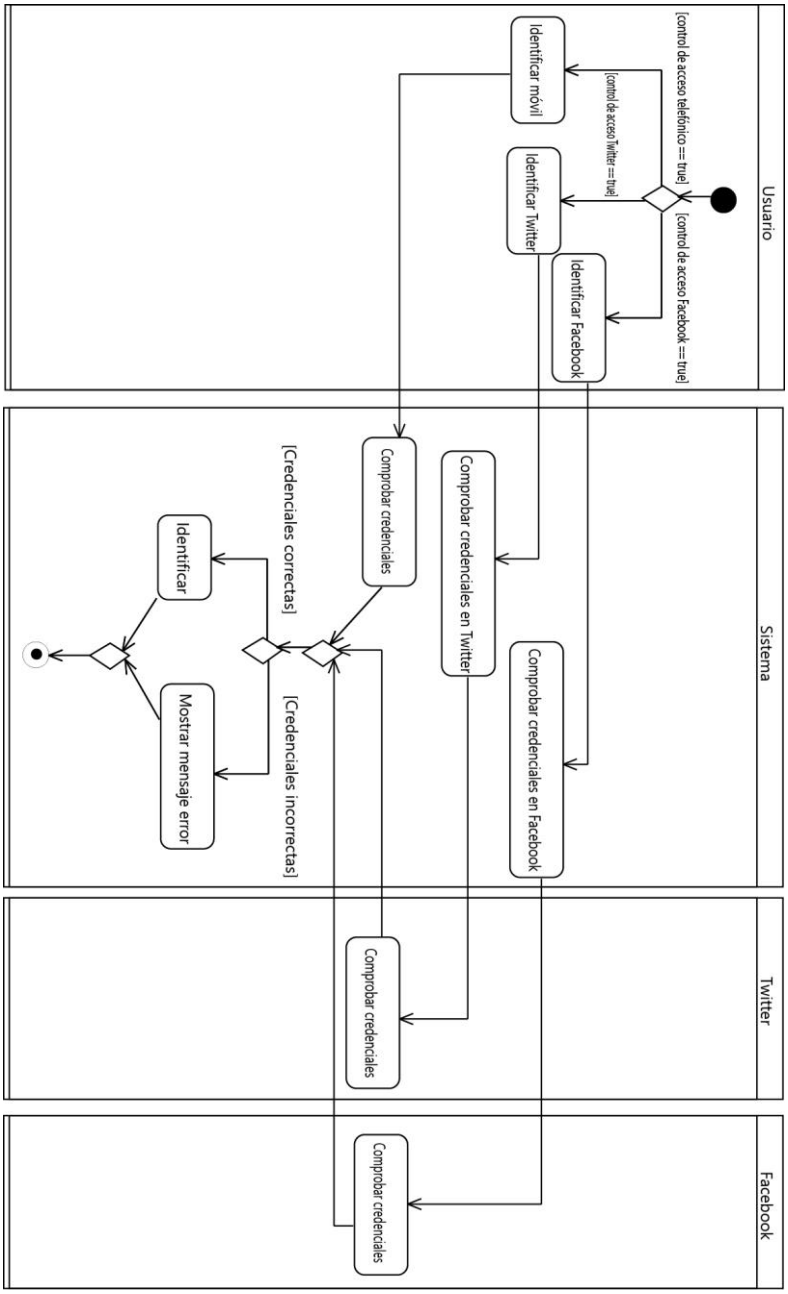


Figura 4.17 Diagrama de actividad para especificar el caso de uso identificarse

Se empleará la técnica de inspección de requisitos para realizar una examinación organizada de los requisitos. Esta técnica fue propuesta inicialmente por Michael Fagan en IBM en 1972 y ha sido mejorada posteriormente (Fagan 2001). Primero, se realiza la *planificación de la validación* donde se propone una fecha de inspección, los artefactos a validar y el número de versión de cada uno de ellos. Tras una reunión entre el ingeniero de requisitos del dominio y el cliente *realizarán la inspección* en reuniones de trabajo para analizar la especificación de requisitos del dominio. Se *elaborará un informe de validación* donde se documenta para cada artefacto analizado las posibles deficiencias a resolver. La Tabla 4.7 muestra un ejemplo de la plantilla de inspección para la evaluación de un diagrama de actividad. El informe generado de validación contiene unos campos con información sobre la inspección (fecha, responsable de la inspección, artefacto observado y número de versión del artefacto) y una relación de los defectos encontrados en la especificación.

Tabla 4.7 Ejemplo de plantilla de inspección

Campo	Valor	
Fecha de la inspección	15/06/2015	
Responsable	David Blanes	
Artefacto revisado	Diagrama de actividad ACT-03	
Versión del artefacto	3	
Defectos encontrados	Localización	Comentario
	Nodo de actividad 02	No se ha especificado la condición de guarda de los flujos entrantes
	Actividad 05	Recomendable cambiar el nombre de la actividad para evitar ambigüedades

4.3.4 Definición del modelo de variabilidad de los requisitos

Los modelos de características se han empleado exitosamente para modelar la variabilidad de una LPS. Sin embargo la variabilidad representada en el modelo de características representa únicamente la variabilidad externa de la LPS. Sin embargo la variabilidad en una LPS estará presente diferentes niveles de abstracción (Babar y Shull 2010): requisitos, descripción de la arquitectura, documentos de diseño, código fuente, código complicado, código enlazado y sistema en ejecución. Esta variabilidad en diferentes artefactos y diferentes etapas del desarrollo motiva a representar la variabilidad en cada uno de los artefactos del proceso de desarrollo de forma separada. Desacoplar el modelo de variabilidad de los artefactos de requisitos hace que pueda mantenerse de una forma consistente, que no sea necesario ni representar información de variabilidad en el modelo de requisitos ni información de trazabilidad hacia los requisitos en el modelo de variabilidad.

La actividad de definición del modelo de variabilidad de los requisitos está compuesta por dos tareas tal como muestra la Figura 4.18. Primero se *identifica la variabilidad* analizando los modelos de casos de uso y los diagramas de actividad. Los puntos de variabilidad se han de recoger en el modelo llamado *modelo de variabilidad de requisitos*.

Durante el *análisis de la variabilidad* de requisitos se identifica la variabilidad de los requisitos y se definen los puntos de variabilidad y sus variantes respecto a los requisitos (Pohl y otros 2005). Los requisitos que difieren entre ellos indican la necesidad de introducir un punto de variabilidad y sus variantes. Para extraer la información de variabilidad de requisitos se examinan los requisitos de la LPS por parte del *ingeniero de requisitos del dominio*. El análisis revela qué requisitos son únicos y cuales difieren dependiendo del producto. Como resultado del análisis de variabilidad, se definen los puntos de variabilidad y las variantes. Para realizar este análisis de variabilidad a nivel de requisitos se utilizan técnicas como el análisis de variabilidad basado en prioridad, o el análisis de variabilidad basado en listas de comprobación (Pohl y otros 2005).

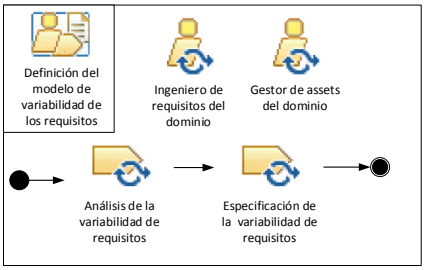


Figura 4.18 Tareas de la definición del modelo de variabilidad de los requisitos

Volviendo al ejemplo de la LPS Savi, en el diagrama de casos de uso para especificar el control de acceso móvil (Figura 4.14) podemos identificar un punto de variabilidad para la *creación del usuario*. Concretamente se identifican las siguientes variantes:

- *Crear usuario*. En el caso de uso base se crea un usuario en el sistema.
- *Vincular Twitter*. En este caso se crea el usuario utilizando la cuenta de la red social Twiter.
- *Vincular Facebook*. En este caso se crea un usuario utilizando las credenciales de la red social Facebook.

Además del ingeniero de requisitos del dominio, en esta tarea participará el *gestor de productos*. El gestor de productos es responsable de decir que parte de la variabilidad de la LPS se ofrece al cliente como variabilidad externa (Pohl y otros

2005). Este conocimiento se utiliza durante esta tarea por parte del gestor de productos para determinar la relación de trazabilidad y los puntos de variabilidad de requisitos. El gestor de productos adicionalmente realiza otras funciones como definir si deber añadirse o modificarse puntos de variabilidad (Pohl y otros 2005).

Una vez definidos los puntos de variabilidad a nivel de requisitos y las variantes posibles para cada uno de ellos, se debe de *especificar el modelo de variabilidad de los requisitos* para representar esta información. El *ingeniero de requisitos del dominio* es el encargado de escoger el lenguaje de representación de la variabilidad de los requisitos adecuado para el problema. En FeDRE no se propone ninguna notación específica siendo el desarrollador libre de escoger la notación que más le convenga. Algunas alternativas posibles son utilizar el modelo de variabilidad ortogonal conocido como *Orthogonal Variability Modelling* (OVM) (Pohl y otros 2005). OVM es un lenguaje simple para superponer la variabilidad sobre cualquier artefacto de desarrollo en un modelo aparte de forma que no interfiere en sus contenidos. Otra alternativa es el *Modelado Delta*, una metodología que expande los lenguajes existentes sin interferir en sus contenidos (Schaefer y otros 2010). El Modelado Delta extiende lenguajes existentes con puntos de variación ejecutables estáticamente. Otra alternativa para representar el modelo de variabilidad es el *Common Variability Language* (CVL) (OMG 2012). CVL es un lenguaje genérico que permite separar el modelado de variabilidad para cualquier LED que haya sido definido utilizando un lenguaje basado en MOF (OMG 2006a). Por último también se puede utilizar un LED que permita expresar la variabilidad. Una vez se selecciona una notación concreta para representar el modelo de variabilidad de requisitos el experto de reutilización define el modelo de variabilidad de requisitos, dando por finalizada la especificación de requisitos.

4.3.5 Verificación del modelo de variabilidad de los requisitos

Una vez se ha definido el modelo de variabilidad de requisitos debe de realizarse una verificación para comprobar que el modelo de variabilidad de requisitos es correcto. En esta actividad participarán el ingeniero de requisitos del dominio y el experto de reutilización. La Figura 4.19 muestra las tareas que componen esta actividad. Primero, se *diseña la verificación* estableciendo los objetivos de la misma y eligiendo la técnica deseada para realizarla. A continuación se debe de *ejecutar la verificación* con la técnica seleccionada: comprobación de modelos o *model checking*, comprobación de satisfacibilidad o *model satisfiability*, etc.

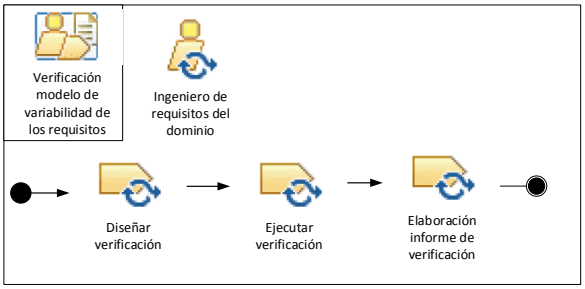


Figura 4.19 Tareas de la verificación del modelo de variabilidad de los requisitos

Por último se realiza una reunión para analizar los resultados de la verificación y *elaborar un informe*. Si existen fallos se debe iterar y volver a definir el modelo de variabilidad de los requisitos, en caso contrario se finaliza quedando completada la especificación de los requisitos y la variabilidad en la ingeniería del dominio.

4.4 Ingeniería de la aplicación

En la ingeniería de la aplicación se derivaran los requisitos para un producto específico tomando como entrada la especificación de requisitos del dominio y la configuración del producto. La Figura 4.20 muestra el proceso para la ingeniería de aplicación de FeDRE. Este proceso se desarrolla de una forma iterativa e incremental y se divide en tres actividades principales:

1. *Derivación de los requisitos del producto*. Los requisitos se obtienen a partir de la configuración del producto que se defina.
2. *Validación de los requisitos del producto*. Se comprueba que la especificación de los requisitos satisface las necesidades del cliente. Podría ser necesario añadir o quitar requisitos del dominio y como consecuencia de debería volver a realizar la derivación de los requisitos del producto o bien especificar requisitos delta hasta completarla.
3. *Especificación de los requisitos delta*. En los casos que es necesario añadir nuevos requisitos delta para completar la especificación se seguirán las reglas propuestas en FeDRE para la ingeniería de la aplicación.

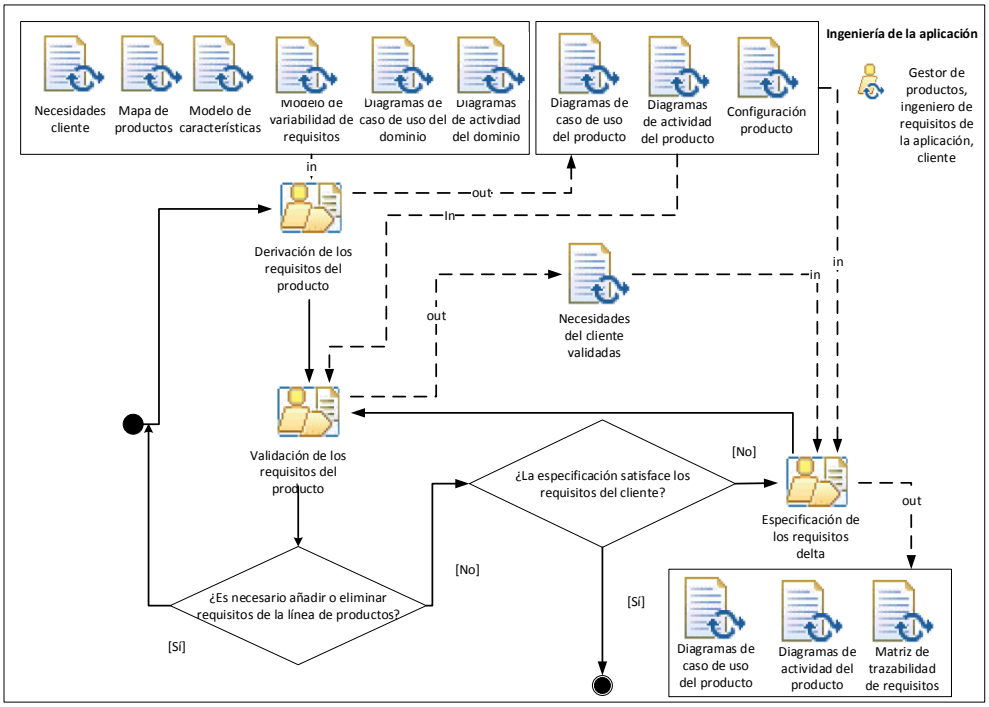


Figura 4.20 Proceso de la ingeniería de aplicación de FeDRE

En el proceso de ingeniería de la aplicación participan tres roles:

- *Ingeniero de requisitos de la aplicación*. Encargado de derivar los requisitos del producto y definir y especificar los requisitos delta (van der Linden y otros 2007) a partir del documento de necesidades del cliente..
- *Gestor de de productos*. Es el responsable de la IR como un todo así como de los productos individuales (van der Linden y otros 2007).
- *Cliente*. Encargado de validar los requisitos del producto para comprobar que satisfacen sus necesidades.

4.4.1 Derivación de los requisitos del producto

En esta actividad se toman seis modelos de entrada desde la ingeniería del dominio: necesidades del cliente, mapa de productos, modelo de características, modelo de variabilidad de los requisitos, el diagrama de casos de uso del dominio y los diagramas de actividad del dominio tal como muestra la Figura 4.21.

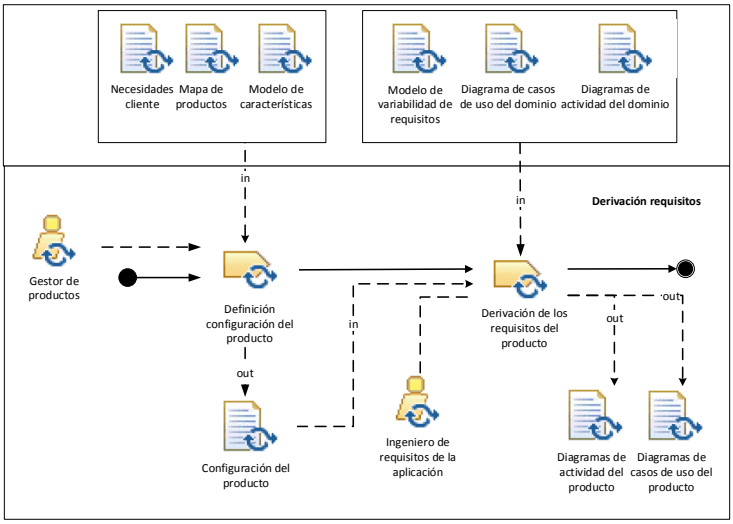


Figura 4.21 Tareas de la derivación de requisitos el producto

El *ingeniero de requisitos de la aplicación* primero *define una configuración del producto* basándose en el modelo de características, las necesidades del cliente y el mapa de productos donde produce como resultado un modelo de configuración del producto. Una *configuración del producto* se define como una selección válida de características que es fija y finita y que define un producto (Voelter y Visser 2011). Esta tarea es llevaba a cabo por el rol *gestor de productos*, el cual conoce los productos de la LPS y decidirá la configuración más adecuada para maximizar la reutilización cubriendo las necesidades del cliente para el producto concreto.

Volviendo al ejemplo de la LPS Savi la Figura 4.22 muestra un fragmento de una configuración de un producto para la LPS. Esta configuración selecciona las características: soporte social y soporte Facebook. Este modelo de configuración de producto se utiliza como entrada para la *tarea de derivación de los requisitos del producto* junto a la especificación de requisitos del dominio (diagramas de casos de uso y diagramas de actividad) y el modelo de variabilidad de los requisitos.

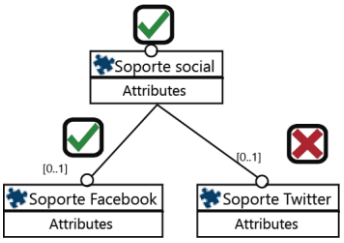


Figura 4.22 Fragmento de la configuración para un producto

Durante la derivación de los requisitos del producto se realiza una *resolución de la variabilidad*. La resolución de variabilidad es el proceso de resolver una característica a una elección posible (variación) entre un conjunto de elecciones posibles (variaciones) (Kulkarni y otros 2012). Existen dos estrategias posibles para resolver la variabilidad de los requisitos (Schaefer 2010): positiva o negativa. Una *variabilidad negativa* parte de un modelo para todos los productos de la LPS que se aumenta con información de variabilidad para determinar qué elementos del modelo estarán presentes en los productos. Por otra parte en una estrategia de *variabilidad positiva* se asocian fragmentos de modelos con características y se componen para una configuración de producto dada. .Por otra parte la resolución de variabilidad

Volviendo al ejemplo de la LPS Savi vamos a ejemplificar la materialización de la variabilidad utilizando una estrategia de variabilidad negativa. El diagrama de casos de uso del dominio para la característica control de acceso de la Figura 4.13 contiene toda la funcionalidad visible. Sin embargo en la configuración del producto no se selecciona la característica de soporte social de Twitter. Esto implica que en la resolución de la variabilidad tenemos que suprimir: el actor Twitter, el caso de uso vincular Twitter y la relación "*extend from*" entre ambos tal como muestra la Figura 4.23.

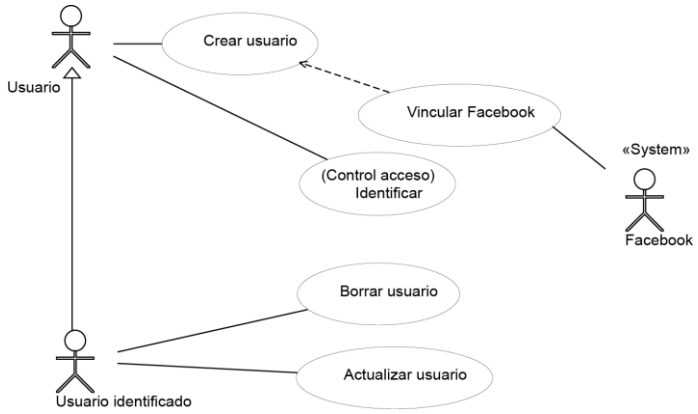


Figura 4.23 Diagrama de casos de uso derivado para la característica control de acceso móvil

De forma similar es preciso resolver los puntos de variabilidad en los diagramas de actividad. Para resolver los puntos de variabilidad en los diagramas de actividad se resuelven las alternativas en los nodos de decisión. Los nodos de decisión se representan gráficamente con un diamante que tiene un flujo de entrada y tiene varios flujos de acción alternativos. Cada uno de estos flujos se activará si se cumple la condición de guarda, la cual se indica en el diagrama como una condición lógica entre corchetes.

En este ejemplo utilizamos también una estrategia de resolución de la variabilidad negativa. La Figura 4.25 muestra el diagrama de actividad derivado para especificar el caso de uso identificar En este ejemplo se incluyen tres particiones de actividad: una para el actor principal que es el usuario, una para el sistema -que siempre es obligatorio incluir en FeDRE- y otra más para representar al actor externo Facebook. Al materializar la variabilidad se debe de suprimir la partición para el sistema externo Twitter dado que no se ha seleccionado la característica de *control de acceso Twitter* en la configuración del producto.

4.4.2 Validación de los requisitos del producto

La validación de requisitos es el proceso de evaluar un entregable de un proyecto para determinar si satisface las necesidades del cliente (Wiegers y Beatty 2013). La validación de requisitos implica comprobar si las entradas, actividades realizadas y los artefactos de los requisitos creados como salidas de la especificación de requisitos satisfacen un determinado criterio. Durante esta actividad se comprobará el grado de satisfacción de las necesidades del cliente. Las *necesidades del cliente* de un producto las define el propio cliente, utilizando sus mismas palabras, para expresar los beneficios que quiere obtener de un producto (Hauser 1993). El cliente puede identificar para un producto usualmente entre 100 y 200 necesidades, incluyendo necesidades básicas (qué se asume que hace el producto), necesidades articuladas (qué se dice que tiene que hacer el producto) y necesidades estimuladas (las que en caso de satisfacerse dejarían muy satisfechos y sorprendidos a los clientes) (Hauser 1993). Estas necesidades del cliente se toman de entrada en la tarea validar las necesidades del cliente en el proceso que muestra la Figura 4.24.

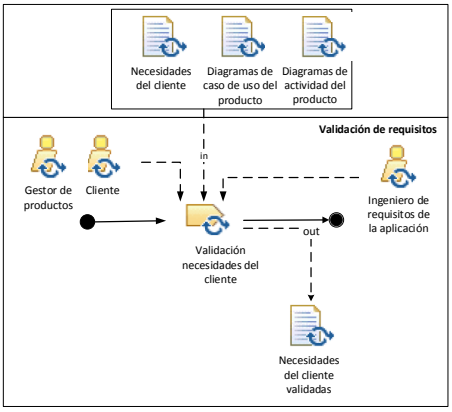


Figura 4.24 Tareas de la validación de los requisitos del producto

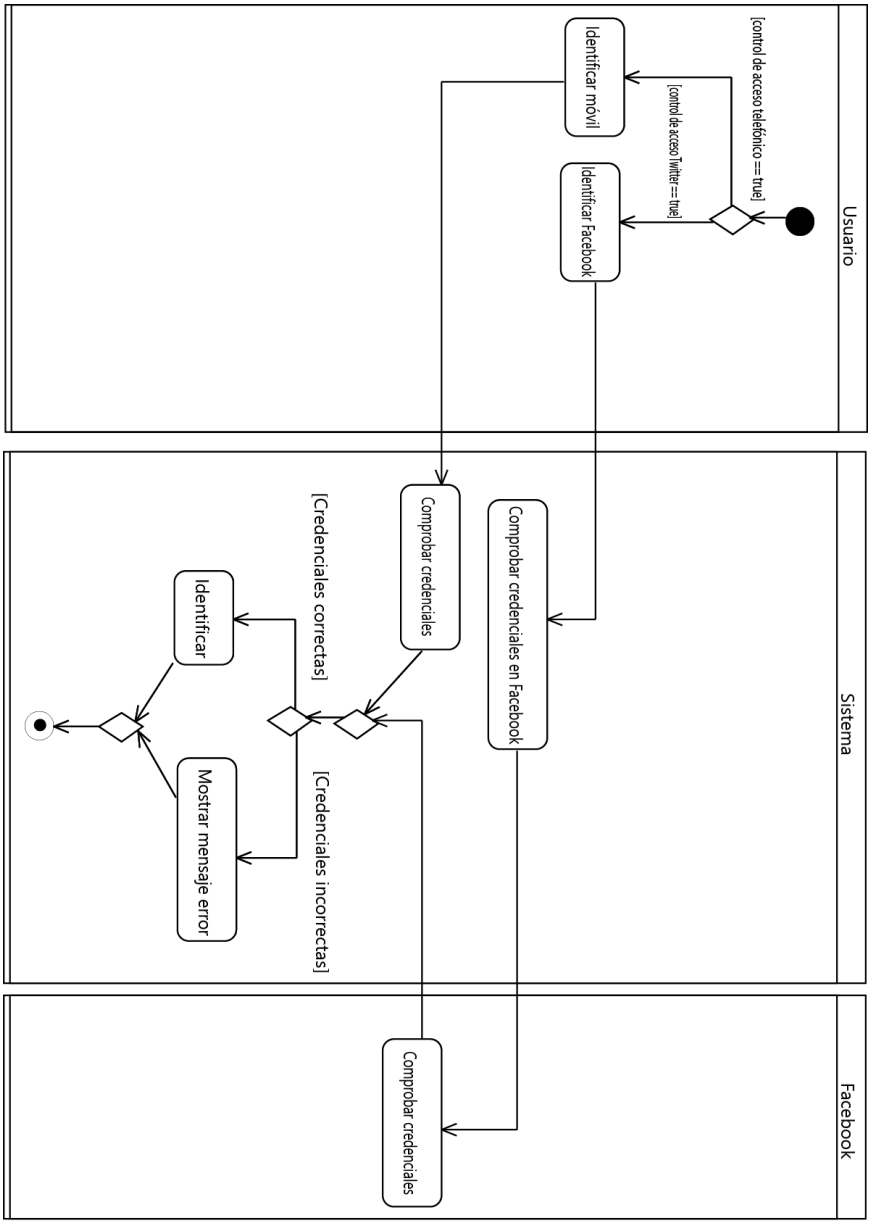


Figura 4.25 Diagrama de actividad derivado para especificar el caso de uso identificar

La Tabla 4.8 muestra un fragmento de la tabla de necesidades para un producto. Se asigna a cada necesidad del cliente: un identificador, un nombre, descripción y un campo para indicar si está soportada o no la necesidad que se utiliza durante la validación. Esta actividad finaliza cuando se ha analizado cada necesidad del cliente y el documento está completo (la columna "soportado" está rellenada para cada necesidad del cliente). Las necesidades pueden cubrirse como casos de uso o acciones, por otra parte la necesidad N21 no está cubierta por ningún artefacto de la especificación de requisitos y deberá de cubrirse mediante la incorporación de una acción en el diagrama de actividad correspondiente. En este ejemplo la necesidad N3 se cubrirá con la incorporación de un nuevo caso de uso.

Tabla 4.8 Ejemplo de tabla de necesidades validadas para el producto

Identificador	**Nombre**	**Descripción**	**¿Soportada?**
N1	Crear usuario con teléfono	Se permite el acceso al sistema con dispositivos móviles.	Sí
N2	Vincular usuario con Facebook	Gestión de acceso mediante Facebook. Se vincula el ID de Facebook con la cuenta del sistema para usarlo para acceder al sistema.	Sí
N3	Vincular usuario con LinkedIn	Gestión de acceso mediante LinkedIn. Se vincula el ID de LinkedIn con la cuenta del sistema para usarlo para acceder al sistema.	No
N4	Identificar usuario	Se identifica un usuario. Por defecto se utilizará la cuenta del sistema	Sí
N5	Borrar usuario	Se borra un usuario del sistema	Sí
N6	Actualizar usuario	Se actualizan los datos de un usuario	Sí
…	…	…	…
N20	Identificar LinkedIn	El usuario requiere identificarse en el sistema con sus credenciales de la red social LinkedIn	No
N21	Llamada a comprobar credenciales en LinkedIn	El sistema recibe una solicitud de identificación mediante la red social LinkedIn. En este caso tiene que hacer una llamada al sistema externo LinkedIn para comprobar las credenciales	No
N22	Comprobar credenciales en LinkedIn	El sistema solicita a LinkedIn una dirección de correo y una contraseña. El sistema externo LinkedIn notifica si esas credenciales son correctas o no	No

4.4.3 Especificación de los requisitos delta

En aquellos casos que existen requisitos que no se pueden satisfacer con una especificación del producto derivada, será necesario especificar requisitos delta. Este proceso se descompone en dos tareas (Figura 4.26): la especificación de los requisitos delta y la construcción de la matriz de trazabilidad de requisitos delta.

En la especificación de requisitos delta el ingeniero de requisitos de aplicación tomará el documento con las *necesidades del cliente validadas*. En un proceso iterativo seleccionará para cada una de las necesidades cómo se van a especificar. Una necesidad podría cubrirse con un caso de uso o una acción dentro de un diagrama de actividad. Para representar esta notación se utiliza un estereotipo basado en UML llamado delta que se presenta en la siguiente sección.

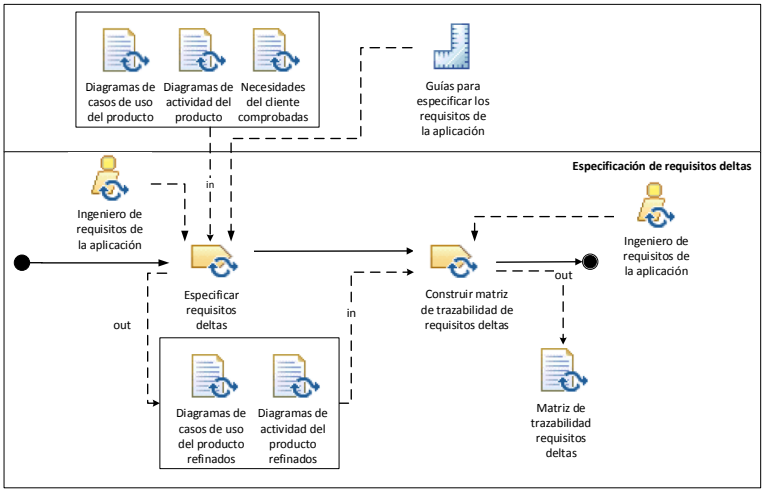


Figura 4.26 Tareas de la especificación de requisitos delta

4.4.3.1 El estereotipo delta

En ocasiones es necesario extender un elemento de un modelo con propiedades específicas para adaptarse a un dominio concreto. Por ejemplo un actor no puede representar per se a un sistema externo y deseamos redefinir un nuevo tipo de actor para soportar esta propiedad. En estos casos UML provee tres mecanismos de extensión para extender elementos de un modelo: etiquetas, restricciones y estereotipos. Formalmente un estereotipo define como debe extenderse una metaclase para permitir el uso de una terminología dependiente de un dominio concreto de forma que extienda o reemplace a la metaclase original (OMG 2007).

Se ha creado un perfil UML que permite representar los requisitos delta en la especificación de requisitos del producto. Estos requisitos delta se etiquetan con el estereotipo «delta». El estereotipo definido que se aplica en la tarea de especificar requisitos delta de la actividad de especificación de requisitos delta a los elementos *Activity Partition*, *Actors*, *Opaque Actions* y *Use Cases*.

El estereotipo puede aplicarse tanto a casos de uso como a diagramas de actividad. Se han definido un conjunto de guías para asistir al ingeniero de requisitos de la aplicación. A nivel de diagramas de casos de uso se pueden añadir nuevos actores y nuevos casos de uso. Por otra parte se puede modificar un requisito existente en un diagrama de casos de uso cuando es necesario. En este caso se utilizará la relación de herencia para expresar que un caso de uso delta especializa al caso de uso original. No se considera el borrado de requisitos dado que si el ingeniero de requisitos considera borrar un requisitos entonces se tendrá que configurar el modelo de características si se debe borrar de toda la familia de productos o eliminar de la configuración del producto si simplemente se quiere suprimir de un producto específico.

Por otra parte es posible añadir requisitos delta en los diagramas de actividad. De acuerdo con las guías de la ingeniería de la aplicación si se añade un nuevo actor se debe de añadir una nueva partición de actividad. Si se añade una nueva actividad en una partición existente, en este caso se añade un elemento *Opaque Activity* etiquetada con el estereotipo «delta» para indicar que es una nueva acción que se añade al flujo de acción. La Figura 4.27 muestra la especificación de casos del control de acceso una vez se han especificado los deltas necesarios. Podemos comprobar que se ha añadido un nuevo caso de uso llamado "vincular *LinkedIn*" con el objetivo de cubrir la necesidad N3. Además dado que se ha añadido un nuevo caso de uso es preciso añadir un nuevo actor que representa al sistema externo *LinkedIn*.

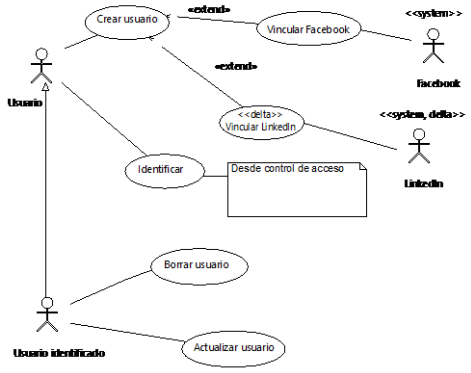


Figura 4.27 Diagrama de casos de uso tras especificar los deltas

De forma similar se deben de añadir los requisitos delta en los diagramas de actividad. La Figura 4.28 muestra el diagrama de actividad para especificar el caso de uso identificar tras especificar los requisitos delta. Existen tres particiones de actividad que se obtuvieron de la derivación del producto que se muestra en el ejemplo: partición de usuario, partición de sistema y partición del sistema externo Facebook. En este caso de acuerdo con las guías se debe de añadir una nueva partición para el sistema externo *LinkedIn* con el estereotipo delta. Por otra parte para especificar la necesidad N20 identificar *LinkedIn* se añade la acción identificar *LinkedIn* en la partición de usuario. De forma análoga se añaden las acciones Comprobar credenciales en *LinkedIn* para cubrir la necesidad N21 y la acción comprobar credenciales de la partición de actividad *LinkedIn* para cubrir la necesidad N22. Una vez se han producido las versiones refinadas de los diagramas de casos de uso y diagramas de actividad con los requisitos delta es necesario construir la matriz de trazabilidad de los requisitos delta.

4.4.3.2 La matriz de requisitos delta

Con el objetivo de mantener la trazabilidad entre los requisitos delta y los requisitos afectados por los cambios se elabora un artefacto llamado *matriz de trazabilidad de requisitos delta*. Este artefacto se representa en forma de tabla con tres columnas: requisito delta, tipo de requisito y requisitos impactados. En la columna requisitos delta se escribe el nombre del nuevo requisito, en tipo se indica el tipo de requisito delta (partición de actividad, actor, acción o caso de uso). Por último se escribe una relación de los requisitos que son impactados por los requisitos delta.

La Tabla 4.9 muestra la matriz de trazabilidad de requisitos delta obtenida para el caso de uso y el diagrama de actividad que se muestra en el ejemplo.

Tabla 4.9 Matriz de requisitos delta

Requisito delta	Tipo	Requisito relacionado
CU-30 Vincular LinkedIn	Caso de uso	ACT-09 Crear usuario, ACT-10 LinkedIn
ACT-10 LinkedIn	Actor	CU-30 Vincular LinkedIn
PA-10 LinkedIn	Partición de actividad	AC-42 Comprobar credenciales
AC-40 Identificar LinkedIn	Acción	ND-02 Nodo decisión, AC-41 comprobar credenciales en LinkedIn
AC-41 Comprobar credenciales en LinkedIn	Acción	AC-40 Comprobar credenciales, AC-42 Comprobar credenciales (partición LinkedIn)
AC-42 Comprobar credenciales (partición LinkedIn)	Acción	PA-10 LinkedIn

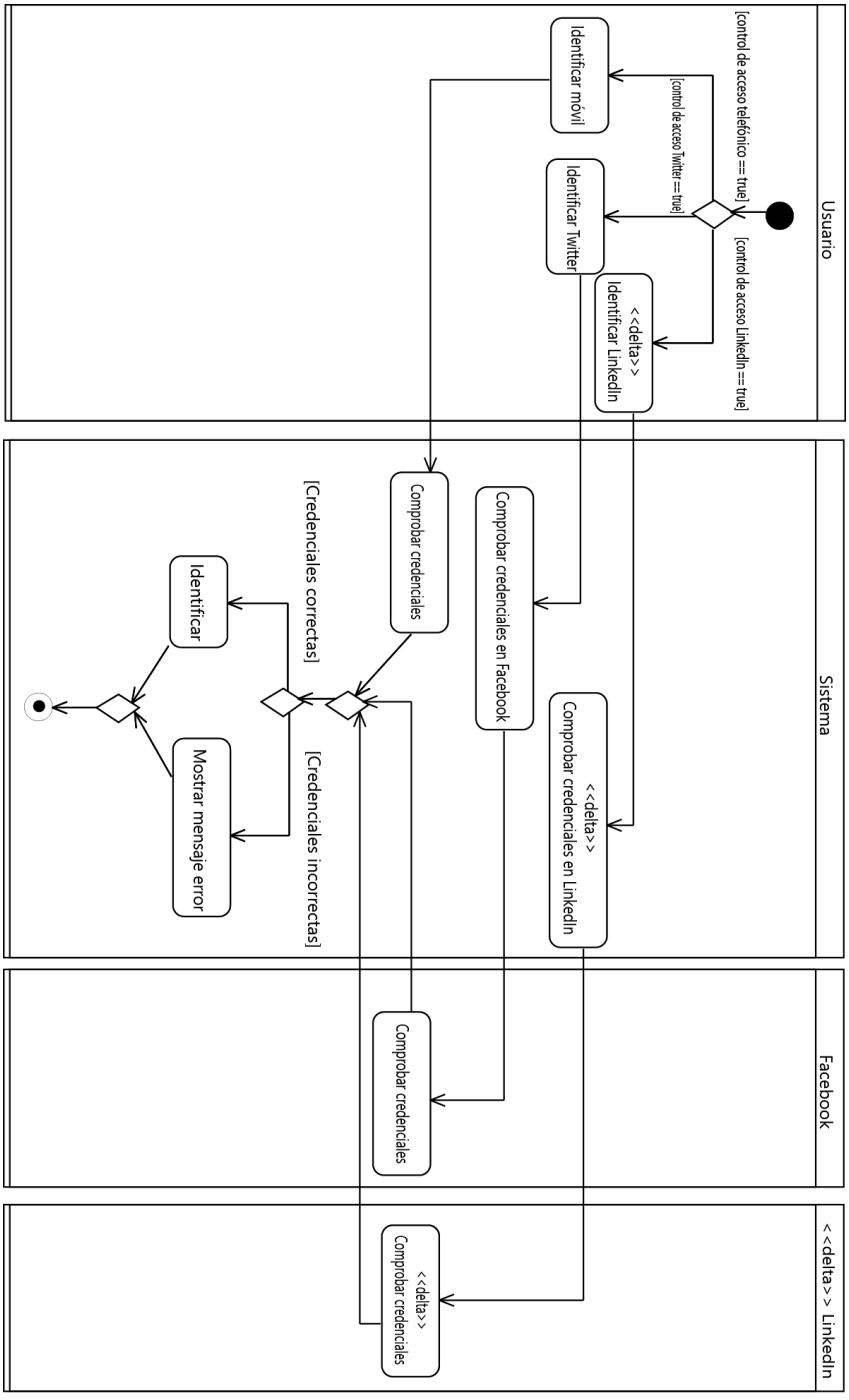


Figura 4.28 Diagrama de actividad tras la especificación de los deltas

4.5 Conclusiones

En este capítulo de ha presentado la principal contribución metodológica de esta tesis: la definición del método FeDRE que permite especificar los requisitos de la familia de productos a partir de los artefactos de *scoping*. Una vez definidos los requisitos, durante la ingeniería de aplicación, se define un proceso para derivar los requisitos y especificar los requisitos delta en caso de que sea necesario.

Durante la ingeniería del dominio se toman como entrada los artefactos de la actividad de *scoping*, donde se establecen los límites y la estrategia de reusabilidad de la LPS. Se utilizan un conjunto de guías para definir y especificar los requisitos de la LPS, representados mediante casos de uso y diagramas de actividad. Esta especificación se debe de validar con el cliente para asegurarse de que se cubren todas las demandas del cliente. Si la validación es positive el experto de reúso elabora una modelo de variabilidad de los requisitos con la notación más adecuada para el dominio del problema.

En la ingeniería del dominio se derivan los requisitos para un producto en un proceso iterativo e incremental. Primero se define la configuración del producto y se resuelven los puntos de variabilidad para derivar los requisitos del producto. En los casos que haya que añadir o excluir una característica del producto se obtendrán nuevamente los requisitos. Para comprobar si la especificación de requisitos es adecuada se realiza una validación con el cliente, donde se elaborar una lista de necesidades que el ingeniero de requisitos de la aplicación elabora. Si es necesario se refina la especificación de requisitos del producto añadiendo los requisitos delta con las guías definidas de FeDRE.

En el siguiente capítulo se muestra una aproximación tecnológica para soportar el proceso de FeDRE para definir y especificar los requisitos tanto en los procesos de ingeniería del dominio como en la ingeniería de la aplicación.

Capítulo 5. Aproximación tecnológica para la derivación de requisitos del producto

Uno de los problemas de las LPS es tratar con un gran número de características que pueden afectar a multitud de puntos de variabilidad en la especificación de requisitos. Esto implica que la derivación de los requisitos para un producto concreto puede resultar muy costosa si no se da un soporte automatizado.

En este capítulo se presenta una aproximación tecnológica con el objetivo de derivar los requisitos para un producto de una forma automatizada gracias a la aplicación del paradigma de desarrollo de software dirigido por modelos.

En la sección 5.1 describe la aproximación tecnológica basada en un multimodelo para expresar la variabilidad de la línea de productos software y la variabilidad interna de los requisitos.

La sección 5.2 describe la aproximación tecnológica que se ha desarrollado para dar soporte a la derivación automáticamente los requisitos de un producto. Esta aproximación está basada en el paradigma de DSDM y dos transformaciones de modelos.

La sección 5.3 presenta las conclusiones del capítulo.

5.1 Aproximación tecnológica para requisitos de la LPS

Uno de los principales problemas del desarrollo LPS consiste en expresar, manejar y resolver correctamente la variabilidad en los distintos artefactos que intervienen en el ciclo de vida. Esta cuestión no siempre resulta fácil dado que un dominio puede requerir el uso de múltiples modelos de requisitos donde se precisa para cada uno de ellos representar su variabilidad correspondiente.

En esta sección se presenta la aproximación propuesta basada en un multimodelo para representar y la variabilidad externa de la línea de productos y la variabilidad interna de los requisitos. Esta aproximación permite además la resolución automatizada de la variabilidad a partir de la configuración del producto.

5.1.1 Introducción al multimodelo

El desarrollo de LPS aplicando una estrategia de DSDM requiere del uso de modelos a diferentes modelos de abstracción para describir el sistema. Estos modelos que representan las vistas se especifican por separado, cada una de ella se especifica con un propósito determinado, sin embargo es necesario definir las relaciones entre las diferentes vistas para representar los aspectos o cualidades que no tienen cabida en los modelos individuales.

La solución propuesta se basa en el uso de un multimodelo que permite emplear distintas vistas (vista de variabilidad externa, vista de requisitos) así como relaciones entre elementos en las distintas vistas. En este contexto utilizamos la definición propuesta en (Barkmeyer y otros 2003) y refinada por (González Huerta 2014), definiendo finalmente un **multimodelo** como:

"*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*".

En un trabajo previo se utilizó el multimodelo para guiar al arquitecto software en la derivación y mejora de arquitecturas de producto en un entorno de DSDM. Este multimodelo incluía inicialmente cuatro vistas (González Huerta 2014): la de variabilidad, la funcional, la de calidad y la de transformación. En este trabajo se ha adaptado este multimodelo para dar soporte la variabilidad, distinguiendo dos niveles: la *variabilidad externa*, representada en la vista de variabilidad, y la

variabilidad interna, representada en la vista de requisitos, para derivar los requisitos del producto a través de transformaciones de modelos.

El objetivo de esta estrategia es provee de un marco de trabajo para modelar la variabilidad de los requisitos de una forma independiente del dominio.

El multimodelo se compone de dos vistas:

- **La vista de variabilidad**. Esta vista expresa la variabilidad externa de la LPS en términos de características comunes y variables dentro de la LPS. El elemento principal del elemento es la característica, que es un aspecto o característica del sistema visible por el usuario (Clements y Northrop 2007). La vista de variabilidad se representa utilizando una variante (Gómez y Ramos 2010) del modelo del modelo de características basado en cardinalidad (Czarnecki y Kim 2005).
- **La vista de requisitos**. Expresa la variabilidad a nivel de requisitos que realiza la variabilidad externa. Se expresará dicha variabilidad utilizando el lenguaje CVL (OMG 2012), el cual permite expresar la variabilidad en un LED. CVL permite establecer puntos de variabilidad sobre un modelo base definido en un LED. Esta variabilidad se define mediante un lenguaje compatible con el estándar MOF (OMG 2006).

Se define una relación entre las características del modelo de variabilidad y los elementos *Choice* de la vista de requisitos.

En las siguientes secciones se describirán las dos vistas utilizadas para representar la variabilidad externa de la LPS (variabilidad externa) y la vista de variabilidad de requisitos (variabilidad interna).

5.1.2 La arquitectura del multimodelo

Para la definición del metamodelo que soporta el multimodelo se sigue una arquitectura en dos capas. Por un lado se define un metamodelo llamado núcleo o *core* del multimodelo que contiene las entidades esenciales para representar un multimodelo Por otra parte se define un metamodelo específico para representar la especificación de requisitos de la línea de productos.

5.1.2.1 El multimodelo genérico o core

Se definió un metamodelo con los elementos esenciales de multimodelo genérico llamado "*core*". De acuerdo con la Figura 5.1 el multimodelo está compuesto de modelos de vista (*ViewpointModel*) y relaciones (*MultimodelRelationship*). Estos modelos de vista tendrán tantas entidades y relaciones entre estas como sea necesario. Las clases *ViewpointModel*, *Entity* y *Relationship* se representan con líneas de puntos para representar que son conceptos pertenecientes a cada vista.

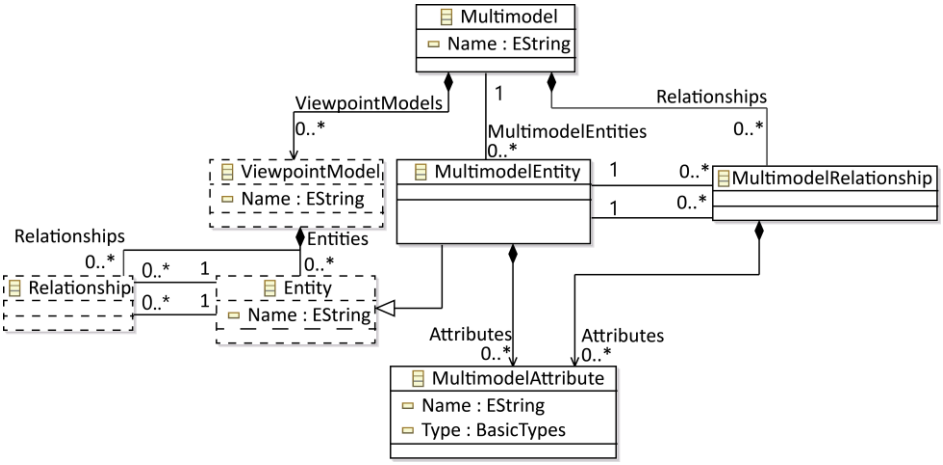


Figura 5.1 Estructura genérica del multimodelo

El multimodelo contendrá entidades propias (*MultimodelEntities*) que extiendes las entidades pertenecientes a los modelos de vista. Además el multimodelo contendrá relaciones entre las entidades del multimodelo para representar relaciones entre elementos de diferentes vistas (*MultimodelRelationship*). Por otra parte se podrán añadir atributos adiciones (*MultimodelAttribute*) con el objetivo de capturar información adicional asociada a constructos que no tendrían cabida en las entidades y las relaciones de los modelos de vista.

Además, se pueden definir restricciones sobre las relaciones y entidades, Una relación no puede tener como origen y destino la misma entidad. Para garantizar que se cumple esta restricción de integridad se puede definir en el propio metamodelo del multimodelo una restricción análoga a la mostrada en el Listado 5.1.

Listado 5.1 Restricción OCL sobre las relaciones en el multimodelo Core

```
OCL:
context Multimodel
inv: DifferentFromTo :
self.from <> self.to
```

Las relaciones entre los elementos de las diferentes vistas definidas en el multimodelo pueden ser de diferentes tipos. Se ha definido en el multimodelo un conjunto de relaciones posibles:

- *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 (2004) 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).

Por otra parte una dirección puede tener dos direcciones:

- *Bidireccional***:** la relación es en ambos sentidos.
- *Unidireccional***:** la relación es en un único sentido.

5.1.2.2 Un multimodelo para representar los requisitos de la LPS

El multimodelo genérico se extiende con el objetivo de soportar las dos vistas que representan la variabilidad externa de la LPS y la variabilidad interna de los requisitos. La Figura 5.2 muestra el metamodelo del multimodelo extendido. La clase contenedora es la clase *Multimodel* la cual extiende a la clase del multimodelo genérico con el mismo nombre.

Este multimodelo contiene dos vistas:

- *Vista de variabilidad*. Dicha clase se representa en el metamodelo con la metaclase *VariabilityViewpointodel*. Dicha clase extiende a la metaclase *FeatureModel* del metamodelo Features y a la metaclase *ViewPointModel*.
- *Vista de requisitos*. Dicha clase se representa en el metamodelo con la metaclase *RequirementsViewpointodel*. Dicha clase extiende a la metaclase *BVRModel* del metamodelo *BVR* y a la metaclase *ViewPointModel*.

Por otra parte el multimodelo contiene las siguientes entidades:

- *EFeature*. Extiende una *Feature* del metamodelo *Features*. Extiende la metaclase *MultimodelRelatableEntity* puesto que participará en relaciones definidas en el multimodelo.
- *EChoice*. Extiende un elemento *Choice* del metamodelo de *BVR*. Extiende la metaclase *MultimodelRelatableEntity* puesto que participará en relaciones definidas en el multimodelo.

Por último se define un tipo de relación:

- *FeatureToVSpec*. Dicha relación establece una correspondencia entre un elemento *EFeature* y un elemento *EChoice*.

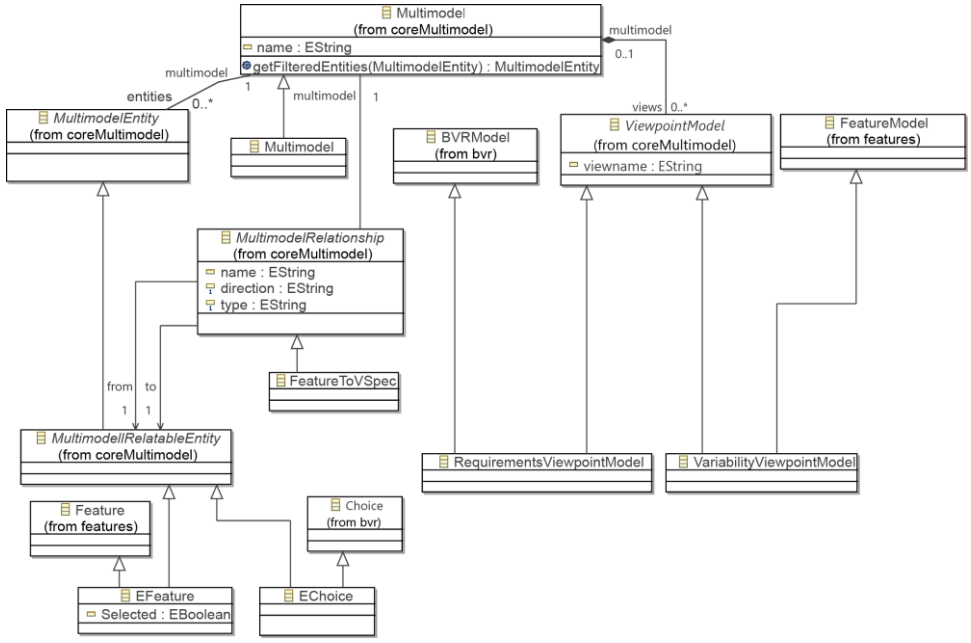


Figura 5.2 Metamodelo del multimodelo de requisitos

5.1.2.3 Vistas del multimodelo

Este multimodelo se compone inicialmente de dos vistas que representan la variabilidad externa de la LPS y la variabilidad interna de los requisitos. A continuación se definen sendos metamodelo propuestos para representar respectivamente cada una de las vistas.

5.1.2.3.1 La vista de variabilidad de la LPS

La vista de la variabilidad del multimodelo permite expresar la variabilidad externa de la línea de productos. Esta variabilidad se expresa mediante características las cuales son funcionalidades o cualidades del sistema visibles por el usuario. Una de las principales notaciones es el modelo de características propuesto por Kang y otros (1990). Sin embargo con el objetivo de permitir un modelo de características con cardinalidades se ha utilizado el metamodelo para expresar la variabilidad externa propuesto por (Gómez 2012).

La Figura 5.3 muestra un extracto del metamodelo definido para soportar el modelo de características con cardinalidades. La clase contenedora principal es el modelo de características (*FeatureModel*). Este modelo de características está compuesto de características (*Features*) que se pueden estructurar en grupos (*Groups*). A diferencia como otras notaciones como la propuesta por Kang y otros (1990), en esta notación se diferencia entre la notación del grupo y la notación de las características hijas. La cardinalidad de grupo restringe cuantas características se pueden instanciar sin importar el número de instancias de la característica.

Además pueden establecerse relaciones entre características (*Relationships*). Se distinguen los siguientes tipos de relaciones:

- *Implicación (A -> B).* Si una instancia de una característica A existen al menos debe de existir una instancia de la característica B.
- *Bi-condicional (A <-> B).* Si una instancia de una característica A existen al menos debe de existir una instancia de la característica B y viceversa.
- *Exclusión (A >-< B).* Si una instancia de una característica A existe, no puede existir ninguna instancia de la característica B y viceversa.
- *Uso (A-->B).* Esta relación se define a nivel de configuración e indica que una instancia específica de una característica A se relaciona con una o más instancias específicas de B definidas por su límite superior (n).

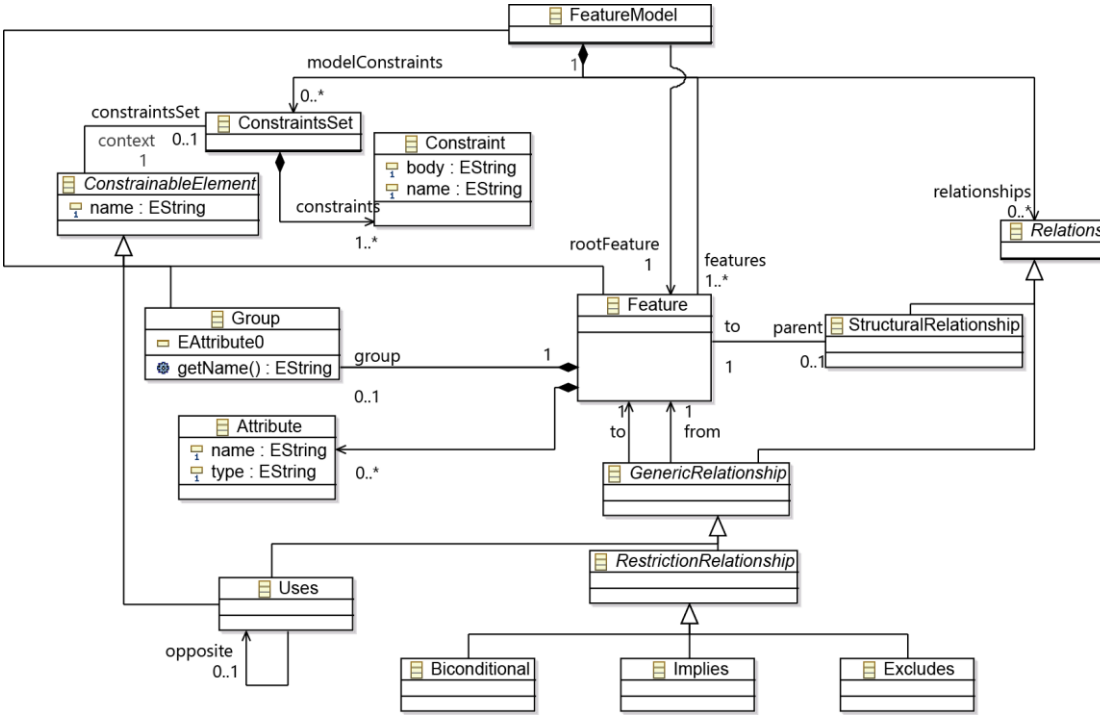


Figura 5.3 Fragmento del metamodelo de variabilidad externa

Por último pueden definirse restricciones (*Constraints*) sobre el modelo. Estas restricciones se definen mediante el lenguaje *Feature Modeling Constraint Language* (FMCL) (Gómez 2012), con una sintaxis y semántica similar a OCL.

La Figura 5.5 muestra un ejemplo de un modelo de caracteristicas para representar la variabilidad externa de la línea de productos Savi. Este diagrama tiene forma de árbol donde la característica raíz es Savi. Esta característica tiene varias características hijas, de las cuales las se indican como relaciones opcionales las que tienen un punto blanco en el diagrama: *información usuario, números de emergencia y soporte social.*

Por otra parte las relaciones de obligatoriedad se indican con un punto negro en el extremo de la relación. En este caso las siguientes características hijas de la característica Savi son obligatorias: *control acceso, contacto, destino, tracking, idioma, localización y soporte social.*

Otro tipo de relación entre características es la de implicación. En el diagrama de características de Savi podemos encontrar una relación de implicación entre *información usuario y control acceso*, que indica que la primera solo podrá estar seleccionada en una configuración si control acceso está seleccionada. De forma

análoga la característica *tracking* Web solo podrá seleccionarse si control acceso Web está seleccionada en una configuración.

Otro tipo de elementos que se utilizan en el modelo de características son los grupos. Los grupos de alternativas, indicados con un triángulo blanco en la notación, permiten seleccionar una y sólo una característica de las características hijas. En este caso la característica *idioma* sólo permite seleccionar entre una de sus características hijas: *idioma estándar* o *múltiples idiomas*.

Otro tipo de grupos son los grupos de selección, indicado en la notación gráfica con un triángulo negro. En el grupo que parte de control acceso se debe de seleccionar como mínimo una de las características hijas y como máximo dos.

5.1.2.3.2 La vista de requisitos

La vista de variabilidad externa representa la variabilidad de alto nivel desde el punto de vista del usuario/cliente. Sin embargo la variabilidad interna de los requisitos se define a un nivel de granularidad más bajo, que se va a representar mediante la vista de requisitos. Para expresar dicha variabilidad interna en los requisitos se ha seleccionado el lenguaje BVR, un lenguaje que surge de la propuesta de standard CVL y que está diseñado para expresar la variabilidad en cualquier lenguaje específico de dominio.

La Figura 5.4 muestra un extracto del metamodelo BVR que da soporte a la variabilidad en los requisitos.

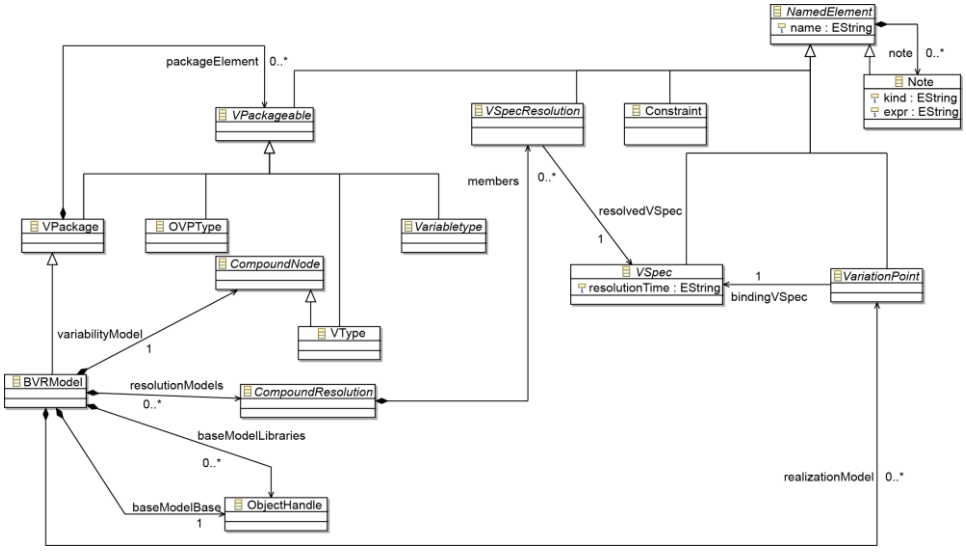


Figura 5.4 Fragmento del metamodelo de variabilidad de requisitos

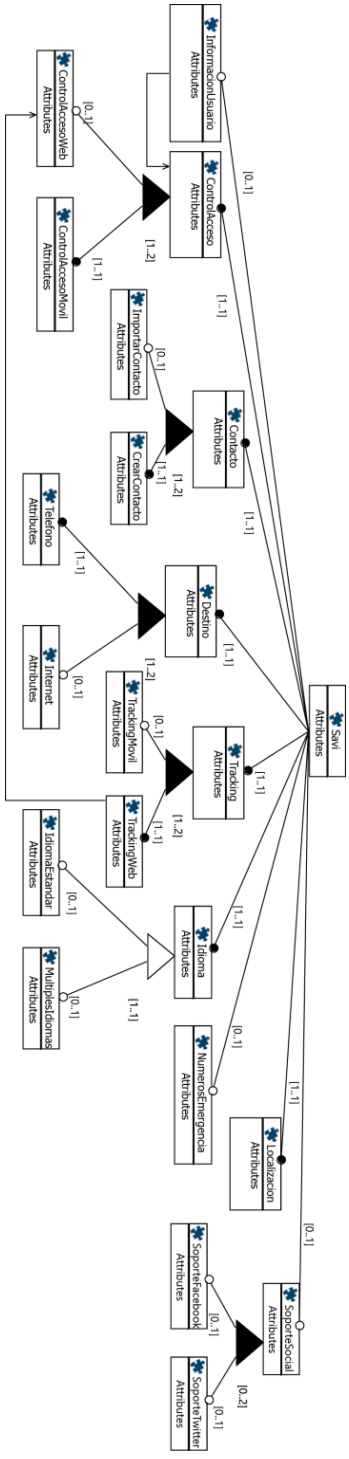


Figura 5.5 Ejemplo de modelo de variabilidad de la LPS

La clase *BVRModel* es el contenedor de la capa de abstracción, la capa de realización y los modelos base. Esta metaclase en concreto contiene el resto de elementos de una especificación BVR:

- Un *modelo de variabilidad (BVR)* que es instancia de la metaclase *CompoundNode*. Contiene el modelo de variabilidad de los requisitos. Este modelo se representa como un árbol de objetos instancia de la metaclase *VNode*. Un *VNode* representa restricciones lógicas implícitas en la resolución de un elemento hijo.

- Un *modelo* (o modelos) *de realización*. El modelo de realización contiene los puntos de variabilidad utilizados para representar la realización de un modelo de variabilidad para un modelo de resolución dado. Los puntos de variabilidad definen los cambios que se realizan en el modelo base para satisfacer la configuración del producto especificada. El modelo de realización es una instancia de la metaclase *VariationPoint*.

- Un *modelo* (o modelos) *de resolución*. Definen la configuración del producto o productos. Es instancia de la metaclase *CompoundResolution*. Un *CompoundResolution* define la raíz de un árbol que contiene elementos *VSpecResolution*.

- Un *modelo base*. Se utiliza como punto inicial para crear productos en el lenguaje base. Es instancia de la clase *ObjetHandle*, la cual identifica un objeto del modelo base.

- Un *modelo* (o modelos) *de librería*. Estos modelos contienen los fragmentos que se utilizaran aditivamente para crear productos a partir del modelo base. Es instancia de la clase *ObjetHandle*, la cual identifica un objeto del modelo base.

El modelo de variabilidad y el modelo de resolución están relacionados, puesto que el modelo de resolución define cómo se resuelve un modelo de variabilidad para producir una determinada instancia de producto. La Figura 5.6 muestra la relación entre los *VSpec*, del modelo de variabilidad, y los *VSpecResolution*, del modelo de resolución.

En el modelo de variabilidad el elemento principal es el *VSpec*, el cual representa a son puntos de decisión que deben de resolverse. Un *VSpec* apunta a un *Target*, el cual indica la entidad a la que hace referencia (por ejemplo un subconjunto del modelo base).

Respecto al modelo de realización, el *VSpecResolution* resuelve un determinado VSpec. Los *VSpecResolution* se organizan en forma de árbol, copian parcialmente

la estructura en árbol de los *VSpec* que se van a resolver. La metaclase *ChoiseResolution* especializa al *VSpecResolution*, resolviendo positivamente (*PosResolution*) o negativamente (*NegResolution*) un VSpec. Una *ChoiceResolution* se asocia con cuatro metaclases:

- Una *Choice* resuelta. Una *Choice* es un VSpec del *modelo de variabilidad BVR* que representa una decisión sí o no. Cuando se asocia un punto de variabilidad a un Choice dependerá si la resolución es un *PosResolution* o un *NegResolution* para determinar si el punto de variación se ejecuta.
- Una ocurrencia de una *ChoiceOccurrence* resuelta. Es similar a un *Choice*, pero hace referencia a un *VType* que tiene la habilidad de definir una estructura subordinada en un *VNode*.
- Un clasificador resuelto (*VClassifier*). Representa el concepto de grupo. Puede tener un sub-árbol en forma de *CompoundNode* que define una especificación de variabilidad subordinada (*VNode*). Cuando se especifica un punto de variabilidad repetible a un *VClassifier*, este se aplicará una vez por cada resolución del VClassifier durante la materialización.
- Una ocurrencia de un *VClassOccurrence*. Es un concepto de grupo similar al *VClassifier*. La diferencia es que el *VClassOcurrence* hace referencia a un *VType* y no tiene definido su propio sub-árbol.

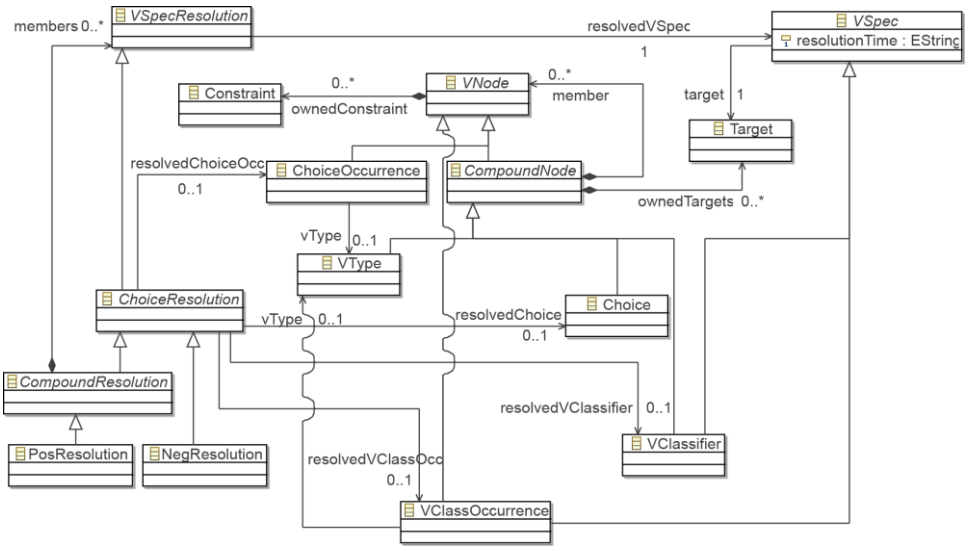


Figura 5.6 Relaciones entre VSpec y el VSpecResolution

Por otra parte los *VSpec* se relacionan con los puntos de variabilidad del modelo de realización. Existen cinco tipos de metaclases que especializan a la metaclase *VariationPoint*:

- *RepeatableVariationPoint*. Es un punto de variación que puede aplicarse múltiples veces durante la materialización. Puede enlazarse sólo con un *VClassifier* o *VClassOccurrence* y aplicarse una vez por cada resolución que le referencia.
- *ChoiceVariationPoint*. Es un punto de variabilidad que puede enlazarse con una Choice. Durante la materialización de una resolución la Choice determina si el punto de variación se aplicará o no.
- *ParametricVariationPoint*. Es un punto de variación que depende de un parámetro y tiene que enlazarse a una variable. Durante la materialización el valor asignado durante la resolución se utiliza como parámetro.
- *StagedVariationPoint*. Es un punto de variación que hace referencia a ocurrencias de un *VType*, un *VClassOccurrence* o un *ChoiceOccurrence*. El punto que realiza la ocurrencia puede variar entre ocurrencias del mismo tipo.
- *OpaqueVariationPoint*. Es un punto de variación ejecutable y dependiente del dominio cuya semántica no está definida por BVR. Es responsabilidad del LED la ejecución de este punto de variación.

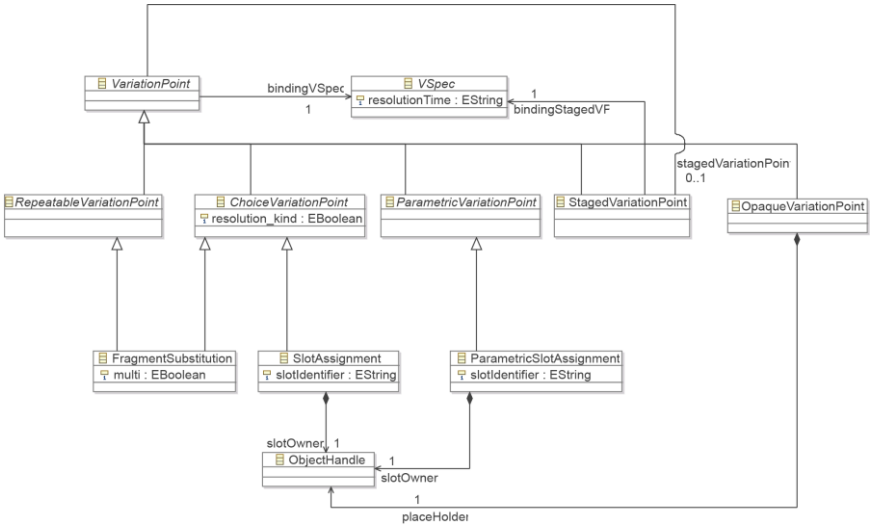


Figura 5.7 Relación entre puntos de variabilidad y VSpec

Los *FragmentSubstitution* emparejan un elemento de origen del modelo base con uno varios fragmentos de reemplazo del modelo de librería para indicar que si se incluye ese punto de variabilidad en el producto se realizará la sustitución correspondiente. La Figura 5.8 muestra las relaciones entre los puntos de variación y los reemplazos.

Un *FragmentSubstitution* contiene tres tipos de elementos:

- Opcionalmente uno, o más de uno, *BoundaryElementBinding*. Estos elementos especifican el enlace entre los elementos *placement* y *replacement*.

- Un *PlacementFragment*. Especifica el fragmento que va a ser reemplazado. El *PlacementFragment* se eliminará.

- Opcionalmente un *ReplacementFragmentType*. Especifica el elemento de reemplazo que reemplazará el original.

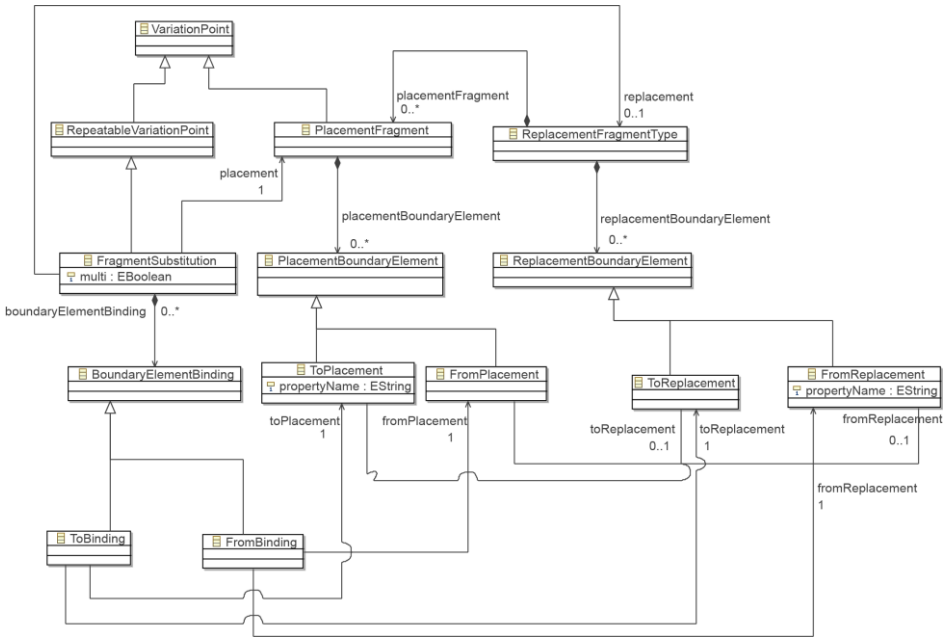


Figura 5.8 Tipos de puntos de variación y los reemplazos

5.1.3 Definición de la variabilidad de requisitos

Una vez definidas las dos vistas del multimodelo y las relaciones entre estas se define la trazabilidad entre la vista de variabilidad de requisitos y los modelos de requisitos. En CVL se definen dos modelos: el modelo base y el modelo librería. Más concretamente, el *modelo base* se utiliza para especificar los requisitos del dominio. Existen diversas alternativas para definir el modelo base y fijar la estructura de dicho modelo y la estrategia de variabilidad a seguir al definir los puntos de variación del modelo de variabilidad de requisitos. Existen diferentes alternativas:

- *Modelo base máximo y estrategia sustractiva*. Se parte de un modelo base completo en el que están presentes todas las posibles variantes y se aplica la aproximación negativa (Sánchez y otros 2007) o sustractiva (Haugen y otros 2010) donde se eliminan los elementos que no están presentes en una configuración del producto determina.

- *Modelo base mínimo y estrategia aditiva*. En esta estrategia se parte de un modelo base donde sólo están presentes los requisitos comunes, sin ningún elemento variable, y se aplica la estrategia positiva (Sánchez y otros 2007) o aditiva (Haugen y otros 2010) para añadir a la especificación de requisitos del producto que han de estar presentes en un determinada configuración del producto.

- *Modelo base intermedio y estrategia mixta*. Algunos elementos opcionales estarán presentes en el modelo base y tendrán que ser sustituidos o eliminados, mientras que otros tendrán que ser agregados al no estar presentes en la configuración del producto.

La Figura 5.9 muestra un ejemplo para cada una de las estrategias para diferentes diagramas de casos de uso. En la primera estrategia (modelo a la izquierda de la figura) el modelo base contiene todas las combinaciones posibles. En el centro se observa la estrategia opuesta, el modelo está vacío y el modelo librería contiene todas las combinaciones posibles. Por último en el lado derecho de la figura se muestra una estrategia mixta donde el modelo base contiene los elementos del caso de uso por defecto y el diagrama contenido en el modelo librería contiene los elementos de los casos adicionales (se incluye el control de acceso con Facebook y/o Twitter).

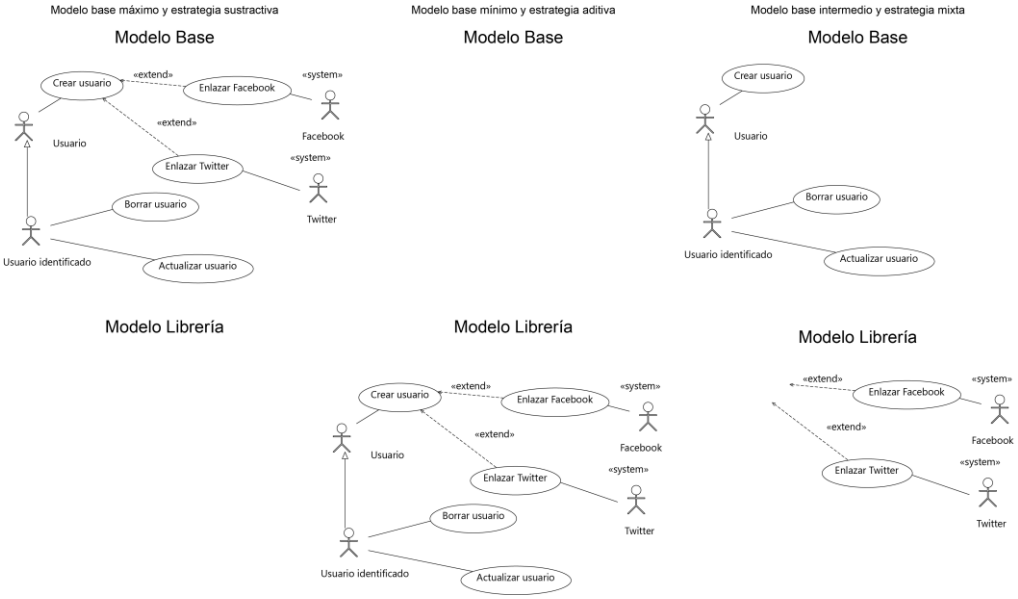


Figura 5.9 Estrategias de variabilidad de los requisitos dependiendo de la estructura del modelo base y librería

Por otra parte la estrategia elegida para *definir el modelo base* también afectará al modelo de librería de CVL. CVL creará los modelos resultantes realizando sustituciones o eliminado elementos del modelo base. En una estrategia de sustitución un conjunto de elementos A del modelo base se reemplaza por otro conjunto de elementos B; tomando como origen del conjunto B o bien elementos del modelo base u otro modelo.

En la *definición de la librería CVL* se define la estructura de la librería CVL. Pueden definirse fragmentos de modelo o bien modelos en sí mismo que se enlazan o agregan desde el modelo base por referencia. Además tal como contemplan los mecanismos de CVL podrían contener variabilidad anidada mediante la definición de variabilidad compuesta.

La trazabilidad entre el modelo de variabilidad de requisitos y el modelo librería se define con el editor CVL utilizando el *modelo de realización*. En este ejemplo vamos a utilizar un modelo base mínimo y una estrategia mínima. Nuestro modelo base en este caso contiene un conjunto de paquetes vacíos. La Figura 5.10 muestra el contenido del modelo base.

Figura 5.10 Contenido del modelo base

Dentro del paquete control de acceso podemos encontrar dos paquetes en el modelo base tal como muestra la Figura 5.11 y un comentario UML que utilizaremos para posteriormente definir los fragmentos de substitución en el modelo librería.

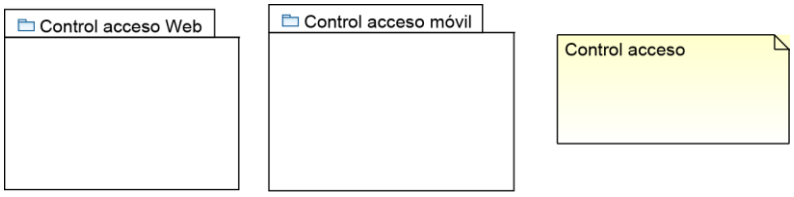


Figura 5.11 Paquetes contenidos dentro del paquete Control de acceso

Por otra parte nuestro modelo librería va a contener el conjunto de combinaciones posibles en el caso de que se resolvieran positivamente todos los *VSpec* asociados. En este caso dentro del paquete en el modelo librería encontraremos el diagrama de casos de uso que se muestra en la Figura 5.12. Este diagrama contiene dos actores, tres casos de uso y las relaciones entre estos que se añadirán al modelo base en el caso de que el *Vspec ControlAcceso* se resuelva positivamente.

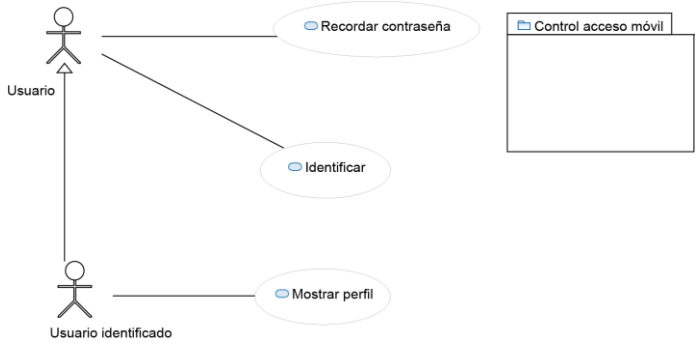


Figura 5.12 Diagrama de casos de uso del modelo librería

Dentro del paquete de control acceso móvil encontramos el diagrama de casos de uso que se muestra en la Figura 5.13. Este diagrama contiene los elementos que se pueden añadir al correspondiente modelo base dependiendo si se resuelven positivamente las *VSpec*: *ControlAcceso*, *ControlAccesoMóvil*, *SoporteFacebook* y *SoporteTwitter*.

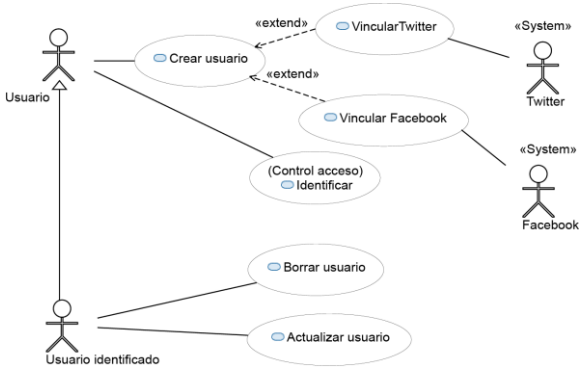


Figura 5.13 Diagrama de casos de uso para el control de acceso móvil en el modelo librería

Para poder relacionar la correspondencia entre el modelo base y el modelo librería es necesario definir el modelo de realización. Volviendo al ejemplo creamos un *PlacementFragment* en el modelo base del caso de uso del modelo base y que se corresponde con la Figura 5.11. Un *PlacementFragment* define un fragmento (un conjunto de los elementos de un modelo) del modelo base que se reemplazará por un *ReplacementFragment* durante la transformación CVL. La Figura 5.14 muestra en la parte superior izquierda el modelo UML del modelo base. En naranja aparece marcado el comentario por el *PlacementFragment*. Además en color morado en el editor nos indica los elementos afectados: en este caso el paquete UML control acceso. En la parte derecha se muestra el modelo realización de Savi donde vamos a definir los *PlacementFragment* del modelo base.

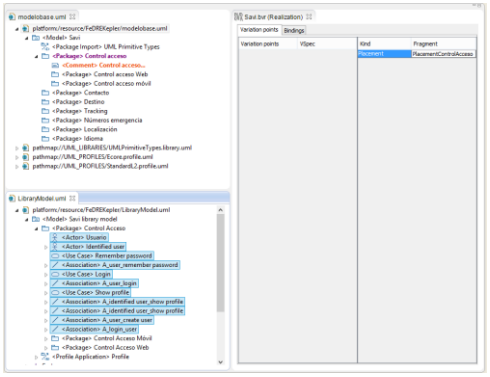


Figura 5.14 Creación del ReplacementFragment para el control de acceso

A continuación se crea el fragmento de reemplazo o *ReplacementFragment,* el cual define un fragmento del modelo librería que se usará como reemplazo de un *PlacementFragment* del modelo base. Volviendo al ejemplo la Figura 5.15 muestra la definición del fragmento de reemplazo. En la parte inferior izquierda se seleccionan los elementos del modelo UML librería. De forma análoga la definición del modelo base se muestran en color azul los elementos del modelo librería UML y en morado los elementos afectados, en este caso el paquete control acceso. En este caso se van a seleccionar en el fragmento de sustitución los elementos del modelo librería correspondientes a la Figura 5.12: los actores usuario y usuario identificado y la relación de herencia entre ellos además de los casos de uso recordar contraseña, identificar y mostrar perfil. Adicionalmente se incluyen las relaciones de asociación entre los actores y los casos de uso.

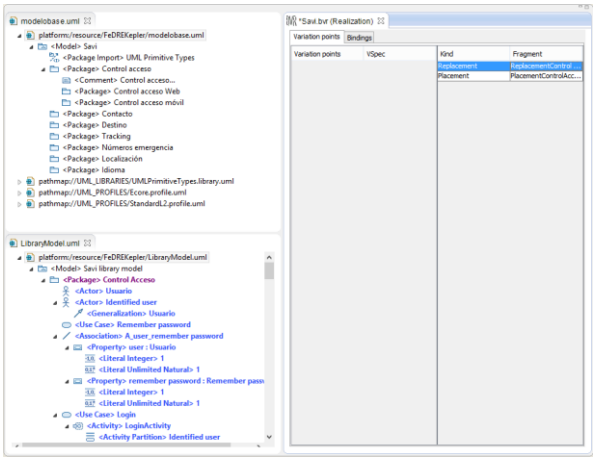


Figura 5.15 Creación del ReplacementFragment para el control acceso

A continuación se definen puntos de variación. Un punto de variación enlaza un *PlacementFragment* del modelo base con un *ReplacementFragment* del modelo librería. Además este punto de variación apunta a un *VSpec* del modelo de variabilidad CVL de forma que cuando se resuelva positivamente ese VSpec se llevará a cabo la sustitución en el modelo base. La Figura 5.16 muestra la creación del punto de variación para el control de acceso. En el editor del modelo de realización en la columna izquierda se sitúa la tabla con los puntos de variación y el *VSpec* relacionado con él. A la derecha aparecen marcados los correspondientes *PlacementFragment* y *ReplacementFragment* asociados a ese punto de variación. A la izquierda de la figura se muestran en la parte superior los elementos asociados a ese punto de variación del modelo base y en la parte inferior los elementos del modelo de librería asociados.

Una vez se define el punto de variación se especifica un *binding*. Los *binding* relacionan el punto de variación con los elementos del modelo base y modelo librería a bajo nivel y son necesarios para poder ejecutar la transformación CVL.

La definición del modelo de realización finalizará cuando se hayan definido todos los puntos de variación necesarios para cada uno de los *VSpec* del modelo de variabilidad CVL. Para la definición de los puntos de variación dentro del diagrama de casos de uso de control acceso web se procede de forma similar aunque este caso se añade la complejidad de tener que tratar con cuatro *VSpec* que pueden modificar este diagrama en el modelo base.

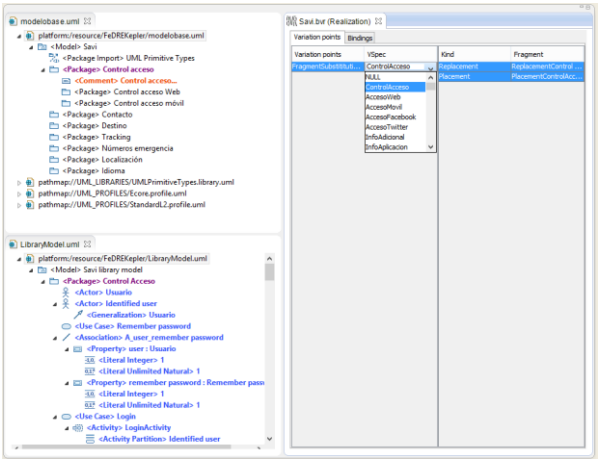


Figura 5.16 Definición del punto de variación para el Control Acceso

Con CVL también es posible definir fragmentos *binding* y *placement* sobre el editor gráfico de UML Papyrus tal como indica la Figura 5.17. En este caso definimos un *PlacementFragment* para el comentario *ControlAcceso*. De forma similar se van a crear cuatro *Placement* en este diagrama uno para cada *VSpec*: ControlAcceso, ControlAccesoMovil, SoporteFacebook, SoporteTwitter.

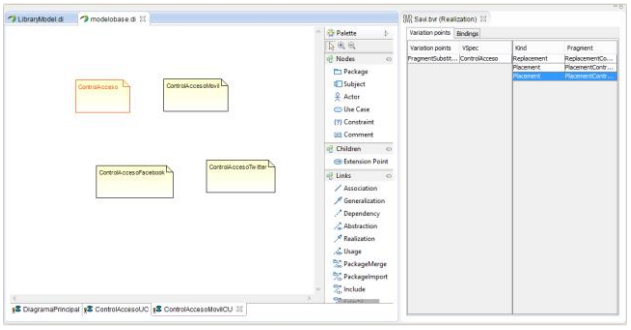


Figura 5.17 Definición de los PlacementFragment en el diagrama de casos de uso de control de acceso móvil en el modelo base

En el modelo de librería seleccionamos sobre el diagrama de casos de uso correspondiente al control de acceso móvil los elementos que se añadirán al modelo librería en caso de que el control de acceso se resuelva positivamente. De forma análoga la Figura 5.18 muestra el fragmento de reemplazo asociado al control de acceso móvil. En este caso se seleccionan los casos de uso crear usuario, borrar usuario, actualizar usuario, los actores usuario y usuario identifica y las relaciones correspondientes

La Figura 5.18 muestra a la derecha el editor del modelo de resolución donde se crear un nuevo *ReplacementFragment* y la izquierda el color azul se muestra el caso de uso identificar el cual está asociado al control de acceso. En color amarillo se muestra en actor usuario debido a que será afectado en el caso de que se realice la sustitución.

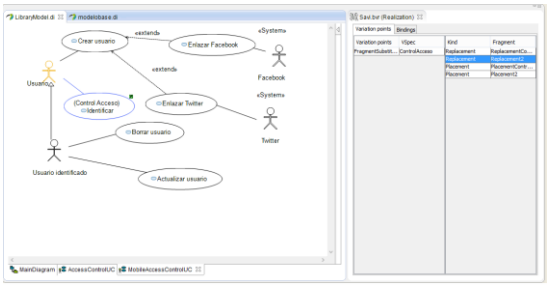


Figura 5.18 Definición del ReplacementFragment para el control de acceso en el diagrama de casos de uso de control de acceso móvil en el modelo librería

De forma análoga la Figura 5.19 muestra en fragmento de reemplazo asociado al control de acceso móvil. En este caso se seleccionan los casos de uso crear usuario, borrar usuario, actualizar usuario, los actores usuario y usuario identifica y las relaciones correspondientes.

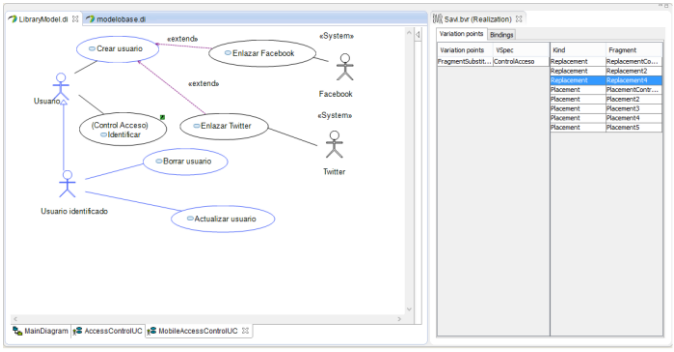


Figura 5.19 Definición del ReplacementFragment para el control de acceso móvil en el diagrama de casos de uso de control de acceso móvil en el modelo librería

5.2 Aproximación tecnológica para la derivación de los requisitos del producto

Con la aproximación tecnológica utilizando en el lenguaje CVL que se presenta en esta sección es posible resolver los puntos de variación de los requisitos y derivar una especificación para un producto específico gracias al DSDM.

Primeramente se define una *configuración del producto*. Esta configuración es un subconjunto de características del modelo de características de la LPS.

Este proceso está compuesto por dos transformaciones de modelos. Con la primera *obtendremos un modelo de resolución CVL* a partir de una configuración del producto. En la segunda transformación *obtendremos la especificación del producto mediante la transformación CVL* que a partir del modelo de resolución y la especificación de requisitos del dominio nos permite derivar los requisitos del producto.

Por último se *especifican los requisitos delta* en los casos que sea necesario.

5.2.1 Definición la configuración del producto

Para derivar los requisitos de un producto es necesario definir una configuración válida del producto compuesto por los requisitos funciones que el producto debe cumplir. Para dar soporte a la tarea de configuración se emplea un modelo de configuración sobre el que seleccionan las características. Posteriormente se da soporte tecnológico para validar dicha configuración mediante la integración de la herramienta FaMa (ISA 2011).

5.2.2 La configuración del producto

El primer paso para lleva a cabo la derivación los requisitos del producto consiste en definir una configuración de producto. La configuración del producto es un subconjunto del modelo de características de LPS que cumple las restricciones de integridad. A nivel tecnológico la configuración del producto consistirá en un conjunto de características seleccionadas de la vista de variabilidad del multimodelo. Para poder seleccionar características se define en la metaclase *EFeature* un atributo booleano llamado *selected*. Para ilustrar esta aproximación seguiremos con el ejemplo basado en la LPS Savi. La Figura 5.21 muestra un ejemplo para la configuración de este producto.

El proceso de configuración del producto se soporta mediante la infraestructura definida para la edición de multimodelos. Mediante el framework EMF y el editor por defecto de ficheros XMI podemos acceder a la vista de varabilidad

para definir la configuración del producto. La Figura 5.20 muestra la definición de la configuración en el multimodelo. La entidad EFeature del multimodelo contiene el atributo select que nos permite seleccionar mediante true o false las características de la vista de variabilidad que seleccionaremos.

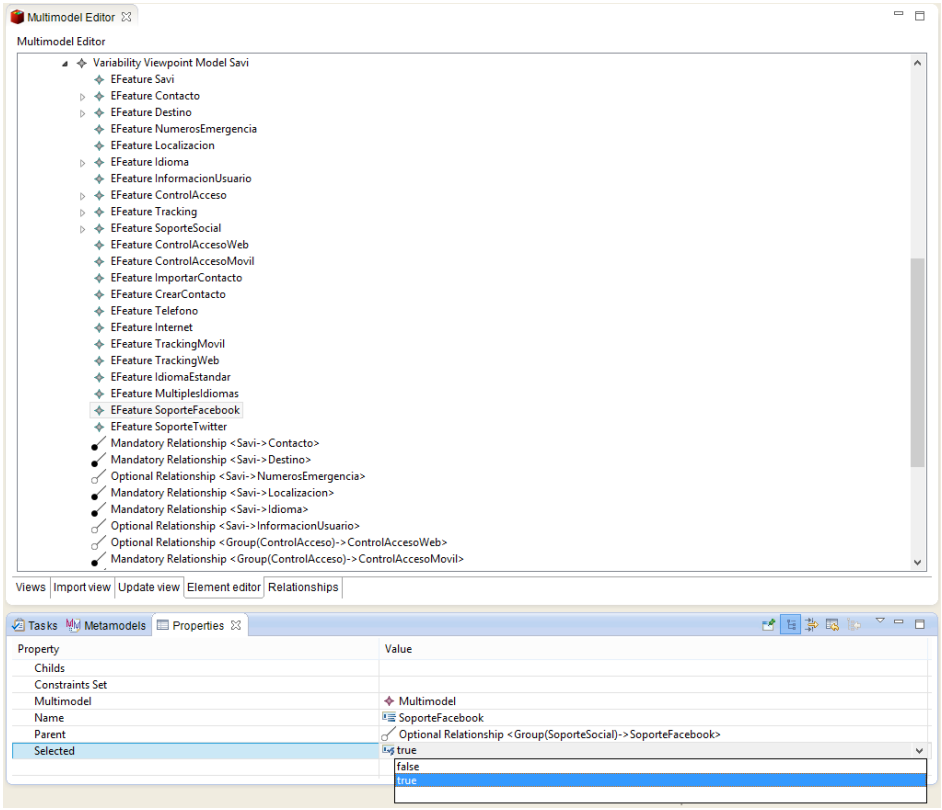


Figura 5.20 Edición del multimodelo para definir la configuración de producto

5.2.3 Validación de la configuración del producto

En la vista del multimodelo de características se define como configuración válida a aquella que satisface las restricciones definidas en el modelo (cardinalidades, relaciones de exclusión, implicación, etc.). Formalmente se define un *modelo de características* Mc como un conjunto de configuraciones es donde $M_c \subseteq C$, siendo C el conjunto de posibles combinaciones de características (Janota y Botterweck 2008). La *configuración de características* o configuración del producto $C = \{c_1,c_2,..,c_n\} \in C$ se dice que es una configuración válida de Mc sí y sólo sí, $c \in Mc$ (Janota y Botterweck 2008).

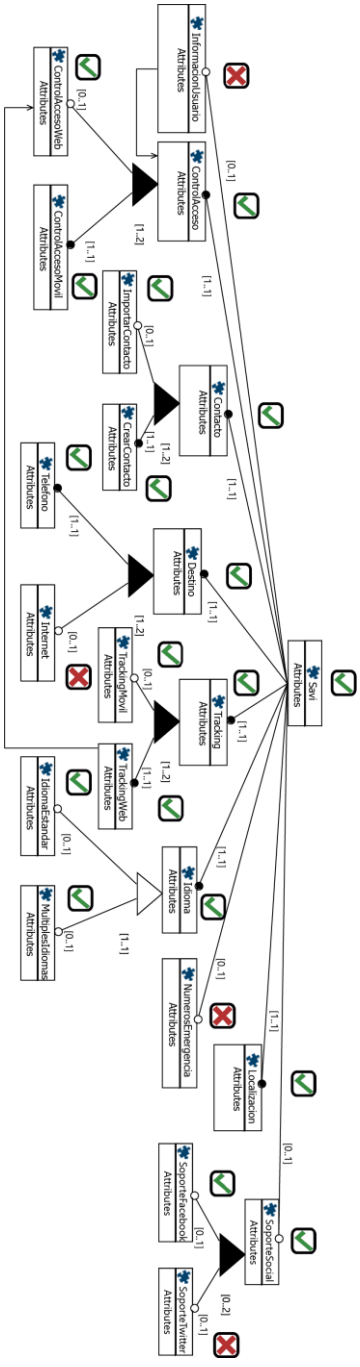


Figura 5.21 Ejemplo de configuración de producto

El análisis de la lista de variabilidad se lleva a acabo ejecutando una transformación de modelos para convertir la vista de variabilidad del multimodelo a una modelo compatible con la herramienta FaMa (ISA 2011). FaMa es una herramienta para en análisis de modelos de variabilidad basada en *Constraint Programming*. Primero se genera un modelo de variabilidad FaMa a partir de la vista de variabilidad del multimodelo mediante el conjunto de transformaciones propuestas por (Khachan 2012). En la Figura 5.22 se muestra el fichero que contiene la vista de variabilidad externa de la familia de productos para gestionar las órdenes de pago en una tienda online.

El Listado 5.2 muestra el modelo FaMa obtenido a partir de la vista de variabilidad. Se indica de forma tabulada el nombre de la característica y a continuación con dos puntos la lista de características hijas con la cardinalidad en caso de que proceda. A continuación se muestra un listado con las restricciones sobre el modelo de características.

Listado 5.2 Modelo FaMa obtenido

```
%Relationships
Savi: Contacto Destino [NumerosEmergencia] Localizacion Idioma
[InformacionUsuario] ControlAcceso Tracking [SoporteSocial] ;
ControlAcceso: [1,2] { ControlAccesoWeb ControlAccesoMovil };
Contacto: [1,2] { CrearContacto ImportarContacto };
Destino: [1,2] { Telefono Internet };
Tracking: [1,2] { TrackingMovil TrackingWeb };
Idioma: [1,1] { IdiomaEstandar MultiplesIdiomas };
SoporteSocial: [0,2] { SoporteFacebook SoporteTwitter };

%Constraints
InformacionUsuario REQUIRES ControlAcceso;
TrackingWeb REQUIRES ControlAccesoWeb;
```

Una vez obtenido el modelo de características de FaMa, el otro artefacto necesario para llevar a cabo la validación de la consistencia es el modelo de la configuración del producto. Este se deriva mediante de una transformación de modelo a texto que extrae la lista de características seleccionadas en la vista de variabilidad del multimodelo. El Listado 5.3 muestra la lista de características obtenidas para la configuración del producto.

Listado 5.3 Configuración del producto derivada

```
Contacto ControlAcceso ControlAccesoMovil ControlAccesoWeb CrearContacto
Destino Idioma ImportarContacto Localizacion MultiplesIdiomas
NumerosEmergencia Savi SoporteFacebook SoporteSocial Telefono Tracking
TrackingMovil TrackingWeb
```

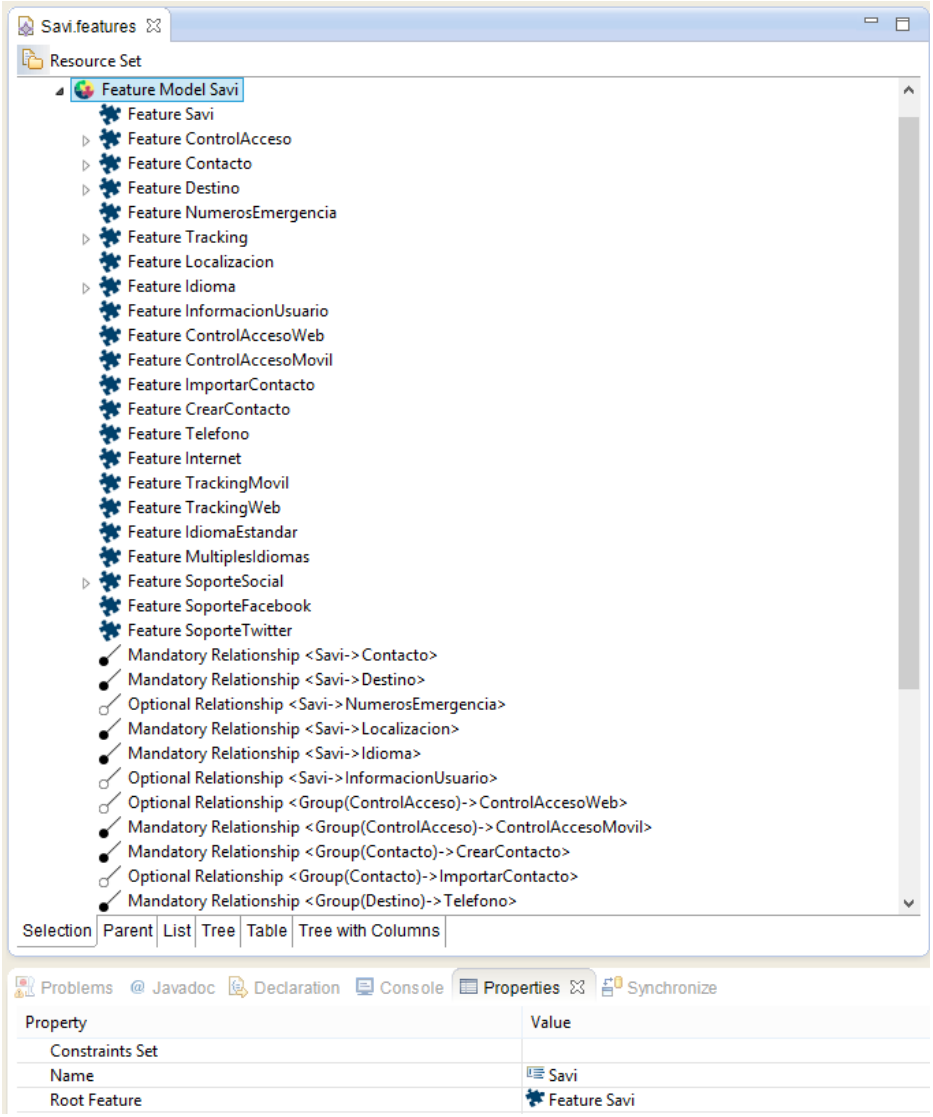


Figura 5.22 Fichero con la vista de variabilidad

El modelo de características y la configuración del producto en el formato de FaMa se utilizan para realizar una consulta a dicha herramienta con el objetivo de consultar si el producto es válido. El resultado de la herramienta es un booleano que indica *true* si la configuración del producto es válida o *false* en caso contrario. La Figura 5.23 muestra el resultado de la validación de consistencia para la configuración que se muestra en la Figura 5.21.

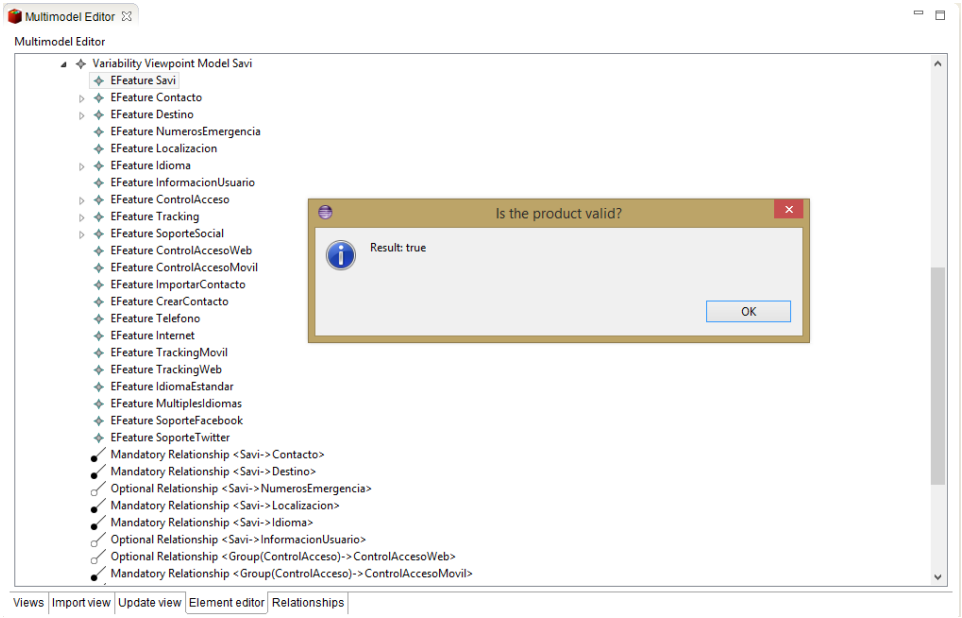


Figura 5.23 Resultado de la validación de consistencia

5.2.4 Derivación de los requisitos del producto

Esta actividad empleará dos transformaciones de modelos para derivar los requisitos del producto: obtención del modelo de resolución CVL y derivación de los requisitos de la aplicación. La primera transformación toma como entrada el multimodelo y la configuración del producto y produce un modelo de resolución CVL. Este modelo de resolución CVL se toma como entrada en la segunda transformación se toma como entrada el modelo de variabilidad CVL y el modelo base (compuesto como diagramas de caso de uso diagramas de actividad) para derivar los requisitos del producto a través de transformaciones CVL.

En las siguientes subsecciones se describe en detalle el proceso para obtener el modelo CVL que representa la variabilidad de los requisitos así como la obtención de la especificación de requisitos del producto en desarrollo.

5.2.4.1 Obtención del modelo de resolución CVL

Una vez se ha obtenido una *configuración del producto válida* para el producto el siguiente paso es obtener el modelo de resolución CVL. Esta transformación toma como entradas el modelo configuración del producto validado y el multimodelo que contiene las relaciones entre las dos vistas. Como resultado se genera un árbol de resolución CVL, el cual contiene la resolución de la

variabilidad del productor desarrollo, teniendo en cuenta las características seleccionadas en la configuración del producto.

El *modelo de variabilidad de requisitos* se representa utilizando la notación CVL en su segunda revisión (Haugen y Øgård 2014). Como ejemplo de esta notación la Figura 5.25 muestra el modelo de variabilidad de requisitos para la LPS Savi. El uso de un modelo de un modelo de variabilidad externo que representa la variabilidad de la LPs y un modelo de variabilidad interno que representa la variabilidad interna de los requisitos nos permite una mayor flexibilidad (por ejemplo el modelado de requisitos *crosscutting* (Nuseibeh 2004)). En el ejemplo de Savi la selección del *SoporteSocial* va a tener impacto en la variabilidad de más de un requisito. Primeramente afectará al control de acceso web dado que aparece unas nuevas *EChoice* hijas para la *EChoice ControlAcceso*: *ControlAccesoFacebook* y *ControlAccesoTwitter* que se resolverán positivamente dependiendo de que se selecciones las *EFeature SoporteFacebook* y *SoporteTwitter*. De forma análoga en el caso de que se seleccionen las *EFeature SoporteFacebook* y *SoporteTwitter* se resolverán positivamente las dos *EChoice ImportarFacebook* e *ImportarTwitter*.

Por otra parte el multimodelo contiene la trazabildad entre las *EFeatures* seleccionadas de la vista de variabilidad con los *EChoices* de la variabilidad de requisitos tal como muestra la Figura 5.24. Cada vez que se seleccione una *EFeature* con el campo *selected* como true se resolverá positivamente el *EChoice* seleccionado.

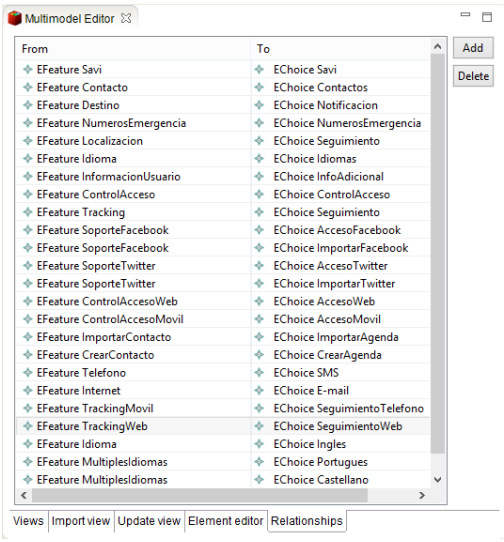


Figura 5.24 Relaciones en el multimodelo entre la variabilidad externa e interna

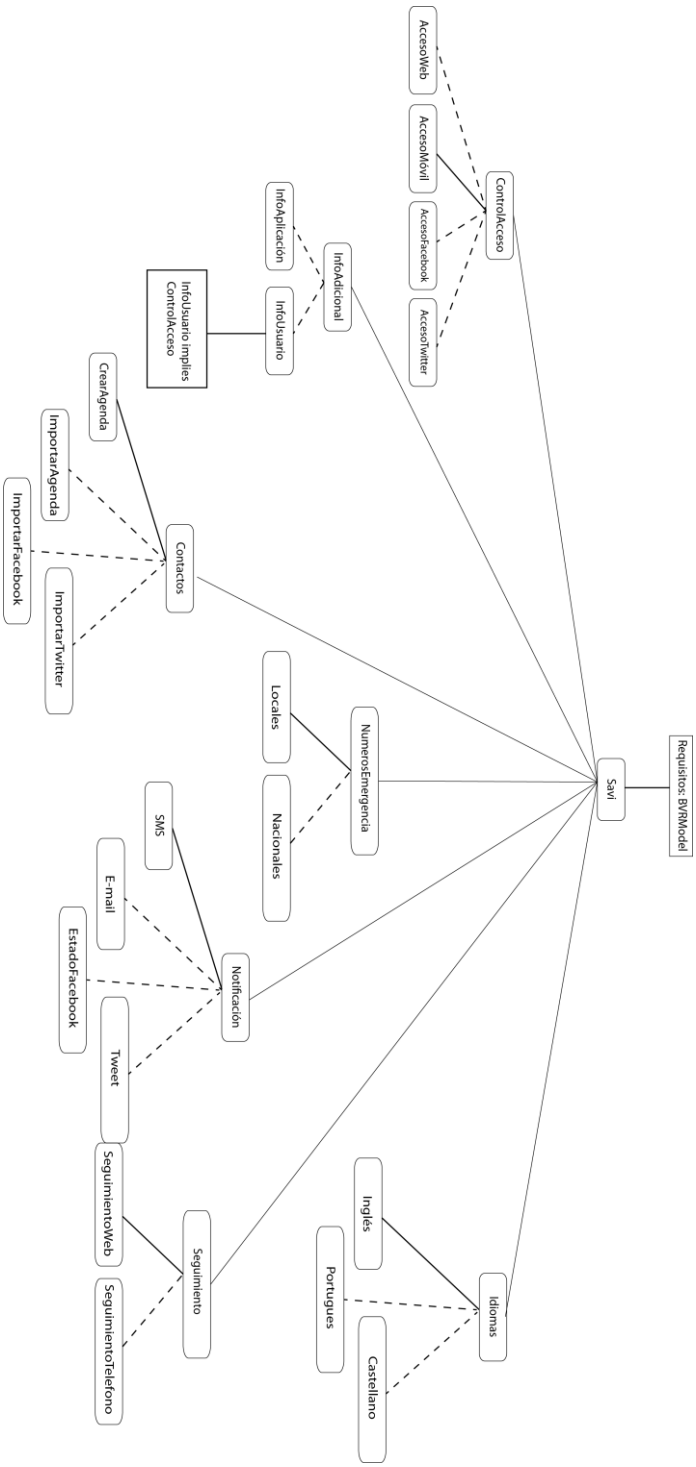


Figura 5.25 Modelo de variabilidad de requisitos de la LPS Savi

Finalmente se obtiene el modelo de resolución CVL. La Figura 5.26 muestra el modelo contextual del editor para crear el modelo de resolución para la configuración que hemos definido en el multimodelo (Figura 5.26).

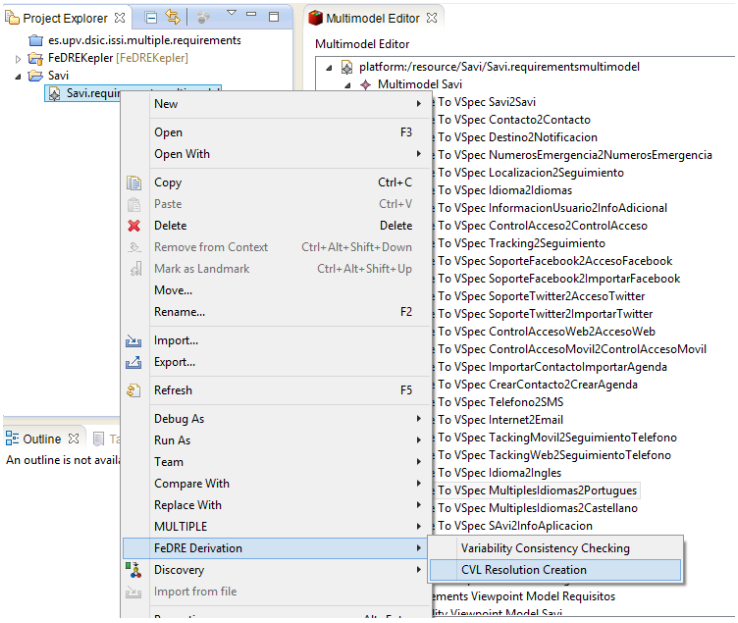


Figura 5.26 Creación modelo resolución

Este modelo tiene una estructura en forma de árbol similar al modelo de variabilidad de CVL. La Figura 5.27 muestra el modelo de resolución obtenido. Esta estructura puede observarse si abrimos el modelo BVR con el editor por defecto de EMF tal como se muestra en la Figura 5.28. El modelo BVR puede contener de forma opcional uno o más modelos de resolución. En este caso el elemento raíz del modelo de resolución es la *PosResolution* Savi. Esta contiene varios *PosResolution*, siguiendo la estructura del modelo de variabilidad CVL: uno de ellos es ControlAcceso. ControlAcceso conocen tres *PosResolution*: AccesoWeb, AccesoMovil, AccesoFacebook. Sin embargo se resolverá negativamente AccesoTwitter dado que la *feature* SoporteTwitter no ha sido seleccionada en la configuración del producto.

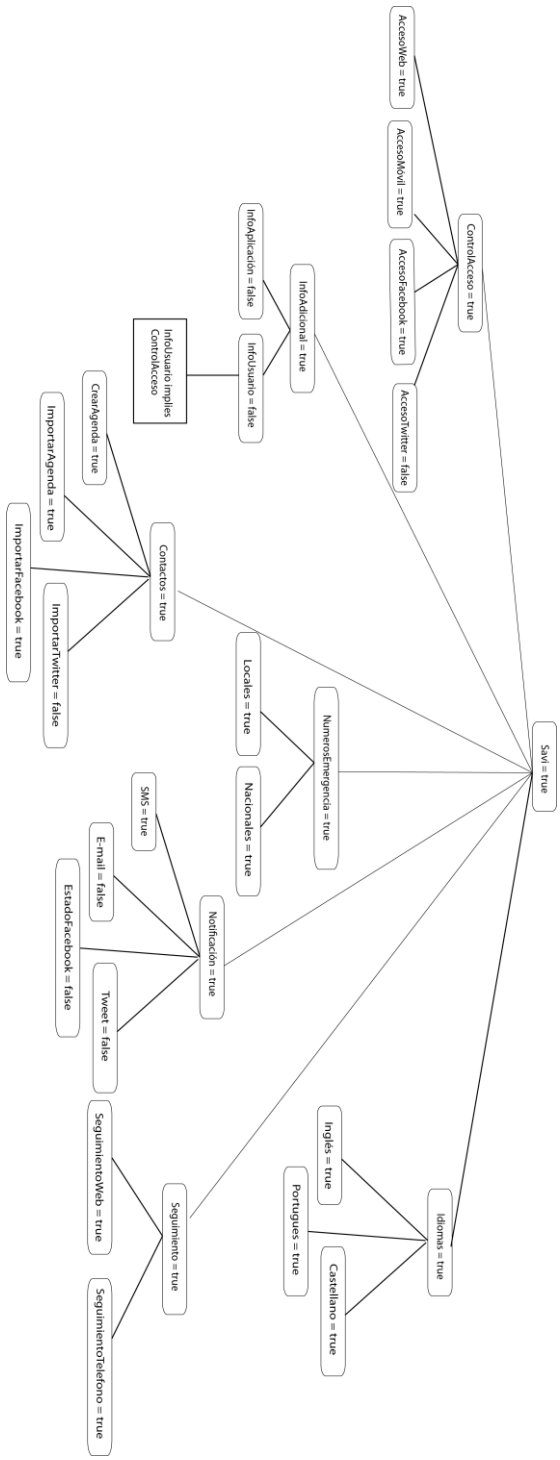


Figura 5.27 Modelo de resolución obtenido para el producto

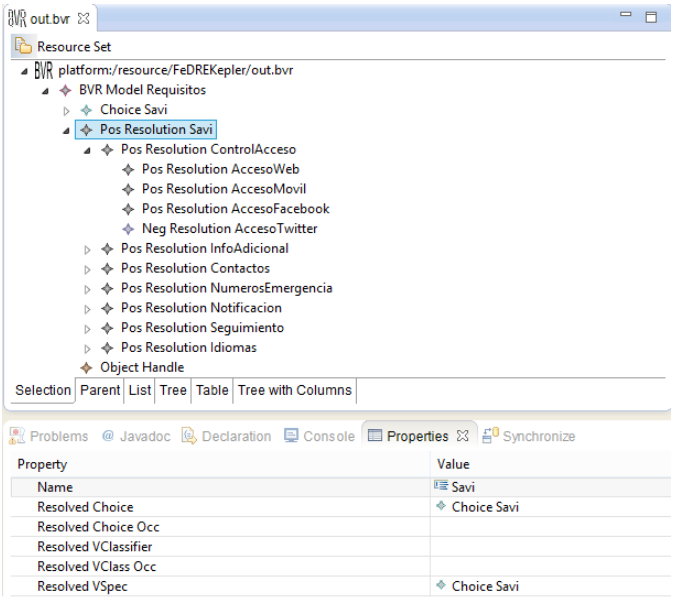


Figura 5.28 Modelo de resolución generado en un editor en árbol

5.2.4.2 Validación del modelo de resolución CVL

Una vez obtenido el modelo de resolución puede validarse para comprobar para que es correcto utilizando el editor de CVL (ver Figura 5.29). El modelo de resolución debe de satisfacer la condición de que el número de *VSpec* resueltos sea igual al número de *VSpec* del modelo de variabilidad BVR. El Listado 5.4 muestra la restricción OCL que debe de cumplir un modelo de resolución CVL. El resultado será un mensaje de indicando true (ver Figura 5.12) o por el contrario un mensaje de error indicando el número del *VSpec* que debe de resolverse para satisfacer la restricción de integridad.

Listado 5.4 Restricción OCL de integridad del modelo de resolución

```
OCL :
context VSpecResolution
inv VSpecResChildrenCorrespondsToVSpecChildren :
self.childResolution->size() >= self.resolvedVSpec.childVSpec->size()
and
not (self.resolvedVSpec.childVSpec->exists (vSpec | not
(self.childResolution->exists(vRes | vRes.resolvedVSpec == (VSpec)))))
```

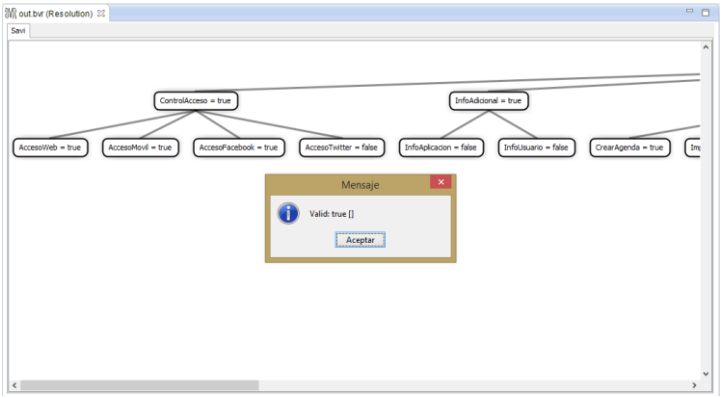


Figura 5.29 Validación del modelo de resolución CVL obtenido a partir de la transformación

5.2.5 Derivación de los requisitos del producto

Esta transformación toma como entrada el modelo resolución CVL generado en la transformación anterior y la especificación de requisitos del dominio sobre la que se definió la variabilidad haciendo uso a la transformación CVL. En esta transformación se resuelven los puntos de variabilidad (ver sección 5.1.3) y los *binding* que se definieron previamente durante la ingeniería del dominio. La transformación CVL se ejecuta desde el editor de resoluciones de CVL tal como muestra la Figura 5.30.

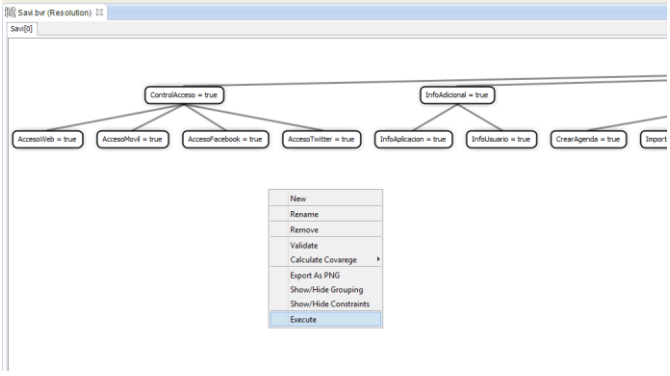


Figura 5.30 Ejecución de la transformación CVL

Como resultado se nos genera un modelo UML realizando las sustituciones necesarias en el modelo base dependiendo de las *VSpec* resueltas positivamente en el modelo de resolución generado a partir de nuestro multimodelo. La Figura 5.31 muestra el modelo UML resultante tras aplica la transformación CVL para el control de acceso y control de acceso móvil.

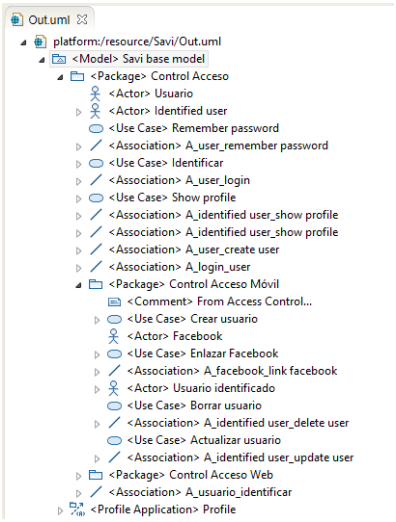


Figura 5.31 Modelo resuelto para el producto

5.2.6 Especificación de los requisitos deltas

Una vez derivada la especificación de requisitos podría ser necesario especificar los requisitos deltas. Para dar soporte a la especificación de los requisitos deltas en esta aproximación tecnológica se utiliza la herramienta de modelado Papyrus. La Figura 5.32 muestra el diagrama de perfiles donde se define el perfil UML con la herramienta Papyrus y que fue presentado en el proceso FeDRE (sección 4.4.3.1).

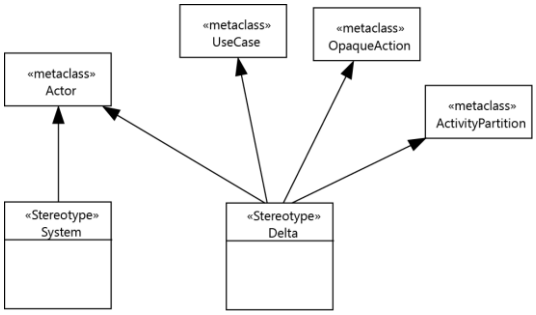


Figura 5.32 Profile de UML definido en Papyrus

El diagrama de perfiles UML es un diagrama de estructura que define estereotipos, etiquetas, valores y restricciones personalizadas. Los perfiles permiten adaptar o personalizar metamodelos existentes con construcciones específicas para un dominio particular. Los perfiles pueden aplicarse dinámicamente a un modelo. La Figura 5.33 muestra la aplicación del perfil FeDRE a un modelo de requisitos.

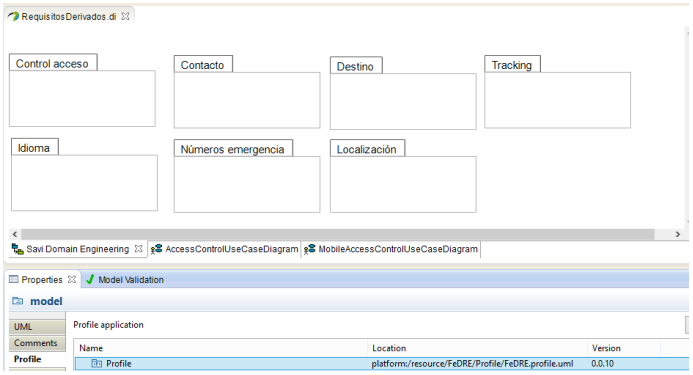


Figura 5.33 Aplicación del perfil FeDRE a un modelo de requisitos

Los perfiles definen estereotipos, tipos de datos, tipos primitivos y enumeraciones. Los estereotipos son clases de perfiles que definen cómo se extiende una metaclase existente como parte de un perfil. En FeDRE definimos dos estereotipos: el estereotipo «system» para representar sistemas externos y el estereotipo «delta» para representar requisitos delta tal como muestra la Figura 5.32.

Los estereotipos siempre extienden otras metaclases. La extensión de una metaclase por un estereotipo se representa por la relación *extensión*, la cual es un tipo de asociación que se usa para indicar que las propiedades de una metaclase se extienden a través de un estereotipo. La notación de una *extensión* se representa con una flecha con un triángulo de color negro que parte del estereotipo a la metaclase extendida. La Figura 5.32 contiene una relación de extensión que parte del estereotipo «system» a la metaclase *actor*. A su vez el estereotipo «delta» se aplica a las metaclases *actor, use case, opaque action y activity partition*. Finalmente durante la especificación de requisitos aplicamos los estereotipos en Papyrus seleccionando sobre el elemento el qué estereotipo aplicar como muestra la Figura 5.34.

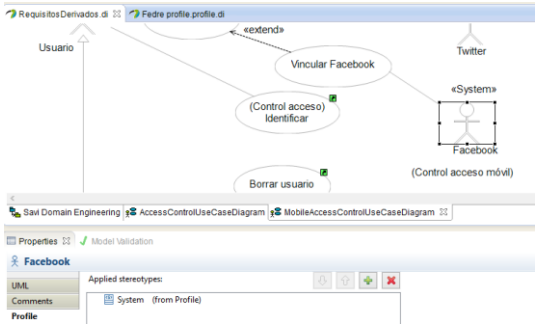


Figura 5.34 Aplicación del estereotipo «system» a un actor

5.3 Conclusiones

En este capítulo se ha descrito la aproximación tecnológica que soporte el proceso FeDRE. La aproximación soporte los dos procesos de una LPS: ingeniería del dominio e ingeniería de la aplicación.

Durante la ingeniería del dominio se da soporte a la especificación de los requisitos del dominio mediante un multimodelo. Este multimodelo está compuesto de dos vistas: la vista de variabilidad externa que se representa mediante un modelo de características. Por otra parte la vista de variabilidad interna representa la variabilidad de la especificación de requisitos del dominio. La vista de variabilidad se representa utilizando en lenguaje CVL. El multimodelo contiene las entidades de estas dos vistas y la trazabilidad entre las *Features* de la vista de variabilidad externa y las *Choices* del modelo de variabilidad interna.

Durante la ingeniería de la aplicación se da soporte para derivar los requisitos del producto a partir de la configuración de un producto. La configuración del producto se realiza seleccionando un conjunto de *EFeatures* del multimodelo. Una vez definida la configuración de producto, se valida para comprobar que se satisfacen las restricciones del modelo de variabilidad, mediante un *plugin-in* que se basa en FaMa. Una vez definida la configuración del producto se derivan los requisitos del producto a partir de dos transformaciones. En la primera transformación se crea un modelo de resolución CVL que contiene el conjunto de *Choices* que se resuelven positivamente del modelo de variabilidad de CVL. Una vez obtenido el modelo de resolución se valida mediante el validador de CVL para comprobar que todas las *Choices* del modelo de variabilidad de CVL se han resuelto positivamente o negativamente en el modelo de resolución. Si una *Choice* se resuelve positivamente implicará que se realizará una sustitución en el modelo base por un elemento del modelo librería. Una vez realizada la transformación de CVL se realizan los cambios necesarios en el modelo base para obtener finalmente un modelo UML que contiene la especificación de requisitos del producto. Por último hemos definido un perfil UML que nos permite especificar los requisitos delta sobre la especificación del producto derivada con la herramienta Papyrus.

Capítulo 6. Una herramienta para la definición y especificación de los requisitos de una LPS y la derivación de requisitos de productos

En esta sección se presenta el prototipo implementado para dar soporte la instanciación de la especificación de requisitos de un producto a partir de la especificación del dominio. Esta herramienta se implemente como un conjunto de *plug-ins* en el marco de trabajo *Eclipse Modelling Framework*.

La sección 6.1 presenta la problemática a resolver por la herramienta.

La sección 6.2 presenta la implementación de la arquitectura del metamodelo basada en el metamodelo conceptual presentado en la sección 5.1.1.

La sección 6.3 muestra la infraestructura definida para el soporte para la creación de editores para multimodelos de requisitos. Además se muestra el editor implementado para editar instancias de requisitos que implementan la arquitectura definida en el punto anterior. Por último se muestran los *plug-ins* implementados para soportar la derivación de requisitos.

La sección 6.4 presenta las conclusiones de este capítulo.

6.1 Introducción

Los sistemas software nunca permanecen estáticos a lo largo del tiempo. Esto implica que es preciso añadir nueva funcionalidad como demanda a nuevos requisitos funcionales o realizar cambios debido a diferentes causas (mantenimiento, migraciones, etc.). Una de las consecuencias del cambio continuo en un sistema representado por un multimodelo es que no solo cambian los elementos de internos de una vista y las relaciones de elementos de una de las vistas con elementos de otras vistas; sino que además puede incluso que pueda requerirse la incorporación y/o eliminación de vistas del sistema.

Aunque inicialmente el multimodelo para representar los requisitos de una LPS contiene dos vistas es preciso realizar una implementación lo suficientemente flexible para incluir otras vistas (por ejemplo añadir una nueva vista de requisitos no funcionales y permitir establecer relaciones hacía la vista de variabilidad de los requisitos funcionales).

Sin embargo dar solución en este contexto no siempre es fácil utilizando las herramientas del marco tecnológico EMF. Eclipse define Ecore, una implementación del estándar *Essential MOF* (EMOF) (OMG 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). Este hecho resulta complicado utilizando las herramientas tradicionales de EMF donde cada vez que se realiza un cambio en un metamodelo deja inservibles las instancias que se habían generado previamente,

Así pues para conseguir la gestión de un gestor de multimodelos que representen un sistema 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.

Todas estas operaciones deben comportarse como una transacción de modo que tras las modificaciones se mantenga la consistencia de todas las instancias. 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.
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 implementación del multimodelo (el multimodelo conceptual se presenta en el capítulo) se basa en el soporte para trazabilidad propuesto para la herramienta de gestión de modelos MOMENT (Gómez y otros 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.

6.2 Implementación de la arquitectura del multimodelo

La arquitectura definida para soportar nuestra infraestructura se basa en dos capas. Por un lado, se ha definido un metamodelo del núcleo del multimodelo (*Core Multimodel*) que soporta la implementación en *Eclipse Modeling Framework* (EMF) (Steinberg y otros 2008). A la hora de definir un multimodelo concreto se extenderán las metaclases de este metamodelo *core*. Esta arquitectura ha sido definida con el fin de automatizar la generación de editores de multimodelos de requisitos. Además, se ha desarrollado una infraestructura 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.

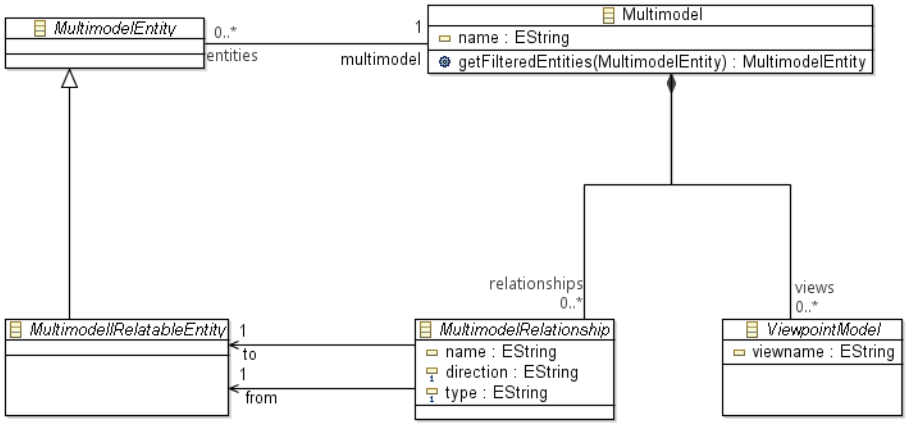


Figura 6.1 Metamodelo del multimodelo core

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 se referencian como recursos externos en el multimodelo. Para minimizar el acoplamiento entre los metamodelos, se aplica el patrón proxy (Gamma y otros 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 la metaclase *ViewpointModel* y de la metaclase del metamodelo externo que representa el modelo en dicho metamodelo externo.

Para cada clase que participa en una relación crearemos una metaclase *proxy* (Gamma y otros 1995). Estas clases también heredan de la metaclase *RelatableEntity* que soporta la definición de las relaciones entre elementos de las distintas vistas.

Como se ha comentado anteriormente el usuario puede hacer uso de un meta-modelo que hereda del multimodelo básico o Core. Para crear un meta-modelo de un multimodelo personalizado por parte del usuario tiene que crear un nuevo meta-modelo mediante el asistente EMF. Este asistente tiene que heredar del metamodelo básico o *Core*.

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 entre las definidas en (González 2014). 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.

6.3 Una infraestructura para generar automáticamente editores de requisitos en LPS

Para dar soporte a la definición de multimodelos de requisitos se define una infraestructura para editar y gestionar multimodelos que implementan la estructura genérica del multimodelo. Primero el usuario crea un metamodelo personalizado para soportar el multimodelo. Una vez definido el metamodelo del multimodelo personalizado se genera automáticamente el código el editor mediante las utilidades de generación de código de EMF.

6.3.1 Generación de editores de multimodelos

El usuario puede registrar metamodelos que representan vistas en el metamodelo personalizado. El usuario puede incluir como vista cualquier metamodelo basado en EMF que represente una vista. El usuario carga las vistas necesarias, las entidades y establece las relaciones necesarias adaptándose al dominio del problema. En la Figura 6.2 se muestra el metamodelo definido para soportar el multimodelo de requisitos.

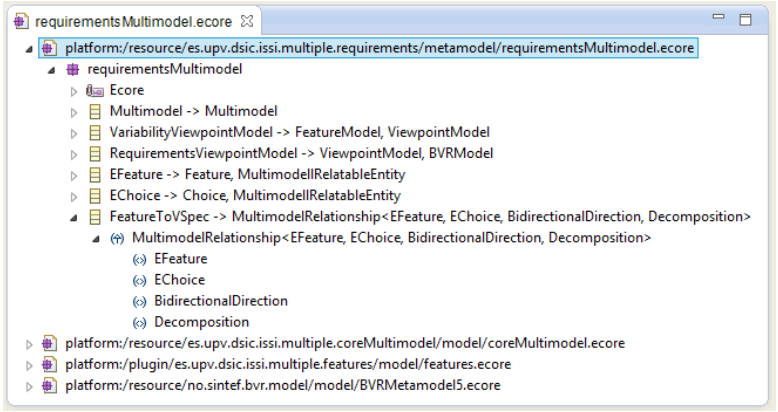


Figura 6.2 Definición del multimodelo personalizado

Una vez se crea el metamodelo personalizado, el usuario podrá generar el código necesario para editar multimodelos que sean instancia del metamodelo del multimodelo personalizado. La Figura 6.3 muestra el generador definido para crear el multimodelo personalizado.

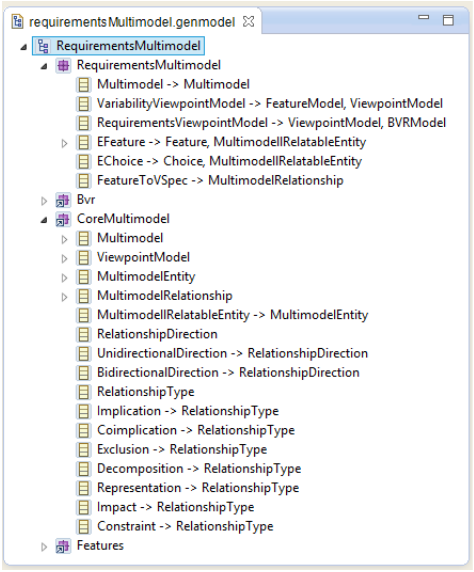


Figura 6.3 Generador del multimodelo personalizado

Una vez se ha construido el metamodelo del multimodelo personalizado el siguiente paso es registrar los elementos de este. La razón de tener que realizar el registro reside en la necesidad de indicarle al editor del multimodelo de qué clase de vistas este soporta, qué tipo de entidades contiene y qué tipo de relaciones pueden instanciarse en un modelo. Este mecanismo se basa en el mecanismo propuesto por Eclipse de puntos de extensión y extensiones, el cual se describe a continuación brevemente. Un *plug-in* en Eclipse puede declarar extensiones y puntos de extensión. Este marco tecnológico basado en la extensión da soporte a la extensibilidad, y es uno de los fundamentos principales en los que se sustenta la plataforma Eclipse (Hargrave y otros 2005). Este soporte se basa en dos conceptos: el punto de extensión y la extensión.

Un *punto de extensión* se define en un *plug-in* con el *namespace* de este *plug-in*. Esto quiere decir que el nombre del *plug-in* define un *namespace* para los puntos de extensión que define el *plug-in*. Un punto de extensión recibe extensiones y define un esquema de las extensiones que acepta. Este esquema define en el lenguaje *Extensible Markup Language* (XML) las características de una *extensión*. La Figura 6.4 muestra la definición del punto de extensión para el punto de extensión definido para el multimodelo Core. Este punto de extensión contiene el identificador *"es.upv.dsic.issi.multiple.coreMultimodel.custommultimodel "*.

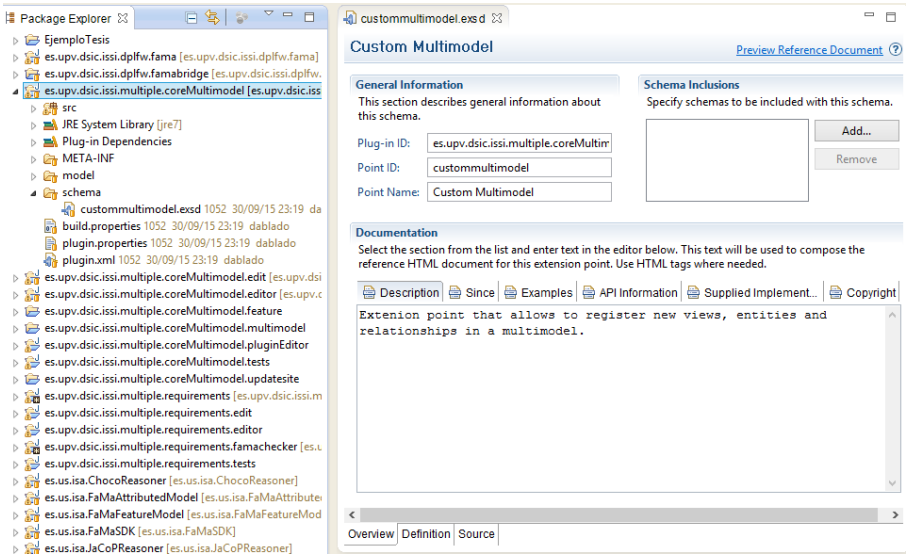


Figura 6.4 Definición del punto de extensión

Las *extensiones* de un *plug-in* se definen en el fichero *XML manifest* de un *plug-in* del proyecto que contiene el código del multimodel personalizado. En este caso nuestro proyecto llamado *"es.upv.dsic.issi.multiple.requirements"* contiene la definición del multimodelo personalizado y el código del editor. Dentro de este proyecto editamos su fichero plugin.xml y creamos la extensión que registra los elementos del multimodelo personalizado. En la Figura 6.5 se muestra la edición de la extensión.

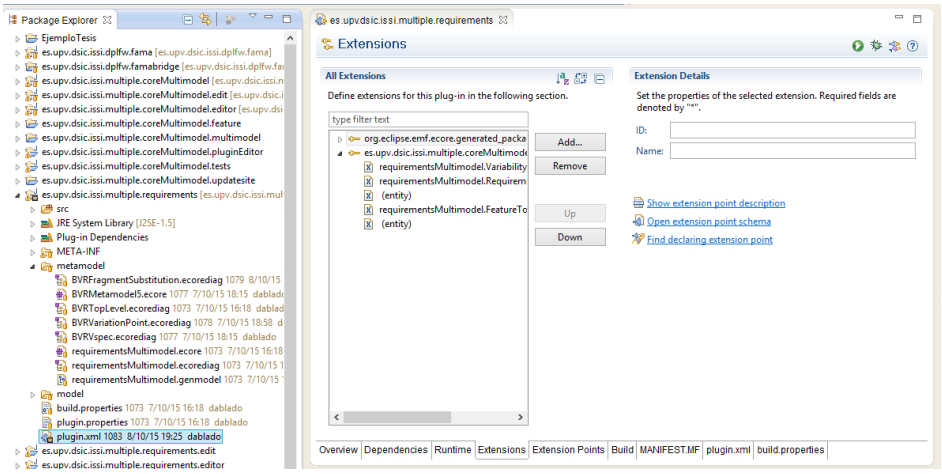


Figura 6.5 Definición de la extensión para registrar el multimodelo personalizado

Por último con el objetivo de facilitar la edición de las extensiones de un metamodelo personalizado se ha desarrollado en *plug-in* que es capaz de crear un punto de extensión tal como muestra la Figura 6.6. Este asistente abre el fichero *plugin.xml* y crea de forma automática una extensión del punto de extensión *"es.upv.dsic.issi.upv.es.coreMultimodel"* agregando las entradas necesarias para registrar las vistas, entidades y relaciones.

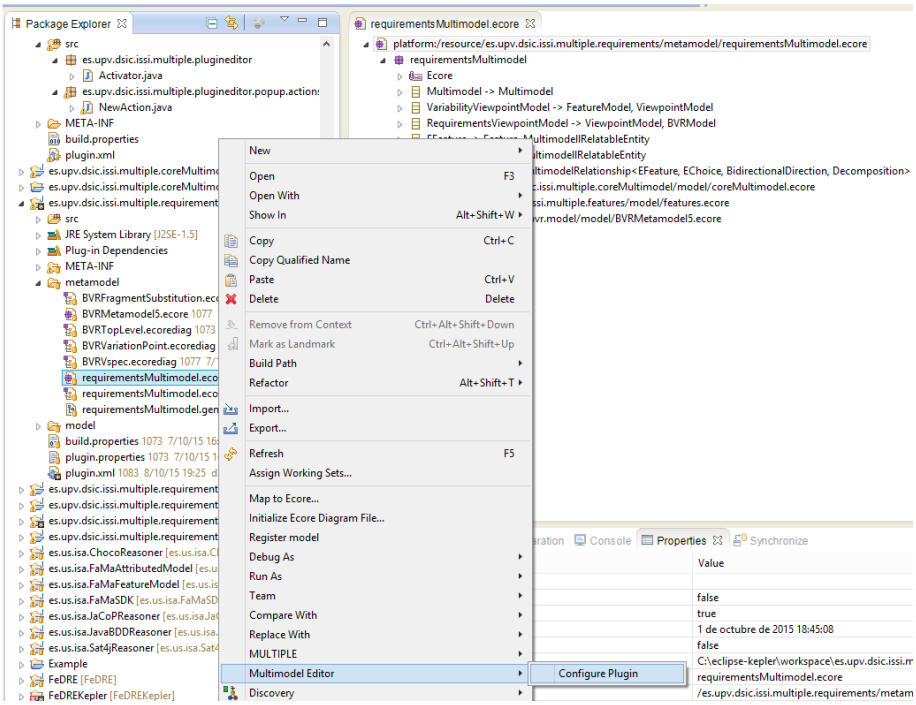


Figura 6.6 Asistente para registrar los elementos de un multimodelo personalizado

Una vez finalizada la definición, creación del código del editor EMF del multimodelo personalizado y el registro del mismo creando una extensión que extienda el punto de extensión "*es.upv.dsic.issi.multiple.coreMultimodel.custommultimodel"* podemos utilizar el editor definido para crear instancias del multimodelo personalizado para especificar los requisitos de una línea de productos.

6.3.2 Edición de multimodelos de requisitos para LPS

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. 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.

6.3.2.1 Importar vista

Podemos crear una vista a partir de cero. Para eso el editor nos permite elegir entre las vistas disponibles que se han registrado previamente en el punto de extensión *es.upv.dsic.issi.multiple.coreMultimodel.custommultimodel* de los elementos del multimodelo. En la Figura 6.7 se muestra la pestaña del editor tras pulsar el botón de añadir vista, mostrando un cuadro de diálogo con las dos vistas que hemos registrado previamente: la de variabilidad y la de variabilidad de requisitos.

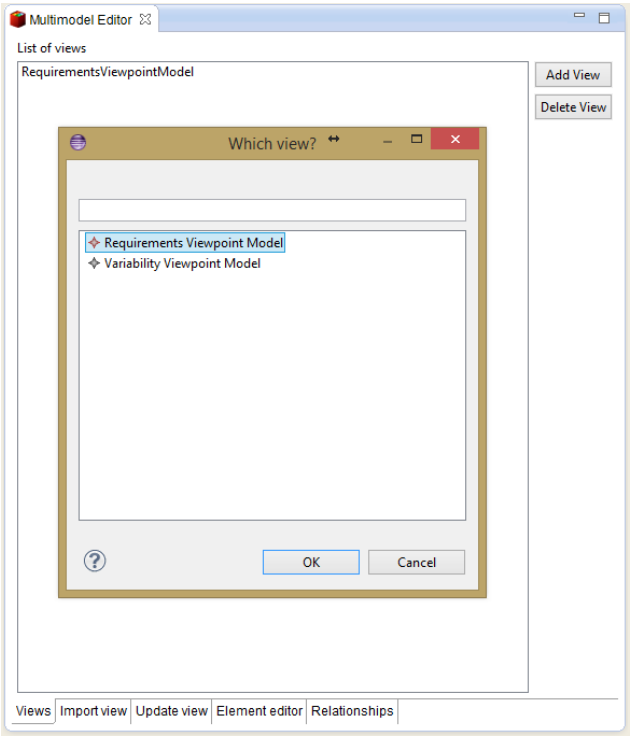


Figura 6.7 Vista del editor del multimodelo para la edición de vistas

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. La Figura 6.8 muestra el editor cargando un modelo con un modelo de características que se importará en la vista de variabilidad del multimodelo.

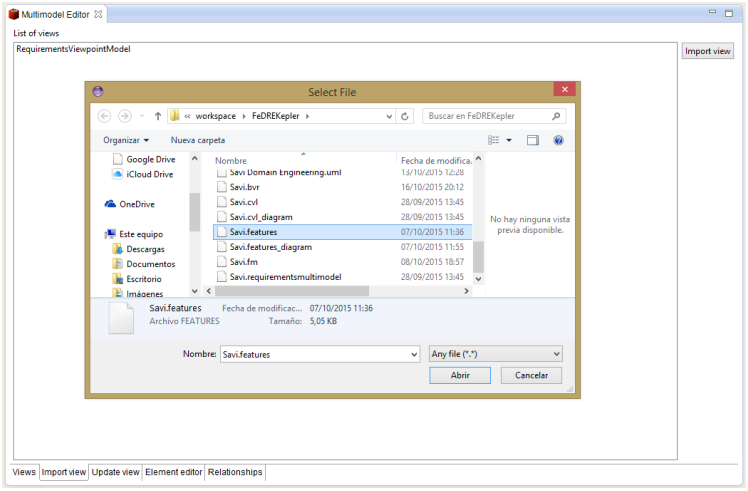


Figura 6.8 Pestaña para importar vistas en el editor del multimodelo

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). A continuación muestra en el Listado 6.1 el pseudocódigo del proceso de importación de vistas.

Listado 6.1 Pseudocódigo para importar la vista

```
VistaAImportar = CargaVista();
            if (not Multimodel.contiene(VistaAImportar){
                        VistaAdaptada = ConvierteVista(VistaAImportar);
            Multimodelo.vistas.Añade(VistaAdaptada);
}
```

El método *ConvierteVista* (Listado 6.2) 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. A continuación se muestra el pseudocódigo del método *ConvierteVista*:

Listado 6.2 Pseudocódigo para convertir la vista

```
ConvierteVista (VistaOriginal){
        VistaConvertida = CrearVistaVacía (TipoVista);
        VistaConvertida.atributos = VistaOriginal.atributos;
        VistaConvertida.características = VistaOriginal.características;
        for (VistaOriginal.entidades)
        {
                EntidadExtendida = ConvertidadEndidad(Entidad);
                if (DebeExtenderse (Entidad)
                {
                        VistaConvertida.Añadir(EntidadExtendida);
                }
        }
        return VistaConvertida;
}
```

6.3.2.2 Actualizar vista

Por último, el editor también permite actualizar una vista ya existente del multimodelo. Para ello se ha integrado una herramienta llamada *EMFCompare* (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 6.9. 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 detectados. Esto permite editar con herramientas externas modelos que han sido importados al multimodelo, aun cuando sus entidades formen parte de relaciones en dicho multimodelo. En el ejemplo hemos cambiado el nombre de la EFeature *Control_acceso* por el nombre *ControlAcceso*.

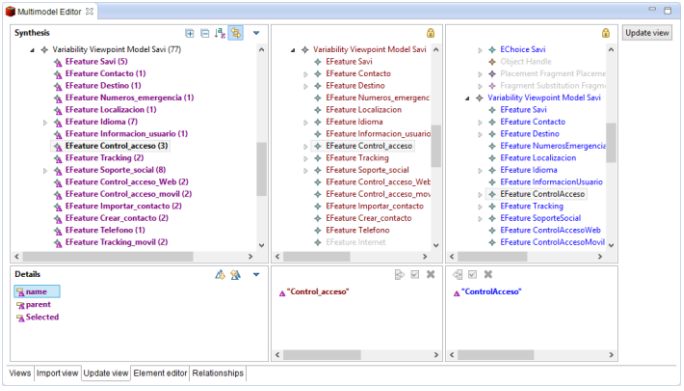


Figura 6.9 Actualización de la vista de variabilidad

Si decidimos aceptar el cambio en la *EFeature* Control_acceso en ese caso nos aparece una pantalla donde nos permite aceptar el cambio e integrarlo en el multimodelo (Figura 6.10).

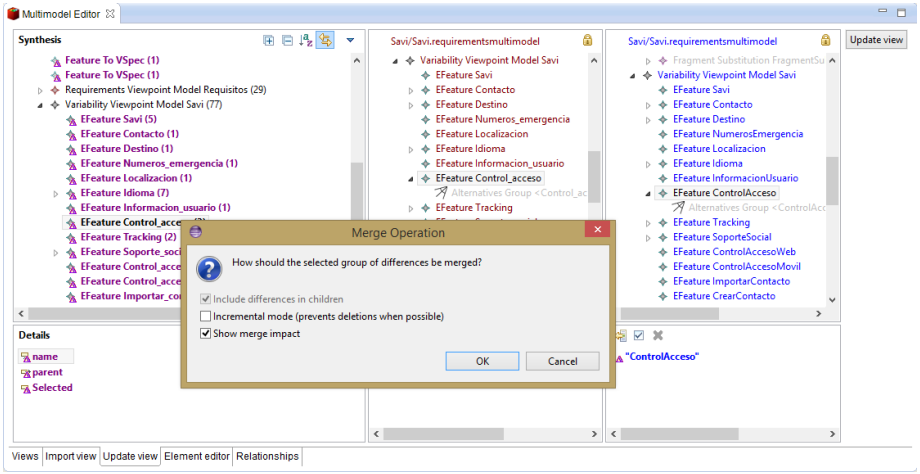


Figura 6.10 Aplicación de la operación de actualización

A continuación se nos muestran los impactos que tendrá en el multimodelo la aceptación de este cambio. La Figura 6.11 muestra los cambios requeridos para poder completar la operación. El campo *name* de la *EFeature* se va afectado por el cambio de nombre mencionado anteriormente.

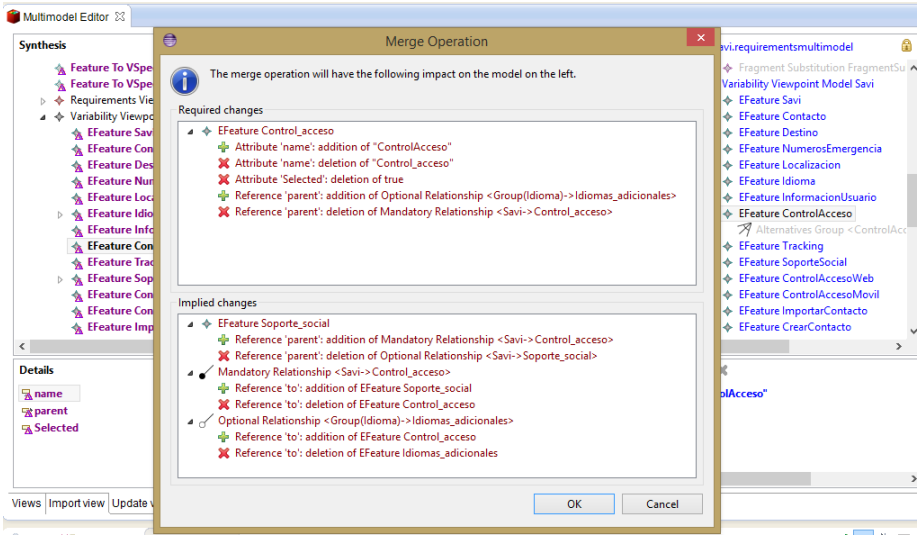


Figura 6.11 Cambios requeridos

Finalmente una vez realizados los cambios se actualiza el metamodelo. Si navegamos al editor en árbol en el multimodelo podemos observar como se ha realizado el cambio correctamente tal como muestra la Figura 6.12. Se ha realizado el cambio en la *EFeature* seleccionada.

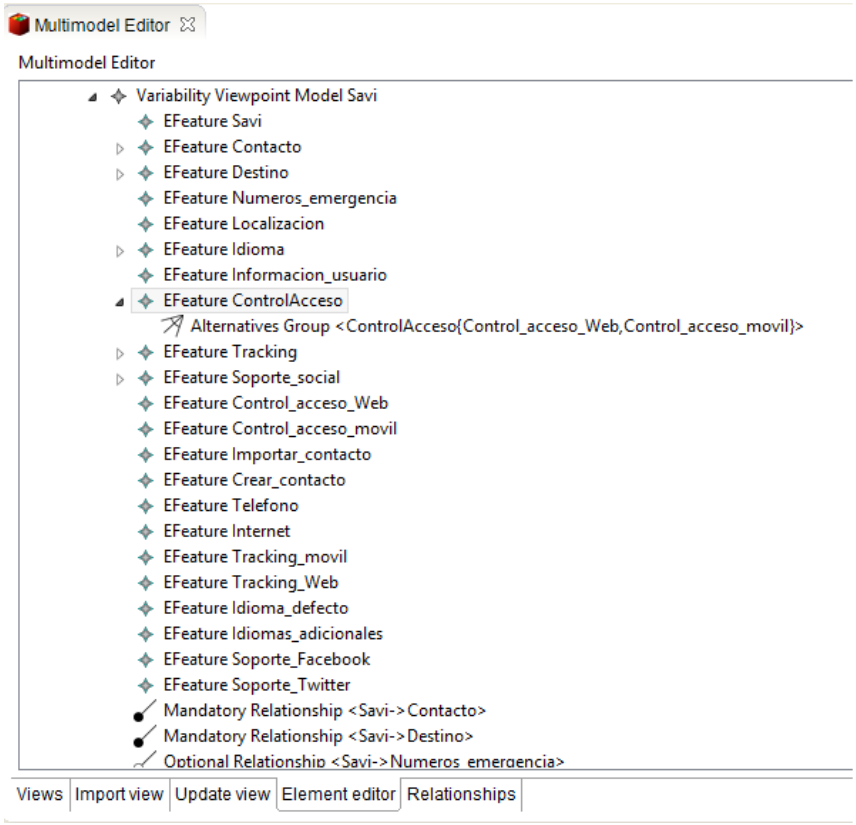


Figura 6.12 Multimodelo tras el cambio realizado

6.3.2.3 Editor en árbol

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, vistas y relaciones. La Figura 6.13 muestra la pestaña del editor en árbol del editor del multimodelo. En este caso podemos ver la instancia para representar la LPS de Savi.

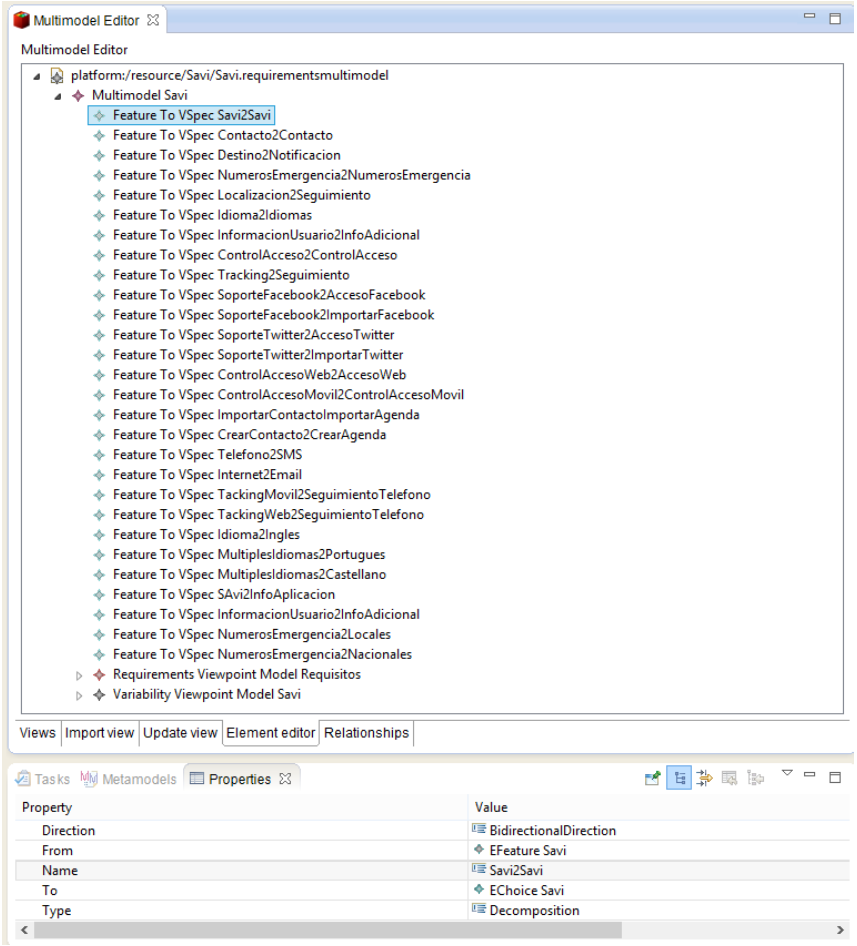


Figura 6.13 Editor en árbol del editor de multimodelos

6.3.2.4 Editor de relaciones

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. La Figura 6.14 muestra la pestaña de edición de relaciones del editor del multimodelo. Cuando se crea una relación nueva se muestra un desplegable si se hace clic con el ratón en los campos *from* o *to*. Este desplegable muestra todas las entidades del multimodelo que son del tipo especificado. Siguiendo el ejemplo de la Figura 6.14 la relación del tipo *EFeatureToEChoice* define en el campo *from* el tipo EFeature, en consecuencia se muestran todas las entidades EFeature del multimodleo.

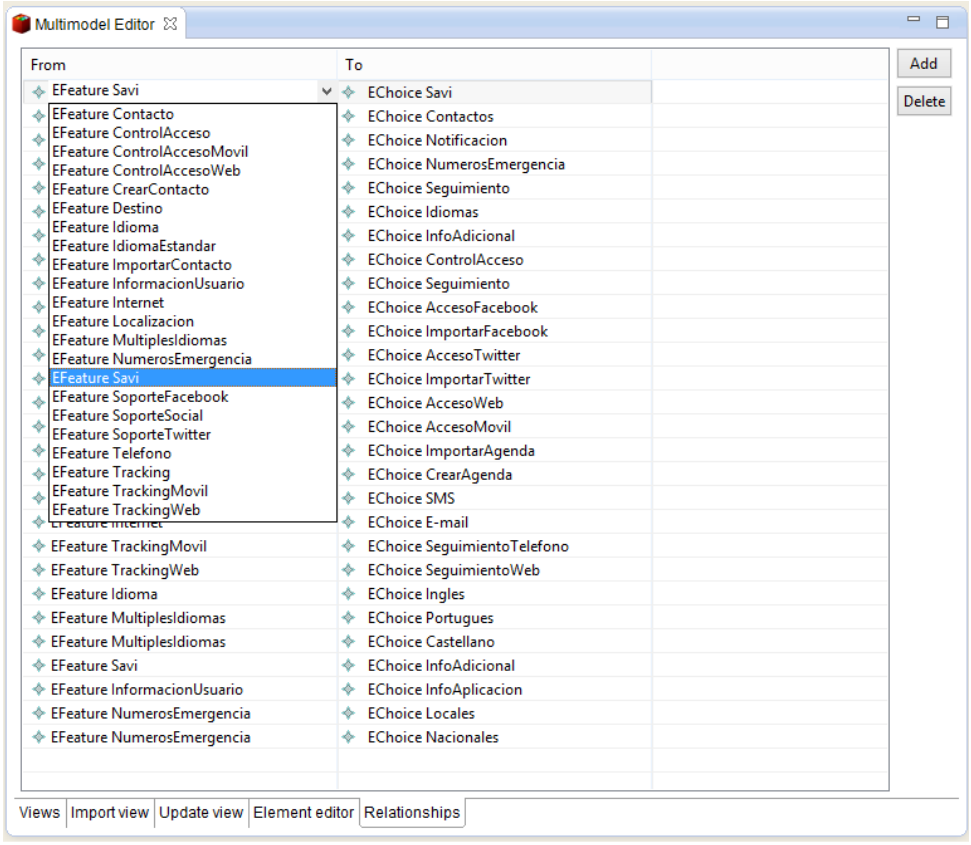


Figura 6.14 Editor de relaciones del multimodelo

6.3.3 Soporte a la derivación

Adicionalmente se implementó un *plug-in* para soportar la derivación de los requisitos del producto para dar soporte a la aproximación tecnológica presentada en la sección 5.2. Este *plug-in* provee por una parte un validador de variabilidad externa y por otra parte es capaz de producir un modelo de resolución CVL para derivar los requisitos del producto. La Figura 6.15 el menú contextual con las dos opciones de derivación implementadas en el plug-in.

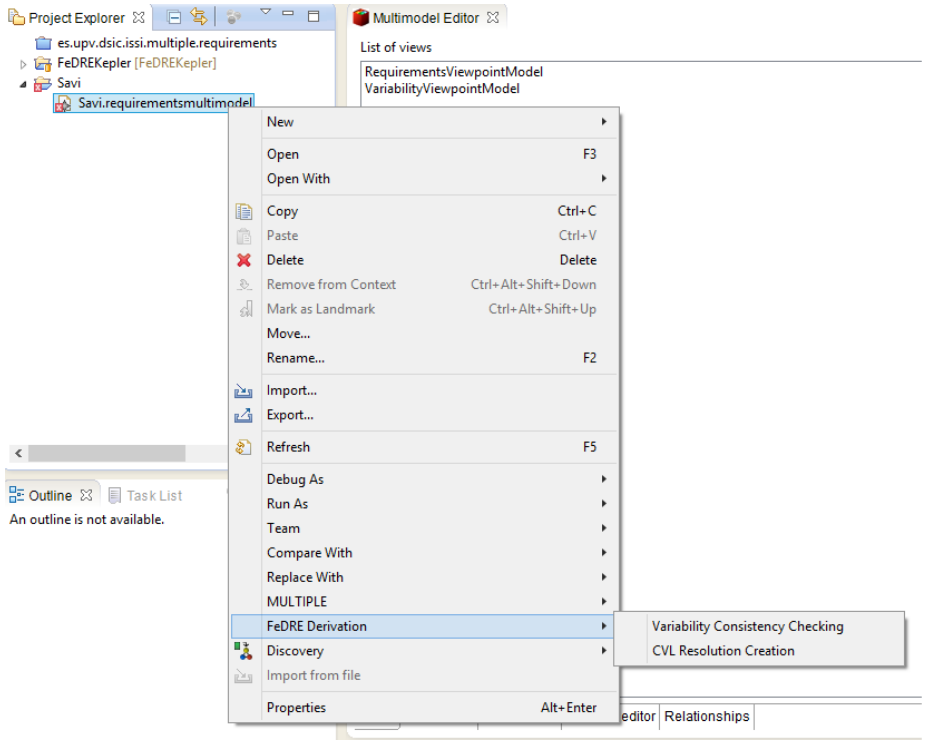


Figura 6.15 Opciones de soporte de la derivación en FeDRe

6.3.3.1 Validador de variabilidad externa

El validador de variabilidad externa se implementa para dar soporte a la aproximación tecnológica presentada en la sección 5.2.3. Este validador integra el marco tecnológico FaMa (ISA Research Group 2011). La Figura 6.16 muestra la arquitectura de FaMa la cual provee los siguientes elementos:

- El *QuestionTrader*. Son interfaces que interactúan con el núcleo de FaMa, llamado FaMa FW.
- El *núcleo* de FaMa. Comunica las diferentes extensiones entre ellas.
- *Razonadores*. Provee diferentes razonadores de terceros.
- *Operaciones*. Incorporan nuevas operaciones de razonamiento. Por ejemplo contar y listas productos, detección y explicación de errores y filtros de productos.
- *Selectores*. Seleccionan en tiempo de ejecución el razonador para realizar preguntas al razonador basados en un criterio definido previamente (rendimiento, precisión, etc.).

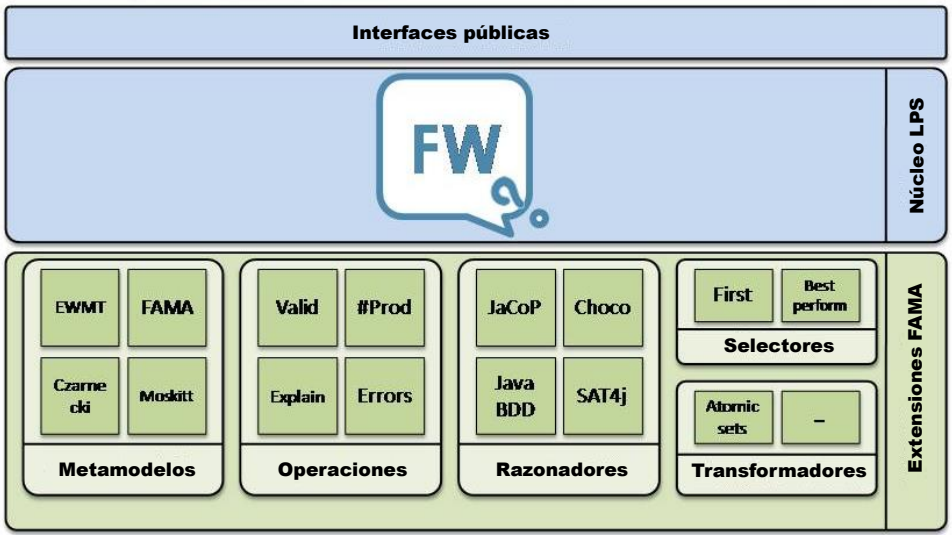


Figura 6.16 Arquitectura de FaMa

FaMa desde su versión 0.9.2 es compatible con OSGi de modo que hace posible la integración con otras herramientas. En nuestro caso se han diseñado una serie de plug-ins auxiliares que son capaces de interactuar con el razonador Choco de FaMa. El plug-in *"es.upv.dsic.issi.dplfw.fama"* se utiliza para crear los *bundles* de FaMa tal como muestra la Figura 6.17. El plug-in *"es.upv.dsic.issi.dplfw.fama"* llamada desde el plug-in *"es.upv.dsic.issi.multiple.requirements.famachecker"* que es que consumirá los servicios de FaMa invocando al *QuestionTrader*.

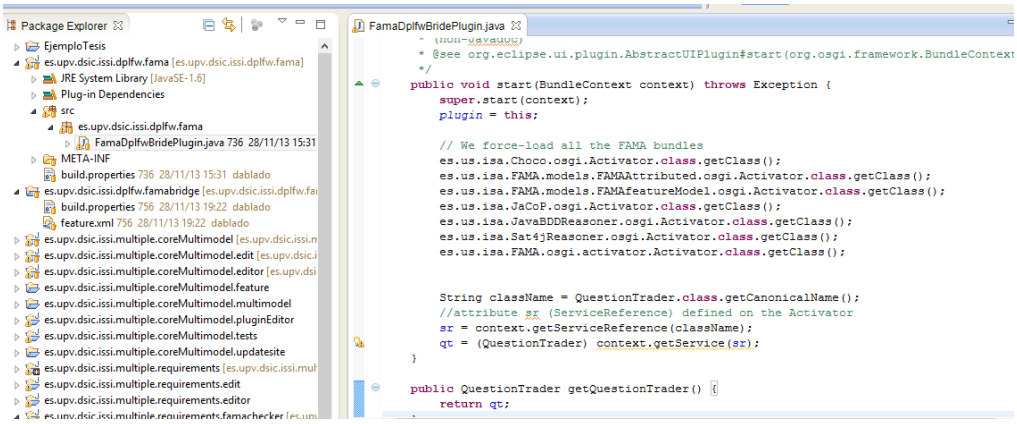


Figura 6.17 Consumo de los bundles de FaMa

El Listado 6.3 muestra el pseudocódigo el plug-in de validación. A grandes rasgos este plug-in es capaz de cargar un modelo de variabilidad a partir de la vista de variabilidad del multimodelo. Este modelo se define en la instancia *QuestionTrader* de FaMa mediante su método *setVariabilityModel(ModeloVariabildiad)*. Por otra parte se le consulta si el producto que forman el conjunto de *EFeatures* del metamodelo (tienen el campo *selected* con el valor true). Por último se realiza una consulta al *QuestionTrader* para comprobar que la configuración es válida, siendo el resultado de la consulta un valor booleano.

Listado 6.3 Pseudocódigo de la validación de la variabilidad

```
ValidaModelo(){
        Multimodelo multimodelo = cargaMultimodelo();
        QuestionTrader qt = BundelFama.obtenerQuestionTrader();
        ModeloVariabilidad mv = CargaModeloVarabiolidad();
        qt.defineModeloVariabilidad(mv);

        ConfiguracionProducto p = ExtraeCaracteristicas(multimodelo);

        PreguntaValida pv = qt.CreaaPregunta();
        pv.estableceProducto(p);
        qt.pregunta(pv);
        mostrarResultado(qt.esProductoValido);
}
```

6.3.3.2 Generación del modelo de resolución CVL

Otra de las funcionalidades del plug-in de la derivación de FeDRe es soportar la integración con la herramienta CVL para derivar los requisitos del producto a partir del multimodelo. Esta implementación da soporte a la aproximación tecnológica presentada en 5.2.4.

El Listado 6.4 muestra el pseudocódigo de la función principal este plug-. Esta función extrae la vista de variabilidad de requisitos del multimodelo. Esta vista es un modelo de CVL modificado, dado que las *Choices* del modelo original se convirtieron en *EChoices* para que pueda ser posible relacionar las *Choices* del modelo de variabilidad CVL con las *EFeatures* en el multimodelo. Para poder trabajar con la herramienta CVL y ejecutar la transformación CVL (ver capítulo 5.2.4) es necesario realizar un casting inverso y convertir todas las *EChoice* en *Choice* para que la herramienta CVL pueda trabajar con el modelo.

Una vez convertido del modelo de variabilidad CVL para poder procesarlo con el motor de transformación CVL, necesitamos crear un modelo de resolución CVL. CVL. Este modelo se basa en la configuración del producto definida en el multimodelo desde el que se invoca la derivación.

Listado 6.4 Pseudocódigo de la generación del modelo CVL

```
CrearModeloCVL(){
        Multimodelo multimodelo = cargaMultimodelo();
        VistaRequisitos vistareq = multimodelo.cargaVistaRequisitos();

        ModeloCVL modelocvl = new ModeloCVL();
        ModeloRequisitos modeloreq = convierteVistaRequisitos(vistareq);
        Modelocvl.estableceVistaRequisitos(modeloreq);
        ModeloResolucion modeloresolucion = CrearModeloResolucion(modeloreq);
        Modelocvl.listaModelosResolucion.
         añadirModeloResolucion(modeloresolucion);
        Modeloreg.setModeloRealizacion(vistaget.getModeloRealizacion);

        return modeloreq;
}
```

El modelo de resolución se crea siguiendo la misma estructura en árbol del modelo de variabilidad. CVL El Listado 6.5 muestra el pseudocódigo de este método. Se recorren todas las *Choices* del modelo de variabilidad CVL y se resuelve positivamente o negativamente dependiendo del resultado del método isSelected(Choice); el cual devolverá true si existe en el metamodelo un *EFeature* seleccionada en la configuración (campo *seleted* seleccionado como *true*) y existe una relación en desde esa EFeature al EChoice que se va a comprobar. En caso de que el resultado de la comprobación sea true, el método *AddPosResolution* añade en el modelo de resolución un elemento *PosResolution* que indica que la *Choice* del modelo de variabilidad CVL se va a resolver positivamente. En caso negativo se crea un elemento *NegResolution* llamando al método *AddNegaResolution*.

Listado 6.5 Pseudocódigo de la generación del modelo de resolución CVL

```
CrearModeloResolucion(ModeloRequisitos modeloreq){
            ModeloResolucion modelores = new ModeloResolucion();
        Iterator iterator = modeloreg.getChoices();
            while(iterator.hasNext()){
               Choice choice = iterator.next();
               if(isSelected(choice)){
                        modeloreq.addPosResolution(choice));
            }else{
                                modeloreq.addNegResolution(choice));
                     }
            }

            return modeloreq;
}
```

Como resultado se crea un modelo CVL que contiene la vista de variabilidad convertida (todos los *EChoice* se convierten en elementos *Choice*), el modelo de realización original y se le añade el modelo de resolución correspondiente a la configuración del producto del multimodelo. El fichero resultado contiene extensión "bvr" que es correspondiente a la versión 2.0 de CVL.

6.4 Conclusiones

En este capítulo se ha presentado de funcionalidad de la herramienta creada para dar soporte la aproximación tecnológica presentada en el capítulo 5.

En primer lugar, se ha utilizado una arquitectura de dos niveles donde se define un metamodelo para un multimodelo genérico o *core*, el cual contiene las primitivas básicas para representar a un multimodelo, y un metamodelo para representar el multimodelo personalizado el cual extiende el multimodelo *core* con las entidades necesarias para representar los requisitos en el dominio correspondiente.

Por otra parte se ha implementado un editor de multimodelos que permite editar cualquier multimodelo que esté basado en el metamodelo del multimodelo *core*. Este editor puede crear una vista a partir de cero (de las vistas registradas en el multimodelo) o importar una vista a partir de un modelo XMI existente compatible con EMF. Por otra parte es posible actualizar una vista del multimodelo mediante la integración el componente EMF Compare el cual nos muestra las diferencias entre el multimodel actual y los cambios que implicaría actualiza la nueva versión de la vista que queremos actualizar. La herramienta nos permite seleccionar qué cambios queremos aplicar finalmente al multimodelo. Por otra parte se integra un editor en árbol para editar los elementos del multimodelo: vistas y relaciones.

Adicionalmente, se ha implementado una infraestructura para soportar la definición de multimodelos que representan los requisitos de una LPS. En esta infraestructura se define un metamodelo personalizado con las vistas necesarias para el problema específico. Una vez definido este metamodelo para el multimodelo personalizado es posible generar automáticamente mediante las utilidades de generación de código de EMF editores de multimodelos. Aunque inicialmente el multimodelo de FeDRE utiliza sólo dos vistas para representar los requisitos la infraestructura nos permite añadir nuevas vistas en caso de que fuera necesario (por ejemplo añadir una vista de requisitos no-funcionales).

Por último se integra en el editor de multimodelos un *plug-in* para la derivación de un producto a partir de un multimodelo que realiza dos funciones: permite validar la configuración del producto definida a partir de la de variabilidad del multimodelo a partir de las *EFeatures* que están seleccionadas. La segunda funcionalidad es crear un modelo CVL a partir del multimodelo. Este modelo CVL se obtiene a partir en la vista de variabilidad de requisitos del multimodelo, añadiendo un modelo de resolución obtenido a partir de la configuración del producto actual. Este modelo de resolución puede editarse con el editor de CVL para derivar los requisitos, obteniendo un modelo UML como resultado.

Capítulo 7. Validación empírica de FeDRE

En este capítulo de presentan las sendas validaciones empíricas realizadas a ambos procesos de FeDRE.

La sección 7.1 presenta el cuasi-experimento para validar las guías de la ingeniería del dominio.

La sección 7.2 presenta el cuasi-experimento para validar las guías de la ingeniería de la aplicación.

La sección 7.3 presenta las conclusiones de este capítulo.

7.1 Cuasi-experimento para la validación de la ingeniería del dominio

En esta sección se presenta el cuasi-experimento realizado con el objetivo de analizar la utilidad de las guías de la ingeniería del dominio de FeDRE. Para conseguir este propósito se ha aplicado un cuasi-experimento aplicando las guías presentadas en Wohlin y otros (2000). Las etapas de este cuasi-experimento que se presentan en las siguientes secciones son: diseño, preparación y ejecución, recogida de datos y análisis de datos. Además se analizarán en una sección las amenazas a la validez del experimento.

7.1.1 Diseño del cuasi-experimento

En primer lugar, se definen los objetivos y el plan del cuasi-experimento. El objetivo de este cuasi-experimento, según el paradigma *Goal-Question-Metric* (GQM) (Basili y otros 1998) es: *analizar* FeDRE *con el propósito* de evaluar *respecto a* su facilidad y utilidad percibida *desde el punto de vista* de un conjunto de ingenieros de requisitos.

El **contexto** de este cuasi-experimento es el modelado de requisitos de una aplicación real en un contexto de desarrollo de LPS. La LPS seleccionada es la aplicación de móviles llamada Savi. Se ha escogido Savi para aplicar las guías de la ingeniería del dominio de FeDRE debido a que es un ejemplo de aplicación real.

Se han definido dos **variables subjetivas dependientes**: *facilidad de uso percibida* y *utilidad percibida*. Para medir las dos variables después de aplicar las guías de la ingeniería del dominio de FeDRE se aplican dos instrumentos de medida propuestos en la evaluación de métodos de modelado de requisitos basados en la percepción del usuario (Abrahão y otros 2011). En concreto se han adaptado dos variables basadas en el mencionado instrumento de medición a partir del Modelo de Aceptación Tecnológica (Davis 1989):

- *Facilidad de Uso Percibida* (FUP): el grado en que una persona cree que utilizando FeDRE le ahorrará esfuerzo. Esta variable representa un juicio perceptual sobre el esfuerzo requerido para aprender y usar FeDRE.
- *Utilidad Percibida (UP):* el grado en que una persona cree que FeDRE le va a ayudar a alcanzar sus objetivos perseguidos. Esta variable representa el juicio perceptual de la efectividad de FeDRE.

Se han definido un conjunto de preguntas cerradas para medidas estas variables basadas en la percepción. Estas preguntas se han combinado en una en una encuesta de 7 elementos, 4 eran correspondientes a la variable FUP y 3 a la variable UP. Las preguntas se han formulado usando una escala Likert (Likert 1932) utilizando un formato de preguntas 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 enunciaron dos hipótesis a contrastar:

- $H1_0$: FeDRE se percibe como difícil de utilizar, *H1a*: FeDRE es percibido como fácil de utilizar.
- $H2_0$: FeDRE no es percibido como útil, *H2a*: FeDRE es percibido como útil.

Por otra parte se ha definido una solución para la especificación de requisitos considerada como la correcta del ejercicio. El objetivo de esta solución es compararla con las soluciones de los sujetos que participan en el cuasi-experimento con el objetivo de analizar en qué grado se ha aplicado correctamente FeDRE de una forma eficiente y efectiva. Con este objetivo se han definido cuatro *variables objetivas dependientes*:

- *Efectividad_CU* que se calcula como el cociente entre el número de casos de usos identificados por el sujeto correctamente y el número total de casos de usos correctos.
- *Efectividad_SCE* que se calcula como el cociente entre el número de escenarios alternativos identificados por el sujeto correctamente y el número total de escenarios correctos.
- *Eficiencia_CU* que se calcula como el cociente entre el número de casos de usos identificados por el sujeto correctos y el tiempo utilizado por el sujeto para identificar los casos de uso.
- *Efectividad_SCE* que se calcula como el cociente entre el número de escenarios alternativos identificados por el sujeto correctamente y el tiempo utilizado por el sujeto para identificar los escenarios alternativos.

La Tabla 7.1 muestra la planificación del experimento. Antes de la sesión del cuasi-experimento se realizó una *sesión de entrenamiento* de 30 minutos con el objetivo de introducir los conceptos básicos de FeDRE y conceptos esenciales

de la IR (ej. notación de casos de uso), y la descripción del ejercicio en sí. Después de la sesión de entrenamiento se ejecutó la sesión con el ejercicio. Por último cuando los sujetos finalizaron el ejercicio tuvieron que rellenar un cuestionario con el objetivo de evaluar FeDRE.

Tabla 7.1 Planificación del cuasi-experimento para validar las guías de la ingeniería del domino

Grupo	Actividad
Entrenamiento (30 min.)	Introducción a FeDRE Explicación del ejercicio
Sesión (90 min.)	Tarea 1: identificar los casos de uso Tarea 2: especificar los casos de uso Rellenar cuestionario para evaluar FeDRE

Se han definido varios documentos con el objetivo de ser instrumentos del casi-experimento: presentaciones para la sesión de entrenamiento, una explicación del método, un formulario para recopilar los datos y finalmente un cuestionario. Estos documentos se han utilizado por los sujetos, que han sido escogidos de un grupo de ingenieros del software asociados. Se le preguntó a los sujetos sobre su experiencia en el área y como resultado se observó que ninguno de ellos tenía experiencia previa en ese contexto. En consecuencia no se consideró necesario establecer una clasificación de los sujetos basada en la IR para LPS. En la primera sesión participaron 8 estudiantes de la Universitat Politècnica de València (España). En la segunda sesión participaron 6 estudiantes de la Universidad Federal de Bahía (Brasil).

7.1.2 Preparación y ejecución del cuasi-experimento

Respecto a la actividad de *scoping*, todos los artefactos (ej. modelo de características, especificación de las características y mapa de productos) se crearon por un analista del dominio y un experto del dominio que fueron asistidos por un experto de *scoping* con más de 6 años de experiencia en actividades de *scoping* en LPS. En análisis de mercado se realizó basándose en otros productos similares y con el mismo objetivo que Savi, que están disponibles en la AppStore. La funcionalidad de estos productos se ha incluido en el modelo de características de la familia de productos Savi, la cual incluye 27 características identificadas.

Se ha seleccionado un conjunto de características para el cuasi-experimento, asumiendo que FeDRE es una propuesta flexible a las especificaciones de requisitos incrementales. La selección de estas características se basó en el criterio de seleccionar las que estaban presentes en más productos de acuerdo con el mapa de productos.

Cada una de estas 27 características del modelo de características se especificó de acuerdo a la plantilla de especificación de características. Adicionalmente durante la actividad de *scoping* se definió una lista de productos del dominio de aplicaciones de gestión de emergencias, con el objetivo de especificar el artefacto mapa de productos. Respecto a la especificación de requisitos para la actividad de ingeniería del dominio, dos analistas de requisitos del equipo crearon el artefacto glosario basado en los modelos de la actividad de *scoping*. Un total de 16 términos relevantes se identificaron para el dominio.

Los artefactos creados para la actividad de *scoping* (modelo de características, especificación de características y mapa de producto) y el artefacto glosario, creado por el ingeniero del dominio en la actividad del método especificación de requisitos, se han utilizado para crear los requisitos funcionales y la matriz de trazabilidad aplicando las guías de FeDRE para la ingeniería del dominio.

7.1.3 Recogida de datos

Los datos para este cuasi-experimento se han recopilado durante la especificación de requisitos de la ingeniería del dominio de FeDRE. Los requisitos funcionales de la LPS los han especificado 14 estudiantes de doctorado de dos universidades (España y Brasil). A estos estudiantes se les pidió aplicar las guías para especificar los requisitos de la ingeniería del dominio (Apéndice A) con el objetivo de responder a las siguientes preguntas:

i. ¿*Qué* características pueden agruparse para especificar los casos de uso?
ii. ¿*Cuáles* serán los casos de uso a especificar para una característica o conjunto de características?
iii. ¿*Dónde* deben de especificarse los casos de uso?
iv. ¿*Cómo* va a especificarse cada caso de uso en términos de pasos?

A continuación se muestra cómo se responde a cada una de estas preguntas por parte de los sujetos.

7.1.3.1 ¿Qué características se agrupan para especificar los casos de usos?

En este paso se analizan que todas las características incluidas en la unidad de incremento en la iteración actual. Una unidad de incremento es un conjunto de características cuyos requisitos se van a especificar. Para cada una de las iteraciones los sujetos tienen que decidir qué características se especificaran como casos de uso. De acuerdo con la primera tarea de las guías de la ingeniería de dominio de FeDRE muchos de los sujetos escogieron comenzar la iteración

con la característica *control de acceso* y sus características hijas, debido a que comparten la misma funcionalidad (tarea 1 de las guías). Existen dos maneras válidas de implementar e importar un contacto (una opcional con *control de acceso web*; y otra obligatoria con *control de acceso móvil*) de acuerdo con las guías (pasos 1.2 y 1.4). Algunos usuarios decidieron que estas características no se especificaran como casos de uso y de forma alternativa se especificaron como escenarios alternativos en los casos de uso relacionados con la característica *control de acceso*. De forma similar algunos sujetos especificaron las características *importar con Facebook*, *importar con Twitter* como escenarios alternativos para un casos de uso de la característica *importar contacto*. Muchos sujetos aplicaron este criterio para las características *destino SMS*, *destino Twitter*, *destino Facebook* y *destino e-mail* como escenarios alternativos de los casos de uso relacionados con la característica *destino*. Las características: contacto, importar contacto, añadir contacto, destino y números de emergencia fueron especificados mayoritariamente como casos de uso por parte los sujetos. Desafortunadamente algunos sujetos no decidieron especificar escenarios alternativos como se sugería en las guías, ignorando la variabilidad del modelo de características.

7.1.3.2 ¿Cuáles serán los casos de uso a especificar para una característica o conjunto de características?

Después de decidir qué características necesitan ser especificadas como casos de uso el siguiente paso consiste en identificar qué casos de uso deben de asociarse con cada característica. Además debe de rellenarse de forma incremental la matriz de trazabilidad que relaciones los casos de uso con las características.

La mayoría de los casos de uso identificados por los sujetos a partir de las características seleccionadas se presentan a continuación:

- Característica *control de acceso*: crear usuario, identificarse, mostrar perfil, recordar clase de paso y enviar e-mail.
- Característica *control de acceso Web*: actualizar usuario, borrar usuario.
- Característica *contacto*: mostrar contacto, borrar contacto, actualizar contacto.
- Característica *añadir contacto*: añadir contacto.
- Característica *importar contacto*: recuperar contactos, importar contactos.
- Característica *destino*: enviar notificación.
- Característica *números de emergencia*: crear número de emergencia, borrar número de emergencia, actualizar número de emergencia.

La solución considerada como correcta para las características proporcionadas contenía un total de 17 casos de uso.

7.1.3.3 ¿Dónde deben de especificarse los casos de uso?

De acuerdo con las reglas de FeDRE los casos de uso con un comportamiento similar y que se han identificado simultáneamente relacionados en dos características con el mismo padre deben de ser relocalizados con el objetivo de evitar la especificación redundante del mismo comportamiento. Cuando esto ocurre el caso de uso tiene que ubicarse únicamente una vez en el nivel de característica padre. Una vez que se han identificado para todas las características los casos de uso es posible empezar a modelar los casos de uso. Se crea un paquete de caso de uso para cada característica que contiene casos de usos, y se crea un diagrama de caso de uso para incluir los casos de uso, actores y relaciones entre ellos.

7.1.3.4 ¿Cómo va a especificarse cada caso de uso en términos de pasos?

Después de identificar los casos de uso y relacionarlos con las características, los sujetos especificaron cada caso de uso tomando en cuenta las variaciones del modelo de características.

7.1.4 Análisis de datos

Se han llevado a cabo dos análisis para los datos recogidos del cuasi-experimento: uno cuantitativo y otro cualitativo. El análisis cuantitativo relacionado con las variables objetivas (efectividad y eficiencia) observadas durante la ejecución de las dos tareas. Como no se ha utilizado un grupo de control, el análisis se ha realizado con el objetivo de identificar deficiencias en las guías y mejorar la redacción de estas. El análisis cualitativo se ha realizado empleando el conjunto de respuestas cerradas del cuestionario post-experimento, el cual rellenaron los sujetos una vez finalizadas las tareas del cuasi-experimento. Esta información se analizó con el objetivo de comprobar los resultados de la percepción de facilidad de uso y utilidad, y su relevancia estadística para contrastar las hipótesis del cuasi-experimento.

7.1.4.1 Análisis cuantitativo

Respecto a la efectividad en la tarea 1 del cuasi-experimento, se ha medido el cociente del número correcto de casos de uso que se han identificado entre el número total de casos de uso modelados en la solución (*Efectividad_CU*). Los usuarios fueron capaces de identificar correctamente un 62,1% de los casos de uso totales de la solución acordada en la tarea 1. Para comprobar la efectividad en la tarea 2 se compararon los escenarios alternativos con los escenarios alternativos que se acordaron en la solución considerada como correcta (*Efectividad_SCEN*). Los resultados muestran que esta variable tuvo una media de 0,523 lo cual significa que los sujetos especificaron correctamente el 52,3% del total de casos de uso (Tabla 7.2).

Adicionalmente para cada tarea se han medido el tiempo empleado y la eficiencia. Los resultados muestran que los sujetos invirtieron en torno a unos 51 minutos para completar la tarea 1, con un valor para la variable *Eficiencia_CU* de 0,216 (números de casos de uso correcto / tiempo). Los sujetos tardaron en torno a 39 minutos para completar la tarea 2 para la variable *Eficiencia_SCEN* con un valor de 0,048 (número de escenarios alternativos correctos / tiempo).

Tabla 7.2 Media y desviación estándar para las variables analizadas

Variable	Media	Desviación estándar
Efectividad_CU	0,621	0,182
Efectividad_SCEN	0,523	0,447
Tiempo tarea 1	51,86	13,091
Eficiencia_CU	0,216	0,085
Tiempo tarea 2	39	19,247
Efectividad_SCEN	0,0484	0,048

Finalmente se ha realizado un análisis cualitativo analizando las respuestas a las preguntas abiertas del cuestionario. Por ejemplo algunos sujetos sugirieron reformularlas algunas guías para evitar ambigüedades durante la definición de casos de uso (ej. identificando grupos de características que comportan la misma funcionalidad) o durante la especificación de casos de uso (ej. definiendo escenarios alternativos). Otros sujetos sugirieron incluir en las guías reglas para tratar con las relaciones entre características (*include* / *extend*). El análisis de estos datos cuantitativos reveló algunas cuestiones importantes que ayudaron a mejorar las guías de la ingeniería del dominio de FeDRE.

7.1.4.2 Análisis cualitativo

En esta sección se discuten los resultados del análisis cualitativo realizado para contrastar las hipótesis establecidas. Todos los resultados que se presentan se han obtenido utilizando la herramienta estadística SPSS v20 con un valor alfa de 0,05. Se ha analizado si las medias de las respuestas a las cuestiones relativas las variables subjetivas eran significativamente mayores que el valor neutral Likert (igual a 3). En este caso el valor neutral es el valor medio del rango de 1 a 5 de medición de las dos variables subjetivas se han considerado como un intervalo de escala (Carifio y Perla 2007). Ambas variables tienen una media sobre el valor neutral

Para verificar ambas hipótesis con los datos primero se ha seleccionado el test más apropiado para analizar los datos obtenidos. El primer paso es analizar si los datos recopilados se ajustan a una distribución normal. Como el tamaño de las muestras es menor de 50 se aplicó el test Shapiro-Wilk para verificar si los datos seguían una distribución normal. Los resultados del test de normalidad muestran que ambas variables se distribuyen de forma normal dado que los valores obtenidos son mayores que 0,05 (Tabla 7.3).

Tabla 7.3 Análisis de las variables FUP y UP

	Media	Desviación	Shapiro-Wilk	Alfa	t-test[5]
Facilidad de uso	3,857	0,813	0,696	0,833	0,002
Utilidad percibida	3,880	0,771	0,066	0,722	0,001

A continuación se realiza un test para comprobar si los resultados obtenidos son significativamente mayores al valor neutral 3. Dado que ambas variables siguen una distribución normal se realiza una prueba t-test para muestras independientes con un valor de test de 3. Los p-values obtenidos son menores que 0,05 con los que ($p \leq \alpha$). Como consecuencia se pueden anular las dos hipótesis nulas lo que significa que FeDRE es percibido por los participantes como fácil de usar y útil (Tabla 7.3).

7.1.5 Amenazas a la validez

Las principales amenazas a la **validez interna** del cuasi-experimento son: evaluación del diseño, experiencia del sujeto, intercambio de evaluación entre evaluadores, y la comprensibilidad de los documentos. El *diseño de la evaluación* podría haber afectado los resultados debido a la selección de características que se tomaron como entrada en FeDRE con el objetivo de extraer los requisitos.

[5] P-values para la prueba one-tailed t-test

Se ha intentado mitigar esto considerando un conjunto de características que implicaban utilizan el conjunto completo de reglas de las guías de la ingeniería del dominio de FeDRE. *La diferencia de experiencia de los sujetos* no afecta a los resultados, debido al hecho de que ninguno de los sujetos que participó en este cuasi-experimento tenía experiencia en el desarrollo de LPS. El intercambio de información se mitigó debido a que se monitorizó a los participantes mientras realizaban las tareas. Los sujetos realizaron el experimento en dos sesiones establecidas en España y Brasil y sin relación entre los sujetos de modo que se descartó el intercambio de información entre ellos. El factor *compresión del material* se mitigo eliminando todos los posibles malos entendidos antes de cada sesión.

Otro factor a tener en cuenta es la amenaza a la **validez externa**. Un factor a mitigar es la *representatividad de los resultados*. Para mitigar esta amenaza, haciendo más representativas las tareas, la complejidad del ejercicio se ha ajustado para que los sujetos fueran capaces de aplicar cada regla de las guías al menos una vez teniendo en cuenta que la duración del experimento estaba limitada a 90 minutos.

La mayor amenaza para la **validez del constructo** del cuasi-experimento fue la robustez del cuestionario, relacionada con las dos hipótesis. Esta robustez fue probada aplicando el test alfa de Conbrach para cada una de las respuestas cerradas del cuestionario que se midieron con las variables FUP y UP. Los resultados del test produjeron unos resultados de 0,833 y 0,722 respectivamente. Estos resultados fueron por encima del umbral mínimo de aceptación α=0,70 (Maxwell 2002).

Por último se analiza la **validez de conclusión del experimento**. En este caso se mitiga el factor de *validez del test estadístico aplicado*. Este factor fue mitigado aplicando los test más relevantes en el campo de la ingeniería del software empírica (Maxwell 2002).

7.2 Cuasi-experimento para la validación de la ingeniería de la aplicación

Se ha realizado un segundo cuasi-experimento con el objetivo de evaluar la utilidad de FeDRE en la ingeniería de la aplicación utilizando las guías propuestas por Wohlin y otros (2000). La principal razón para seleccionar un cuasi-experimento para realizar la validación es la ausencia de aproximaciones de IR para LPS que soporten la derivación de requisitos de un producto y la especificación de los requisitos delta. Este hecho incremente la dificultad para llevar a cabo experimentos controlados que comparen la efectividad y la efectividad de FeDRE respecto otras aproximaciones de IR para LPS.

Las siguientes subsecciones proveen detalles de las fases del cuasi-experimento, que son: diseño del cuasi-experimento, preparación y ejecución, recogida de datos, análisis de los resultados. Además se describen las posibles amenazas a la validez y qué medidas se han adoptado para mitigarlas.

7.2.1 Diseño del cuasi-experimento

En primer lugar, se definen los objetivos y el plan del cuasi-experimento. El objetivo de este cuasi-experimento, según el paradigma *Goal-Question-Metric* (GQM) (Basili et al. 1998) es: *analizar* FeDRE *con el propósito* de evaluar *respecto a* su facilidad, utilidad percibida e intención de uso percibida *desde el punto de vista* de un conjunto de ingenieros de requisitos.

El **contexto** del cuasi-experimento es la definición y especificación de los requisitos de un producto en una línea de productos software diseñada por ingenieros de requisitos noveles. Los participantes tienen que validar y especificar los requisitos delta para un producto perteneciente a una familia de productos de tiendas en Internet.

En este cuasi-experimento se han definido tres **variables subjetivas dependientes**: *facilidad de uso percibida, utilidad percibida e intención de uso*. Para medir estas variables, después de aplicar FeDRE, se ha utilizado un instrumento de medición propuesto para la evaluación de métodos de modelado de requisitos basado en percepciones del usuario (Abrahão y otros 2011). Concretamente se adaptan tres variables basadas en percepción para dicho instrumento, las cuales se basan en constructor del *Modelo de Aceptación Tecnológica* (MAT) (Davis 1989):

- *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 mínimo.

- *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.

Se han definido una serie de preguntas para medir estas tres variables basadas en la percepción. Estas preguntas se combinaron en un cuestionario formado por 9 preguntas. Las respuestas se codificaron mediante una escala Likert (Likert 1932) de 5 puntos con el formato de afirmaciones y respuestas escaladas de totalmente de acuerdo a totalmente en desacuerdo. El orden de las preguntas se alteró de forma aleatoria con el fin de evitar el sesgo de respuesta sistemática (Hu y otros 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 sobre FeDRE.

Por otra parte se han definido las siguientes **hipótesis**:

- $H1_0$: FeDRE se percibe como difícil de usar por los participantes que lo aplican / *H1a*: FeDRE se percibe como fácil de usar por los participantes que aplicaron el método.

- $H2_0$: FeDRE se percibe como no útil por los participantes que lo aplican / *H2a*: FEDRE se percibe como útil por los participantes que lo aplican.

- $H3_0$: Los participantes no tienen intención de usar FeDRE en el futuro / *H3a*: Los sujetos tienen intención de utilizar FeDRE en el futuro.

Adicionalmente antes de la ejecución del cuasi-experimento, se definió una especificación de requisitos definida como solución correcta por los tres autores de FeDRE. El objetivo de la solución considerada como correcta es analizar el grado en el que los sujetos aplicaron FeDRE de forma efectiva y

eficiente, comparando los resultados obtenidos con la solución acordada. Con este objetivo se definieron seis **variables dependientes**:

- *Precisión de las necesidades del cliente (Precisión_NED)*. Esta variable se mide como el cociente entre el nº de necesidades del cliente identificadas correctamente por el sujeto, dividido por el nº total de necesidades del cliente identificadas por el sujeto (necesidades del cliente correctamente identificadas más las necesidades del cliente identificadas como falsos positivos). Este variable se calcula aplicando la formula (1).

- *Exhaustividad de las necesidades del cliente (Exhaustividad_NED)*. Es el cociente entre el nº de necesidades del cliente identificadas correctamente dividido entre el nº total de necesidades del cliente (número de necesidades del cliente más el nº de necesidades del cliente clasificadas como positivos negativos). Se calcula aplicando la formula (2).

- *Efectividad de casos de uso (Efectividad_CU)*. Se calcula como el cociente entre el nº de deltas identificados correctamente por el sujeto dividido por el nº total de casos de uso correctos. Se calcula aplicando la formula (3).

- *Eficiencia de casos de uso (Eficiencia_CU)*. Se calcula como el cociente entre el nº de casos de uso deltas correctos, divididos por el tiempo total empleado en la identificación de los casos de uso delta. Se calcula aplicando la formula (4).

- *Efectividad de diagramas de actividad (Efectividasd_DA)*. Se calcula como el cociente entre el nº de acciones delta que el sujeto identifica correctamente entre el número total de acciones correctas. Se calcula aplicando la formula (5).

- *Eficiencia de diagramas de actividad (Eficiencia_DA)*. Se calcula como el cociente entre el nº de acciones delta correctas que el sujeto identifica correctamente dividido por el tiempo total en la especificación de las actividades deltas. Se calcula aplicando la formula (6).

$$Precisión\_NED = \frac{n^{o}\,de\ necesidades\ del\ cliente\ correctamente\ identificadas}{n^{o}\,total\ de\ necesidades\ del\ cliente\ identificadas} \quad (1)$$

$$Exhaustividad\_NED = \frac{n^{o}\,de\ necesidades\ del\ cliente\ identificadas\ correctamente}{n^{o}\,total\ de\ necesidades\ del\ cliente} \quad (2)$$

$$Efectividad\_CU = \frac{n^{o}\,de\ casos\ de\ uso\ delta\ identificadores\ correctamente}{n^{o}\,total\ de\ casos\ de\ uso\ identificados} \quad (3)$$

$$Eficiencia\_CU = \frac{n^{o}\,de\ casos\ de\ uso\ delta\ identificados\ correctamente}{tiempo\ total\ empleado\ en\ la\ identificación\ de\ casos\ de\ uso\ delta} \quad (4)$$

$$Efectividad\_DA = \frac{n^{o}\,de\ acciones\ delta\ identificadas\ correctamente}{n^{o}\,total\ de\ acciones\ delta\ correctas} \quad (5)$$

$$Eficiencia\_DA = \frac{nº\ de\ acciones\ delta\ identidicadas\ correctamente}{tiempo\ total\ empleado\ en\ la\ especificación\ de\ acciones\ delta} \quad (6)$$

7.2.2 Preparación y ejecución del cuasi-experimento

En esta sección se describe la preparación y ejecución del cuasi-experimento. La Tabla 7.1 muestra la planificación del cuasi-experimento. Antes del ejercicio se realizó una sesión de entrenamiento de 45 minutos para explicar los conceptos previos de la IR para LPS (ej. la notación de casos de uso, tipo de relaciones en un diagrama de casos de uso, etc.), el método FeDRE y las tareas del cuasi-experimento. Después de la sesión de entrenamiento, se llevó a cabo el cuasi-experimento. Esta sesión se compuso de dos tareas. Finalmente cuando se finalizaron las tareas, los sujetos rellenaron un cuestionario de evaluación.

Tabla 7.1 Planificación del cuasi-experimento de la ingeniería de la aplicación

Grupo	Actividad
Entrenamiento (45 min.)	Introducción a FeDRE Explicación del ejercicio
Sesión (90 min.)	Tarea 1: Validar los requisitos del producto Tarea 2: especificar los requisitos delta Rellenar cuestionario para evaluar FeDRE

7.2.2.1 Preparación del experimento

Se han preparado diversos **documentos**[6] como instrumentos para el cuasi-experimento: trasparencias para la sesión de entrenamiento, explicación del proceso, formulario de recogida de datos y un documento con la descripción de la LPS con las tareas a realizar y un cuestionario de evaluación.

Se seleccionó un grupo de estudiantes de doctorado en diversos temas de ingeniería del software. Se ha diseñado un cuestionario demográfico para categorizar los participantes en función de su experiencia y conocimiento previo, así como su pertenencia al grupo de España o Brasil para comprobar si existían

[6] El material experimental se encuentra disponible en la dirección: http://bit.ly/fedre2015

diferencias entre los resultados. La primera sesión se compuso de 8 estudiantes de doctorado de la Universitat Politècnica de València (España). Los participantes españoles son expertos en diversos temas relacionados con la ingeniería del software pero novatos en el desarrollo de LPS. El segundo grupo se compuso de 7 estudiantes de doctorado de la Universidad Federal de Bahía (Brasil). Los participantes de Brasil son expertos en desarrollo de LPS en el contexto de investigación.

Respecto al material experimental se ha elaborado un modelo de características que representa una familia de productos de tiendas en Internet. Este modelo de características contiene 23 características y puede producir un total de 3969 productos válidos. A partir de este modelo de características se definieron y especificaron los requisitos de la familia de productos software (diagramas de caso de uso y diagramas de actividad del dominio) aplicando las reglas de la ingeniería del dominio de FeDRE (sección A.1). Finalmente se definió el modelo de variabilidad de los requisitos, representado en CVL.

Una vez definidos los artefactos de la ingeniería del dominio, se prepararon los artefactos de la ingeniería del producto. Para los artefactos para el producto concreto *ÓrdenesCoin* se creó una configuración que se muestra las características seleccionadas con un tic verde (Figura 7.1). Esta configuración se utilizó como entrada en la derivación de los requisitos del producto donde asistidos con la herramienta se derivaron los requisitos del producto.

Por último se elaboró la tabla de necesidades del cliente con el objetivo de que los sujetos comprobaran si cada uno de los requisitos estaban soportados en la especificación de los requisitos derivada.

Después de crear la configuración del producto *OrdernesCoin* los autores del material experimentan realizaron la transformación de modelos. Como resultado se generó un modelo de resolución CVL. Este modelo de resolución se utilizó en la transformación CVL para derivar la especificación de los requisitos del producto que se utilizó en el cuasi-experimento. La Figura 7.2 muestra el diagrama de casos de uso derivado para el producto OrdernesCoin que define el pago. Igualmente se derivaron los diagramas de actividad que especifican los casos de uso, como ejemplo en la Figura 7.4. Puede encontrarse todo el material experimental en el Apéndice C.

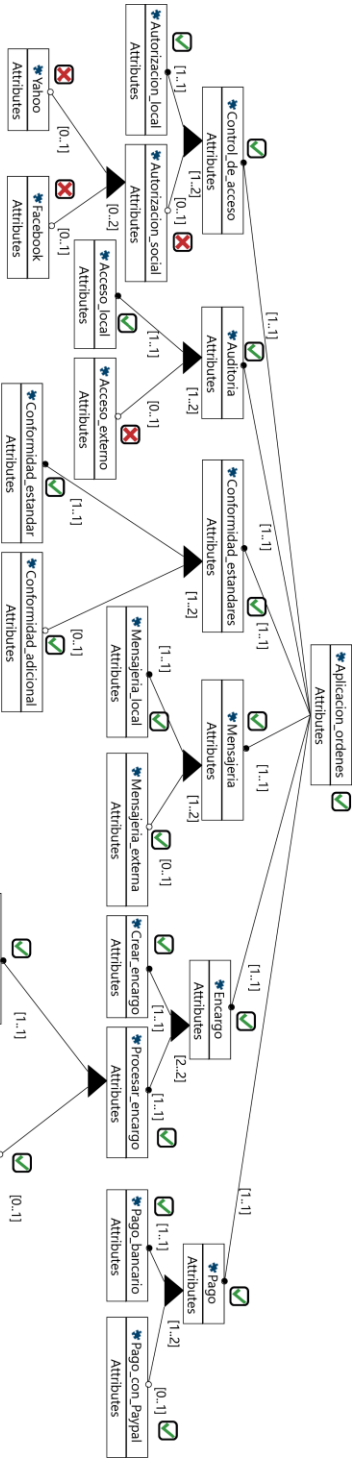


Figura 7.1 Configuración del producto OrdenesCoin

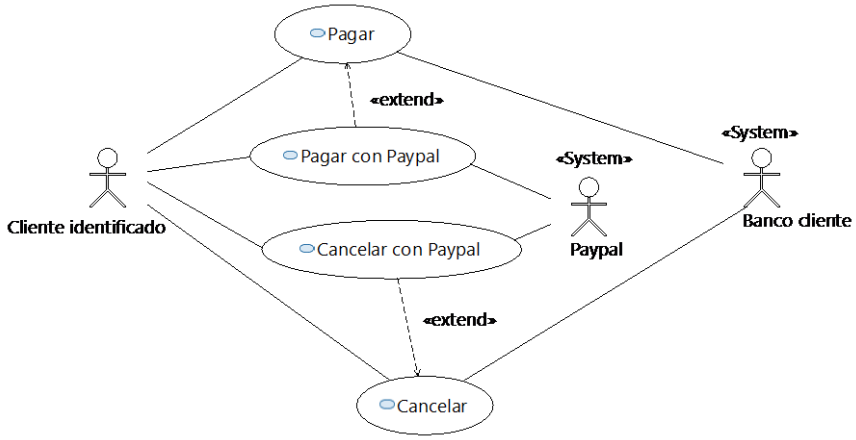


Figura 7.2 Diagrama de casos de uso después de derivar los requisitos del pago

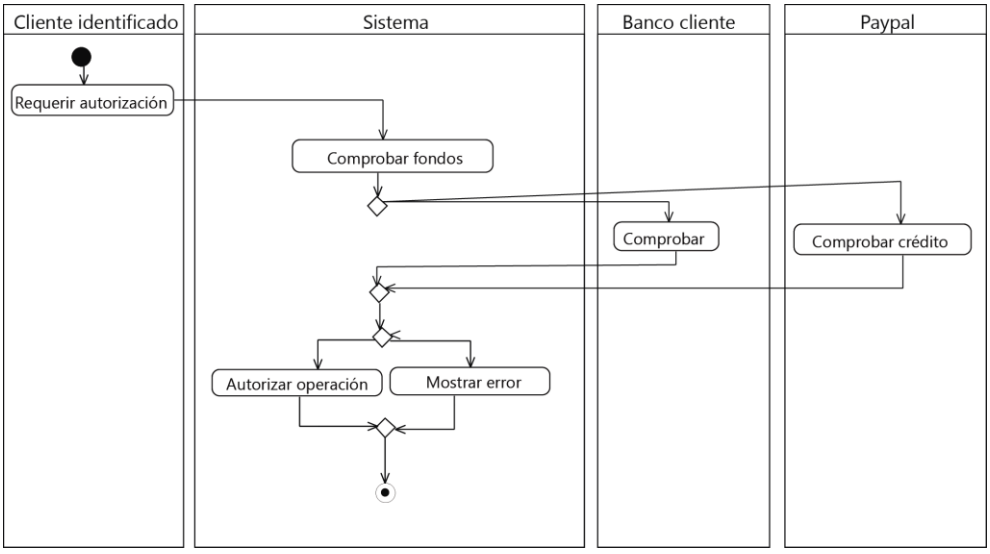


Figura 7.3 Diagrama de actividad para especificar el pago

7.2.2.2 Tareas experimentales

Durante la primera tarea experimental los sujetos llevaron a cabo la validación de requisitos de FeDRE para comprobar si las necesidades se cubrieron completamente con la especificación de requisitos del producto derivada. Si un participante detecta que los requisitos derivados no son completos (ej. no satisfacen completamente la lista de necesidades), entonces los participantes llevaron a cabo la especificación de requisitos delta para completar la especificación derivada. En esta tarea se utiliza como artefacto de entrada los

requisitos derivados. Sin embargo si el participante considera que la especificación de requisitos del producto no debe de completarse, no sería necesario ejecutar la tarea 2.

7.2.2.2.1 Validación de requisitos

En esta primera tarea los sujetos tienen que comprobar qué necesidades del usuario se soportan en la derivación de requisitos especificada. Se proveen dos modelos para completar esta tarea: el diagrama de casos de uso y el diagrama de actividad. La Tabla 7.2 muestra las necesidades del cliente para el producto *OrdersCoin*. Lo sujetos tienen que comprobar si cada una de las necesidades se soportan o no. Entre estas necesidades algunas se cubrirán mediante casos de usos: N1, N2, N5 y N6. Otras necesidades del cliente se cubren mediante acciones de los diagramas de actividad: N3, N4, N7, N8. Sin embargo algunas necesidades del cliente están relacionadas con nueva funcionalidad: el pargo con Bitcoins: N9, N10, N11, N12. Como consecuencia si los sujetos ejecutan correctamente la primera tarea, entonces ejecutarán la segunda para especificar los requisitos delta.

7.2.2.2.2 Especificación de requisitos delta

En esta tarea el sujeto lleva a cabo la especificación de los requisitos delta aplicando las guías de FeDRE para la ingeniería de la aplicación. Dependiendo de su granularidad algunas de las necesidades del cliente se especificaran como elementos de los diagramas de caso de uso o como elementos de los diagramas de actividad. Esta tarea tomo entrada la lista de necesidades del cliente que no se cubrieron con los requisitos derivados. En este cuasi-experimento las necesidades N9 y N10 tiene que cubrirse con casos de uso delta mientras que las necesidades N11 y N12 tienen que cubrirse con diagramas de actividad.

La Figura 7.4 muestra el diagrama de casos de uso tras especificar los requisitos delta. La necesidad del cliente N9 requiere que el producto soporte el pago con Bitcoins. Después de aplicar las guías de FeDRE las necesidades del cliente se cubren con el nuevo caso de uso pagar con Bitcoins, que se etiqueta con el estereotipo «delta».

Tabla 7.2 Tabla de necesidades del cliente

Necesidad	Descripción
N1 Pagar	Con el pago con tarjeta de crédito el cliente identificado en el sistema tiene que introducir su número de tarjeta de crédito, el número de seguridad que se encuentra en el código de seguridad y la fecha de vencimiento de la tarjeta. Con estos datos el cliente identificado requiere autorización al sistema. El sistema comprobará la disponibilidad de saldo en la cuenta.
N2 Cancelar	El cliente identificado cancele la compra y solicita la devolución. En este paso el sistema requiere la devolución del importe. Para realizar esto el sistema transfiere el importe del pedido cancelado al Banco Cliente.
N3 Comprobación fondos bancarios	El sistema comprobará la disponibilidad de saldo en la cuenta asociada a la tarjeta de crédito del cliente solicitando al Banco Cliente que se haga una comprobación de fondos.
N4 Realizar transferencia bancaria	Para realizar una devolución el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario.
N5 Pagar con Paypal	En ocasiones es posible que el cliente identificado cancele la compra y solicite la devolución. Para realizar esto el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario. En este caso se transfiere el importe de la compra devuelta al Banco Cliente.
N6 Cancelar con Paypal	En el caso de que se hubiera realizado el pago mediante Paypal el cliente identificado solicita cancela la compra y solicita la devolución. El sistema tiene que requerir la devolución en este caso. ÓrdenesCoin efectúa un ingreso en la cuenta Paypal del cliente.
N7 Comprobación fondos en Paypal	Paypal debe de comprobar la cantidad de crédito de un cliente para verificar que se puede realizar la transacción.
N8 Realizar trasferencia a cuenta Paypal	Paypal confirma si se ha realizado correctamente un ingreso en la cuenta de un cliente
N9 Pago con Bitcoins	El pago mediante BitCoins funciona de forma similar a los otros sistemas. Sin embargo con el Bitcoin no existe un banco y en su lugar existe un Cartera Virtual que almacena los Bitcoins. La Cartera Virtual debe verificar la cantidad de saldo.
N10 Cancelar con Bitcoins	Si el pago se realizó con Bitcoins, el cliente puede requerir cancelar su compra con Bitcoins y solicitar la devolución. ÓrdenesCoin se encarga de requerir la devolución. En este caso se debe de realizar una transferencia a la Cartera Virtual.
N11 Comprobar disponibilidad saldo en cartera Virtual	La Cartera Virtual de un cliente comprueba si se tiene saldo suficiente para que el cliente efectúe un pago.
N12 Realizar trasferencia a la cartera Virtual de un cliente	La Cartera Virtual recibe una transferencia en por parte del sistema para realizar una devolución.

De forma similar la necesidad del cliente N10 requiere la devolución con Bitcoins. Para cubrir la N10 se añade el nuevo caso de uso cancelar con Bitcoins. Finalmente el actor Cartera Virtual, etiquetado con el estereotipo «delta», para representar un tercer actor que juega un papel similar a los actores Paypal y banco cliente.

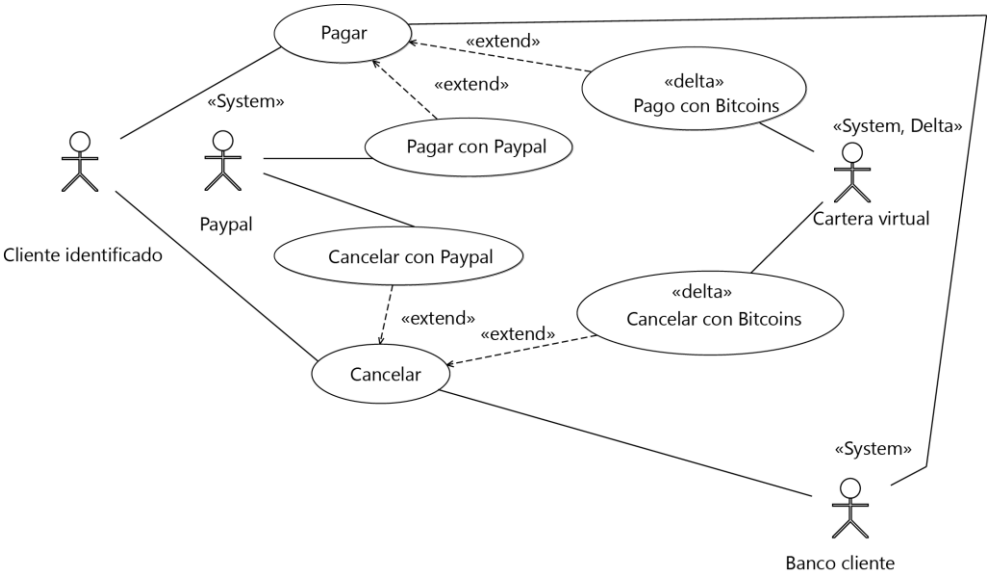


Figura 7.4 Diagrama de casos de uso para el pago tras especificar los deltas

Una vez se han definido los requisitos deltas en los diagramas de casos de uso, tienen que especificarse los requisitos deltas en los diagramas de actividad. La Figura 7.5 muestra el diagrama de actividad que especifica el caso de uso pago tras la especificación de los deltas para soportar la necesidad del cliente N11. Primero se añade una nueva partición de actividad para el actor Cartera Virtual, como es un requisito delta también se especifica con el estereotipo «delta». Finalmente, se añaden los flujos de control necesarios en el diagrama de actividad: uno del nodo de decisión en la partición del sistema hacia la actividad comprobar fondos, y otro que parte de esta acción a un nodo de decisión en la partición del sistema. Una vez que se han añadido los requisitos delta al diagrama de actividad, se considera que la necesidad N11 ya está soportada para el producto ÓrdenesCoin.

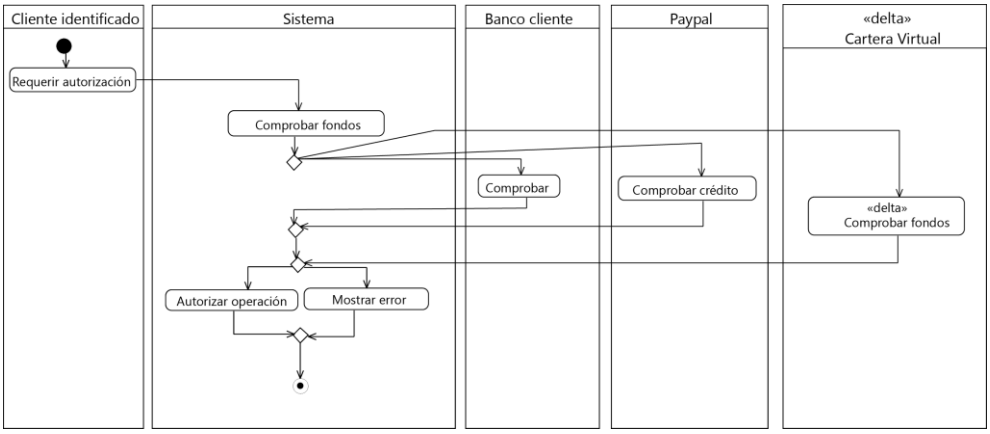


Figura 7.5 Diagrama de actividad del pago tras especificar los deltas

Por último, se define la matriz de trazabilidad de requisitos deltas, la cual se completa con la información de los nuevos requisitos deltas. Un extracto de esta matriz de trazabilidad se muestra en la Tabla 7.3. Una vez el sujeto experimental rellena la matriz de trazabilidad requisitos delta, la segunda tarea del cuasi-experimento se ha completado. Finalmente los participantes tendrán que responder a un cuestionario que contiene las preguntas cerradas para medir las variables subjetivas de percepción (FUP, UP y la ITU).

Tabla 7.3 Matriz de trazabilidad de los requisitos delta

Requisito delta	Tipo	Requisitos relacionados
Pago con Bitcoins	Caso de uso	Pago (caso de uso), Cartera Virtual (actor)
Cancelar con Bitcoins	Caso de uso	Cancelar (use case), Cartera Virtual (actor)
Comprobar Cartera Virtual	Partición de actividad	Comprobar fondos Cartera Virtual (acción)
Comprobar fondos (Cartera Virtual)	Acción	Cartera Virtual (partición de actividad)

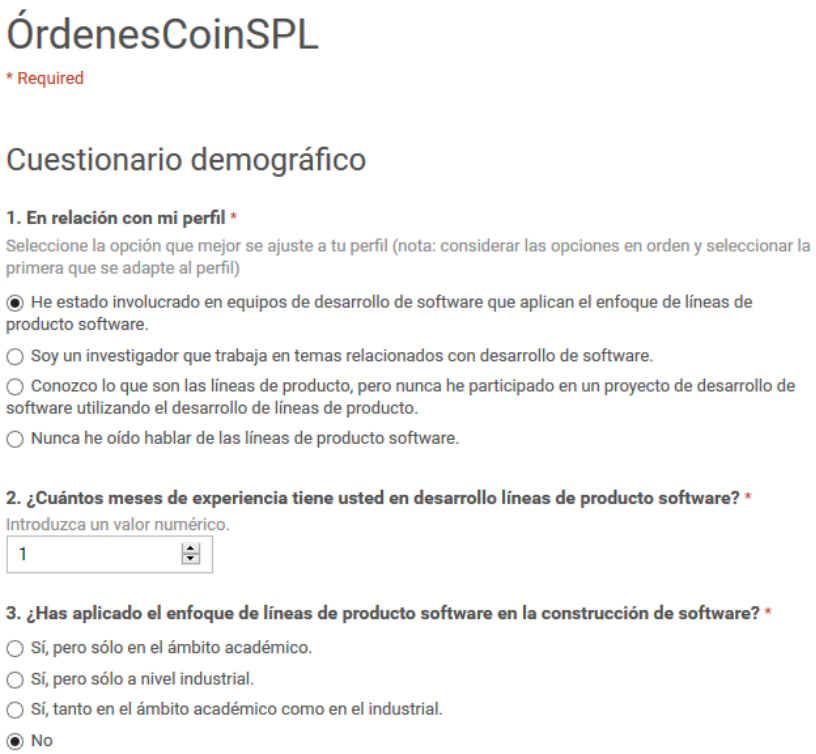
7.2.3 Ejecución del cuasi-experimento

El cuasi-experimento se realizó en una habitación individual, sin permitir interacción entre los sujetos. Los conductores del experimento aclararon las preguntas y dudas de los temas que surgieron durante las sesiones experimentales. Los datos para este cuasi- experimento se recogieron durante la ejecución de las dos tareas FEDRE que conforman el cuasi-experimento: la validación requisitos del producto, y la especificación de requisitos delta. Un formulario en línea se utilizó para llevar a cabo la adquisición de datos, descrito en la siguiente sección.

7.2.4 Recogida de datos

Los datos para este cuasi-experimento se han recogido mediante un formulario online. Los participantes introducían la información de la ejecución durante la ejecución de cada una de las tareas del ejercicio.

Previamente se le pidió que introdujeran un ID asignado por el evaluador y que rellenaran un cuestionario demográfico (Figura 7.6).


ÓrdenesCoinSPL

* Required

Cuestionario demográfico

1. En relación con mi perfil *

Seleccione la opción que mejor se ajuste a tu perfil (nota: considerar las opciones en orden y seleccionar la primera que se adapte al perfil)

- (•) He estado involucrado en equipos de desarrollo de software que aplican el enfoque de líneas de producto software.
- ( ) Soy un investigador que trabaja en temas relacionados con desarrollo de software.
- ( ) Conozco lo que son las líneas de producto, pero nunca he participado en un proyecto de desarrollo de software utilizando el desarrollo de líneas de producto.
- ( ) Nunca he oído hablar de las líneas de producto software.

2. ¿Cuántos meses de experiencia tiene usted en desarrollo líneas de producto software? *

Introduzca un valor numérico.

1

3. ¿Has aplicado el enfoque de líneas de producto software en la construcción de software? *

- ( ) Sí, pero sólo en el ámbito académico.
- ( ) Sí, pero sólo a nivel industrial.
- ( ) Sí, tanto en el ámbito académico como en el industrial.
- (•) No

Figura 7.6 Fragmento del cuestionario demográfico

En la primera tarea se le mostró a los sujetos los diversos artefactos que forman la especificación de los requisitos para el producto *ÓrdenesCoin* obtenida automáticamente con la herramienta. Tras leer la documentación se les mostró a los sujetos una lista de necesidades que el cliente demandaba para el producto, debiendo de seleccionar en un *checkbox* cuales estaban soportadas y cuáles no (Figura 7.7).


ÓrdenesCoinSPL

* Required

Tarea 1

A continuación tras leer la especificación indica cuales de las necesidades para el producto OrdersCoin están cubiertas con la especificación de requisitos.

Para cada una de las necesidades enunciadas debes decidir si se soporta. Para tomar esta decisión has de consultar los siguientes anexos:

Anexo III - Diagramas de caso de uso del dominio
http://dablado.webs.upv.es/fedre/files
/Anexo%20III%20-%20Diagramas%20de%20caso%20de%20uso%20del%20dominio.pdf

Anexo IV - Diagramas de actividad del dominio
http://dablado.webs.upv.es/fedre/files
/Anexo%20IV%20-%20Diagramas%20de%20actividad%20del%20dominio.pdf

N1 Pagar *

Con el pago con tarjeta de crédito el cliente identificado en el sistema tiene que introducir su número de tarjeta de crédito, el número de seguridad que se encuentra en el código de seguridad y la fecha de vencimiento de la tarjeta. Con estos datos el cliente identificado requiere autorización al sistema. El sistema comprobará la disponibilidad de saldo en la cuenta asociada a la tarjeta de crédito del cliente solicitando al sistema gestión del Banco Cliente que se haga una comprobación de fondos. Si los fondos son suficientes entonces el sistema autoriza la trasferencia y se realiza el pago, en caso contrario se muestra un mensaje de error.

○ Sí
○ No

Figura 7.7 Fragmento del formulario online mostrando la lista de necesidades del cliente

Respecto a la tarea 2 se les pidió a los participantes que descargaran un Eclipse de una URL donde se encontraba una versión con los diagramas UML en el editor Papyrus con la especificación de requisitos obtenida previamente (Figura 7.8).

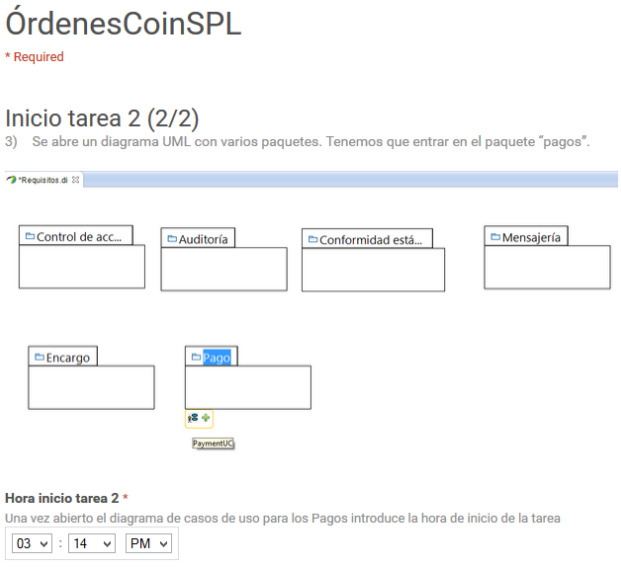


Figura 7.8 Instrucciones previas para completar la tarea 2

Los sujetos tuvieron que seguir las reglas indicadas para modelar los requisitos delta que detectaron tras ejecutar la tarea 1 de validación. Los sujetos debían introducir los requisitos delta tanto en los casos de uso como en los diagramas de actividad de la especificación (Figura 7.9).

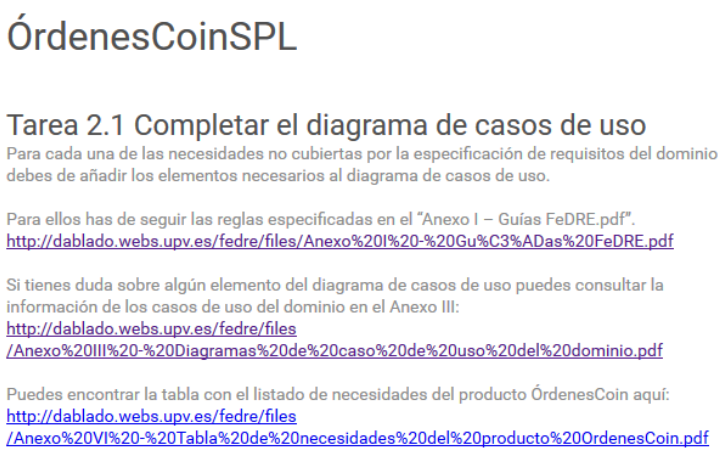
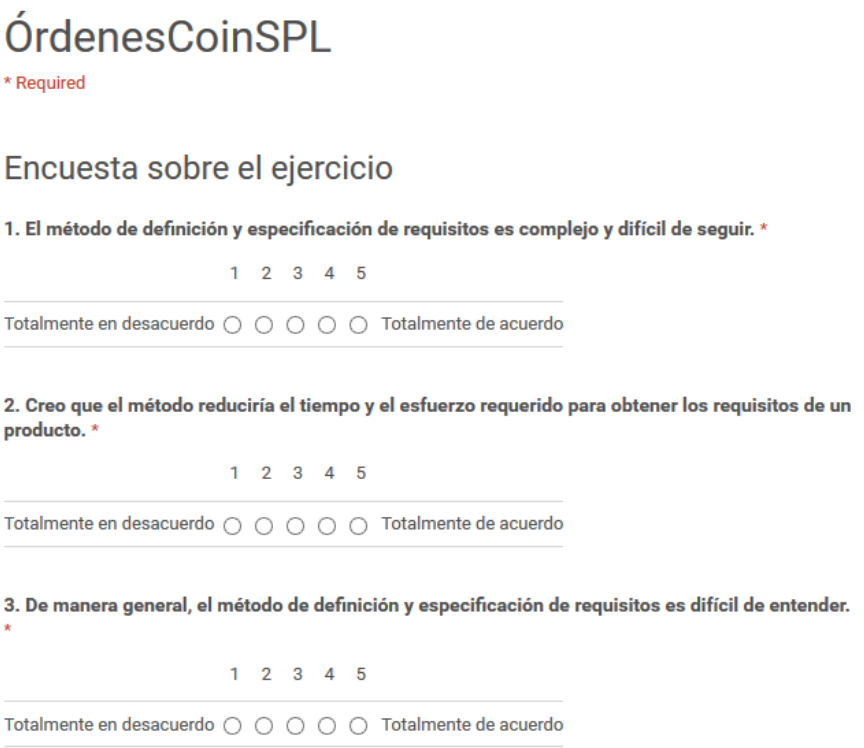


Figura 7.9 Instrucciones para completar la primera parte de la tarea 2

Una vez realizado el ejercicio con el editor de UML los sujetos debían rellenar matriz de trazabilidad con los requisitos delta que consideraron oportuno añadir a la especificación (Figura 7.10).

Figura 7.10 Matriz de trazabilidad de los requisitos delta

Por último se pidió a los sujetos que valoraran el método y las guías propuestas para definir y especificar los requisitos delta (Figura 7.11).



ÓrdenesCoinSPL

* Required

Encuesta sobre el ejercicio

1. El método de definición y especificación de requisitos es complejo y difícil de seguir. *

1 2 3 4 5

Totalmente en desacuerdo ○ ○ ○ ○ ○ Totalmente de acuerdo

2. Creo que el método reduciría el tiempo y el esfuerzo requerido para obtener los requisitos de un producto. *

1 2 3 4 5

Totalmente en desacuerdo ○ ○ ○ ○ ○ Totalmente de acuerdo

3. De manera general, el método de definición y especificación de requisitos es difícil de entender. *

1 2 3 4 5

Totalmente en desacuerdo ○ ○ ○ ○ ○ Totalmente de acuerdo

Figura 7.11 Formulario de evaluación de FeDRE

7.2.5 Análisis de los resultados

Se realizaron dos tipos de análisis sobre los datos recogidos: uno cualitativo y uno cuantitativo, dependiendo de la naturaleza de cada variable. El *análisis cuantitativo* se realizó sobre las variables de precisión y exhaustividad en la primera tarea, y sobre la eficacia y eficiencia durante la ejecución de la segunda tarea. Este análisis se realizó con el objetivo de evaluar el uso de FeDRE y la búsqueda de eventuales diferencias en el rendimiento dependiendo del perfil del participante (ej. formación y conocimiento). El *análisis cualitativo* se realizó mediante el análisis de las respuestas a las preguntas cerradas en el cuestionario post-experimento, que se rellenó por los sujetos después de la ejecución del cuasi-experimento. Esta información se analizó con el fin de evaluar las tres variables basadas en la percepción del usuario. Todos los resultados presentados se obtuvieron mediante el uso de la herramienta estadística SPSS v20 con un valor alfa de 0,05.

7.2.5.1 Análisis cuantitativo

En primer lugar se analizan las preguntas del cuestionario demográfico con el objetivo de analizar el conocimiento previo de los sujetos con la IR para LPS. Las preguntas 4, 7 y 8 fueran respondidas correctamente por todos los sujetos de modo que se excluyen como factor en el análisis demográfico.

Tabla 7.4 Pesos de las respuestas a las preguntas del cuestionario demográfico

Nº	Respuestas	Valor
1	He estado involucrado en equipos de desarrollo de software que aplican el enfoque de líneas de producto software.	3
	Soy un investigador que trabaja en temas relacionados con desarrollo de software.	2
	Conozco lo que son las líneas de producto, pero nunca he participado en un proyecto de desarrollo de software utilizando el desarrollo de líneas de producto.	1
	Nunca he oído hablar de las líneas de producto software.	0
3	Sí, pero sólo en el ámbito académico.	1
	Sí, pero sólo a nivel industrial.	2
	Sí, tanto en el ámbito académico como en el industrial.	3
	No	0
5	Una característica es todo lo que puede variar en una línea de producto software. Una característica puede ser un artefacto software, un documento, etc.	2
	Una característica es un aspecto visible para el usuario de un sistema o producto software y que normalmente se organizan en modelos de características.	3
	No estoy seguro.	1
6	Existe una única especificación de requisitos para todos los productos software desarrollados en las líneas de producto software.	1
	Debe de especificarse una nueva especificación de requisitos a partir de cero para cada producto de las líneas de producto software.	2
	Existe una especificación de requisitos común para las líneas de producto software. Además cada producto tiene su propia especificación de requisitos derivada a partir de la especificación de requisitos de las líneas de producto software.	3
9	Un requisito que no tiene correspondencia con ningún requisito de la línea de productos software.	3
	Es un término que genérico de la ingeniería de requisitos y que no tiene relación con las líneas de producto software.	2
	Es un requisito específico de la ingeniería del dominio.	1

Para el resto de preguntas, se estableció una valoración del perfil. Esto nos permite analizar las diferencias existentes entre los valores de las variables dependientes debido al factor Perfil. Para calcular este valor a las respuestas a las preguntas 1, 3, 5, 6 y 9 se les aplican los valores de la Tabla 7.4.

Por otra parte 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. Finalmente el factor Perfil se calcula agregando los valores ponderados de las preguntas 1, 2, 3, 5, 6 y 9.

A continuación se comprueba si la distribución de los datos se distribuye de forma normal. Dado que el número de observaciones N es inferior a 50, se realiza para cada variable el test de Shapiro-Wilk con el objetivo de comprobar si las variables se distribuyen normalmente. Se efectúan las pruebas de normalidad sobre las variables cuyos resultados se muestran en la Tabla 7.5. Este análisis se realiza con dos factores con el fin de descartar las diferencias significativas entre las poblaciones de participantes sin conocimientos de LPS (Perfil=0) y con conocimiento de LPS (Perfil=1) o entre las poblaciones de participantes de España (Sesión=0) o de Brasil (Sesión=1).

Para aquellas variables normalmente distribuidas ($p \geq 0,05$) se analiza la significancia mediante la prueba t-test para dos muestras independientes con los factores *Perfil* y *Sesión* (tiempo tarea 1, eficiencia_CU, tiempo tarea 2). Para las variables que se distribuían de forma no normal se analizó la significancia de la prueba mediante la prueba Mann-Whitney para dos muestras independientes con el factor *Perfil* y *Sesión* (Precisión_NED, Exhaustividad_NED, Efectividad_CU, Efectividad_ACT, Eficiencia_ACT). La Tabla 7.6 muestra que los resultados con significancia estadística para el tiempo de tarea 1, Efectividad_CU y tiempo de tarea 2. El resto de variables no tienen significancia estadística para los factores de sesión y experiencia ($p \geq 0,05$).

Tabla 7.5 Resultados de las pruebas de normalidad

Variable	**Sesión**		**Experiencia**	
Factor	**0**	**1**	**0**	**1**
Precisión_NED	0,000	0,029	0,000	0,022
Exhaustividad_NED	*	*	*	*
Tiempo tarea 1	**0,092**	**0,967**	**0,077**	**0,988**
Efectividad_CU	*	*	*	*
Eficiencia_CU	**0,637**	**0,113**	**0,944**	**0,173**
Efectividad_ACT	0,000	0,000	0,000	0,000
Eficiencia_ACT	0,000	0,017	0,000	0,025
Tiempo tarea 2	**0,533**	**0,113**	**0,754**	**0,173**

Tabla 7.6 Resultado de las pruebas de significancia de las diferencias entre poblaciones esos de las respuestas a las preguntas del cuestionario demográfico

Variable	Sesión	Experiencia
Precisión_NED	0,512	0,739
Exhaustividad_NED	0,350	0,414
Tiempo tarea 1	**0,020**	**0,002**
Efectividad_CU	0,350	**0,013**
Eficiencia_CU	0,106*	0,702*
Efectividad_ACT	0,679	0,866
Eficiencia_ACT	0,141	0,155
Tiempo tarea 2	**0.014**	**0.001**

*Son constantes

La Tabla 7.7 muestra los resultados del análisis de la media y desviación típica para las variables objetivas. Respecto a la tarea 1 se han medido la precisión y exhaustividad de las necesidades del cliente. La precisión (Precisión_NED) es una medida de la precisión para identificar las necesidades del cliente, con un valor de 0,866 sobre un valor máximo de 1. La exhaustividad corresponde a los positivos verdaderos con un valor con una media de 0,977. El tiempo para completar la tarea 1 se analiza en la Tabla 7.6 debido a que existen diferencias significativas con los factores de sesión y perfil.

Tabla 7.7 Mediana y varianza para las variables distribuidas normalmente

Variable	Mediana	Varianza
Precisión_NED	1	0,031
Exhaustividad_NED	1*	0
Eficiencia_CU	9:52	38997,940
Efectividad_ACT	1	0,103
Eficiencia_ACT	4:56,25	53302,747

*Son constantes

Respecto a la tarea 2 se ha medido la eficiencia_CU (número de casos de uso delta identificados correctamente / número total de casos de uso deltas correctos) con una media de 0,966 sobre 1. La eficiencia_CU se analiza en la Tabla 7.8 debido a que existían diferencias significativas entre los factores de sesión y perfil. La Efectividad_ACT (número de acciones delta que el sujeto identificó correctamente / número total de acciones delta correctas) tiene un media de 0,816 sobre 1. La Eficiencia_ACT (número de acciones delta correctas que el sujeto identificó / el tiempo total utilizado en la especificación de acciones delta) con un valor medio de 0,171 sobre 1. Los resultados muestras que la efectividad para calcular casos de usos fue ligeramente superior a la efectividad para identificar actividades en el diagrama de actividad.

La Tabla 7.8 muestra los resultados para el tiempo de la tarea 1, la efectividad de CU y el tiempo de tarea 2 para los factores de sesión y perfil. El tiempo utilizado para realizar la tarea 1 fue ligeramente superior en el grupo de Brasil, que tenía un mayor conocimiento en LPS que el grupo de España. Similarmente el tiempo para llevar a cabo la tarea 1 fue ligeramente superior para los sujetos con experiencia. En nuestra opinión los sujetos con mayor experiencia en requisitos para LPS tomo más tiempo debido que tomaban en cuenta más factores para decidir si una necesidad del cliente estaba soportado en la especificación de requisitos. Se observa un fenómeno similar en los tiempos para realizar la tarea 2: los sujetos con menos experiencia realizaron la experiencia en un menor tiempo. Este fenómeno afecto a la variable Eficiencia_CU: el grupo de España y los sujetos con menos experiencia fueron más eficientes.

Tabla 7.8 Media y desviación típica para las variables tiempo tarea 1, efectividad_CU y tiempo tarea

Variable	Factor	Media	Desviación Estándar
Tiempo tarea 1	Sesión (=0)	9,13	6,151
	Sesión (=1)	20,86	10,653
	Perfil (=0)	10,22	6,629
	Perfil (=1)	21,17	11,635
Eficiencia_CU	Sesión (=0)	0,113	0,292
	Sesión (=1)	0,080	0,016
	Perfil (=0)	0,112	0,027
	Perfil (=1)	0,076	0,015
Tiempo tarea 2	Sesión (=0)	35	4,504
	Sesión (=1)	52	12,342
	Perfil (=0)	34,78	4,816
	Perfil (=1)	53,83	12,432

7.2.5.2 Análisis cualitativo

Al igual que en caso de las variables objetivas, el primer test comprueba si hay diferencias significativas asociadas al factor Perfil o a la Sesión. Para seleccionar la prueba a aplicar se selecciona primero el test Shapiro-Wilk dado que el número de muestras es menor que 50 con el objetivo de comprobar si las variables se ajustan a una distribución normal. La Tabla 7.9 muestra el resultado de dicha prueba para las variables subjetivas Sesión y Perfil.

Tabla 7.9 Resultados de las pruebas de normalidad de las variables subjetivas por sesión y perfil

Variable	Sesión		Perfil	
	ES (=1)	BR (=2)	1	2
FUP	**0,539**	**0,081**	**0,431**	**0,820**
UP	0,144	0,039	0,115	0,030
IU	**0,792**	**0,091**	**0,701**	**0,070**

El resultado de esta prueba nos ayuda seleccionar el test adecuado para comprobar si existen diferencias significativas entre las poblaciones dependiendo de los factores. Los resultados muestran que las variables FUP y UP se distribuyen de acuerdo a una distribución normal ($p \geq 0,05$), aplicándose en este caso el test t paramétrico unilateral para muestras independientes (Juristo y Moreno 2001). Para la variable UP que no se ajusta a una distribución normal se realiza la prueba no paramétrica Mann-Whitney (Conover 1998). La Tabla 7.10 muestra el resultado de dichas pruebas, descartándose la existencia de diferencias significativas para las diferentes poblaciones y en consecuencia el análisis de las variables de realizará de manera conjunta.

Tabla 7.10 Resultado de las pruebas de significancia de las diferencias entre poblaciones

Variable	Sesión	Perfil
FUP	0,498	0,087
UP	0,813	0,857
IU	0,855	0,227

Las variables subjetivas se analizan comparando si el valor medio de cada variable es significativamente mayor que el valor neutral Likert (igual a 3 en este caso). El rango de valores posibles de las respuestas va de 1 a 5 para cada una de las variables subjetivas que se ha considerado en la escala del intervalo (Carifio y Perla 2007). Cada una de las variables tiene un valor sobre el valor neutral 3 (Tabla 7.11).

Tabla 7.11 Análisis de las variables FUP, UP e IU

Variable	Media	Deviación estándar	Shapiro-Wilk	Alpha Cronbach	t-test
FUP	3.58	0.196	0.397	0.825	0.011
UP	3.80	0.238	0.014	0.898	0.012
IU	3.60	0.208	0.097	0.842	0.012

Para contrastar la hipótesis de las variables primero tenemos que seleccionar el test más adecuado. Se realiza la prueba Shapiro-Wilk con el objetivo para saber si las variables se distribuyen normalmente dado que el tamaño de las muestras es menor que 50. De acuerdo con los resultados las variables FUP e IU de distribuyen normalmente mientras que UP no se distribuye normalmente al ser el resultado del test menor que 0,05. Para comprobar la significancia de las variables FUP e IU sobre el valor neutral 3 se realizará el test paramétrico para variables independientes t con un valor de test de 3. Los resultados de los p-values son menores que 0,05 ($p \leq \alpha$) y como consecuencia podemos rechazar las hipótesis nulas asegurando que FeDRE se percibe como fácil de utilizar y los participantes tendría intención de usarlo. Respecto a la variable UP se realiza el

test Wilcoxon con un valor de test de 3. El resultado de 0,012, inferior a 0,05, nos permite rechazar la hipótesis nula permitiendo asegurar que FeDRE se percibe como útil.

7.2.6 Amenazas a la validez

En esta sección se explican los principales problemas que pueden poner en peligro de validez del cuasi-experimento. Se van a analizar cada uno de los factores que proponen los autores (Cook y Campbell 1979).

7.2.6.1 Validez interna

La validez interna es relevante en aquellos estudios que establecen una relación causal (Runeson y Höst 2008). En este casi-experimento las amenazas principales a la validez interna son: diseño de la evaluación, experiencia de los sujetos, intercambio de información entre participantes y la comprensibilidad de los materiales.

El *diseño de la evaluación* puede afectar los resultados como consecuencia de la selección de los artefactos utilizados en el material experimental. Este efecto se ha intentado mitigar considerando un conjunto de artefactos que obligara a utilizar el conjunto de reglas de la ingeniería de la aplicación de FeDRE para especificar los requisitos delta.

Otro factor que podría influir es la *experiencia de los sujetos*. Se hizo rellenar un cuestionario demográfico con el objetivo de conocer los conocimientos previos de los sujetos en la IR para LPS. Se han realizado test estadísticos con el objetivo de analizar si existían diferencias significativas con los factores sesión (Brasil o España) y la experiencia (tiene o no tiene). Por otra parte se realizó una sesión de entrenamiento en ambas sesiones con el objetivo de dotar y equilibrar el conocimiento del participante en los métodos de IR para LPS de acuerdo con su perfil de ingeniero de requisitos novel.

Otro factor a controlar es el *intercambio de información* debido a que puede alterar el resultado de los distintos experimentos. Sin embargo las sesiones se realizaron en dos sesiones en países distintos entre sujetos que no tenían relación entre ellos. Además se solicitó a los participantes que devolvieran el material experimental una vez finalizado el cuasi-experimento en todas las sesiones.

Por último la *comprensibilidad de los materiales* podría afectar a los resultados. Se realizaron dos versiones del material experimental una en español y otra en inglés para la sesión en Brasil. Se resolvieron las dudas y malentendidos acerca de los materiales durante las sesiones experimentales.

7.2.6.2 Validez externa

La validez externa hace referencia a la veracidad aproximada de las conclusiones que implican generalizaciones entre diferentes contextos. En este caso las mayores amenazas a la validez externa fueron la representatividad de los resultados y el tamaño y complejidad de las tareas.

El *diseño de la evaluación* podría restringir la generalización de los resultados debido a la selección de los modelos de requisitos que se tenían que validar para comprobar si satisfacían las necesidades del cliente. Se han seleccionado un conjunto de necesidades que el sujeto tenía que evaluar si estaba satisfechas en los modelos de requisitos proporcionados. La selección se realizó garantizando que hubiera la suficiente representatividad para que tuvieran que utilizar en la validación tantos los diagramas de casos de uso como los diagramas de actividad.

Otro factor que amenaza a la validez externa es el *tamaño y complejidad de las tareas*. Se han utilizado unas tareas de un tamaño medio con el objetivo de llegar una solución de compromiso entre un ejercicio que tuviera la representatividad suficiente y una complejidad adecuada para que los participantes pudieran completar las tareas en un tiempo limitado.

7.2.6.3 Validez de constructo

La validez del constructo puede verse afectada por las *métricas aplicadas* en el análisis cuantitativo y por la fiabilidad del cuestionario. Para mitigar esta amenaza su definieron las variables subjetivas basándose en el Método de Aceptación de Tecnología (TAM), un método utilizado comúnmente 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). Los resultados obtenidos superaron el umbral mínimo aceptable de (α=0,70) (Maxwell 2002).

7.2.6.4 Validez de las conclusiones

En la validez de las conclusiones las principales amenazas son la recopilación de datos y la validez de las pruebas estadísticas aplicadas.

Sobre el primer factor, *recopilación de datos*, se aplicó el mismo procedimiento en cada sesión con el objetivo de que cada variable dependiente se calculara mediante la aplicación de la misma fórmula.

Respecto a la *validez de las pruebas estadísticas aplicadas*, se utilizaron las pruebas comúnmente aceptadas en la comunidad de la ingeniería del software empírico con el objetivo de mitigarlas (Maxwell 2002).

7.3 Conclusiones

En este capítulo de han presentado los dos cuasi-experimentos realizados para validar respectivamente las guías de la ingeniería del dominio y la ingeniería de la aplicación de FeDRE. En el primer cuasi-experimento se ha analizado la eficacia, eficiencia, facilidad de uso percibida y utilidad percibida de las guías de la ingeniería del dominio de FeDRE. En el segundo cuasi-experimento se ha analizado eficacia, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso de las guías de la ingeniería del dominio de FeDRE.

El primer cuasi-experimento modelaba los requisitos de una LPS de aplicaciones móviles para la notificación de emergencias con el objetivo de validar las guías de la ingeniería del dominio de FeDRE. Los resultados muestran que FeDRE fue percibido por los sujetos como fácil de usar y útil para especificar los requisitos en esta particular LPS. Por otra parte los sujetos aportaron retroalimentación importante para reformular algunas reglas de las guías que podrían inducir a confusión.

En el segundo cuasi-experimento validaba los requisitos de una LPS centrada en el comercio electrónico con el objetivo de validar si cumplía los requisitos para un producto en concreto. La mayor parte de los sujetos fueron capaces de identificar correctamente qué necesidades estaba cubiertas y qué requisitos debían de añadirse como requisitos delta.

Finalmente aunque los resultados de los cuasi-experimentos son prometedores como una evaluación previa de FeDRE, es necesario analizar si se producen los mismos resultados con participantes más experimentados y con un conjunto de requisitos más amplio.

Capítulo 8. Conclusiones y trabajos futuros

En este capítulo se presentan las principales conclusiones obtenidas durante el desarrollo del trabajo y el análisis de los objetivos de investigación que se han alcanzado. Por otra parte se presentan las contribuciones derivadas de esta investigación y los trabajos futuros previstos.

La sección 8.1 presenta las conclusiones del trabajo realizado.

La sección 8.2 presenta el contexto de investigación donde se ha realizado la tesis.

La sección 8.3 presenta los resultados obtenidos de este trabajo y las publicaciones realizadas en el contexto de la tesis.

La sección 8.4 presenta las becas y galardones obtenidos por el doctorando.

La sección 8.5 presenta los trabajos futuros que plantea el trabajo de tesis.

8.1 Conclusiones

En últimos años se han propuesto diversas aproximaciones para soportar la IR en el desarrollo LPS, si bien siguen existiendo múltiples carencias. La mayoría de aproximaciones de IR para LPS se centran meramente en la extensión de técnicas tradicionales de IR para representar la variabilidad durante la ingeniería del dominio, sin tener en cuenta cómo debe de ser resuelta esa variabilidad y cómo se han de incorporar los requisitos específicos del producto (requisitos delta). Por otro lado, la mayoría de propuestas que soportan la derivación de los requisitos del producto no siempre permiten representar dichos requisitos delta. Es por ello que hay una carencia de procesos integrados que permitan la especificación de requisitos a nivel de ingeniería de dominio, incluyendo su variabilidad, y que permitan la resolución de la misma de manera automatizada, dando soporte a la especificación de requisitos delta específicos de producto.

Con el objetivo de dar solución a esta problemática en esta tesis doctoral se ha propuesto y validado empíricamente, mediante dos cuasi-experimentos, un proceso para la definición y especificación de los requisitos de una familia de productos software y de los productos generados a partir de esta. A continuación se analizará el grado de cumplimiento de cada una de las metas definidas en la sección 1.3.

8.1.1 Definición de un proceso de requisitos para LPS basado en una estrategia de DSDM

Respecto a esta meta se ha definido FeDRE, un proceso integrado que cubre tanto la ingeniería del dominio como la ingeniería de la aplicación, dando soporte así mismo a la especificación de requisitos delta. Durante la ingeniería del dominio se toman como entrada los artefactos del *scoping*, incluyendo el modelo de características. FeDRE toma como punto de partida el modelo de características, un artefacto al que están más habituados los expertos del dominio (Oliveira y otros 2014), en contraste con otras propuestas para LPS que, en su mayoría, parten de los modelos tempranos de requisitos. Desde nuestro punto de vista utilizar los modelos de características como entrada al proceso de ingeniería del dominio de requisitos resulta más adecuada, puesto que el modelo de características debe define en la actividad de *scoping* donde se tiene en cuenta la planificación de reúso. A partir del modelo de características se definen y especifican de forma iterativa e incremental los requisitos de la familia de productos. Esta especificación se valida con el cliente con la técnica de inspección de requisitos. Una vez definida la especificación de los requisitos de la familia de productos el siguiente paso es definir el modelo de variabilidad de

requisitos en un modelo independiente de la propia especificación. Con esta representación de la variabilidad en un modelo diferente evitamos sobrecargar los modelos de requisitos con información de variabilidad, mejorando la mantenibilidad de dichos modelos. Por otra parte el modelo de características no es suficiente para especificar la variabilidad de los requisitos dado que es complicado establecer la información de variabilidad del modelo de características a otros artefactos de requisitos (Bühne y otros 2004). La utilización de un modelo de variabilidad de requisitos resulta más adecuada que utilizar el modelo de características de la LPS, debido a que el modelo de características se encuentra a un nivel de granularidad mayor y se evita la ambigüedad que puede producirse entre la variabilidad de los diferentes artefactos de desarrollo (Pohl y otros 2005). El proceso finaliza con la verificación del modelo de variabilidad de requisitos.

Los requisitos de la ingeniería del dominio se utilizan en el proceso de ingeniería de la aplicación. Una vez se define una configuración del producto, se derivan los requisitos del producto a desarrollar. En FeDRE definimos una actividad de validación de los requisitos del producto mediante el artefacto que contiene las necesidades del cliente. Si no es posible satisfacer las necesidades del cliente mediante los requisitos del producto derivados se especifican los requisitos delta. FeDRE al permitir especificar los requisitos delta evita el problema de que la especificación de requisitos del productos se limite únicamente a partir de los requisitos de la LPS (Djebbi y otros 2007), mejorando la flexibilidad en la especificación de los requisitos del producto.

8.1.2 Definición de una aproximación tecnológica de IR para LPS basada estrategia de DSDM

A continuación se presentan los resultados respecto a la meta de definición de una aproximación tecnológica para soportar el proceso FeDRE. Durante la ingeniería del dominio se definen los mecanismos necesarios de la definición y especificación de los requisitos de la LPS mediante una estrategia de DSDM. Los requisitos se representan en un multimodelo que contiene dos vistas: de variabilidad y de requisitos. La vista de variabilidad representa la variabilidad de la LPS y se representa mediante un modelo de características. Por otra parte la vista de requisitos representa la variabilidad interna de la especificación de requisitos de la LPS. La vista requisitos se representa utilizando en lenguaje CVL. El lenguaje CVL permite representar en un modelo la variabilidad de un DSL, en este caso los modelos UML de casos de uso y diagramas de actividad, evitando representar la variabilidad sobre la misma especificación de requisitos. Por otra parte el multimodelo contiene las dos vistas y las relaciones entre ellas.

De esta forma se establece la variabilidad a dos niveles de granularidad distintos; evitando utilizar el modelo de características para representar la variabilidad de los requisitos, el cual puede resultar no ser adecuado para este propósito (Bühne y otros 2004).

Como parte del soporte a la estrategia de DSDM se han definido los metamodelos necesarios. Para la definición del metamodelo que soporta el multimodelo se sigue una arquitectura en dos capas. Por un lado se define un metamodelo llamado núcleo o *core* del multimodelo que contiene las entidades esenciales para representar un multimodelo Por otra parte se define un metamodelo específico para representar la especificación de requisitos de la línea de productos. Esta arquitectura nos permite incorporar en un futuro nuevas vistas de una forma flexible dependiendo del dominio de la LPS (ej. incluir una con la variabilidad de la arquitectura referencia).

Respecto a la aproximación tecnológica para cubrir la ingeniería de la aplicación, se da soporte a la derivación de los requisitos de un producto a partir de la configuración de dicho producto y del multimodelo. Esta definición de la configuración se realiza seleccionando las características de la vista de variabilidad del multimodelo que se incluirán en la configuración del producto. Esta configuración de producto se puede validar, para comprobar que se satisfacen las restricciones del modelo de variabilidad, mediante la herramienta de verificación de modelos de características FaMa. Una vez definida la configuración del producto se derivan los requisitos del producto a partir de dos transformaciones. En la primera transformación se crea un modelo de resolución CVL que contiene el conjunto de puntos de variabilidad o *Choices* que se resuelven positivamente del modelo de variabilidad de CVL. Una vez obtenido el modelo de resolución se valida mediante el validador de CVL para comprobar que todas las *Choices* del modelo de variabilidad de CVL se han resuelto positivamente o negativamente en el modelo de resolución. Si una *Choice* se resuelve positivamente implicará que se realizará una sustitución en el modelo base por un elemento del modelo librería. Una vez realizada la transformación de CVL se realizan los cambios necesarios en el modelo base para obtener finalmente un modelo UML que contiene la especificación de requisitos del producto. De este modo se da soporte completamente a la derivación de los requisitos del producto, evitando uno de los mayores inconvenientes de adopción de las aproximaciones de IR para LPS que es la falta de soporte mediante herramienta (Alves y otros 2010).

Una vez derivados los requisitos podemos especifciar los requisitos delta, en los casos que sea necesario. Hemos definido un perfil UML en Papyrus que permite especificar los requisitos delta utilzando esta herramienta de modelado.

8.1.3 Validación empírica mediante dos cuasi-experimentos

Actualmente muchas de las aproximaciones de IR para LPS no realizan validaciones cualitativas o cuantitativas de sus procesos (Alves y otros 2010). Con el objetivo de suplir esta carencia se ha definido como meta la validación empírica del proceso FeDRE. Para ello, se han llevado a cabo dos cuasi-experimentos para comprobar la facilidad de uso percibida y la utilidad de uso del proceso FeDRE en la ingeniería del dominio y la ingeniería de la aplicación. Adicionalmente en el segundo cuasi-experimento se analizó la intención de uso del proceso en el futuro.

El *primer cuasi-experimento* consistía en especificar los requisitos de una LPS de aplicaciones móviles para la notificación de emergencias con el objetivo de validar las guías de la ingeniería del dominio de FeDRE. El cuasi-experimento involucró a 14 sujetos, 8 de la Universitat Politècnica de València (España) y 6 de la Universidad Federal de Bahía (Brasil). Entre los sujetos encontramos estudiantes de máster y doctorado, y profesionales del desarrollo de software, algunos de ellos con experiencia industrial desarrollando LPS. En el análisis cualitativo se evaluó la facilidad de uso percibida y la utilidad percibida de los participantes cuando emplean FeDRE en la definición y especificación de los requisitos de la LPS a partir de los artefactos de *scoping*. Del análisis cualitativo se desprende que los sujetos apreciaron el proceso de ingeniería del dominio de FeDRE como fácil de usar y útil. Adicionalmente se realizó un análisis cuantitativo de la efectividad y eficiencia, que nos han permitido constatar que, en general, la aplicación del proceso permitió a los participantes definir y especificar los requisitos de la LPS correctamente. Por último se analizaron las respuestas a las preguntas abiertas del cuestionario de evaluación para obtener retroalimentación sobre FeDRE.

Desde un punto de vista investigador, el cuasi-experimento nos ha permitido analizar la factibilidad del proceso durante la definición y especificación de requisitos de una aplicación real, aunque en un contexto controlado. Los participantes, tras aplicar el proceso, nos proporcionaron sugerencias para reformular algunas reglas en las guías, para evitar ambigüedad durante de la definición y especificación de los requisitos.

Desde un punto de vista práctico, esta es una primera validación con un número reducido de participantes y con perfiles heterogéneos. Son necesarias más replicaciones con un mayor número de participantes y grupos más homogéneos para analizar factores que puedan interactuar (por ejemplo la experiencia o la habilidad).

En el *segundo cuasi-experimento* se validaron los requisitos de un producto, derivados a partir de una LPS centrada en el comercio electrónico, para comprobar que se satisfacían las necesidades del cliente. Se seleccionó un grupo de estudiantes de doctorado en diversos temas de ingeniería del software. La primera sesión se compuso de 8 estudiantes de doctorado de la Universitat Politècnica de València (España). Los participantes de dicho grupo eran expertos en diversos temas relacionados con la ingeniería del software pero noveles en el desarrollo de LPS. El segundo grupo estaba compuesto por 7 estudiantes de doctorado de la Universidad Federal de Bahía (Brasil), expertos en desarrollo de LPS tanto en el ámbito industrial como en el de investigación. El experimento contenía dos tareas experimentales. En la primera se evaluaron las necesidades del cliente y una especificación de requsitios derivados a partir de la configuración de un producto. Los participantes validaron si estos requisitos derivados satisfacían las necesidades del cliente. En caso de que todas la necesidades del cliente no estuvieran satisfechas se ejecutó la segunda tarea experimental donde los participantes especificaron los requisitos deltas necesarios para satisfacer todas las necesidades del cliente. Durante el análisis cualitativo se evaluaban la facilidad de uso percibida, utilidad percibida e intención de uso futura de los participantes cuando aplicaban el proceso de validación y especificación de requisitos, siendo los resultados positivos. Durante el análisis cuantitativo se analizaron la efectividad y eficiencia aplicando FeDRE durante la ingeniería de la aplicación.

Desde el punto de vista investigador, el cuasi-experimento 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 proceso. Como resultado del análisis cuantitativo podemos deducir que la mayor parte de los sujetos fueron capaces de identificar correctamente qué necesidades estaban cubiertas y qué requisitos delta debían de especificarse. Por último, pero no menos importante, su obtuvo realimentación de los participantes que aplicaron el proceso.

Desde el punto de vista práctico al ser este un primer estudio con un número muy reducido de participantes, con dos perfiles heterogéneos y que solo arroja resultados preliminares sobre la utilidad, facilidad de uso e intención de uso del proceso de derivación de FeDRE es preciso realizar más replicaciones. En el futuro se tiene previsto replicar este cuasi-experimento en grupos más homogéneos y con un mayor número de participantes para analizar posibles interacciones de factores como la experiencia o la habilidad.

8.2 Contexto de la investigación

Esta tesis doctoral se ha realizado en el contexto del grupo de investigación Ingeniería del Software y Sistemas de Información de la Universitat Politècnica de València.

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: Usabilidad en la Ingeniería de Líneas de Producto Orientadas a Servicios*. Financiado por el Ministerio de Ciencia e Innovación, Gobierno de España con referencia PHB2011-0015. Programa Hispano-Brasileño de Cooperación Interuniversitaria. Duración: enero del 2013 al 31 de diciembre de 2013. Investigadores principales: Silvia Abrahão (UPV) & Eduardo Santana de Almeida (coordinador UFBA).
- *Proyecto MULTIPLE: Multimodeling Approach for Quality-Aware Software Product Lines*. Financiado por el Ministerio de Ciencia e Innovación, Gobierno de España con referencia TIN2009-13838. Duración desde octubre de 2009 a septiembre de 2013. Investigador principal: Silvia Abrahão.
- *Transformación de Modelos Dirigida por Atributos de Calidad.* Financiado por la Universitat Politècnica de València, referencia PAID-06-07-3286. Duración desde diciembre de 2007 a diciembre de 2009.
- *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.

8.3 Resultados

En este trabajo de tesis se han realizado dos publicaciones en revistas, una de ellas indizada en el ranking JCR. Respecto a las publicaciones en congresos se ha realizado una publicación en una conferencia internacional catalogada de nivel A según el ranking ERA-CORE, una publicación en el congreso internación SBCARS y una demo en la conferencia MODELS, cuya conferencia principal se encuentra indizadas en categoría "B" en el índice ERA-CORE. Actualmente se encuentra en proceso de revisión una publicación con el título "*An Approach for Specifying Product Requirements in Software Product Line Development*", en la revista indizada en el ranking JCR: Requirements Engineering Journal. A continuación se describen las publicaciones en mayor detalle.

8.3.1 Publicaciones derivadas de la tesis

En esta sección se presentan las publicaciones derivadas de este trabajo de tesis.

8.3.1.1 Revistas internacionales

- *Defining and Validating a Feature-Driven Requirements Engineering Approach*. Raphael Pereira de Oliveira, David Blanes, Javier Gonzalez-Huerta, Emilio Insfran, Silvia Abrahão, Sholom Cohen, Eduardo Santana de Almeida. Journal of Universal Computer Science (JUCS). Factor de impacto: 0,762 (ISI Web of Science JCR report 2012).
- *A Comparative Study on Model-Driven Requirements Engineering for Software Product Lines*. David Blanes Dominguez; Emilio Insfran. Revista de Sistemas e Computação 2(1), pp. 3 - 13. Universidade Salvador (Brazil), 2012. ISSN 2237-2903.

8.3.1.2 Actas de congresos internacionales

- *A Multimodel Approach for Specifying the Requirements Variability on Software Product Lines*. David Blanes, Javier González, Emilio Insfran. 23rd International Conference on Information Systems Developments (ISD), pp. 329-336, Varazdin, Croacia (2014). ISBN 978-953-6071-43-2. Catalogado como "A" en el índice Era-Core.
- *A Feature-Driven Requirements Engineering Approach for Software Product Lines*. Raphael Oliveira, Emilio Insfran, Silvia Abrahao, Javier González, David Blanes, Sholom Cohen, Eduardo Almeida. 7th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS 2013), Brazilia-DF, Brazil. IEEE Computer Society, 2013. Observaciones: galardonado como la tercera mejor contribución del congreso.
- *A tool for the automatic generation of multimodel editors*. Tool Demostrations track, 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). CEUR Workshop Proceeding, 2015. Conferencia indizada como Era-Core "B".

8.3.1.3 Artículos en proceso de revisión

- *An Approach for Specifying Product Requirements in Software Product Line Development*. David Blanes, Javier González, Emilio Insfran. Requirements Engineering Journal (REJ). Factor de impacto: 0,882 (ISI Web of Science JCR report 2014) (En proceso de revision).

8.3.1.4 Otras publicaciones

Se ha publicado un informe técnico en el año 2011:

- *Review of Requirement Engineering Approaches for Software Product Lines*. Autores: David Blanes, Emilio Insfran. Entidad: Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València.

8.3.2 Publicaciones durante la fase de formación

En esta sección se presentan los resultados se presentan los resultados obtenidos en fase de formación durante el Máster de Ingeniería del Software, Métodos Formales y Sistemas de Información impartido en el Departamento de Sistemas Informáticos y Computación de la Universitat Politècnica de València.

8.3.2.1 Publicaciones en revistas

- A Requirements Engineering Approach for the Development of Multi-Agent Systems. David Blanes Dominguez; Emilio Insfrán; Silvia Abrahao. International Journal of Software Engineering and Its Application (IJSEIA). 4, pp. 1 - 14. 2010. ISSN 1738-9984.

8.3.2.2 Actas de congresos internacionales

- *Requisitos de Comportamiento Social para Sistemas Multi-Agente*. XXXVI Conferencia Latinoamericana de estudios de informática (CLEI 2010). Lorena Rodríguez; David Blanes; Emilio Insfrán. Asunción, Paraguay. Fecha de realización: 18/10/2010., .pp. 1 - 9. Facultad Politécnica-Universidad Nacional de Asunción, Universidad Autónoma de Asunción.

- *Towards an Architecture for Ensuring Product Quality in Model-Driven Software Development*. 11th International Conference on Product-Focused Software Process Improvement (PROFES 2010). Javier González Huerta; David Blanes; Emilio Insfrán; Silvia Abrahao Gonzales. Limerick, Irlanda. Fecha de realización: 21/06/2010. "Short Paper".pp. 1 - 4. Springer. Catalogado como "B" en el índice Core.

- *A Requirements Modeling Approach for the Development of Multi-Agent Systems using Gaia*. David Blanes Domínguez; Emilio Insfrán; Silvia Abrahao. International Conference on Advanced Software Engineering & Its Applications (ASEA 2009) Jeju Island, Korea, Fecha de realización: 10/12/2009, pp. 245 - 252. Springer. Nombrado mejor artículo del congreso.

- *Requirements Engineering in the Development of Multi-Agent Systems: A Systematic Review*. David Blanes; Emilio Insfrán; Silvia Abrahao. 10th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2009) Burgos, España, Fecha de realización: 23/09/2009.

8.4 Becas y galardones

- Ayuda FPU convocatoria 2009. Entidad/es financiadora/s: Ministerio de Economía y Competitividad. Fecha de inicio: 01/05/2011, 48 meses. Cuantía total: 45.026,64.
- Premio al tercer mejor artículo en el congreso "7th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS 2013)".

8.5 Trabajos futuros

En este documento se ha presentado el trabajo realizado durante la tesis doctoral. Sin embargo esto no significa el fin del trabajo de investigación en el área. Como objetivo a corto y medio plazo se deben de afrontar las tareas pendientes hasta liberar la solución definitiva de acuerdo con el modelo de investigación y transferencia tecnológica aplicado en la tesis. En esta tesis se han cubierto las seis primeras fases de la metodología de investigación (Gorschek y otros 2006) quedando abierta la validación realista. Como trabajo futuro se deben de realizar experimentos para realizar validaciones en un contexto industrial con el objetivo de lanzar finalmente la solución final.

Otro trabajo futuro abierto tiene como objetivo añadir una notación para representar los RNF de la LPS (ej. modelos de metas (Chung y otros 2000)). Quedando abierto el problema sobre cómo representar debidamente los RNF: como una nueva vista del multimodelo o dentro de la propia vista de requisitos (funcionales). El trabajo futuro estudiará los efectos positivos y negativos de ambas estrategias con el objetivo de encontrar una representación idónea. Además se tiene previsto extender las guías de la ingeniería de la aplicación para poder definir y especificar los RNF.

Otro trabajo pendiente en sobre el multimodelo es la integración de la variabilidad de la LPS y la variabilidad de requisitos con la variabilidad de la arquitectura propuesta en la tesis de González-Huerta (2014), estudiando las posibles relaciones entre los elementos de dichas vistas y la trazabilidad entre ellas.

Respecto al proceso FeDRE se tiene previsto añadir mayor integración con la actividad de *scoping*. Una de las mejoras posibles es definir actividades en el proceso que cubran la inclusión de los requisitos delta, cuya reutilización vaya a ser necesaria en otros productos, dentro de la especificación de requisitos del dominio. Esta mejora incluiría la definición de métricas para validar el grado de reutilización y de coste de integración del requisito delta para decidir si sería rentable o no incluirlo en la LPS.

Por otra parte se tiene previsto mejorar la herramienta que soporta a FeDRE. Por una parte se tiene previsto mejorar la importación y actualización de la vista, sustituyendo el código actuar en EMF dinámico por un lenguaje de transformación (ej. ATL (Jouault y otros 2006)). Esta extensión permitirá realizar la transformación de las entidades de una vista a entidades del multimodelo.

Respecto al editor del multimodelo se tiene previsto extender el editor en árbol para permitir la edición del multimodelo con una sintaxis concreta gráfica con el objetivo de mejorar aspectos de usabilidad como resultado de los comentarios de los sujetos durante el cuasi-experimento para validar la ingeniería del dominio.

Otro problema abierto es la mejora del soporte a la derivación de requisitos del producto mediante CVL. El soporte en la versión actual de CVL deriva un modelo UML que parte del modelo base y sobre el que se le realizan las resoluciones de los puntos de variabilidad. Sin embargo no es posible con la tecnología actual reconstruir los diagramas resueltos.

Apéndice A Guías de FeDRE

En este apéndice se presentan las guías propuestas en el proceso FeDRE. Estas guías se dividen en dos bloques: las guías para la ingeniería del dominio y las guías para la ingeniería de la aplicación. El material está en inglés puesto que las guías se definieron en este idioma para preparar el material experimental durante el primer cuasi-experimento.

A.1 Guías de la ingeniería del dominio

Task 1: Define Domain requirements

1.1. Identify ***which*** *Features* or *group of Features* share functionality. For each Feature or group of Features, identify if it needs Use Cases:

1.1.1.Root mandatory *Features* or intermediate mandatory *Features* must have *Use Cases*;

1.1.2. Mandatory leaf *Features* may have *Use Cases*, or can be specified as extension points on the activity diagrams that specify the *Use Cases* in the parent *Feature*;

1.1.3. Root optional *Features* or intermediate optional *Features* must have *Use Cases;*

1.1.4. *Use Cases* identified for leaf optional *Features* should be specified as *Use Cases that extends Use Cases* in the parent *Feature* with extension points, or extension points on the activity diagrams that specify the *Use Cases* in the parent *Feature;*

1.1.5. Intermediate alternative *Features (OR)* must have *Use Cases;*

1.1.6. *Use Cases* identified for leaf alternative *Features (OR)* should be specified *Use Cases that extends Use Cases* in the parent *Feature* with extension points or as extension points on the activity diagrams that specify the *Use Cases* in the parent *Feature;*

1.1.7. Intermediate alternative *Features (XOR)* must have *Use Cases;*

1.1.8. *Use Cases* identified for leaf alternative *Features (XOR)* should be specified as *Use Cases that extends Use Cases* in the parent *Feature* with extension points, or extension points on the activity diagrams that specify the *Use Cases* in the parent *Feature;*

1.2. Identify ***what*** specific *Use Cases* for each feature or group of features are;

1.2.1. *Use Cases* identified for children *Features* (that have the same parent) with similar behavior must be specified just once in the parent *Feature* (***where***);

1.2.2. Each *Feature*, which has one or more identified *Use Case*, should have a *Use Case Package*. Each *Use Case Package* should have a *Use Case Diagram*;

1.2.3. For each *Use Case Diagram:*

1.2.3.1. *Identify Actors*

1.2.3.1.1. If one Actor specializes another actor, then add an *Inheritance Relationship* from the specialized actor to the base actor.

1.2.3.1.2. If the *Actor* participates in a *Use Case* then add a association relationship from the *Actor* to the *Use Case*

1.3. If one Use Case (base) includes another Use Case (inclusion), then add a *Includes To* Relationship from the base Use Case to the Inclusion Use Case

1.3.1.1. If one Use Case (extension) extends the behavior of another Use Case (base), then add a Extend Relationship form the extension Use Case or the base Use Case

1.3.1.2. For each **extension point** through Use Cases, create an *Extend Relationship* form the extension Use Case or the base Use Case

Task 2: Specify Domain requirements

2.1. Create two *Activity Partitions*. One for each *Actor* associated with the specified *Use Case*, and another one for the **System**.

2.1.1. If there are more *Actors* associated to the use case: create a new *Activity Partition* for each *Actor.*

2.2. Create an *Initial Node* in the activity partition associated with the main *Actor*. The main *Actor* is the *Actor* that has an association relationship with the current *Use Case* that is been specified. If multiple *Actors* have an association to the *Use Case* then the main actors is who starts the flow of activity.

2.3. Add the Activity Diagram elements.

2.3.1. Add Actions and Control Flows to describe the System as a series of actions with the control flowing sequentially from one action to the next.

2.3.1.1. If an Activity has a precondition then a new precondition is added in the diagram in the activity

2.3.1.2. If an Activity has a postcondition then a new postcondition is added in the diagram in the activity

2.3.1.3. Add *Control Flows* among Activity Diagram elements when it is needed.

2.4. *Decision Nodes* should be placed when multiple alternative flows are possible.

2.4.1. For each alternative a Guard must be defined.

2.5. A *Merge Node* should be placed when multiple flows has the same target activity.

2.6. A *Fork Node* is added when the execution of multiple concurrent activities are possible. A *Fork Node* is a control node that splits a flow into multiple concurrent flows.

2.7. A *Join Node* is used when it is needed to synchronize multiple flows. A *Join Node* has multiple incoming edges and one outgoing edge.

2.8. For each **extension point** a new decision node is created

2.8.1. For each alternative a new outgoing *Control Flow* will be created. A guard must be defined for each new outgoing control flow.

2.9. Create a *Final Activirty Node* at the end of the System activity partition.

A.2 Guías de la ingeniería de la aplicación

1. For each not covered *need*.

1.1. If the *need* is a functional block, it will be specified as a new *use case*(s).

1.2. If the *need* is part of a functional block, it is specified as an *action* in a use case specification.

Task 1: Identify requirements deltas.

2. For each need that must be specified such as use case o set of use cases. Select the *package* and the *use case diagram* where the use case should be specified.

Identify use cases for each use case diagram

2.1. Add the necessary uses cases in the selected use case diagram. Tag every *use case* with the "delta" stereotype.

2.2. For each identified use case update, insert a row in the deltas requirements traceability matrix. Put in the column requirement delta field the name of the identified use case. In the type field put "use case".

2.3. If an actor participates in a new delta use case, a new association is added from the actor to the use case

2.4. In the deltas requirements traceability matrix, select the row where is the delta use case and add in the "related requirements" column the actor name.

2.5. If the new use case (base) includes another use case (inclusion), then add a includes relationship from the new use case to the Inclusion use case

2.6. In the requirements deltas traceability matrix add the name of the included use case in the related requirements field.

2.7. If the new use case (extension) extends the behavior of another use case (base), then add an extend relationship form the new use case to the base use case.

2.8. In the requirements deltas traceability matrix add the name of the extended use case in the related requirements field.

Identify actors

2.9. Each new (delta) actor is tagged with the "delta" stereotype

2.10. For each new (delta) actor, insert a new row in the requirements deltas traceability matrix. In the "requirement delta" column put the actor name. In the "type" field put "actor".

2.11. If the actor extends an existing actor, then add a generalization relationship from the new actor to the actor that will be specialized

2.12. If the actor represents an external system, then tag the actor with the «system» stereotype.

Task 2: Specify Requirements Deltas.

3. For each need that have to be specified as *action* in an *activity diagram*, select the activity diagram and add the necessary elements. Create a new *activity partition* for each new (delta) actor.

3.2. Add *actions* and *control flows* to describe the System as a series of actions with the control flowing sequentially from one action to the next.

3.2.1. Each (delta) action is tagged with the "delta" stereotype

3.2.2. For each identified action, update inserting a row in the deltas requirements traceability matrix. Put in the column requirement delta field the name of the identified action. In the type field put "action".

3.3. Add *control flows* among activity diagram elements (actions, decision nodes, merge nodes) when it is needed.

3.4. *Decision nodes* should be placed when multiple alternative flows are possible.

3.4.1. For each alternative, a *guard* must be defined.

3.5. A *merge* node should be placed when multiple flows has the same target activity.

3.6. A *join* node is used when it is needed to synchronize multiple flows. A join node has multiple incoming edges and one outgoing edge.

3.6.1. For each alternative from a decision node, a new outgoing control flow will be created. A guard must be defined for each new outgoing control flow.

3.7. Create a *final activity* node at the end of the System activity partition.

Apéndice B Material del cuasi-experimento de la ingeniería del dominio

En este apéndice se adjunta el contenido del material experimental del primer cuasi-experimento realizado para validar las guías de la ingeniería de dominio de FeDRE. Este material experimental está compuesto de un boletín con instrucciones generales y cuatro apéndices. Además se elaboró un cuestionario para recoger las opiniones de los usuarios sobre las guías de FeDRE. El material está escrito en inglés puesto que el material del experimento se realizó en ese idioma.

B.1 Bulletin

Problem description: the Savi Product Line

Savi is a product line of mobile applications for the management of notifications in emergencies (such as robberies, kidnapping, etc.). In a risky situation, the user can send notifications to his/her contacts. With this information, the contact can track the user and notify the responsible such as the police.

In order to use the Savi application, the user must be registered in the system. The user can create an account by signing up through the registration form. Alternatively, the user can create an account by using his/her Twitter or Facebook account. In this case, the user provides the login information in Twitter or Facebook, and then the system links the account with the user's Twitter/Facebook account.

Once the user is registered, then he/she can be identified in the system. If the user forgets the password then a new password can be requested by filling the "remember password" form. In this case, the system will validate the data and send back an e-mail to the user with the new password.

When the user is identified in the application, a main screen will be shown. The user could access to his/her profile and edit his/her data (e.g. name, phone number, e-mail, etc.).

The most important function in the system is the emergency notification. The user can select a subset of contacts from the Savi contact list and send them an emergency notification. This contact list is different from the phone's contact list. The registered user can create a Savi contact manually; in this case, the registered user fills a form with the contact data: contact name, phone number, and e-mail address. Alternatively, the user can import a contact to the Savi contact list from an external system (e.g. the phone contact list). If the application has the social support, then the contact can be imported by using his/her Facebook/Twitter contact list. Finally, the user could update or delete a contact, in this case, the Savi contact list will be shown and then the user could select a contact to be updated/deleted.

The emergency notifications can be sent to each contact one way depending of the application. Savi can send notifications to emergency numbers (e.g. police, firefighters, etc.) by SMS. Alternatively, if the social support is enabled then the user can send emergency notifications by using Twitter o Facebook.

Since the emergency numbers vary depending on the current location, user could add emergency numbers by filling the "add new emergency number" form if the

application has support. The user can eventually update or delete an existing emergency number; in this case, the list of emergency numbers is shown and the registered user can select an emergency number to be updated/deleted.

The user can also log in by using a traditional web browser. The web interface only displays an extended version of the user data profile (e.g. associated Twitter account, associated Facebook account) and allows deleting the user account (this function will not be available in the application interface).

Task 1: identify use cases

Fill in the start time (hh:mm):

In order to achive this task you must to read the following documents:

- *Feature Specification (B.2)*. Cointains a description for each Feature in detail.
- *Glossary (B.3)*. It contains a list of domain vocabulary necessary to understad the Feature Specification.
- *Product Map (B.4)*. It contains the relation between each feature from the Feature Model with the products of the Software Product Line.

The details of the structure and use of each document can be found in the section A.1

Please for each feature in the Feature Model apply the FeDRE guidelines (A.1 –*Task I: Identify Use Cases*).

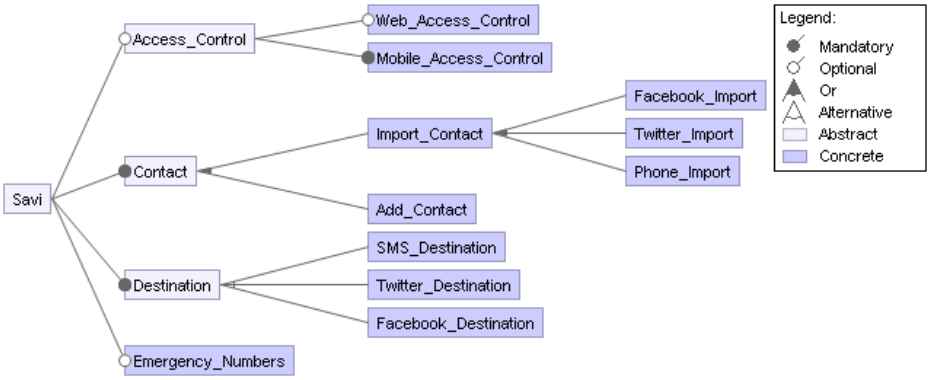


Figure 1. Feature Model

Build the Traceability Matrix

The Traceability Matrix represents the relationships among features and functional requirements (use cases). The rows in the matrix show the use cases and the columns show the features.

To build the traceability matrix, follow the FeDRE guidelines (A.1, *Task 1: Identify Use Cases*).

In the table below, for the set of Use Cases identified in Task 1.1:

1.1.1. Write down the name of the Use Cases in each row.

1.2.2. Indicate the relationships among features and Use Cases using an "X" in the corresponding cell.

1.2 Identify the Use Cases taking as input the Feature Model shown in Figure 1.

1.2.1 In the pages 5-6, draw the resulting use cases diagrams obtained after applying the FeDRE guidelines (1).

Use Case Name	Access Control	Web Access Control	Mobile Access Control	Contact

Use Case Name	Import Contact	Facebook Import	Twitter Import	Phone Import

Use Case Name	Destination	SMS Destination	Twitter Destination	Facebook Destination

Use Case Name	E-mail Destination	Emergency Numbers

Fill in the finish time (hh:mm):

Task 2 - Specify Use Cases

Fill in the start time (hh:mm):

2.1 - From the set of Use Cases identified in Task 1, write down the name of the Use Cases that are related to the Access_Control feature, and their respective child features.

- Access_Control:

- Web_Access_Control:

- Mobile_Access_Control:

2.2 - For each one of the identified Use Cases, perform the Use Case Specification by using the template provided on Table B.1 **(use as many templates as needed). You should follow the FeDRE guidelines described in A.1 *- page 2, Task 2: Specifiy Use Cases.***

Table B.1 Requirements specification template

Use Case ID	<<ID>>	
Use Case Name	<<NAME>>	
Description	<<DESCRIPTION>>	
Associated feature(s) [1..*]	<<ASSOCIATED_FEATURE(S)>>	
Actor(s) [0..*]	<<ACTOR(S)>>	
Pre-condition	<<PRE_CONDITION>>	
Post-condition	<<POST_CONDITION>>	
Includes To	<<USE CASE NAMES>>	
Extends From	<<USE CASE NAMES: CONDITION>>	
Main Success Scenario		
Step	***Actor Action***	***Blackbox System Response***
<<STEP>>	<<ACTOR_ACTION>>	<<BLACKBOX_SYSTEM_POSITIVE_RESPONSE>>
Alternative Scenario name [0..*]		<<ALTERNATIVE_SCENARIO_NAME>>
Condition		<<CONDITION>>
Associated feature(s) [0..*]		<<ASSOCIATED_ FEATURE(S)>>
Step	**Actor Action**	**Blackbox System Response**
<<STEP>>	<<ACTOR_ACTION>>	<<BLACKBOX_SYSTEM_POSITIVE_RESPONSE>>

You cand find additional information to understad the Specification template in A.1 - *Task 2: Specify Use Cases.*

Use Case ID	UC-01	
Use Case Name		
Description		
Associated feature(s) [1..*]		
Actor(s) [0..*]		
Pre-condition		
Post-condition		
Includes To		
Extends From		
Main Success Scenario		
Step	***Actor Action***	***Blackbox System Response***
Alternative Scenario name [0..*]		
Condition		
Associated feature(s) [0..*]		
Step	**Actor Action**	**Blackbox System Response**
Alternative Scenario name [0..*]		
Condition		
Associated feature(s) [0..*]		
Step	**Actor Action**	**Blackbox System Response**

Fill in the finish time (hh:mm):

B.2 Feature Specification

Fill in the start time (hh:mm):

[F001] ACCESS CONTROL			
Priority:	High	Variability:	Optional
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	-
Child features:	F002, F003	Type:	Abstract
Description:	The user can create an account in the system by using the registration form or importing the data from a social network account (Twitter, Facebook). If the user forgets the password, then he/she can request a new one. In this case, the System will send an e-mail with the new password. When the user is registered, then he/she can sign up by entering the username and password. Moreover, it is possible to modify the user profile.		

[F002] WEB ACCESS CONTROL			
Priority:	High	Variability:	Optional
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	-
Child features:	-	Type:	Concrete
Description:	A user can access through the web interface. The web interface only allows to show an extended version of the user data profile (e.g. associated Twitter account, associated Facebook account, etc.) and deleting the user account (this function will not be available in the application interface).		

[F003] MOBILE ACCESS CONTROL			
Priority:	High	Variability:	Mandatory
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	-
Child features:	-	Type:	Concrete
Description:	The mobile access control shows a basic user profile. It is not possible to delete a Savi account from the mobile application.		

[F004] CONTACT			
Priority:	High	Variability:	Mandatory
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	-
Child features:	F005, F006	Type:	Abstract
Description:	The contact (s) are those who will receive the emergency notifications. The application will store important information (such as e-mail, phone number, and so on) about the person to be contacted. The contact feature allows to add a contact and, eventually, to import from different sources.		

[F005] IMPORT CONTACT			
Priority:	Medium	Variability:	Mandatory
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	F004
Child features:	F004, F005, F006	Type:	Abstract
Description:	The user will import information about the intended contact to be added in the list. In this process the phone number, Facebook profile, Twitter profile and others personnel data are examples of retrieved information.		

[F006] FACEBOOK IMPORT			
Priority:	Low	Variability:	Optional
Requires:	F013	Excludes:	-
Binding time:	Compile	Father feature:	F005
Child features:	-	Type:	Concrete
Description:	The user imports a contact from Facebook to the Savi contact list.		

[F007] TWITTER IMPORT			
Priority:	Low	Variability:	Optional
Requires:	F012	Excludes:	-
Binding time:	Compile	Father feature:	F005
Child features:	-	Type:	Concrete
Description:	The User imports a contact from Twitter to the Savi contact list.		

[**F008] PHONE IMPORT**			
Priority:	Low	Variability:	Optional
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	F005
Child features:	-	Type:	Concrete
Description:	The User imports his/her contacts from the phone contact list to the Savi contact list.		

[F009] ADD CONTACT			
Priority:	High	Variability:	Mandatory
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	F004
Child features:	-	Type:	Concrete
Description:	The user will provide information about the contact to be added in the list. In addition, the user has the choice of importing a contact. Phone number, Facebook ID, Twitter ID and e-mail are example of information provided by the user.		

[F010] DESTINATION			
Priority:	Medium	Variability:	Mandatory
Requires:		Excludes:	-
Binding time:	Compile	Father feature:	-
Child features:	-	Type:	Concrete
Description:	The system sends a warning message for each selected contact on the contact list.		

[F011] SMS DESTINATION			
Priority:	Low	Variability:	Mandatory
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	F010
Child features:	-	Type:	Concrete
Description:	The system sends a SMS message to the Savi contacts		

[F012] TWITTER DESTINATION			
Priority:	Low	Variability:	Optional
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	F010
Child features:	-	Type:	Concrete
Description:	The user requests to send a tweet to his/her contacts		

[F013] FACEBOOK DESTINATION			
Priority:	Low	Variability:	Optional
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	F010
Child features:	-	Type:	Concrete
Description:	The user requests to send Facebook message to his/her contacts		

[F014] EMERGENCY NUMBERS			
Priority:	Low	Variability:	Optional
Requires:	-	Excludes:	-
Binding time:	Compile	Father feature:	-
Child features:	-	Type:	Concrete
Description:	The emergency numbers vary depending on the current location. The registered user could add emergency numbers by filling the "add new emergency number" form. The registered user can also update or delete an existing emergency number; in this case the list of emergency numbers is shown and the registered user can select an emergency number to be updated/deleted.		

B.3 Glossary

This document describes the most technical definitions in the Savi Software Product Line. The glossary goal is to explain the main terms in the Domain, in order to share a common vocabulary between the developers

Common Alert Protocol	The Common Alerting Protocol (CAP) is an XML-based data format for exchanging public warnings and emergencies between alerting technologies. CAP allows a warning message to be consistently disseminated simultaneously over many warning systems to many applications. CAP increases warning effectiveness and simplifies the task of activating a warning for responsible officials.
Contact	It represents a person to be contacted in an emergency. It includes relevant information like e-mail, phone number, Facebook ID, Twitter ID.
Contact List	It is a collection of contacts sorted in an alphabetical order.
Destination	It is the way that the contact will receive the emergency message.
Emergency Number	It is a telephone number that allows contacting with local emergency services. The emergency number depends on the region.
Facebook	Social Web Site. It is a site where the user can share information with other contacts.
Geographic coordinate system	A geographic coordinate system is a coordinate system that enables every location on the Earth to be specified by a set of numbers and/or letters. The coordinates are often chosen such that one of the numbers represents vertical position, and two or three of the numbers represent horizontal position.
Internet Connection	It is the access to the Internet through the user phone or web access.
Language	It is the language in which the contact will receive the message.
Location	It is the exact point, on the Earth's surface, retrieved by the geographic coordinate system.
Phone	Device that contains the Savi application.
Region	A region is a medium-scale area where an emergency number is shared.
Route	A way or course taken by the User in getting from a starting point to a destination.
SMS	Short Message Service (SMS) is a text messaging service that allows the exchange of short text messages between mobile devices.
Twitter	Microblogging service. It is a site where the user can share small message.
User	It represents the person who uses the application.

B.4 Product Map

There are planned five products in this SPL:

- *Savi Lite*. It is the simplest version and convers just de essential functionality. It allows sending emergency messages by SMS.
- *Savi Standard*. This product adds to the Savi Little the option of importing contacts from the phone agenda and show a list of emergency numbers.
- *Savi Social*. This product adds "social" functionality such as: import contacts from Facebook, Twitter and send emergencies by using this both social networks.
- *Savi Pro*. It adds the option of track the position in the phone.
- *Savi Ultimate*. It is the most complete product of the family.

The functionality for each product can be seen in the product mapping. This artifact relates features from the feature model with every product.

Features / Products	Savi Lite	Savi Standard	Savi Social	Savi Pro	Savi Ultimate
Access_Control			X	X	X
Web_Access_Control			X	X	X
Mobile_Access_Control			X	X	X
Contact	X	X	X	X	X
Add_Contact	X	X	X	X	X
Import_Contact		X	X	X	X
Facebook_Import			X	X	X
Twitter_Import			X	X	X
Phone_Import		X	X	X	X
Destination	X	X	X	X	X
SMS_Destination	X	X	X	X	X
Twitter_Destination			X	X	X
Facebook_Destination			X	X	X
Emergency_Numbers		X	X	X	X

B.5 Survey about the use of Feature-Driven Requirements Engineering approach

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.	I believe that the SPL requirements specifications obtained with the FeDRE approach are disorganized, unclear, unconcise and ambiguous	O	O	O	O	O	I believe that the SPL requirements specifications obtained with the FeDRE approach are organized, clear, concise and non-ambiguous
2.	I believe that the FeDRE approach has enough expressiveness to represent functional SPL requirements	O	O	O	O	O	I believe that the FeDRE approach has not enough expressiveness to represent functional SPL requirements
3.	It is difficult for me to follow the guidelines proposed by FeDRE approach	O	O	O	O	O	It is easy for me to follow the guidelines proposed by FeDRE approach
4.	The guidelines for specifying SPL functional requirements are easy to learn	O	O	O	O	O	The guidelines for specifying SPL functional requirements are difficult to learn
5.	I believe that FeDRE approach would increase the time required to specify SPL requirements	O	O	O	O	O	I believe that FeDRE approach would reduce the time required to specify SPL requirements
6.	Overall, I found the FeDRE approach to be useful	O	O	O	O	O	Overall, I found the FeDRE approach to be unuseful
7.	The FeDRE approach is complex and difficult to follow	O	O	O	O	O	The FeDRE approach is simple and easy to follow.

Please answer the following questions:

8. Do you have any suggestion on how to make this requirements specification method more easy to use?

9. What are the reasons that will make usable or not this method in the future?

Please write any other comment or suggestion related to requirements specification method that you want to do in the space below:

_____

_____

_____

_____

Apéndice C Material del cuasi-experimento de la ingeniería de la aplicación

En este apéndice se presenta el material experimental utilizado durante el cuasi-experimento para validar la ingeniería de la aplicación. El material experimental está compuesto de un boletín principal con la descripción del ejercicio y documentos anexos con información adicional.

C.1 Boletín

Descripción del ejercicio

Durante este ejercicio se van a especificar los requisitos delta de un producto en un contexto de desarrollo de líneas de producto software. La Figura C.1 muestra el diagrama SPEM con el proceso.

El ejercicio se centrará en una tarea de este proceso:

Tarea 1. Validar los requisitos del producto.

Se proporcionan los requisitos obtenidos automáticamente con la herramienta para el producto. Se deben de validar estos requisitos para comprobar que se satisfacen completamente las necesidades del cliente. Si no se satisfacen en ese caso hay que ir a la tarea 2 para especificar los requisitos delta que completarán la especificación. Si se considera que esta especificación satisface las necesidades del cliente en ese caso el proceso termina.

Tarea 2. Especificar los requisitos delta para el producto.

Partiendo de los requisitos obtenidos en la tarea obtener *requisitos del producto* (diagramas de caso de uso y actividad del producto) se deberá *especificar los requisitos delta* que sean necesarios para satisfacer las necesidades del cliente.

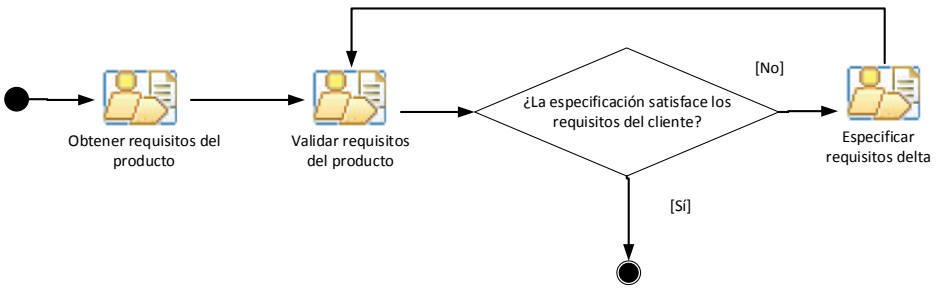


Figura C.1 Proceso de especificación de los requisitos de un producto

Descripción la de línea de productos: Órdenes

La compañía Trey Research™ ofrece servicios de comercio electrónico a sus clientes. Uno de sus desarrollos más notorios es la línea de productos software "Órdenes" la cual provee una solución a la gestión de una tienda de productos a través de Internet. La línea de productos incluye la funcionalidad necesaria para realizar la gestión del control de acceso, auditoría de los diarios del sistema (*logs*), aseguramiento de la conformidad con los estándares, gestión de mensajes para comunicarse con los proveedores, gestión y proceso de los pedidos y la gestión de pagos y devoluciones, que pueden ser mediante tarjeta de crédito y/o PayPal™. En este ejercicio nos centraremos en este último apartado, la gestión de pagos.

Como información adicional en **los anexos III** (sección C.3) **y IV** (sección C.4) se muestran los diagramas de casos de uso y la especificación, mediante diagramas de actividad, de la gestión de pagos de la familia de productos Órdenes. Además en el **anexo V** se muestra el diagrama de variabilidad CVL de los requisitos de la línea de productos.

Requisitos del producto: ÓrdenesCoin

En los últimos años ha tenido un auge adicional el uso de criptomonedas o monedas digitales. Para satisfacer a los usuarios que desean efectuar el pago de sus productos con estas monedas emergentes, la aplicación *ÓrdenesCoin* viene a cubrir este segmento del mercado. En concreto se dará soporte a la moneda Bitcoin, la cual ha ganado fama dado su diseño basado en código abierto.

En el **anexo II** (sección C.2) se muestran las características de la línea de productos que se incluirán en *ÓrdenesCoin*. El producto tiene que ser capaz de gestionar el control de acceso al sistema mediante un usuario y contraseña (autorización local). Se incluirá también la gestión de los *logs* del sistema (auditoría) aunque sólo de forma local, no permitiendo el almacenamiento de los *logs* en servicios Cloud. *ÓrdenesCoin* también soportará la conformidad de los estándares: tanto la conformidad estándar (suficiente para ventas nacionales) como la adicional (satisface los requisitos para realizar ventas internacionales). Por otra parte se da soporte a la mensajería tanto local como externa mediante servicios en la nube. También se dará soporte a crear y procesar encargos. Este proceso de los pedidos podrá hacerse mediante el sistema (proceso local) o servicios en la nube (proceso externo).

Respecto a la gestión de pagos *ÓrdenesCoin* soportará el pago (por defecto bancario), pago mediante Paypal y pago mediante Bit Coins. Con el pago (bancario) el cliente identificado en el sistema tiene que introducir su número de tarjeta de crédito, el número de seguridad que se encuentra en el código de

seguridad y la fecha de vencimiento de la tarjeta. Con estos datos el cliente identificado *requiere autorización* al sistema. El sistema *comprobará la disponibilidad de saldo* en la cuenta asociada a la tarjeta de crédito del cliente solicitando al sistema gestión del Banco Cliente que se haga una *comprobación de fondos*. Si los fondos son suficientes entonces el sistema *autoriza la trasferencia* y se realiza el pago, en caso contrario se muestra un mensaje de error.

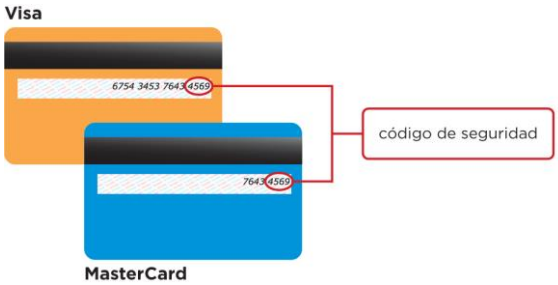


Figura C.2 Código de seguridad que se debe introducir en el pago bancario

El cliente también podrá realizar un pago mediante una cuenta Paypal. En este caso para realizar un pago el cliente *requiere autorización* para abonar el importe del pedido. El sistema cuando recibe la petición debe de proceder a *comprobar la disponibilidad de saldo* para realizar la transacción. En este caso el sistema gestor de Paypal debe de *comprobar la cantidad de crédito* en dicha cuenta para verificar que se puede realizar la transacción. Paypal trasmitirá el resultado de la comprobación al sistema *ÓrdenesCoin* y el sistema autorizará la trasferencia o bien procesará el pago o bien mostrará un mensaje de error.

Por último el pago mediante BitCoins funciona de forma similar. Sin embargo, a diferencia del pago mediante tarjeta bancaria, con el Bitcoin no existe un banco y en su lugar existe un *Cartera Virtual* que almacena los Bitcoins. En esta modalidad de pago el cliente requiere la autorización y el sistema se encarga de comprobar la disponibilidad de saldo. En este caso la Cartera Virtual del cliente debe *verificar la cantidad de saldo*, para comprobar existe saldo suficiente para realizar el pago. La Cartera Virtual trasmitirá el resultado de la comprobación al sistema *ÓrdenesCoin* y el sistema autorizará la trasferencia y se procesará el pago o bien mostrará un mensaje de error.

En ocasiones es posible que el cliente identificado *cancele* la compra solicite la devolución. En este paso el sistema requiere la *devolución* del importe. Para realizar esto el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario. En este caso se *transfiere el importe* de la compra devuelta al Banco Cliente. El Banco Cliente comunica a *ÓrdenesCoin* que la solicitud fue correcta, *procesando la devolución*, o bien comunica el tipo de error ocurrido.

En el caso de que se hubiera realizado el pago mediante Paypal, el cliente identificado puede solicitar *cancelar con Paypal*. El sistema tiene que requerir la devolución. Para ello *ÓrdenesCoin* efectúa un ingreso en la cuenta Paypal del cliente. En este caso Paypal confirma que el ingreso fue realizado correctamente a *ÓrdenesCoin* o informará del tipo de problema ocurrido en caso de que no sea exitosa.

Por último si el pago se realizó con Bitcoins, el cliente solicita la devolución y ÓrdenesCoin se encarga de requerir la devolución. En este caso se debe de *realizar una transferencia* a la Cartera Virtual del cliente. El sistema *procesará la devolución,* en caso de que todo el proceso haya sido correcto, o mostrará un mensaje de error en caso de fallo.

C.2 Anexo II. Configuración del producto

Este documento describe las distintas características seleccionadas en el producto ÓrdenesCoin.

- *Control de acceso*. Este sistema contiene los requisitos del sistema de gestión de acceso al sistema. Este sistema incluye la gestión de los usuarios del sistema, el acceso al sistema al sistema mediante nombre y palabra de paso (password). También se permite modificar un perfil existente.
- *Autorización local*. Sistema básico de control de acceso. Permite únicamente validación de usuarios mediante usuario y password almacenados en una base de datos local.
- *Auditoría*. Este sistema se encarga de almacenar un diario con información del sistema como por ejemplo eventos y excepciones lanzadas por la aplicación. El controlador de los logs es el encargado de gestionarlos. Puede enviar los logs por correo, descargarlos del sistema y revisarlos. También se almacena información de eventos especiales como por ejemplo compras por un valor superior a 10,000€. Esta información puede contener datos sensibles que deben de almacenarse con un nivel alto de seguridad.
- *Acceso local*. Se almacenan los registros del sistema en una base de datos local.
- *Conformidad estándares*. Este sistema se encarga de examinar las órdenes de pedido para cumplir la conformidad con las restricciones gubernamentales para cada tipo de producto. Regularmente se efectuarán consultas en la base de datos para determinar que se cumpla la conformidad con los estándares.
- *Conformidad estándar*. En este caso el sistema asegura una compatibilidad con la regulación del gobierno del país local. Esta solución es suficiente para ventas nacionales.
- *Conformidad adicional*. En este caso existen exportaciones a otros países de modo que la regulación es más restrictiva para algunos productos.
- *Mensajería*. La aplicación se debe comunicar con los proveedores para trasmitirles información como las órdenes de pedido.

- *Mensajería local*. La trasmisión con los proveedores se efectúa a través del propio sistema.
- *Mensajería externa*. Adicionalmente se utilizan servicios externos de otras empresas a través de la tecnología Cloud.
- *Encargo*. Este sistema se encarga de la gestión de las órdenes de encargo.
- *Crear en cargo*. Se encarga de la creación de las órdenes de encargo en el sistema. El rol Web almacena los pedidos en una base de datos. Por otra parte el empleado de pedidos será el encargado de visualizar una orden de encargo y difundir una orden al proveedor apropiado.
- *Procesar encargo*. Una vez se ha realizado un pedido, el cliente identificado se podrá ver el estado del pedido. En esta fase el proveedor será responsable de enviar un acuse de recibo una vez se le haya asignado un pedido. En ese momento el empleado de pedidos actualiza el estado del pedido.
- *Procesamiento local (encargos)*. Se envían las órdenes de encargo a los proveedores mediante la aplicación local.
- *Procesamiento externo (encargos)*. Se trasmitan las órdenes de encargo mediante un servicio situado en un proveedor Cloud. En este caso el proveedor externo envía acuse de recibo mediante una aplicación Cloud en lugar de mandar la información con la aplicación Órdenes.
- *Pago*. Gestiona el pago del producto por parte de los clientes del pedido. Las transacciones se ingresan en el Banco de la Aplicación Órdenes.
- *Pago bancario*. El cliente ingresa el dinero del pedido a través de su banco. El Banco del Cliente efectúa el ingreso una vez se autoriza el pago en el Banco de la Aplicación Órdenes.
- *Pago con Paypal*. El cliente puede realizar el pago mediante la plataforma Paypal. En este caso el Banco de la Aplicación Órdenes pide permiso a Paypal para cargar el importe, Paypal comprueba que se puede realizar la carga (el cliente tiene saldo suficiente).

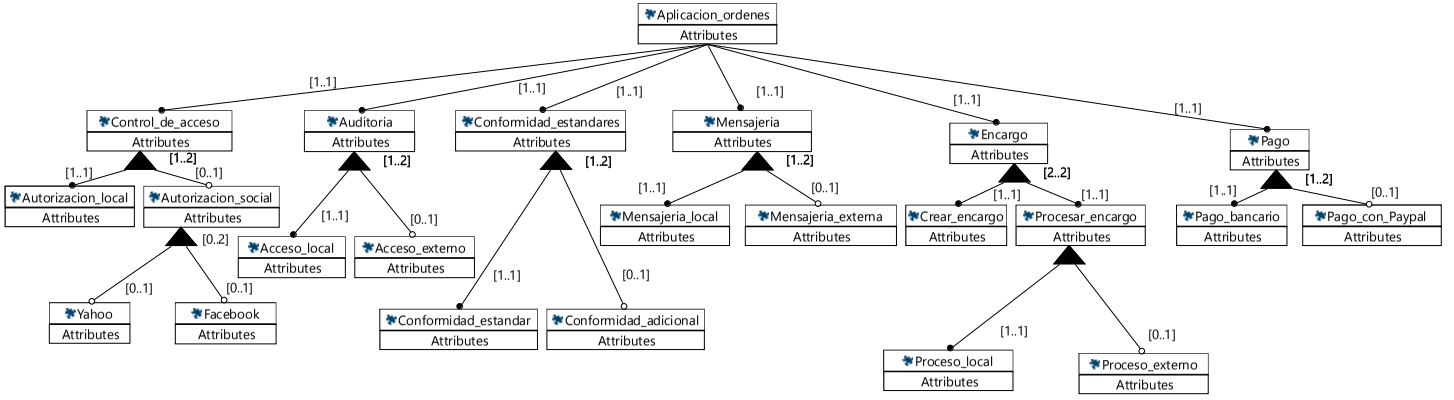


Figura C.3 Modelo de variabilidad de la línea de productos

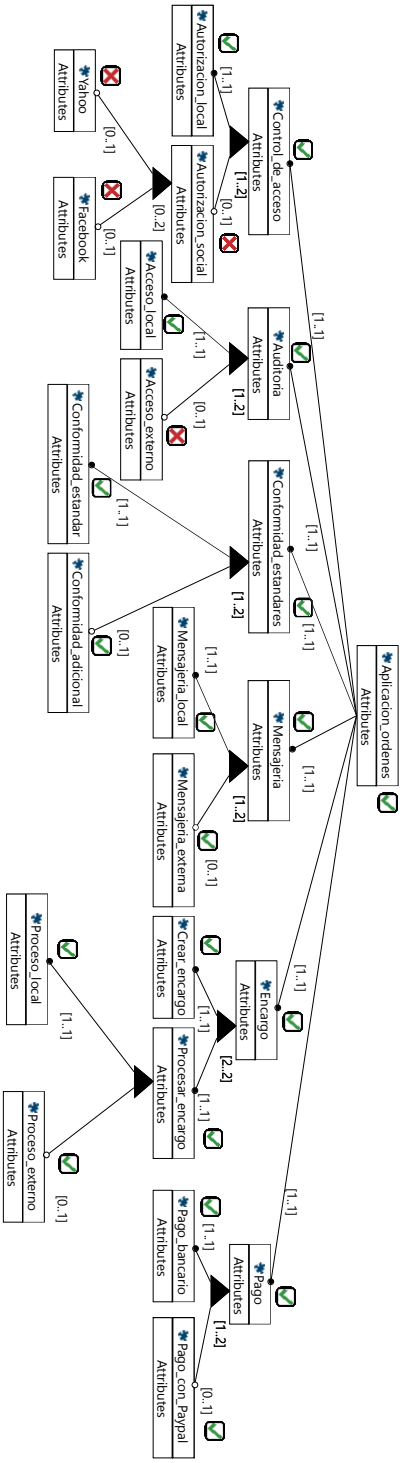


Figura C.4 Configuración del producto Órdenes Coin

C.3 Anexo III. Diagramas de Casos de Uso

En este anexo se muestra el diagrama de casos de uso asociado al Pago. La Figura 1 muestra en la parte superior las características del diagrama de variabilidad del sistema de pago y en la parte inferior el diagrama de casos de uso asociado.

Los cuadros de colores indican la trazabilidad entre las características y los elementos del diagrama de casos de uso.

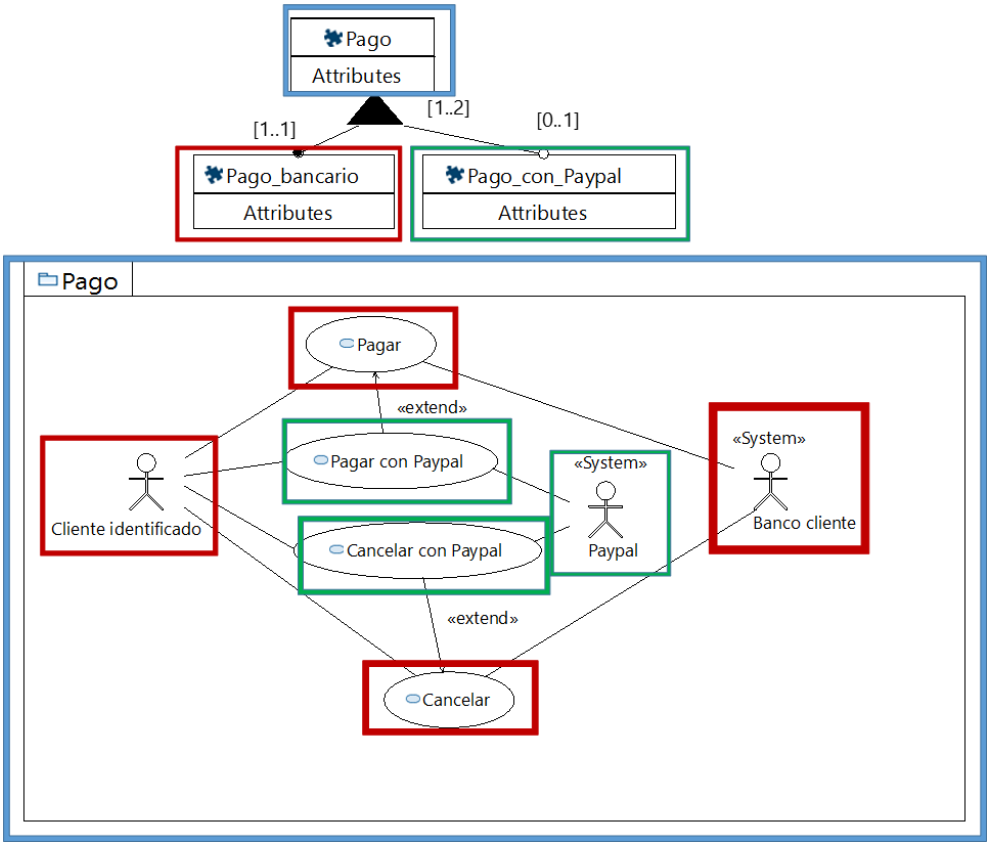


Figura C.5 Trazabilidad entre las características y el diagrama de casos de uso

La Figura 2 muestra el diagrama de casos de uso asociada al Pago.

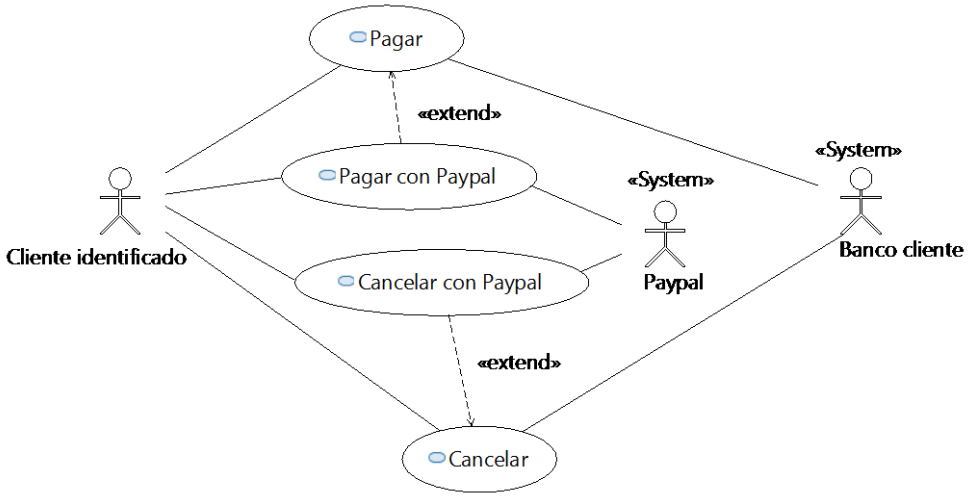


Figura C.6 Diagrama de casos de uso

Descripción de los casos de uso

Pagar

El cliente realiza una compra de un producto. La acción de pagar se realiza por defecto mediante tarjeta de crédito. El cliente identificado en el sistema tiene que introducir su número de tarjeta de crédito, el número de seguridad que se encuentra en el código de seguridad y la fecha de vencimiento de la tarjeta. Con estos datos el cliente identificado requiere autorización al sistema. El sistema comprobará la disponibilidad de saldo en la cuenta asociada a la tarjeta de crédito del cliente solicitando al sistema gestión del Banco Cliente que se haga una comprobación de fondos. Si los fondos son suficientes entonces el sistema autoriza la trasferencia y se realiza el pago, en caso contrario se muestra un mensaje de error.

Pagar con Paypal

Descripción En ocasiones es posible que el cliente identificado cancele la compra y solicite la devolución. En este paso el sistema requiere la devolución del importe. Para realizar esto el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario. En este caso se transfiere el importe de la compra devuelta al Banco Cliente. El Banco Cliente comunica a ÓrdenesCoin que la solicitud fue correcta, procesando la devolución, o bien comunica el tipo de error ocurrido.

Cancelar

Descripción En ocasiones es posible que el cliente identificado cancele la compra y solicite la devolución. En este paso el sistema requiere la devolución del importe. Para realizar esto el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario. En este caso se transfiere el importe de la compra devuelta al Banco Cliente. El Banco Cliente comunica a ÓrdenesCoin que la solicitud fue correcta, procesando la devolución, o bien comunica el tipo de error ocurrido.

Cancelar con Paypal

Descripción En el caso de que se hubiera realizado el pago mediante Paypal el cliente identificado solicita cancela la compra y solicita la devolución. El sistema tiene que requerir la devolución en este caso. Para ello ÓrdenesCoin efectúa un ingreso en la cuenta Paypal del cliente. En este caso Paypal confirma que el ingreso fue realizado correctamente a ÓrdenesCoin o informará del tipo de problema ocurrido en caso de que no sea exitosa.

C.4 Anexo IV. Diagramas de actividad del dominio

Diagrama de actividad del pago

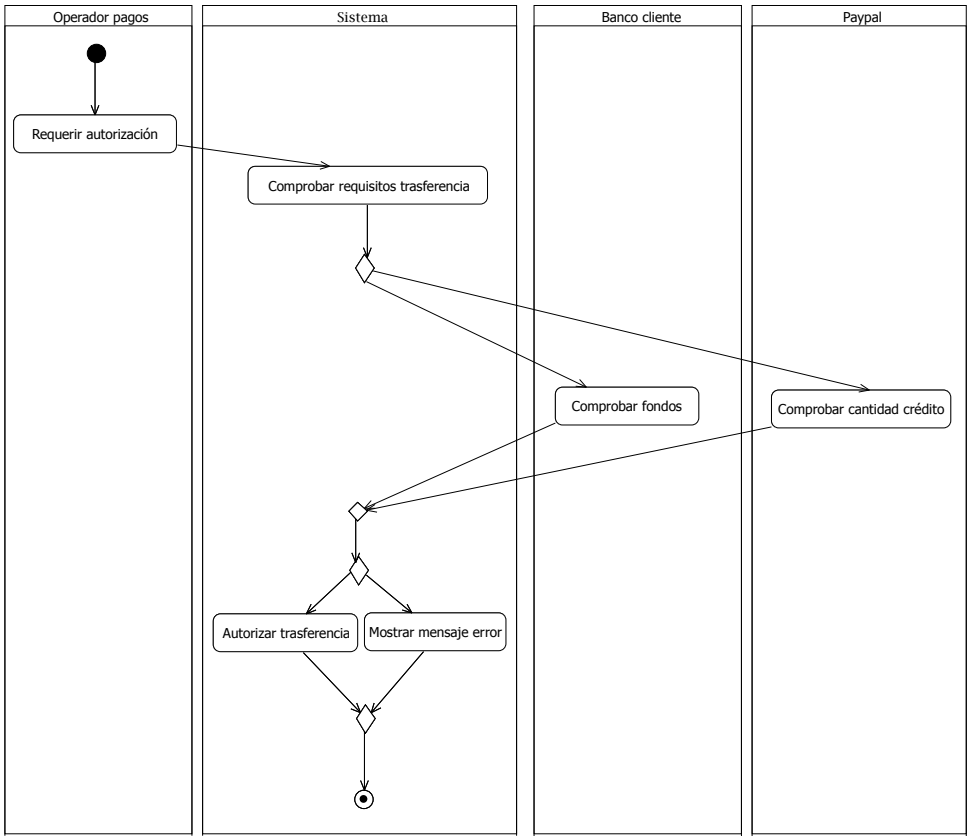
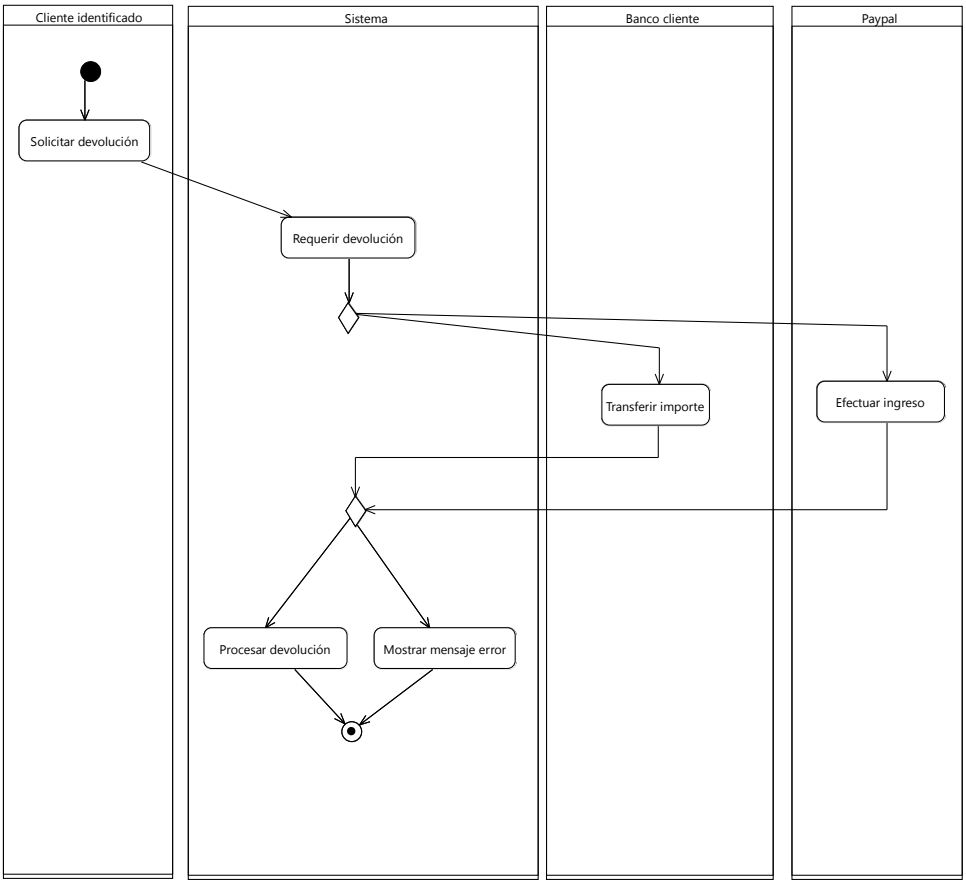
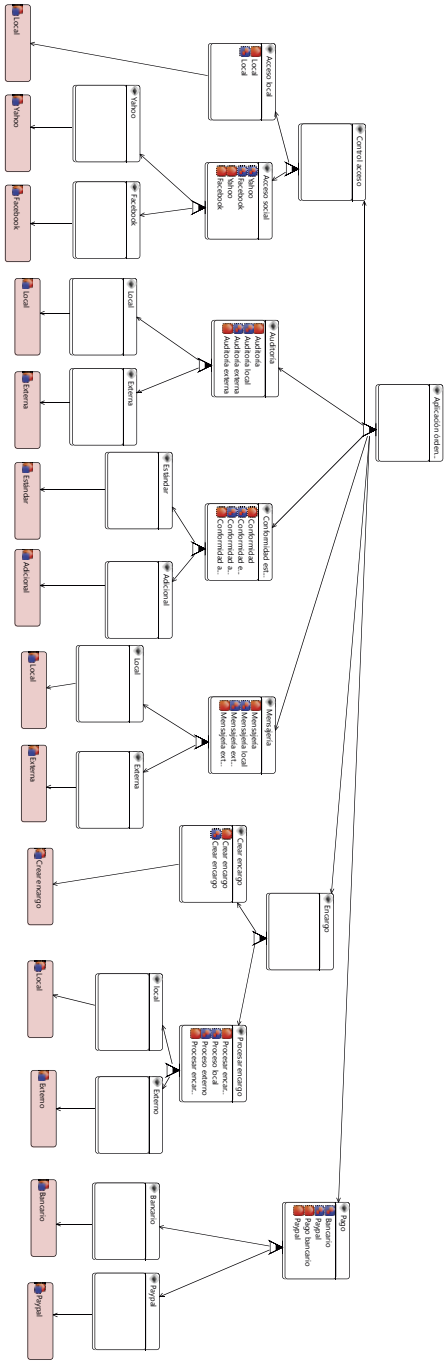


Diagrama de actividad de la devolución



C.5 Anexo V. Diagrama de variabilidad de requisitos



C.6 Anexo VI. Tabla de necesidades del producto ÓrdenesCoin

Necesidad	Descripción
N1 Pagar	Con el pago con tarjeta de crédito el cliente identificado en el sistema tiene que introducir su número de tarjeta de crédito, el número de seguridad que se encuentra en el código de seguridad y la fecha de vencimiento de la tarjeta. Con estos datos el cliente identificado requiere autorización al sistema. El sistema comprobará la disponibilidad de saldo en la cuenta asociada a la tarjeta de crédito del cliente solicitando al sistema gestión del Banco Cliente que se haga una comprobación de fondos. Si los fondos son suficientes entonces el sistema autoriza la trasferencia y se realiza el pago, en caso contrario se muestra un mensaje de error.
N2 Cancelar	En ocasiones es posible que el cliente identificado cancele la compra y solicite la devolución. En este paso el sistema requiere la devolución del importe. Para realizar esto el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario. En este caso se transfiere el importe de la compra devuelta al Banco Cliente. El Banco Cliente comunica a ÓrdenesCoin que la solicitud fue correcta, procesando la devolución, o bien comunica el tipo de error ocurrido.
N3 Comprobación fondos bancarios	El sistema comprobará la disponibilidad de saldo en la cuenta asociada a la tarjeta de crédito del cliente solicitando al sistema gestión del Banco Cliente que se haga una comprobación de fondos.
N4 Realizar transferencia bancaria	Para realizar una devolución el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario. En este caso se transfiere el importe de la compra devuelta al Banco Cliente.
N5 Pagar con Paypal	En ocasiones es posible que el cliente identificado cancele la compra y solicite la devolución. En este paso el sistema requiere la devolución del importe. Para realizar esto el sistema transfiere el importe del pedido cancelado al Banco Cliente en caso de haber optado por el pago bancario. En este caso se transfiere el importe de la compra devuelta al Banco Cliente. El Banco Cliente comunica a ÓrdenesCoin que la solicitud fue correcta, procesando la devolución, o bien comunica el tipo de error ocurrido.
N6 Cancelar con Paypal	En el caso de que se hubiera realizado el pago mediante Paypal el cliente identificado solicita cancela la compra y solicita la devolución. El sistema tiene que requerir la devolución en este caso. Para ello ÓrdenesCoin efectúa un ingreso en la cuenta Paypal del cliente. En este caso Paypal confirma que el ingreso fue realizado correctamente a ÓrdenesCoin o informará del tipo de problema ocurrido en caso de que no sea exitosa.
N7 Comprobación fondos en Paypal	Paypal debe de comprobar la cantidad de crédito de un cliente para verificar que se puede realizar la transacción.

N8 Realizar trasferencia a cuenta Paypal	Paypal confirma si se ha realizado correctamente un ingreso en la cuenta de un cliente
N9 Pago con Bitcoins	El pago mediante BitCoins funciona de forma similar a los otros sistemas. Sin embargo, a diferencia del pago mediante tarjeta bancaria, con el Bitcoin no existe un banco y en su lugar existe un Cartera Virtual que almacena los Bitcoins. En esta modalidad de pago el cliente requiere la autorización y el sistema se encarga de comprobar la disponibilidad de saldo. En este caso la Cartera Virtual del cliente debe verificar la cantidad de saldo, para comprobar existe saldo suficiente para realizar el pago. La Cartera Virtual trasmitirá el resultado de la comprobación al sistema ÓrdenesCoin y el sistema autorizará la trasferencia y se procesará el pago o bien mostrará un mensaje de error.
N10 Cancelar con Bitcoins	Si el pago se realizó con Bitcoins, el cliente puede requerir cancelar su compra con Bitcoins y solicitar la devolución. El sistema (ÓrdenesCoin) se encarga de requerir la devolución. En este caso se debe de realizar una transferencia a la Cartera Virtual del cliente. El sistema procesará la devolución, en caso de que todo el proceso haya sido correcto, o mostrará un mensaje de error en caso de fallo.
N11 Comprobar disponibilidad saldo en cartera Virtual	La Cartera Virtual de un cliente comprueba si se tiene saldo suficiente para que el cliente efectúe un pago.
N12 Realizar trasferencia a la cartera Virtual de un cliente	La Cartera Virtual recibe una transferencia en por parte del sistema para realizar una devolución.

C.7 Cuestionario demográfico para evaluar la experiencia de los participantes en IR para LPS

A continuación se muestran las preguntas del cuestionario demográfico junto con las respuestas posibles a la preguntas, exceptuando la pregunta 2 que tiene una respuesta abierta. Este cuestionario lo rellenaron los participantes antes de realizar las tareas experimentales.

Nº pregunta	Pregunta	Respuestas posibles
1	¿Cuántos meses de experiencia tiene usted en desarrollo líneas de producto software?	He estado involucrado en equipos de desarrollo de software que aplican el enfoque de líneas de producto software. Soy un investigador que trabaja en temas relacionados con desarrollo de software. Conozco lo que son las líneas de producto, pero nunca he participado en un proyecto de desarrollo de software utilizando el desarrollo de líneas de producto. Nunca he oído hablar de las líneas de producto software.
2	¿Cuántos años tiene de experiencia en el desarrollo de LPS?	(Respuesta abierta)
3	¿Has aplicado el enfoque de líneas de producto software en la construcción de software?	Sí, pero sólo en el ámbito académico. Sí, pero sólo a nivel industrial. Sí, tanto en el ámbito académico como en el industrial. No
4	Una línea de productos software es...	Un conjunto de sistemas software que comparten un conjunto común y gestionado de características, desarrollado a partir de un conjunto común de activos software de una manera preestablecida. Un conjunto de productos construido a partir de una plataforma de componentes. Dichos componentes no tienen variabilidad interna. No estoy seguro
5	¿Qué es una característica?	Una característica es todo lo que puede variar en una línea de producto software. Una característica puede ser un artefacto software, un documento, etc.

		Una característica es un aspecto visible para el usuario de un sistema o producto software y que normalmente se organizan en modelos de características. No estoy seguro.
6	En el desarrollo de líneas de producto software...	Existe una única especificación de requisitos para todos los productos software desarrollados en las líneas de producto software. Debe de especificarse una nueva especificación de requisitos a partir de cero para cada producto de las líneas de producto software. Existe una especificación de requisitos común para las líneas de producto software. Además cada producto tiene su propia especificación de requisitos derivada a partir de la especificación de requisitos de las líneas de producto software.
7	En el desarrollo de líneas de producto software una configuración es:	Una combinación de características y / o requisitos no funcionales que definen un producto y que cumple con las restricciones definidas en un modelo de características y / o requisitos no funcionales de la líneas de producto software. El concepto de configuración no es aplicable en el desarrollo de líneas de producto software. Un conjunto de funcionalidades que definen el producto.
8	En este modelo de características mostrado en la parte inferior una configuración VÁLIDA es la formada por el conjunto de características	{Access_Control,Web_Access_Control,Twitter_Access_Control,Facebook_Access_Control} es una configuración válida {Access_Control,Mobile_Access_Control,Twitter_Access_Control,Facebook_Access_Control} es una configuración válida {Access_Control,Mobile_Access_Control,Phone_Access_Control,Twitter_Access_Control} es una configuración válida
9	¿Qué es un requisito delta?	Un requisito que no tiene correspondencia con ningún requisito de la línea de productos software. Es un término que genérico de la ingeniería de requisitos y que no tiene relación con las líneas de producto software. Es un requisito específico de la ingeniería del dominio.

Bibliografía

ABRAHÃO, S., INSFRAN, E., CARSÍ, J.A., GENERO, M. (2011). "*Evaluating requirements modeling methods based on user perceptions: A family of experiments*". Information Sciences, Vol. 181, Num. 16, pp. 3356–3378.

ALFÉREZ, M., MOREIRA, A., AMARAL, V., ARAÚJO, J. (2011). "*Model-Driven Requirements Specification for Software Product Lines*". Model-Driven Domain Analysis and Software Development. pp. 369–386, IGI Global.

ALFÉREZ, M., SANTOS, J.P., MOREIRA, A., GARCIA, A., KULESZA, U., ARAÚJO, J., AMARAL, V. (2009). "*Multi-view Composition Language for Software Product Line Requirements.*". En actas de: Proceedings of the Second international conference on Software Language Engineering (SLE), Denver, USA. pp. 103–122.

ALVES, V., NIU, N., ALVES, C., VALENÇA, G. (2010). "*Requirements engineering for software product lines: A systematic literature review*". Information and Software Technology, Vol. 52, Num. 8, pp. 806–820.

ASADI, M., BAGHERI, E., MOHABBATI, B., GAŠEVIĆ, D. (2012). "*Requirements engineering in feature oriented software product lines*". En actas de: Proceedings of the 16th International Software Product Line Conference (SPLC), New York, New York, USA. p. 36.

ASADI, MOHSEN, BAGHERI, EBRAHIM, GAŠEVIĆ, DRAGAN, HATALA, MAREK, MOHABBATI, B., ASADI, M., BAGHERI, E., GAŠEVIĆ, D., HATALA, M., MOHABBATI, B. (2011). "*Goal-Driven Software Product Line Engineering*". En actas de: Proceedings of the 2011 ACM Symposium on Applied Computing, New York, New York, USA. pp. 691–698.

BABAR, M.A., SHULL, F. (2010). "*Managing Variability in Software Product Lines*". IEEE Software, Vol. 27, Num. 3, pp. 89–94.

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*". Technical Report. National Institute of Standards and Technology. Bureau Drive, Gaithersburg, USA.

BASILI, V.R., ROMBACH, H.D., MEMBER, S., ROMBACH, H.D. (1998). "*The*

TAME Project: Towards Improvement-Oriented Software Environments". IEEE Transactions on Software Engineering, Vol. 14, Num. 6, pp. 758–773.

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.

BAYER, J., MUTHIG, D., WIDEN, T. (2000). "*Customizable domain analysis*". En actas de: First Internation Symposium on Generative and Component-Based Software Engineering, pp. 178–194.

BEYDEDA, S., BOOK, M., GRUHN, V. (2005). "*Model-Driven Software Development*". Development, Vol. 1, Num. 2, pp. 464.

BONIFÁCIO, R., BORBA, P. (2009). "*Modeling scenario variability as crosscutting mechanisms*". En actas de: 8th ACM international conference on Aspect-oriented software development (AOSD), Charlottesville, Virginia, USA. pp. 125–136.

BOSCH, J. (2000). "*Design and use of software architectures: adopting and evolving a product-line approach*". ACM Press/Addison-Wesley Publishing Co.

BRAGANÇA, A., MACHADO, R.J. (2009). "*A model-driven approach for the derivation of architectural requirements of software product lines*". Innovations in Systems and Software Engineering, Vol. 5, Num. 1, pp. 65–78.

BRUN, C., PIERANTONIO, A. (2008). "*Model differences in the eclipse modeling framework*". UPGRADE, The European Journal for the Informatics Professional, Vol. 9, Num. 2, pp. 29–34.

BÜHNE, S., LAUENROTH, K., POHL, K. (2004). "*Why is it not Sufficient to Model Requirements Variability with Feature Models?*". En actas de: Proceedings of the Workshop: Automotive Requirements Engineering (AURE), co-located at RE'04, Nagoya, Japan. pp. 5–12.

CAMPBELL, D.T., STANLEY, J.C. (1982). "*Diseños experimentales y cuasi experimentales en la investigación social*". Editorial Ammorrortu.

CARIFIO, J., PERLA, R.J. (2007). "*Ten Common Misunderstandings, Misconceptions, Persistent Myths and Urban Legends about Likert Scales and Likert Response Formats and their Antidotes*". Journal of Social Sciences, Vol. 3 (3), Num. 3, pp. 106–116.

CHENG, B.H.C., ATLEE, J.M. (2007). "*Research Directions in Requirements Engineering*". En actas de: Future of Software Engineering (FOSE '07), pp. 285–303.

CHUNG, L., NIXON, B.A., YU, E., MYLOPOULOS, J. (2000). "*Non-Functional Requirements in Software Engineering*". Kluwer Academic Publishers.

CLEMENTS, P., NORTHROP, L. (2007). "*A Framework for Software Product Line Practice, Version 5.0*". http://www.sei.cmu.edu/productlines/framework.html.

CONOVER, W.J. (1998). "*Practical nonparametric statistics*". John Wiley & Sons.

COOK, T.D., CAMPBELL, D.T. (1979). "*Quasi-experimentation: Design & analysis issues for field settings*". Houghton Mifflin Company.

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 (GPCE), Tallin, Estonia. pp. 423–437.

CZARNECKI, K., EISENECKER, U. (2000). "*Generative Programming: Methods, Tools, and Applications*". Addison-Wesley.

CZARNECKI, K., KIM, C.H.P. (2005). "*Cardinality-based feature modeling and constraints: A progress report*". En actas de: International Workshop on Software Factories, held at Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), San Diego, California, USA. pp. 16–20.

DAVIS, F.D. (1989). "*Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology*". MIS Quarterly, Vol. 13 (3), Num. 3, pp. 319.

DJEBBI, O., SALINESI, C., DIAZ, D. (2007). "*Deriving product line requirements: The RED-PL guidance approach*". En actas de: Proceedings of the Asia-Pacific Software Engineering Conference (APSEC), Nagoya, Japan. pp. 494–501.

ERIKSSON, M., BÖRSTLER, J., BORG, K. (2005). "*The PLUSS approach - domain modeling with features, use cases and use case realizations*". En actas de: Proceedings on 9th International Conference on Software Product Lines (SPLC), Rennes, France. pp. 33–44.

FAGAN, M.E. (2001). "*Pioneers and Their Contributions to Software Engineering*". En actas de: Proceeding of the first Conference on Software Pioneers, Bonn, Germany.

FAVRE, J.-M. (2004). "*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.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. (1995). "*Design patterns: elements of reusable object-oriented software*". Addison Wesley.

GLINZ, M., HEYMANS, P. eds. (2009). "*A Case Study on Delta Requirements*". Requirements Engineering: Foundation for Software Quality. pp. 45–58., Springer Berlin Heidelberg

GÓMEZ, A. (2012). "*Model Driven Software Product Line Engineering: System Variability View and Process*". PhD thesis, Universitat Politècnica de València.

GÓMEZ, A., RAMOS, I. (2010). "*Cardinality-based feature modeling and model-driven engineering: Fitting them together*". En actas de: Int. Workshop on Variability Modeling of Software intensive Systems (VAMOS), Linz, Austria. pp. 61–68.

GÓMEZ LLANA, A., BORONAT, A., MOLL, J.Á.C.C. (2006). "*Soporte Gráfico para trazabilidad en una herramienta de gestión de modelos.*". Informe técnico. Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València. Valencia.

GONZÁLEZ HUERTA, J. (2014). "*Derivación, Evaluación y Mejora de la Calidad de Arquitecturas Software en el Desarrollo de Líneas de Producto Software Dirigido por Modelos*". PhD thesis, Universitat Politècnica de València.

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.

GRISS, MARTIN L., FAVARO, JOHN, D'ALESSANDRO, M. (1998). "*Integrating feature modeling with the RSEB*". En actas de: Integrating feature modeling with the RSEB, Victoria, British Columbia, Canada.

GRUBER HARGRAVE, B. J., MCAFFER, J., RAPICAULT, P., WATSON, T., O., GRUBER, O., HARGRAVE, B. J., MCAFFER, J., RAPICAULT, P., WATSON, T. (2005). "*The Eclipse 3.0 platform: Adopting OSGi technology*". IBM Systems Journal, Vol. 44 (2), Num. 2, pp. 289–300.

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.

HAUGEN, Ø., ØGÅRD, O. (2014). "*BVR – Better Variability Results*". En actas de: Proceeding of the 8th System Analysis and Modeling: Models and Reusability (SAM), Lecture Notes in Computer Science, Valencia, Spain.

pp. 1–15.

HAUSER, J.R. (1993). "*How Puritan-Bennett used the house of quality*". Spring, Vol. 34, Num. 3, pp. 61–70.

HU, P.J., CHAU, P.Y.K., SHENG, O.R.L., TAM, K.Y. (1999). "*Examining the technology acceptance model using physician acceptance of telemedicine technology*". Journal of Management Information Systems, Vol. 16, Num. 2, pp. 91–112.

IEEE (1990). "*IEEE Standard Glossary of Software Engineering Terminology/IEEE Std 610.12-1990*".

ISA (2011). "*Fama Tool Suite*". http://www.isa.us.es/fama/.

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.

JAVIER GONZÁLEZ-HUERTA (2014). "*Quality Derivation, Evaluation and Improvement of Software Arquitectures on Software Product Lines Development (In Spanish)*".

JOHN, I., EISENBARTH, M. (2009). "*A decade of scoping: a survey*". En actas de: Proceedings of the 13th International Software Product Line Conference, pp. 31–40.

JOUAULT, F., ALLILAIRE, F., BÉZIVIN, J. (2006). "*ATL: a QVT-like transformation language*". Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 719–720.

JURISTO, N., MORENO, A.M. (2010). "*Basics of Software Engineering Experimentation.*". Kluwer Academic Publishers.

KÄKÖLÄ, T., DUEÑAS, J.C. (2007). "*Software Product Lines: Research Issues in Engineering and Management*". Springer Science.

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.

KHACHAN, C. (2012). "*Un Framework para el Análisis Automático de Líneas de Producto Software*". Master thesis, Universitat Politècnica de València,

Valencia.

KRUCHTEN, P. (1999). "*The Rational Unified Process: An Introduction*". Addison Wesley.

KULKARNI, V., BARAT, S., ROYCHOUDHURY, S. (2012). "*Towards Business Application Product Lines*". En actas de: Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems (MODELS), Lecture Notes in Computer Science, Berlin, Heidelberg. pp. 285–301.

LAITENBERGER, O., ROMBACH, D. (2003). "*(Quasi-)experimental studies in industrial settings*". Lecture notes on empirical software engineering. pp. 167–227., World Scientific Publishing Co., Inc.

LIKERT, R. (1932). "*A technique for the measurement of attitudes*". Archives of Psychology, Vol. 22, Num. 140, pp. 1–55.

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.

LOUCOPOULOS, P., KARAKOSTAS, V. (1995). "*System Requirements Engineering*". McGraw-Hill: New York, NY, USA.

MAXWELL, K. (2002). "*Applied statistics for software managers.*". Prentice Hall.

MIKYEONG MOON, YEOM, K., HEUNG SEOK CHAE (2005). "*An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line*". IEEE Transactions on Software Engineering, Vol. 31, Num. 7, pp. 551–569.

MOON, M., YEOM, K., CHAE, H.S. (2006). "*A Metamodel Approach to Architecture Variability in a Product Line*". En actas de: Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components (ICSR), pp. 115–126.

MUSSBACHER, G., ARAÚJO, J., MOREIRA, A., AMYOT, D. (2012). "*AoURN-based Modeling and Analysis of Software Product Lines*". Software Quality Journal, Vol. 20 (3-4), Num. 3-4, pp. 645–687.

MUTHIG, D., JOHN, I., ANASTASOPOULOS, M., FORSTER, T., DÖRR, J., SCHMID, K. (2004). "*GoPhone - A Software Product Line in the Mobile Phone Domain*". Report, Fraunhofer IESE, Kaiserslautern.

NUSEIBEH, B. (2004). "*Crosscutting requirements*". En actas de: Proceedings of the 3rd international conference on Aspect-oriented software development

(AOSD), New York, New York, USA. pp. 3–4.

OBJECT MANAGEMENT GROUP (2012). "*Common Variability Language (CVL) OMG Revised Submission*".

OLIVEIRA, R.P. DE, BLANES, D., GONZALEZ-HUERTA, J., INSFRAN, E., ABRAHÃO, S., COHEN, S., ALMEIDA, E.S. DE (2014). "*Defining and Validating a Feature-Driven Requirements Engineering Approach*". Journal of Computer Science, Vol. 20 (5), Num. 5, pp. 666–691.

OMG (2006)(a). "*Meta Object Facility (MOF) Core Specification Version 2.0*".

OMG (2006)(b). "*Object Constraint Language (OCL) Specification Version 2.0*".

OMG (2008). "*Software & Systems Process Engineering Meta-Model Specification (SPEM) v2.0*".

OMG (2007). "*UML 2.4.1 Superstructure Specification*". http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF.

POHL, K., BÖCKLE, G., VAN DER LINDEN, F. (2005). "*Software Product Line Engineering - Foundations, Principles, and Techniques*". Springer: Berlin.

PRESSMAN, R.S. (2005). "*Ingeniería del software: Un enfoque práctico*". McGraw Hill.

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.

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.

SANTANA NEIVA, D.F. (2009). "*RiPLE-RE: A Requirements Engineering Process for Software Product Lines*". PhD thesis, Universidade Federal de Pernambuco.

SCHAEFER, I. (2010). "*Variability Modelling for Model-Driven Development of Software Product Lines*". En actas de: Proceedings of the 14th international conference on Software product lines (SPLC), Linz, Austria. pp. 85–92.

SCHAEFER, I., BETTINI, L., DAMIANI, F., TANZARELLA, N. (2010). "*Delta-oriented programming of software product lines*". En actas de: Proceedings of the 14th international conference on Software product lines (SPLC), pp. 77–91.

SEPÚLVEDA, S., CRAVERO, A., CACHERO, C. (2015). "*Requirements Modelling Languages for Software Product Lines: a Systematic Literature Review*". Information and Software Technology, Vol. 69, pp. 16–36.

SHAKER, P., ATLEE, J.M., WANG, S. (2012). "*A feature-oriented requirements modeling*

language". En actas de: Proceeding of the 20th IEEE International Requirements Engineering Conference,.

SOMMERVILLE, I. (2011). "*Software Engineering*" 9th edition. Addison Wesley.

STEINBERG, D., BUDINSKY, F., PATERNOSTRO, M., MERKS, E. (2008). "*EMF: Eclipse Modeling Framework*". Addison-Wesley Professional.

VASILEVSKIY, A., HAUGEN, Ø., CHAUVEL, F., JOHANSEN, M.F., SHIMBARA, D. (2015). "*The BVR tool bundle to support product line engineering*". En actas de: Proceedings of the 19th International Conference on Software Product Line (SPLC), New York, New York, USA. pp. 380–384.

VOELTER, M., VISSER, E. (2011). "*Product Line Engineering Using Domain-Specific Languages*". En actas de: 15th International Software Product Line Conference (SPLC), pp. 70–79.

WIEGERS, K., BEATTY, J. (2013). "*Software Requirements*". Microsoft Press.

WOHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M.C., REGNELL, B., WESSLÉN, A. (2000). "*Experimentation in Software Engineering*". Kluwer Academic Publishers.

YU, W., ZHANG, W., ZHAO, H., JIN, Z. (2014). "*TDL: a transformation description language from feature model to use case for automated use case derivation*". En actas de: Proceedings of the 18th International Software Product Line Conference on - SPLC '14, New York, New York, USA. pp. 187–196.

ZAVE, P. (1997). "*Classification of Research Efforts in Requirements Engineering*". ACM Computing Surveys 27, Vol. 29 (4), Num. 4, pp. 315–321.