UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

MARBLE Tool: Herramienta para la Automatización de un proceso de Recuperación de Procesos de Negocio de Sistemas de Información Heredados

María de la Sierra Fernández Ropero

Septiembre, 2011

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

MARBLE Tool: Herramienta para la Automatización de un proceso de Recuperación de Procesos de Negocio de Sistemas de Información Heredados

Autor: María de la Sierra Fernández Ropero

Director: Ricardo Pérez del Castillo.

Tutor Académico: Dr. Ismael Caballero Muñoz-Reja

Septiembre, 2011

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

RESUMEN

En este Proyecto Fin de Carrera se presenta el análisis, diseño y construcción de la herramienta MARBLE Tool. Esta herramienta automatiza MARBLE, un método para la recuperación de procesos de negocio a partir de sistemas de información heredados a fin de contribuir a la modernización del software mediante la preservación y alineamiento de las reglas de negocio que subyacen en dichos sistemas. Para ello, MARBLE establece cuatro niveles de abstracción y tres transformaciones entre modelos de esos niveles. La herramienta no sólo extrae los procesos de negocio asociados a los sistemas de información heredados sino que además representa y visualiza dichos procesos de negocio de forma gráfica. Todos los modelos gestionados por la herramienta se basan en especificaciones recogidas en estándares internacionales. El uso de estos estándares junto con el hecho de que MARBLE Tool ha sido desarrollada como un plug-in de Eclipse facilita su aplicación en la industria de la ingeniería del software así como su interoperabilidad con otras herramientas relacionadas.

La herramienta MARBLE Tool está destinada a aquellas empresas que desean recuperar sus modelos de procesos de negocio que están siendo ejecutados por las versiones actuales de sus sistemas de información. Disponer de estos procesos de negocio permite a las empresas poder aplicar proyectos de modernización del software a fin de mejorar ciertas propiedades del software como la calidad, el diseño, la mantenibilidad, el coste, etc. Al mismo tiempo, los nuevos sistemas de información quedan alineados con los procesos de negocio actuales.

ABSTRACT

This Grade Thesis presents the analysis, design and construction of the tool "MARBLE Tool". This tool automates the method so-called MARBLE for recovering business processes from legacy information systems to contribute to the modernization of software through the preservation and alignment of business rules that underlie these systems. To archive this goal, MARBLE establishes four levels of abstraction and three model transformations between these levels. Besides the recovering of the business processes related to legacy information systems, this tool graphically displays such business processes. All models managed by the tool are based on international standard specifications. The usage of these standards along with the fact that MARBLE Tool has been developed as an Eclipse plug-in provides its adoption by the software engineering industry as well as its interoperability with other related tools.

MARBLE Tool has been created for those companies that want to recover their models of business processes which are executed by current version of their information systems. Such current business processes allows companies to carry out software modernization projects in order to improve some software aspects such as quality, design, maintainability, cost, etc. As a result, the new information systems can be aligned with the actual business processes.

DEDICATORIA

A Sergio, por estar ahí en todo momento dándome todo su apoyo

A mis padres, por hacerme ser la persona que soy ahora

A mi hermano, por su apoyo y preocupación todos estos años

A mi familia, por confiar en mí y animarme cada día a ser mejor

A mis amigos, por todos esos buenos momentos que hemos vivido y que han hecho esta etapa de mi vida más agradable

AGRADECIMIENTOS

A Sergio, por apoyarme, confiar en mí en todo momento y animarme en los momentos difíciles

A mi familia, por todo el ánimo y preocupación que han mostrado por mí

A mis amigos, por su gran apoyo y confianza en mí

A Ricardo, por ayudarme siempre y dedicar parte de su tiempo a resolver todas mis dudas

A Ismael, por su paciencia y dedicación en esta etapa de mi carrera

A todos los profesores de la carrera, porque cada uno puso su granito de arena para que hoy pueda estar aquí

A todas aquellas personas que han pasado por mi vida en estos años y han influido en mí ya que todas, en mayor o menor medida, tienen un hueco en mi memoria

ÍNDICE

Índice de Figuras

Índice de Tablas

Índice de Fragmentos de Código

1. INTRODUCCIÓN

En este primer capítulo se presenta una introducción del Proyecto Fin de Carrera. Además, se justifica y fundamenta la necesidad de implementar el software que se describe, describiendo el posible nicho de mercado para la problemática que ha suscitado la realización de este proyecto. Este capítulo concluye con una descripción de la estructura del resto de la memoria.

1.1. Introducción al tema

Actualmente, las organizaciones manejan gran cantidad de datos e información necesaria para realizar sus actividades de negocio (Loshin, 2010; Redman, 2008). Esta información es almacenada y gestionada como parte de las operaciones típicas de los sistemas de información automatizados (en adelante simplemente sistemas de información). Dichos Sistemas de Información fueron creados para dar soporte a los procesos de negocio de la organización, es decir, a la secuencia de pasos que son necesarios para conseguir los objetivos de negocio de la organización (Jeston, Nelis et al., 2008).

Con el paso del tiempo, los sistemas de información pueden llegar a quedar tecnológicamente obsoletos, ya que fueron creados con tecnologías y/o métodos antiguos cuya vigencia y actualidad puede no tener continuidad para el modelo de negocio organizacional. Por otra parte, la calidad de estos sistemas puede llegar a degradarse por debajo de límites aceptables (Pérez-Castillo, García-Rodríguez de Guzmán et al., 2011a; Polo, Piattini et al., 2003). En cualquiera de estos dos casos, se habla de sistemas de información heredados y se hace necesaria su modernización (i.e., crear nuevas versiones del sistema de forma evolutiva mediante la preservación de su comportamiento intrínseco).

La solución tradicional para llevar a cabo la modernización de un sistema de información heredado es la reingeniería software, la cual propone un proceso a través de tres etapas: ingeniería inversa, restructuración e ingeniería directa (Chikofsky y Cross, 1990). Sin embargo, la reingeniería tiene deficiencias en cuanto a su estandarización y formalización, por lo que en muchos casos suelen fracasar los proyectos abordados (Sneed, 2005). Para solucionar los problemas de estandarización y formalización surge

el concepto de Modernización del Software, que añade al concepto tradicional de reingeniería la aplicación de los principios del desarrollo dirigido por modelos (MDA, *Model Driven Architecture*) (OMG, 2003b). Con el fin de definir y estandarizar la modernización software, OMG lanzó la iniciativa ADM (*Architecture Driven Modernization*) (OMG, 2007) que proporciona un conjunto de especificaciones necesarias para llevar a cabo procesos de modernización.

Para realizar con garantías la modernización de un sistema de información heredado, y siempre teniendo en cuenta que a través de la modernización deben preservarse las reglas de negocio soportadas por el sistema a modernizar, es necesario conocer de la forma más precisa posible la configuración actual de los procesos de negocio que la organización ejecuta apoyándose en el sistema heredado (Pérez-Castillo, García-Rodríguez de Guzmán et al., 2011b). Conocer los procesos de negocio que subyacen en un sistema de información supone un gran desafío, ya que los modelos de procesos de negocio disponibles explícitamente en la organización (por ejemplo, en sus manuales de calidad) podrían estar desfasados con respecto a los realmente ejecutados por las versiones actuales de sus sistemas de información heredados. Esto se debe a que, a lo largo de la vida de la organización, los sistemas de información han ido siendo progresivamente adaptados y modificados a fin de satisfacer nuevas necesidades de negocio, y dichos cambios no se han reflejado de forma directa en los modelos de procesos que inicialmente se modelaron. Esto hace que se conviertan en sistemas de información heredados y embeban mucha lógica de negocio (Paradauskas y Laurikaitis, 2006). Por esta razón, un sistema de información no puede ser descartado completamente ya que la información de negocio embebida puede que no se encuentre presente en ningún otro activo de la organización, tan solo en el propio código adaptado/evolucionado del sistema de información (Heuvel, 2006).

Para ser capaz de recuperar los procesos de negocio tomando como entrada el código fuente de un sistema heredado surge MARBLE (Pérez-Castillo, García-Rodríguez de Guzmán et al., 2011b), una técnica de ingeniería inversa para el descubrimiento de procesos de negocio desde sistemas heredados. MARBLE es un marco ADM que permite la extracción de procesos de negocio desde sistemas de información heredados. Esta parte del proceso de modernización se estructura en torno a cuatro niveles de abstracción y a tres transformaciones entre esos niveles que permiten

reducir progresivamente la diferencia en el nivel de abstracción que existe entre el código fuente y los procesos de negocio.

Este proyecto presenta MARBLE Tool, una herramienta que soporta MARBLE, la cual ha sido desarrollada en forma de un conjunto de plug-ins para Eclipse™ (un entorno de desarrollo software integrado (IDE)). De esta forma se facilita su implantación en la industria y se asegura su futura y fácil extensión e integración con otros plug-ins.

El objetivo del Proyecto Fin de Carrera, por tanto, consiste en el análisis, diseño y construcción de MARBLE Tool, donde los procesos de negocio serán descubiertos y representados en una notación entendible por todos los usuarios.

1.2. Contextualización

El proyecto fin de carrera está alineado con la tesis doctoral "MARBLE: Un Marco de Modernización del Software para la Extracción de Procesos de Negocio desde Sistemas Heredados", ya que propone una herramienta de soporte para las técnicas que se han propuesto en dicha tesis.

Así mismo, el proyecto fin de carrera se encuadra dentro de los siguientes proyectos de investigación: MOTERO: MOdernización de sisTEmas heredados hacia líneas de pROducto (01/04/2011 - 31/3/2013) financiado por la Junta de Comunidades de Castilla-La Mancha y MAESTRO: Modernización de Sistemas Heredados hacia Procesos de Negocio (01/10/2010 - 30/09/2011) financiado por la Universidad de Castilla-La Mancha.

1.3. Estructura del documento

A continuación se describe la estructura de contenidos del presente documento, que consta de siete capítulos y cuatro anexos:

- **Capítulo 2. Objetivos.** Este capítulo detalla los principales objetivos del presente Proyecto Fin de Carrera.
- **Capítulo 3. Estado del Arte.** Este capítulo resume la bibliografía relevante y documentación consultada sobre los temas que conciernen al

presente proyecto. Entre los temas a tratar se encuentran los sistemas de información heredados, la modernización del software, los procesos de negocio, etc.

- **Capítulo 4. Método de Trabajo.** Este capítulo muestra el método de trabajo utilizado para la consecución de los objetivos descritos en el capítulo 2. En concreto, el método de trabajo es el Proceso Unificado de Desarrollo (PUD).
- **Capítulo 5. Resultados.** Este capítulo expone los resultados obtenidos en este Proyecto Fin de Carrera tras la aplicación del PUD, según lo descrito en el capítulo 4.
- **Capítulo 6. Conclusiones y Propuestas.** Este capítulo discute las conclusiones del presente trabajo y se establecen diversas líneas futuras de trabajo.
- **Capítulo 7. Referencias.** Este último capítulo proporciona una lista con las referencias bibliográficas consultadas para la realización de este proyecto.
- **ANEXO I. Notación BPMN.** Este anexo presenta la notación BPMN con sus elementos más importantes y destacados.
- **ANEXO II. Manual de usuario**. Este anexo presenta el manual de usuario de la herramienta desarrollada a lo largo del proceso de desarrollo.
- **ANEXO III. Ejemplo de Aplicación en un Entorno Empresarial**. Este anexo presenta en detalle el ejemplo de aplicación realizado en el capítulo 5, en el que se muestra el uso de la herramienta en entornos reales.
- **ANEXO IV. Script de QVT.** QVT es uno de los estándares utilizados en este proyecto fin de carrera, el cual se encarga de realizar la transformación entre modelos de distintos niveles. En este anexo se muestra el script confeccionado para la transformación QVT implementada en MARBLE Tool.

2. OBJETIVOS DEL PROYECTO

Este capítulo describe los objetivos que se marcan para la consecución del Proyecto Fin de Carrera, así como los medios técnicos que se emplean para su óptima consecución.

2.1. Objetivos

El objetivo principal del Proyecto Fin de Carrera es el desarrollo de una herramienta siguiendo el esquema de análisis, diseño e implementación y utilizando técnicas de ingeniería y métodos contrastados en la industria del software. Más concretamente, el Proyecto Fin de Carrera marca el siguiente objetivo principal (véase Figura 2.1) y los dos sub-objetivos que se enumeran después.

Objetivo Principal : Realizar el análisis, diseño e implementación de una herramienta de ingeniería inversa que permita el descubrimiento y representación de los procesos de negocio que subyacen (i.e., que son soportados) por un sistema de información existente.

Figura 2.1. Objetivo Principal del Proyecto Fin de Carrera

El análisis, diseño e implementación de la herramienta se realiza siguiendo los principios del desarrollo dirigido por modelos (MDA), tomando como marco de referencia la estructura de niveles propuesta por MARBLE (Pérez-Castillo, García-Rodríguez de Guzmán et al., 2011b).

La herramienta se desarrolla en forma de Plug-in para el IDE Eclipse porque (i) garantiza una fácil implantación en la industria del software y (ii) asegura su futura extensión e integración con otros plug-ins.

Para acometer el objetivo principal del Proyecto Fin de Carrera, se han planteado los siguientes sub-objetivos, mostrados en la Figura 2.2:

- **Sub-objetivo1**: Desarrollar las funcionalidades de una herramienta que permita la obtención automática de procesos de negocio a partir de sistemas de información heredados siguiendo la estructura de niveles de abstracción propuesta por MARBLE y las transformaciones que propone.

- **Sub-objetivo2**: Desarrollar las funcionalidades de la herramienta que permita la representación gráfica de los Procesos de Negocio obtenidos mediante un editor gráfico creado con EMF/GMF, que represente los elementos basándose en la notación estándar de BPMN 2.0.

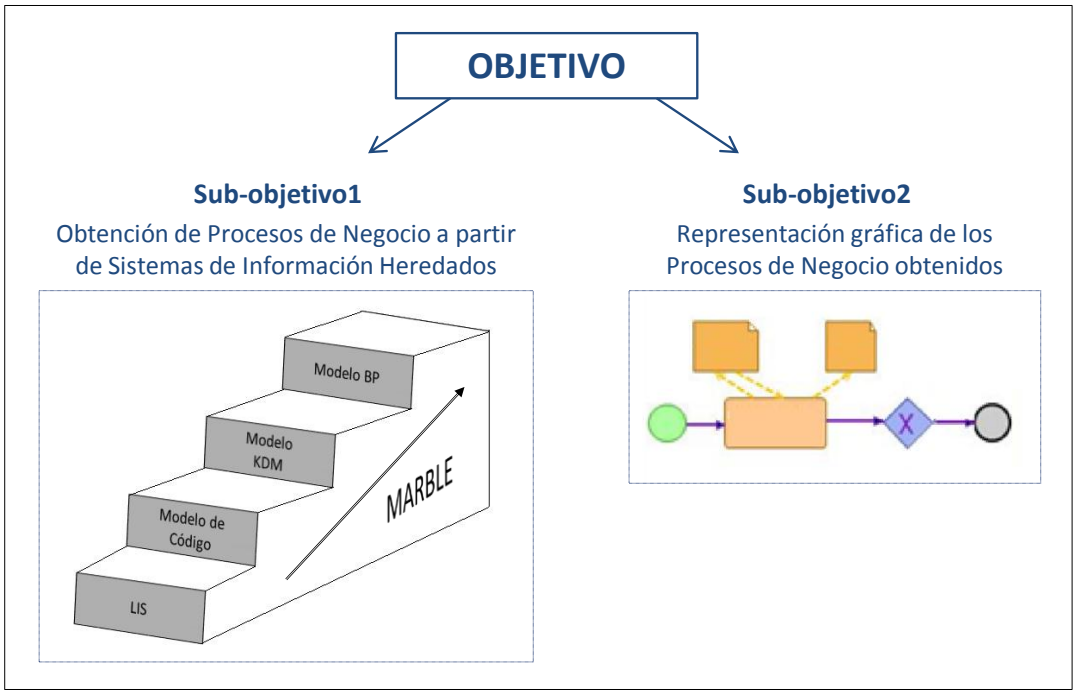


Figura 2.2. Objetivos del Proyecto Fin de Carrera

2.2. Medios Empleados

Las siguientes listas enumeran los recursos técnicos empleados para llevar a cabo el objetivo principal y sub-objetivos marcados.

Recursos tecnológicos (Informáticos)

- El proyecto se ha realizado en un PC con sistema operativo *Microsoft Windows™ 7 Home Premium* con Procesador Intel® Core™ 2 Duo y con 4GB de RAM.
- Etapas de Análisis y Diseño:
  - Para el modelado de diagramas correspondientes al análisis y al diseño de MARBLE Tool se ha utilizado el lenguaje UML 2.0 y la herramienta CASE *Visual Paradigm* (versión 8.1), de la que la UCLM proporciona Licencia Educativa.

    - Para realizar los prototipos iniciales de las interfaces de usuario (GUIs) se ha utilizado la herramienta *Balsamiq Mockups* 2.0.24 (Balsamiq, 2011).
- Etapa de Implementación:
    - Para la implementación se ha escogido el lenguaje de programación Java 1.6 y el entorno de desarrollo integrado *Eclipse™*, distribución *Modeling Tools* (Eclipse, 2010b) en su versión Helios. Concretamente, se han utilizado los paquetes *Graphical Modeling Project* (GMP) y *Eclipse Plug-in Development Environment*.
    - Se han utilizado las librerías *Jdom* 1.1 (Hunter, 2009) para la escritura de documentos XML, la librería *iText* 5.1.1 (1T3XT-BVBA, 2011) para la escritura de documentos PDF, la librería KDM y la librería de *Medini QVT*.
- Etapas de Pruebas:
    - Para la realización de pruebas se ha utilizado los plug-in de Eclipse *JUnit* 4.8 (Gamma y Beck, 2010) y *Eclemma* 1.5.3 (Hoffmann y Janiczak, 2011).
- Como herramientas de propósito general se ha utilizado: *Microsoft Office Word* 2007, *Microsoft Office Project* 2007, *Adobe Reader* 9.0, *Gimp* 2.6, entre otras.

Recursos Materiales

- Libros, artículos de investigación y otra documentación de interés proporcionado por el tutor académico y el director del Proyecto Fin de Carrera.

3. ESTADO DEL ARTE

En este capítulo se presentan los fundamentos y bases teóricas del presente Proyecto Fin de Carrera. En primer lugar, se define el concepto de sistema de información heredado, así como los problemas que representan para las organizaciones (cf. Sección 3.1). En segundo lugar, se presenta el concepto de reingeniería tradicional como una posible solución a los problemas que suponen los sistemas de información heredados así como sus limitaciones actuales (cf. Sección 3.2). En tercer lugar, se define el concepto de modernización del software, como una evolución de la reingeniería tradicional, así como los estándares internacionales que están involucrados en su consecución (cf. Sección 3.3). En cuarto lugar, se trata el concepto de proceso de negocio y su notación como artefacto que puede ayudar a la modernización del software (cf. Sección 3.4). En quinto lugar, se define el método MARBLE en el que se basa el presente Proyecto Fin de Carrera (cf. Sección 3.5). Por último, se describe el entorno de desarrollo que ha sido utilizado para el desarrollo del proyecto (cf. Sección 3.6).

3.1. Sistemas de Información Heredados

Las organizaciones necesitan disponer de sistemas de información que procesen los datos necesarios para realizar las tareas relativas a sus procesos de negocio. Estos sistemas de información pueden llegar a convertirse en Sistemas de Información Heredados (LIS - *Legacy Information System*), es decir, sistemas de información que sobreviven a las diferentes modificaciones a lo largo del tiempo con el fin de satisfacer nuevas necesidades de negocio y adaptarse al cambio continuo del entorno de la organización (Paradauskas y Laurikaitis, 2006).

Debido a que a los sistemas de información heredados se les ha ido añadiendo nuevas funcionalidades para satisfacer nuevos requisitos de la organización, éstos van embebiendo gran cantidad de lógica y reglas de negocio que conforman el flujo de información principal de la organización. Esta información de negocio se va embebiendo durante el mantenimiento incontrolado que sufren los sistemas de información a lo largo del tiempo como consecuencia de la necesidad de ser adaptados a la realidad de los cambios de los procesos de negocio. Por tanto, la información de

negocio podría no estar presente en ningún otro activo de la organización que no sea el código fuente heredado (Sommerville, 2006).

Estos sistemas de información heredados pueden requerir nuevas versiones mejoradas del sistema debido a que éste puede haber quedado tecnológicamente obsoleto, ya que fue creado con tecnologías y/o métodos antiguos en la actualidad, o bien porque los niveles de calidad y mantenibilidad de dichos sistemas han disminuido por debajo de unos límites aceptables (Pérez-Castillo, García-Rodríguez de Guzmán et al., 2011b; Polo, Piattini et al., 2003). En estos casos, el desarrollo de nuevos sistemas de información equivalentes desde cero es desaconsejable, ya que gran parte de la información de negocio embebida podría perderse, pues no está reflejada en ningún otro lugar. Para evitar este problema, es más apropiado llevar a cabo un mantenimiento evolutivo del sistema de información heredado con el fin de preservar la información de negocio embebida. La preservación de la información de negocio embebida requiere la tarea de descubrir y representar previamente dicha información. Esto permite obtener versiones nuevas y mejoradas de los sistemas de información que estén alineados con la operativa real que la organización lleva a cabo (Mens y Demeyer, 2008). Una de las soluciones más contrastada y empleada durante los últimos años para conseguir el mantenimiento evolutivo de sistemas de información heredados es la reingeniería software.

3.2. Reingeniería Software

La reingeniería software se define como el replanteamiento y rediseño de sistemas de información para mejorar propiedades del software tales como la calidad, el diseño, la mantenibilidad, el coste, etc. (Hammer y Champy, 1994). La reingeniería da la oportunidad de incorporar y desarrollar nuevas reglas de negocio que las empresas necesitan comenzar a llevar cabo dentro de un proceso de mejora continua.

El proceso de reingeniería consiste en tres etapas: ingeniería inversa, restructuración e ingeniería directa (Chikofsky y Cross, 1990).

- En la etapa de **ingeniería inversa** se analiza el sistema heredado para identificar los componentes del sistema y sus interrelaciones. Este

proceso construye una o más representaciones del sistema heredado a un nivel mayor de abstracción.

- En la etapa de **restructuración** se toma la representación de la etapa anterior y se transforma, al mismo nivel de abstracción, en una representación que mejora una o varias propiedades del sistema. En cualquier caso, esta representación preserva el comportamiento externo del sistema de información heredado.
- En la última etapa, la **ingeniería directa**, se genera la implementación operativa del sistema a un nivel menor de abstracción que integra las nuevas características.

(Kazman, Woods et al., 1998) presentan las etapas de la reingeniería como un modelo en herradura (véase Figura 3.1). La etapa de ingeniería inversa (lado izquierdo de la herradura) aumenta el nivel de abstracción, la etapa de restructuración (parte superior de la herradura) mantiene el mismo nivel de abstracción, y la etapa de ingeniería directa (lado derecho de la herradura) disminuye el nivel de abstracción.

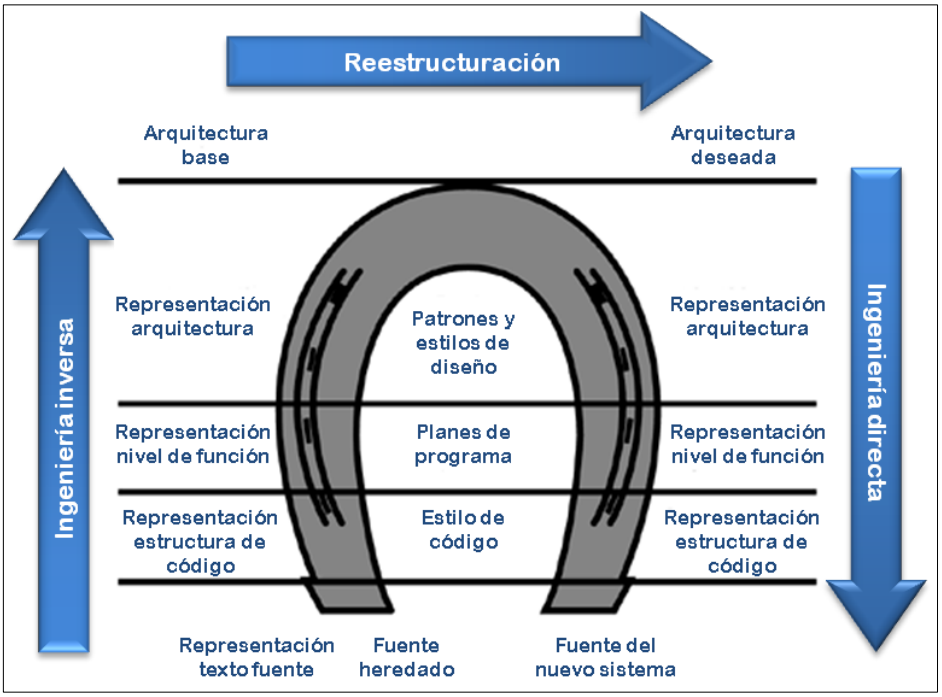


Figura 3.1. Modelo en herradura traducido de (Kazman, Woods et al., 1998)

El proceso de reingeniería software se empezó a aplicar con éxito hace unas décadas en la industria del software. A pesar de ello, algunos estudios revelan que más de la mitad de los proyectos actuales de reingeniería suelen fallar (Sneed, 2005). Los

problemas más importantes que causan este fracaso son la falta de formalización y estandarización del proceso de reingeniería, así como su falta de automatización (Canfora y Di Penta, 2007; Müller, Jahnke et al., 2000).

La escasa estandarización y formalización hace que el proceso de reingeniería no pueda ser integrado y reutilizado en otros proyectos. Esto deriva a su vez en el problema de automatización, ya que al no existir estándares que permitan formalizar la reingeniería es muy difícil automatizar las partes reusables entre otros proyectos, teniendo que para cada nueva iniciativa se tengan que generar nuevas transformaciones específicas *ad hoc*, lo que hace que en ausencia de esta automatización reutilizable se aumenten los costes de mantenimiento.

3.3. Modernización del Software

Para paliar los problemas de la reingeniería software tradicional (debido a la escasa estandarización, formalización y, por tanto, falta de automatización) surge el concepto de Modernización del Software, que aboga por realizar procesos de reingeniería siguiendo los principios del Desarrollo Dirigido por Modelos (MDD – *Model Driven Development*).

En el Desarrollo Dirigido por Modelos (MDD) los artefactos principales del desarrollo son los modelos (no el código fuente de los programas) y la transformación entre modelos. MDD implica una generación semi-automática de la implementación a partir de los modelos.

3.3.1. MDA (Model Driven Architecture)

MDA (*Model Driven Architecture*, Arquitectura Dirigida por Modelos) (OMG, 2003a; OMG, 2003b) es una especificación estándar del OMG (*Object Management Group*) para soportar el MDD. Sus objetivos principales son:

- **Portabilidad**: aumenta la reutilización de las aplicaciones y reduce el coste y la complejidad del desarrollo y administración de las aplicaciones.

- **Interoperabilidad entre plataformas**: utiliza métodos que garantizan que los estándares basados en implementaciones de tecnologías múltiples tengan todos las mismas reglas de negocio
- **Independencia de plataforma**: reduce el tiempo, el coste y complejidad asociada con aplicaciones en diferentes tecnologías.
- **Especificidad del dominio**: a través de modelos específicos del dominio, permite implementaciones rápidas de nuevas aplicaciones.
- **Productividad**: permite a los desarrolladores, diseñadores y administradores de sistemas usar lenguajes y conceptos con los que se sienten cómodos, facilitando la comunicación e integración transparente entre los equipos de trabajo.

MDA, por tanto, aboga por representar todos los artefactos software como modelos y requiere que se diseñen las transformaciones necesarias para generar un modelo a partir de otro. El concepto de modelo ha sido definido por varios autores mediante las siguientes definiciones:

- Un modelo es una descripción o especificación del sistema y de su entorno para un determinado propósito. Un modelo se presenta con frecuencia como una combinación de dibujos y de texto (OMG, 2003a).
- Un modelo de un sistema queda definido como una descripción de (o parte de) un sistema escrita en un lenguaje bien definido. Un lenguaje bien definido es un lenguaje con una forma definida (sintaxis) y significado (semántica) que sea apropiado para ser interpretado automáticamente por un computador (Kleppe, Warmer et al., 2003).
- Un modelo es un conjunto coherente de elementos que cubren ciertos aspectos de diseño, están restringidos a cierto nivel de abstracción, no se necesitan exponer todos los detalles ni necesita ser completo por sí mismo (Mellor, 2003).

MDA define los siguientes tipos de modelos según el nivel de abstracción al que se encuentran representados (véase Figura 3.4):

1. **CIM** (*Computation-Independent Model*). Representa los modelos independientes de la computación que caracterizan el dominio del problema. Este modelo muestra el sistema en el entorno en el que va a

operar y ayuda a representar lo que se espera que haga el sistema de información.

2. **PIM** (*Platform-Independent Model*). Representa los modelos que describen una solución de software que no contiene detalles de la plataforma concreta en que la solución va a ser implementada. Estos modelos surgen como resultado del análisis y diseño. Uno de los lenguajes que permite representar modelos PIM es UML (*Unified Modeling Language).*
3. **PSM** (*Platform-Specific Model*). Son modelos que contienen detalles de la plataforma o tecnología concreta con que se implementará la solución. Combina la especificación de un PIM con los detalles característicos de la plataforma donde se pretende ejecutar.

A la hora de convertir un tipo de modelo en otro, idealmente deben diseñarse y utilizarse transformación automáticas que describen cómo un modelo en un determinado lenguaje y nivel de abstracción puede ser transformado en otro destino (correspondencia entre entidades de ambos modelos). Como consecuencia, todos los modelos representan el mismo sistema pero en diferentes niveles de abstracción.

MDA se complementa con otras especificaciones de OMG que son la base para crear, publicar y administrar modelos en una arquitectura dirigida por modelos, independientemente del tipo de sistema que se va a construir. Estas especificaciones son UML, MOF, XMI y CWM, que se describen brevemente a continuación:

- UML (*Unified Modeling Language*): Notación base para la definición de CWM. A partir de sus modelos visuales se pueden realizar traducciones automáticas a otros lenguajes formales.
- MOF (*Meta Object Facility*): MOF es un lenguaje común y abstracto para la especificación de meta-modelos, que sirve como un modelo común para UML y CWM. MOF es una arquitectura de meta-modelos de cuatro capas como se ve en el ejemplo de la Figura 3.2.
- XMI (*XML Metadata Interchange*): Es un estándar que mapea el MOF al estándar de XML del *World Wide Web Consortium* (W3C). XMI define cómo los tags de XML se usan para representar en XML modelos

serializados que cumplan con el estándar de MOF. Esto facilita el intercambio de modelos entre diversos modelos y herramientas.

- CWM (*Common Warehouse Metamodel*): Es un meta-modelo que especifica interfaces que pueden ser usadas para habilitar el intercambio de metadatos de almacenes de datos e inteligencia de negocio, entre distintas herramientas, plataformas y metadatos en ambientes heterogéneos y distribuidos de almacenes de datos.

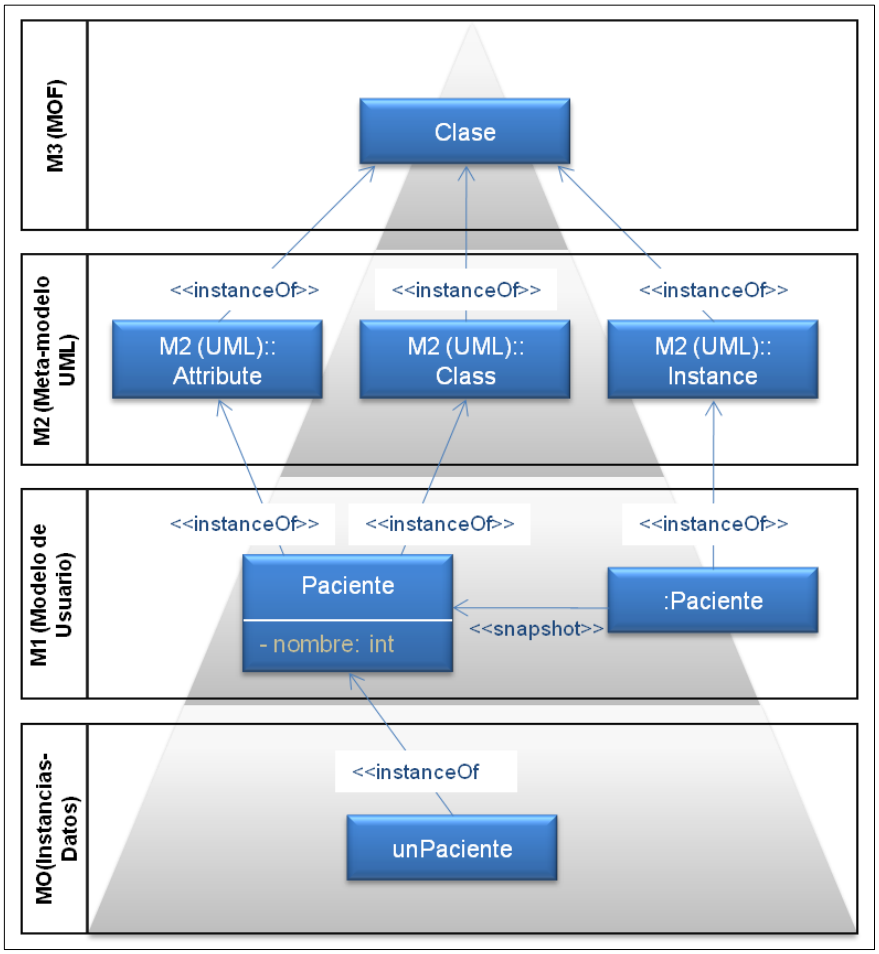


Figura 3.2. Ejemplo de capas de MOF

La Figura 3.3 compara el proceso de desarrollo tradicional y el de MDA y muestra como en los dos enfoques cada etapa del desarrollo produce artefactos que sirven para la siguiente etapa. La principal diferencia radica en la formalización y consistencia con la que se realiza el proceso de transformación del modelo de una fase a otra.

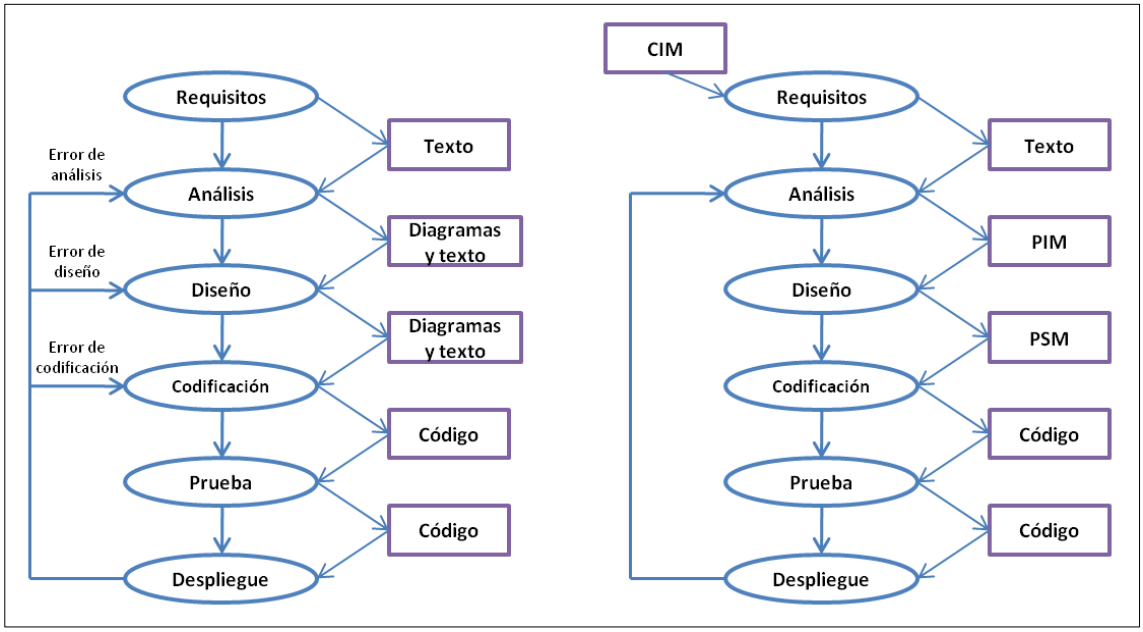


Figura 3.3. Comparativa entre proceso de desarrollo software tradicional y MDA

3.3.2. Iniciativa ADM (Architecture-Driven Modernization)

ADM (*Architecture-Driven Modernization*, Modernización dirigida por la Arquitectura) es una iniciativa propuesta por la OMG cuyo objetivo es la estandarización del proceso de modernización de los sistemas de información heredados centrándose en todos los aspectos de su arquitectura a través de la definición y uso de metamodelos (OMG, 2007).

ADM utiliza transformaciones de modelos para obtener la arquitectura deseada partiendo de su arquitectura actual, todo ello siguiendo el enfoque dirigido por modelos MDA. Por tanto, el modelo de reingeniería en herradura tradicional visto en la Figura 3.1 es mejorado por el modelo de modernización en herradura de la Figura 3.4. La modernización software no reemplaza el proceso de reingeniería tradicional, sino que lo mejora mediante la aplicación de los principios MDA**.**

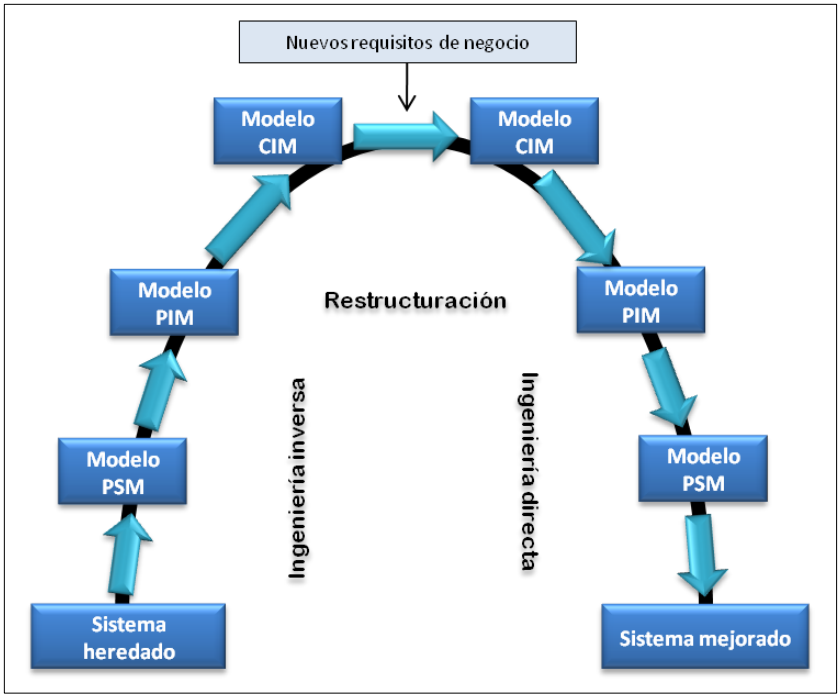


Figura 3.4. Modelo en herradura de ADM

Para conseguir su objetivo ADM define un conjunto de estándares para la modernización. Estos estándares, representados en la Figura 3.5 y definidos en los siguientes sub-apartados, definen los dos principales principios de MDA: (i) modelar todos los artefactos como modelos a diferente nivel de abstracción y (ii) establecer las transformaciones entre estos modelos (OMG, 2003a).

Además de formalizar los artefactos al representarlos como modelos, ADM también permite la formalización de las transformaciones entre modelos mediante el estándar QVT (*Query/Views/Transformations*) propuesto por la OMG (OMG, 2011b). El estándar QVT es descrito en la sección 3.3.4.

3.3.3. Metamodelo KDM (Knowledge Discovery Metamodel)

El metamodelo KDM (*Knowledge Discovery Metamodel*, Metamodelo de Descubrimiento de Conocimiento), reconocido como estándar en ISO/IEC 19506 (ISO/IEC, 2009), especifica un conjunto de conceptos comunes para comprender los sistemas de información heredados para la modernización de software. KDM proporciona infraestructuras para soportar definiciones de conocimiento específicas de dominio, específicas de la aplicación o específicas de la implementación (OMG, 2009).

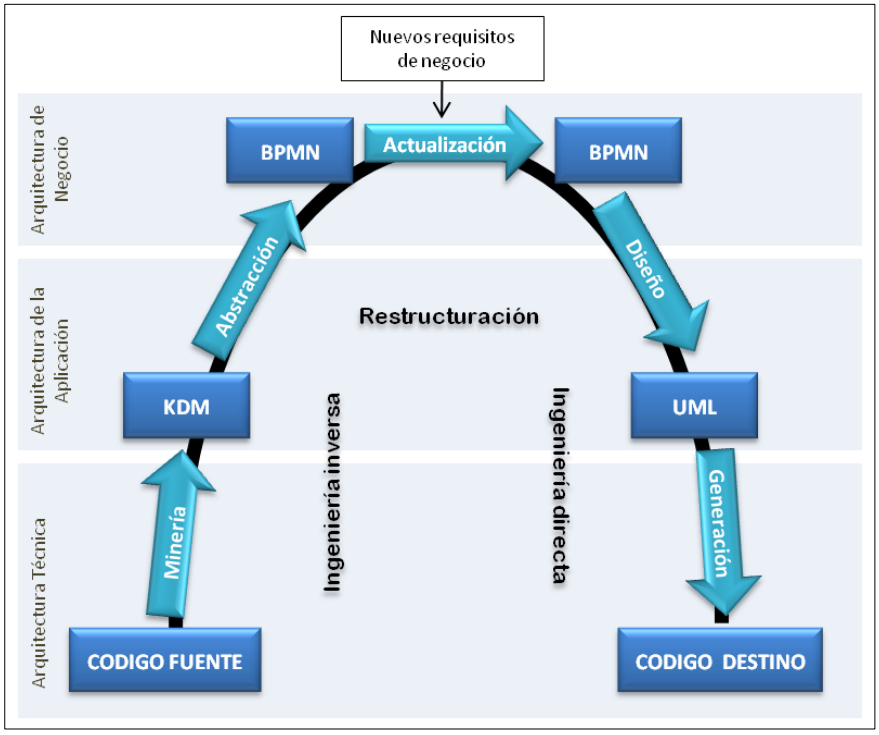


Figura 3.5. Estándares involucrados en las transformaciones ADM. Traducción de (Khusidman, 2008)

El metamodelo KDM se divide en paquetes, agrupados en cuatro capas de abstracción diferentes (véase Figura 3.6), cada uno encargado de modelar una vista o aspecto de un sistema de información heredado. Estos paquetes se describen a continuación:

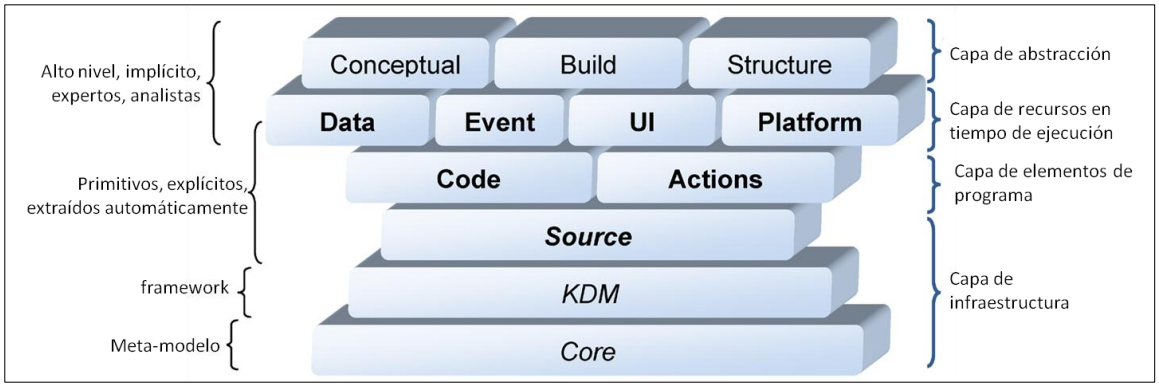


Figura 3.6. Estructura de paquetes KDM (OMG, 2009)

1. **Capa de Infraestructura**: define un pequeño conjunto de conceptos usados sistemáticamente a través de toda la especificación KDM. Tiene tres paquetes:
    a. *Core*: Define las abstracciones básicas de KDM.

   b. *KDM*: Proporciona el contexto compartido por todos los modelos KDM.
   c. *Source:* Define el conjunto de artefactos físicos del sistema de información heredado y permite referenciar partes del código fuente.

2. **Capa de Elementos de Programa**: proporciona una representación intermedia, independiente del lenguaje de programación, para representar los constructores comunes a varios lenguajes de programación. Se estructura en dos paquetes:
   a. *Code:* Define a bajo nivel los procedimientos, tipos de datos, unidades de compilación, etc.
   b. *Action*. Define las acciones llevadas a cabo por los elementos del paquete *code*. Los elementos de ambos paquetes se representan dentro de un modelo de código (elemento *CodeModel*).
3. **Capa de Recursos**: permite representar conocimiento sobre el entorno y los recursos de ejecución utilizados por los sistemas de información heredados. Dispone de cuatro paquetes:
   a. *Data:* Define los aspectos de los datos.
   b. *Event:* Define el modelo de eventos, condiciones y acciones del sistema de información heredado.
   c. *UI:* Define los aspectos de la interfaz de usuario del sistema de información heredado.
   d. *Platform:* Define las características de la plataforma de ejecución.
4. **Capa de Abstracción**: permite representar el conocimiento específico de dominio a la vez que da una visión de negocio de los sistemas de información heredados. Dispone de tres paquetes.
   a. *Conceptual:* Define los elementos específicos de dominio del sistema de información heredado.
   b. *Structure:* Define los componentes estructurales de los sistema de información heredados, es decir, los subsistemas, capas, paquetes, etc.
   c. *Build*: Define los artefactos finales relativos al sistema de información heredado.

3.3.4. QVT (Query/Views/Transformations)

QVT es un estándar del OMG para realizar consultas, obtener vistas y realizar transformaciones sobre modelos MOF (OMG, 2011b). Una consulta (*query*) toma como entrada un modelo y obtiene los elementos específicos de acuerdo a un patrón de búsqueda. Las consultas son realizadas a través del lenguaje declarativo OCL (*Object Constraint Language) (OMG, 2006)*. Una vista (*view*) es una proyección realizada sobre un modelo, que ha sido creada a través de una transformación. Las transformaciones (*transformations*) son operaciones que toman uno o más modelos de entrada para obtener un modelo de salida o resultado. QVT ofrece dos notaciones concretas (gráfica y textual) para definir las transformaciones.

Las características de QVT se mencionan a continuación:

- Soporte a consultas mediante un lenguaje de Consultas: Permite consultar los elementos de diferentes modelos.
- Soporte a transformaciones mediante un lenguaje de Transformación: Permite transformar modelos de entrada y ciertos modelos de salida.
- Naturaleza: Hibrida (Declarativa/Imperativa).
- Sintaxis Abstracta: La sintaxis abstracta de QVT es definida como un metamodelo conforme a MOF.
- Entrada y Salida como Modelos: Los mecanismos de transformación trabajan con modelos que son instancias de metamodelos conformes a MOF.
- Direccional: Las transformaciones pueden ser ejecutadas en las dos direcciones.
- Trazabilidad: Soporta trazabilidad entre los elementos del modelo de entrada y salida.
- Reusabilidad: Permite reutilizar fragmentos de las transformaciones.

Como se indicó anteriormente, la especificación de QVT posee una naturaleza híbrida declarativa/imperativa. La parte declarativa está dividida en una arquitectura de dos niveles y la parte imperativa está constituida por dos mecanismos involucrados en la implementación de las trasformaciones de los dos niveles anteriores. Por tanto, el

lenguaje QVT consiste en varios niveles. Los niveles y las relaciones entre ellos se muestran en la Figura 3.7.

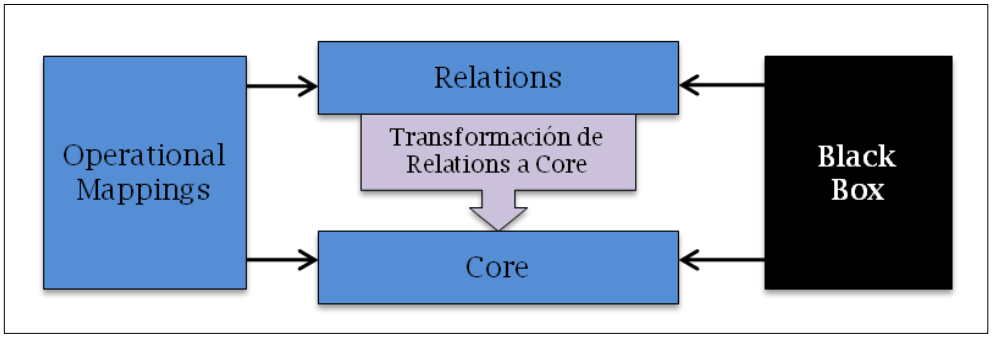


Figura 3.7. Relación entre los metamodelos QVT

En la parte declarativa se encuentran los lenguajes *Core* y *Relations*, que representan la misma semántica pero a dos niveles distintos de abstracción.

- **Core**: Es un pequeño lenguaje que soporta la coincidencia de patrones (*pattern matching*) sobre un conjunto fijo de variables mediante la evaluación de condiciones sobre esas variables contra un conjunto de modelos. En este caso, los modelos para representar la traza de una transformación deben ser definidos explícitamente, ya que no son deducidos a partir de la descripción de la transformación.
- **Relations (QVTr)**: Es una especificación declarativa de las relaciones entre los modelos MOF. El lenguaje *Relations* soporta la coincidencia de patrones de objetos complejos, y crea de manera implícita clases de traza y sus instancias, donde se almacena qué ocurre durante la ejecución de la transformación. Las transformaciones quedan descritas a través de (i) la enumeración de los metamodelos participantes, (ii) un conjunto de reglas que especifican la relación existente entre los términos de los metamodelos, (iii) un conjunto de dominios por regla que se ajusta al conjunto de términos para los que se expresan relaciones y (iv) un conjunto de patrones que cumplen con la estructura de los términos y contienen operaciones OCL. Una regla es de transformación dependiendo si el dominio destino está marcado como *checkonly* o como *enforce*. La marca *checkonly* comprueba si existe una correspondencia válida entre los modelos, mientras que la marca *enforce* fuerza a que los

términos de los respectivos modelos cumplan la relación descrita en la regla.

Por otra parte, en la parte imperativa de QVT se encuentran dos mecanismos adicionales que constituyen implementaciones imperativas de las transformaciones. Estos dos mecanismos son un lenguaje estándar (*Operational Mapping)* y un lenguaje no estándar (*Black-box MOF Operation)*, vistos en la Figura 3.7.

- **Operational Mappings (QVTo)**: Es un lenguaje que se utiliza para implementar una o más *Relations* de una especificación de *Relations*. Esto debe llevarse a cabo cuando resulte demasiado difícil especificar de manera declarativa cómo debe realizarse una *Relations*. Cuando una transformación se implementa completamente mediante *Operational Mappings* se dice que es una **transformación operacional**. Provee extensiones OCL, pero con un estilo más de procedimiento y una sintaxis más imperativa.
- **Black Box**: Se pueden derivar operaciones MOF a partir de *Relations* permitiendo así cualquier implementación de una operación MOF con la misma signatura. El peligro de estas implementaciones de *caja negra* es que tienen acceso a las referencias de los objetos dentro de los modelos, y por ello, pueden realizar cualquier acción sobre dichos modelos.

Entre las herramientas disponibles para diseñar y ejecutar transformaciones QVT se encuentran varias que soportan QVTr como Medini QVT, MOMENT o ModelMorf, y varias que soportan QVTo como Smart-QVT, Borland Together o M2M (Model To Model). En el Proyecto Fin de Carrera se usa la herramienta Medini QVT (ikv++, 2010) que es descrita en mayor profundidad en la sección 3.6.4 para un mejor entendimiento.

3.3.5. Escenarios ADM según el nivel de abstracción

Teniendo en cuenta el nivel de abstracción alcanzado en la etapa de ingeniería inversa (véase Figura 3.4), los modelos de modernización pueden categorizarse en tres tipos (Khusidman y Ulrich, 2007). En cada uno de estos tres tipos el conocimiento y los modelos disponibles son diferentes. En la mayoría de los casos, un nivel mayor de abstracción equivale a más información útil que ofrece más posibilidades durante la fase

de reestructuración. La Figura 3.8 muestra los tres tipos de modernización dependiendo del nivel de abstracción alcanzado.

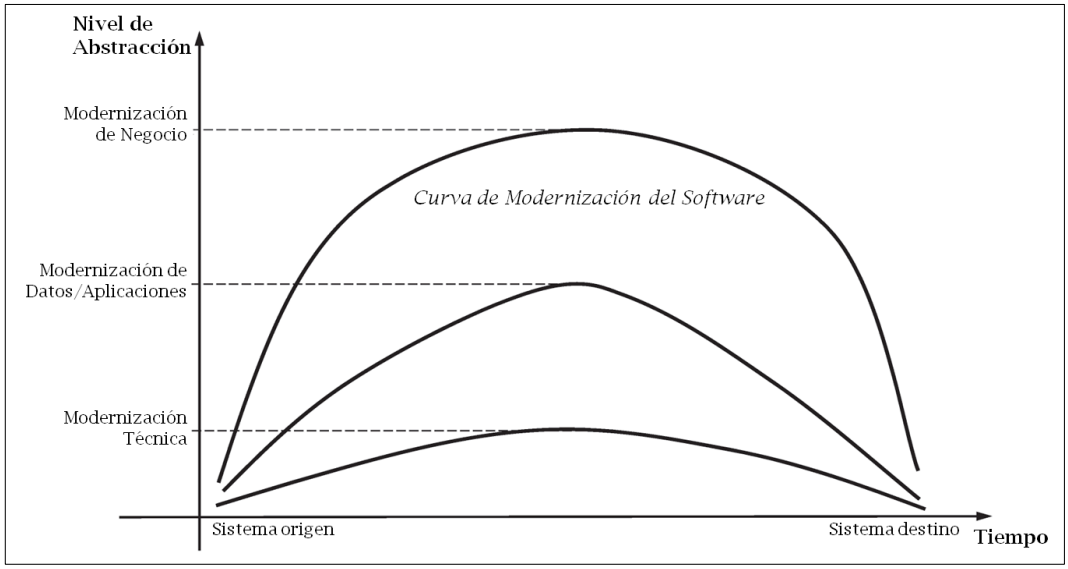


Figura 3.8. Tipos de modelos de modernización

Los tipos o escenarios de modernización con los que es posible encontrarse son los siguientes:

- **Modernización Técnica**: Es la de menor nivel de abstracción y la que comúnmente se aplica a los sistemas de información heredados. Esta modernización se aplica cuando se quiere hacer frente a obsolescencia de la plataforma o el lenguaje. Las nuevas posibilidades técnicas, la conformidad con nuevas normas, la eficiencia del sistema, la facilidad de uso del sistema u otros factores de modernización similares son las razones que justifican la modernización. En algunas ocasiones no es considerado un proceso de modernización en sí ya que sólo se basa en realizar correcciones y modificaciones.
- **Modernización de Datos/Aplicaciones**: Es la de nivel intermedio de abstracción y se centra en la reestructuración del sistema heredado a nivel del diseño de las aplicaciones y/o los modelos de datos involucrados. Durante esta modernización se facilita la reutilización del sistema, se reduce la lógica del sistema que se encuentra deslocalizada y se reduce la complejidad gracias a la aplicación de patrones de diseño.

- **Modernización de Negocio**: Es la de mayor nivel de abstracción y se aplica a nivel de la arquitectura de negocio (por ejemplo, reglas y procesos de negocio). Esta modernización introduce modelos semánticos de negocio que permiten preservar la información de negocio cuando se alinean con los requisitos futuros de la empresa.

  Este tipo de modernización es probablemente el más importante de los tres ya que los cambios realizados en los procesos de negocio son críticos durante la modernización del software (Koskinen, Ahonen et al., 2005). En este último modelo de modernización del software es en el que se centra el proyecto fin de carrera que se presenta.

3.4. Procesos de negocio

Los procesos de negocio (BP por sus siglas en inglés, *Business Process*) definen la secuencia de actividades necesarias para conseguir los objetivos de negocio definidos por una organización. Un proceso de negocio toma una o varias entradas y genera una salida que aporta un valor para los clientes de la organización (Jeston, Nelis et al., 2008; Weske, 2007).

En (WfMC, 1999), los procesos de negocio son descritos como un conjunto de uno o más procedimientos vinculados o actividades que colectivamente realizan el objetivo de negocio dentro del contexto de una estructura organizacional definiendo los roles y las relaciones entre ellos. Estas actividades y procedimientos son ejecutados por los trabajadores y por los sistemas de información por lo que la colaboración entre ellos debe de producirse de manera correcta y eficiente (Weske, 2007).

En los últimos años, y de cara a optimizar el rendimiento empresarial, se ha observado un incremento del interés de las organizaciones en las tareas específicas de gestión de sus procesos de negocio (BPM, *Business Process Management*) (Weske, 2007). Esta gestión conlleva definir e incorporar métodos, técnicas y herramientas para apoyar el diseño, gestión y análisis de los procesos de negocio (Aalst, Hofstede et al., 2003). Una gestión óptima de los procesos de negocio permite adaptarlos a posibles cambios del entorno con el fin de incrementar la satisfacción del cliente, reducir costes, diferenciar sus productos o servicios, etc. (Jeston, Nelis et al., 2008). La relación entre los procesos de negocio y los sistemas de información se resume en la Figura 3.9:

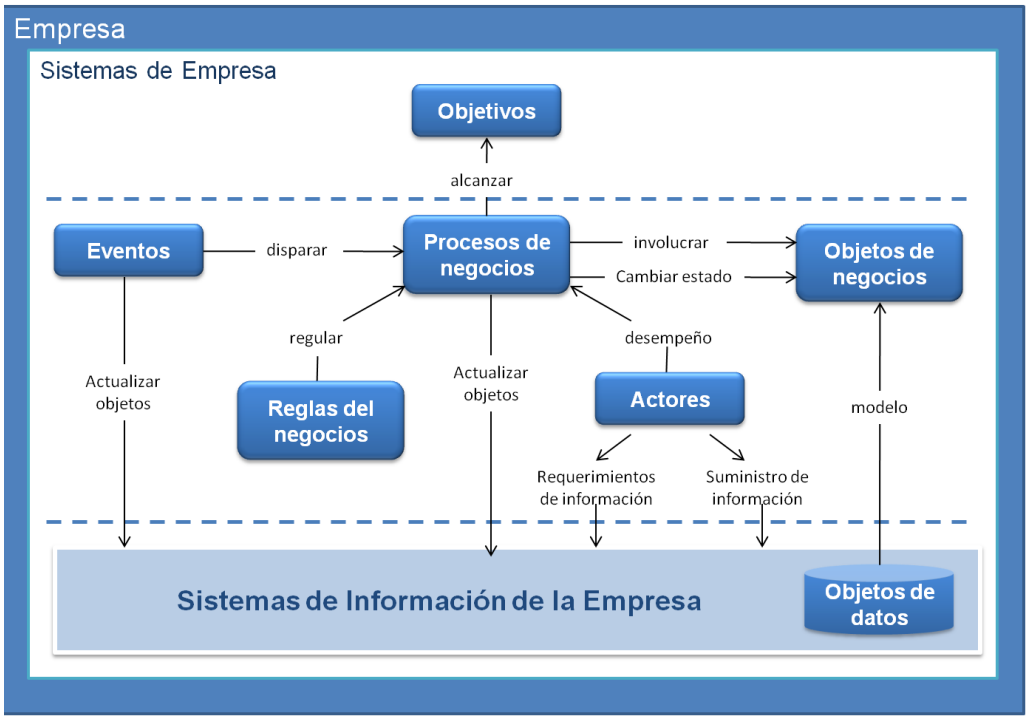


Figura 3.9. Relación entre Procesos de Negocio y Sistemas de Información (Rodríguez Ríos, Fernández-Medina et al., 2007)

Es importante que las organizaciones realicen una gestión de sus procesos de negocio de forma eficaz y continua a fin de mantener y mejorar su nivel de competitividad.

El ciclo de vida de la gestión de procesos de negocio se muestra en la Figura 3.10. Como puede verse, este ciclo de vida consiste en cuatro fases que están relacionadas entre sí y forman una estructura cíclica:

- **Diseño y Análisis**: En esta primera fase los procesos de negocio son identificados, revisados, validados y representados mediante modelos de proceso de negocio. Éstos son expresados en una notación gráfica con el objetivo de facilitar la comunicación entre los procesos y con los propios interesados (*stakeholders*). Además de lo anterior, también se aplican técnicas de modelado como la validación, simulación y técnicas de verificación.
- **Configuración**: Una vez que los procesos de negocio han sido diseñados y verificados se procede a realizar la implementación. Para ello, se elige la plataforma de ejecución, la tecnología de base de datos que va a utilizarse, etc. Una vez que el sistema ha sido configurado, la

implementación necesita ser probada mediante técnicas de *testing* con el fin de detectar problemas.

- **Implantación**: Las instancias del proceso de negocio son inicializadas en esta fase siguiendo la configuración anterior. Además, se realiza un monitoreo de esas instancias con el fin de proporcionar información sobre su estado.
- **Evaluación**: En esta última fase se utiliza la información disponible y se realiza una evaluación y una mejora de los modelos de proceso de negocio y sus implementaciones. La evaluación se realiza mediante el monitoreo de actividades de negocio y mediante técnicas de minería de procesos.

Figura 3.10. Ciclo de vida de los Procesos de Negocio (Weske, 2007)

3.4.1. BPMN (Business Process Model and Notation)

Como parte de la gestión de procesos de negocio, es necesario poder representar los procesos de negocio en una notación entendible por las personas involucradas en su gestión. En este Proyecto Fin de Carrera se ha decidido utilizar la notación BPMN (*Business Process Modeling Notation*) (OMG, 2011a; White y Miers, 2008) por ser una

notación gráfica muy extendida para representar los Diagramas de Procesos de Negocio (BPD, *Business Process Diagram*), actualmente en su versión 2.0.

BPMN es un estándar desarrollado por la OMG que proporciona una notación para el modelado de la gestión de los procesos de negocio, mostrando las interrelaciones entre los distintos componentes, sus fuentes de información y el personal asociado a ellos. Todo esto es descrito mediante una forma entendible para todos los miembros de la organización, desde los analistas de negocio hasta los desarrolladores.

BPMN ofrece las siguientes facilidades que permiten el modelado de los procesos de negocio así como su definición:

- Modelado a nivel de negocio.
- Comprobación de la conformidad de la coreografía.
- Comprobación de la conformidad de las reglas.
- Taxonomías de proceso.
- Interoperabilidad en la ejecución.
- Extensibilidad de la gestión en tiempo real.
- Extensibilidad del lenguaje.
- Reusabilidad del modelo.
- Equivalencias entre el negocio y la tecnología de la información.

BPMN fue desarrollada por la iniciativa BPMI (*Business Process Management Initiative*). En febrero de 2006 BPMN fue adoptada como un estándar de la OMG. A partir de esa fecha se fueron generando nuevas versiones de la notación hasta lanzar la última versión en 2009 (BPMN 2.0) que es la utilizada actualmente. Un resumen de todas las versiones que ha tenido BPMN a lo largo del tiempo se muestra en la Tabla 3.1.

Versión	Fecha de lanzamiento	Desarrollada por	Nº de elementos
BPMN 1.0	Mayo 2004	BPMI	48
BPMN 1.1	Enero 2008	OMG	55
BPMN 1.2	Enero 2009	OMG	55
BPMN 2.0	Agosto 2009	OMG	116

Tabla 3.1. Versiones de BPMN

3.4.1.1. Descripción de la Notación BPMN

La notación BPMN 2.0 (OMG, 2011a) se encuentra organizada en cinco categorías (véase Figura 3.11):

- **Objetos del flujo** (*Flow Objects*)
  - Evento (*Event*)
  - Actividad (*Activity*)
  - Puerta (*Gateway*)
- **Datos** (Data)
  - Objeto de datos (*Data Object*)
  - Entrada de datos (*Data Input*)
  - Salida de datos (*Data Output*)
  - Almacén de datos (*Data Store*)
- **Conectores de los elementos del flujo** (*Connecting Objects*)
  - Flujo de secuencia *(Sequence Flow*)
  - Flujo de mensajes (*Message Flow*)
  - Asociación (*Association*)
  - Asociaciones de datos (*Data Associations*)
- **Categorías de agrupaciones** (*Swimlines*)
  - Contenedor (*Pool*)
  - Compartimento (*Lane*)
- **Artefactos** (*Artifacts*)
  - Agrupación (*Group*)
  - Anotación (*Text Annotation*)

En el Anexo I. Notación BPMN se detallan en profundidad cada uno de los elementos básicos de BPMN.

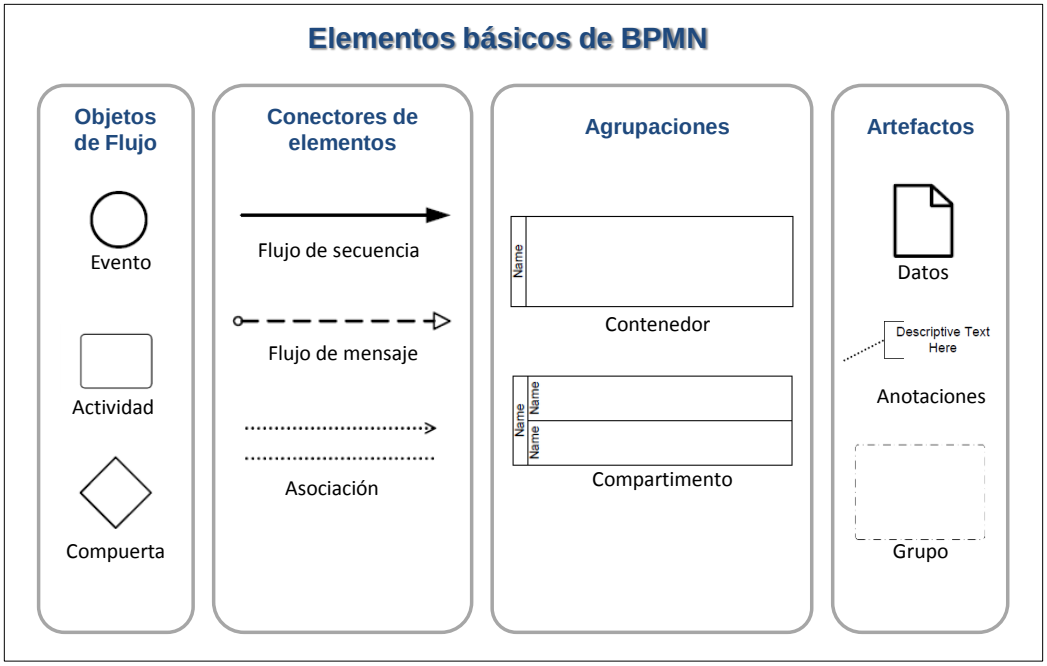


Figura 3.11. Elementos básicos de BPMN 2.0

3.5. MARBLE

MARBLE (*Modernization Approach for Recovering Business processes from LEgacy systems*) (Pérez-Castillo, García-Rodríguez de Guzmán et al., 2011b) es un marco para la obtención de procesos de negocio a partir de sistemas de información heredados, centrándose en la fase de ingeniería inversa del modelo de herradura. MARBLE está basado en KDM (véase apartado 3.3.3), el cual permite realizar representaciones conceptuales abstractas de las diferentes vistas de la arquitectura de los sistemas de información heredados. Posteriormente, ese conocimiento es progresivamente transformado y depurado hasta llegar a los procesos de negocio subyacentes. Para ello, MARBLE se divide en cuatro niveles de abstracción y define tres transformaciones entre ellos como se muestran en la Figura 3.12:

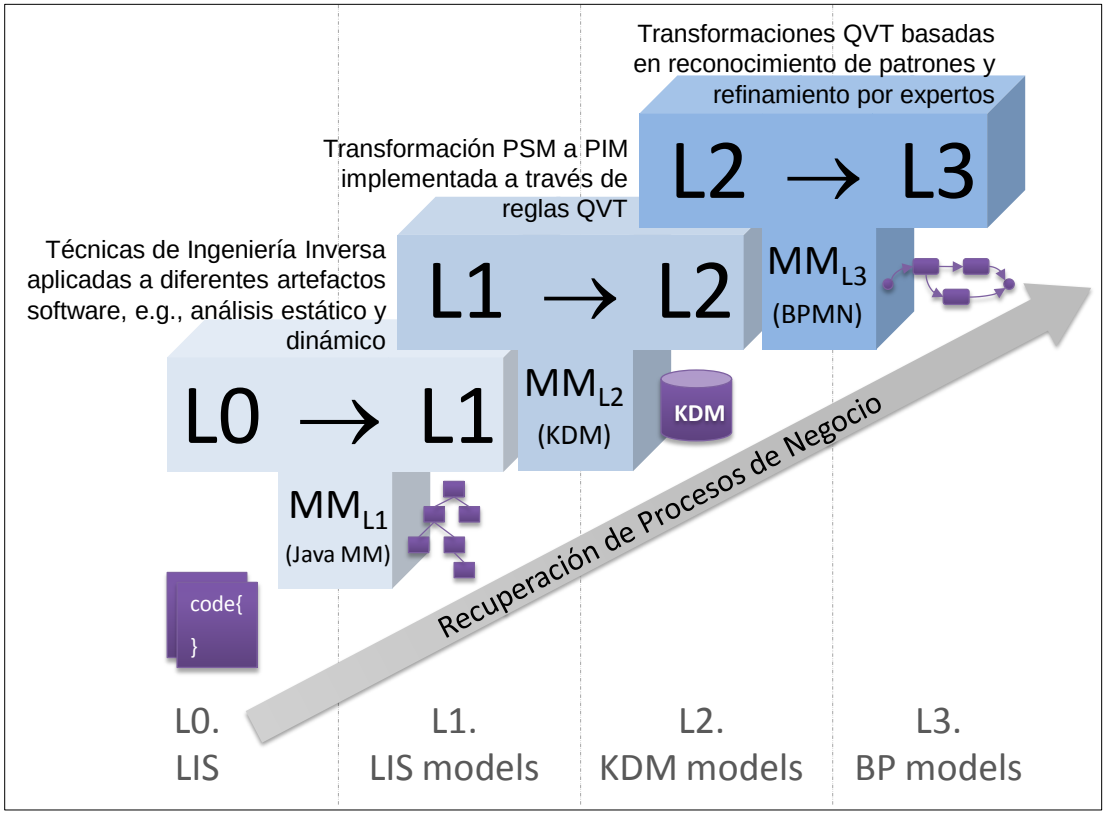


Figura 3.12. Vista general de MARBLE

- **Nivel 0 (L0):** representa al sistema de información heredado de origen en el mundo real, cuyos procesos de negocio asociados se pretende descubrir.
- **Nivel 1 (L1)**: agrupa un conjunto de modelos que representan las diferentes vistas o aspectos de la arquitectura del sistema de información heredado. La transformación L0-a-L1 obtiene modelos PSM desde cada activo o artefacto software heredado. Para ello se recurre a la extracción de información del sistema de información heredado por medio de técnicas de ingeniería inversa clásicas como análisis estático/dinámico, *slicing*, etc. Los diferentes modelos PSM se construyen de acuerdo a metamodelos existentes o creados específicamente para tal fin.
- **Nivel 2 (L2)**: se centra en la representación integrada del sistema de información heredado de acuerdo al metamodelo de KDM. Durante la transformación L1-a-L2 los modelos específicos se transforman hacia un único modelo PIM basado en KDM. Estas transformaciones están desarrolladas en QVT y reducen progresivamente la brecha conceptual entre el sistema de información heredado y los procesos de negocio. El

paso desde el sistema de información heredado (L0) al modelo KDM (L2) no es directo ya que en muchos casos el conocimiento específico a la plataforma puede ser utilizado para inferir los BP. Gracias al nivel intermedio L1 se reduce la perdida semántica común a todo proceso de ingeniería inversa.

- **Nivel 3 (L3)**: este nivel representa los procesos de negocio que son recuperados desde el modelo KDM. Este modelo equivale al CIM y representa un conjunto de diagramas de procesos de negocio. Para la representación de los diagramas de procesos de negocio, MARBLE cuenta con un metamodelo *ex profeso* basado en BPMN. La transformación L2-a-L3 se establece también mediante transformaciones QVT, aunque en este caso están basadas en la detección de patrones de negocio en el modelo KDM. Adicionalmente, un experto de negocio puede refinar los diagramas de procesos de negocio extraídos mediante las transformaciones.

3.6. Entorno Eclipse™

Eclipse (Eclipse, 2011a) es una plataforma de desarrollo de software de código abierto (bajo la licencia *Eclipse Public License*) multiplataforma que permite la integración de diferentes herramientas para dar lugar a un entorno de desarrollo integrado (IDE por sus siglas en inglés *Integrated Development Environment*).

El proyecto Eclipse fue creado por IBM en noviembre de 2001 como sucesor de las herramientas para *VisualAge*. En 2003 se creó la Fundación Eclipse que pasó a hacerse cargo de la administración de Eclipse, siendo Eclipse 3.0 su primera versión.

La Fundación Eclipse es una organización independiente sin ánimo de lucro que se encarga del desarrollo de una serie de proyectos. Estos proyectos dan lugar a marcos de trabajo, herramientas y plug-ins que se unen a la plataforma de desarrollo Eclipse. Muchos de los proyectos desarrollados por Eclipse son escritos en lenguaje Java.

Hasta la fecha, la Fundación Eclipse ha lanzado diversas versiones, recogidas en la Tabla 3.2.

Versión	Fecha de lanzamiento	Versión de la plataforma	Proyectos
Indigo	22-Junio-2011	3.7	http://wiki.eclipse.org/Indigo
Helios[1]	23-Junio-2010	3.6	http://wiki.eclipse.org/index.php/Helios
Galileo	24-Junio-2009	3.5	http://wiki.eclipse.org/Galileo
Ganymede	25-Junio-2008	3.4	http://wiki.eclipse.org/index.php/Ganymede_Simultaneous_Release
Europa	29-Junio-2007	3.3	http://wiki.eclipse.org/index.php/Europa_Simultaneous_Release
Callisto	30-Junio-2006	3.2	http://www.eclipse.org/callisto/callistoprojects.php
Eclipse 3.1	28-Junio-2005	3.1	
Eclipse 3.0	28-Junio-2004	3.0	

Tabla 3.2. Versiones de Eclipse

Entre los proyectos disponibles para la plataforma Eclipse, versión Helios, merece la pena citar Eclipse Modeling Tools (Eclipse, 2010b), que ha sido la utilizada en el presente Proyecto Fin de Carrera para crear el editor gráfico donde se muestran los procesos de negocio obtenidos por la herramienta. Este paquete de modelado contiene una colección de componentes de Eclipse Modeling Project: EMF, GMF, MDT XSD/OCL/UML2, M2M, M2T y EMFT. Estos conceptos son definidos en las secciones siguientes.

3.6.1. Concepto de plug-in Eclipse

Eclipse no es sólo un programa único, sino un programa que permite la extensión de nuevas funcionalidades a través de los llamados plug-ins. Un plug-in puede consumir servicios proporcionados por otros plug-ins o puede extender su funcionalidad para ser consumido por otros plug-ins. Estos plug-ins se cargan dinámicamente por Eclipse en tiempo de ejecución. De hecho, toda la funcionalidad de Eclipse se encuentra en diferentes plug-ins a excepción del núcleo (Clayberg y Rubel, 2008).

Un plug-in es una pequeña unidad de la plataforma Eclipse que se puede desarrollar por separado y puede ser entregado como una caja negra mediante un

[1] Para el desarrollo del presente PFC se ha utilizado la versión Helios ya que era la versión más reciente de todas las disponibles a la fecha de comienzo del mismo.

archivo **.jar*. En un plug-in se encuentra todo el código y los recursos que necesita para funcionar tales como código, archivos de imagen, paquetes de recursos, etc.

3.6.2. Eclipse Modeling Framework Project (EMF)

El proyecto EMF (*Eclipse Modeling Framework*) (Eclipse, 2010a; Steinberg, Budinsky et al., 2008) de Eclipse es un framework de modelado que permite la generación de código para construir herramientas a partir de la definición de un modelo de datos estructurado. A partir de una especificación del modelo descrito en XMI, EMF proporciona herramientas y soporte de ejecución para producir un conjunto de clases Java para el modelo, junto con un conjunto de clases que permiten la visualización y la edición del modelo y un editor básico. EMF une tres importantes tecnologías: Java, XML y UML.

El lenguaje para definir el modelo de datos es el lenguaje Ecore que sigue la jerarquía mostrada en la Figura 3.13, donde se encuentran elementos como *EClass*, *EPackage*, *EAttribute*, *EEnum*, etc.

3.6.3. Graphical Modeling Framework (GMF)

GMF es un plug-in Eclipse que permite crear editores gráficos de modelos a partir de modelos definidos mediante el metamodelo EMF. Es decir, GMF puede ser usado para crear lenguajes específicos de dominio. GMF depende de otros plug-in Eclipse:

- EMF (*Eclipse Modeling Framework*) plug-in: permite definir los metamodelos en EMF. Véase sección 0.
- GEF (*Graphical Editing Framework*): permite la definición de componentes gráficos.
- EMF OCL/Query/Validation/Transaction: permite establecer las restricciones OCL al metamodelo.

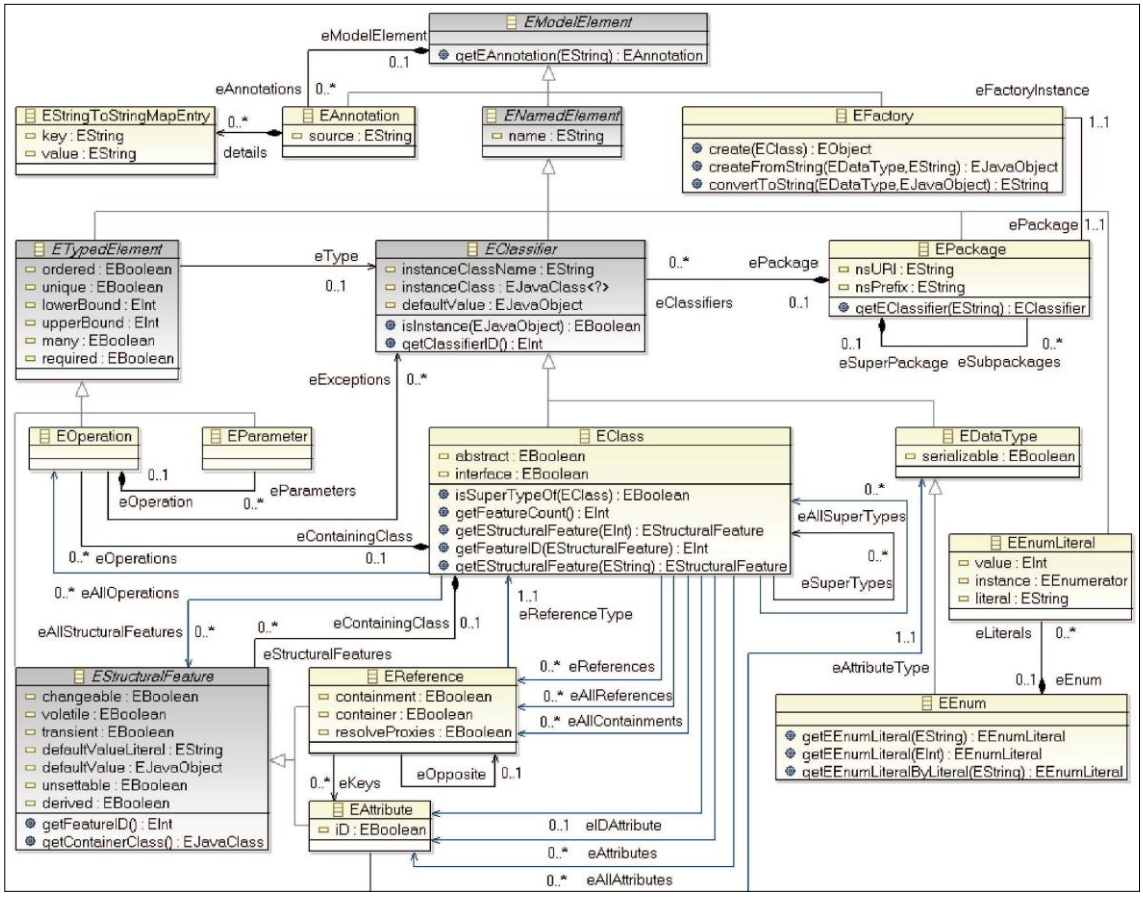


Figura 3.13. Jerarquía de elementos del lenguaje Ecore de EMF

A la hora de llevar a cabo la creación de un editor gráfico con GMF es necesario seguir los pasos que propone el propio GMF (Véase Figura 3.14). En la figura se muestran los componentes de un proyecto GMF que se describen a continuación:

1. **Modelo de dominio (.ecore):** El modelo de dominio es el metamodelo base del Editor Gráfico. Este metamodelo está expresado en el lenguaje ECORE.
2. **Generador de código del modelo EMF (.genmodel):** El modelo generado permite generar automáticamente el código asociado al meta-modelo y a los editores no gráficos.
3. **Modelo de definición gráfica (.gmfgraph):** El modelo de definición gráfica se utiliza para definir las figuras, nodos, links, etc. es decir, todo aquello que se muestra en el diagrama.
4. **Modelo de definición de la herramienta (.gmftool):** El modelo de definición de la herramienta se utiliza para especificar la paleta, herramientas de creación, acciones, etc. para los elementos gráficos.

5. **Definición de correspondencia o Mapping (.gmfmap):** El modelo de definición de correspondencia es el que relaciona los tres modelos creados: el de dominio, el gráfico y el de herramienta.
6. **Modelo generador (.gmfgen):** Es el encargado de generar todo el código del diagrama.

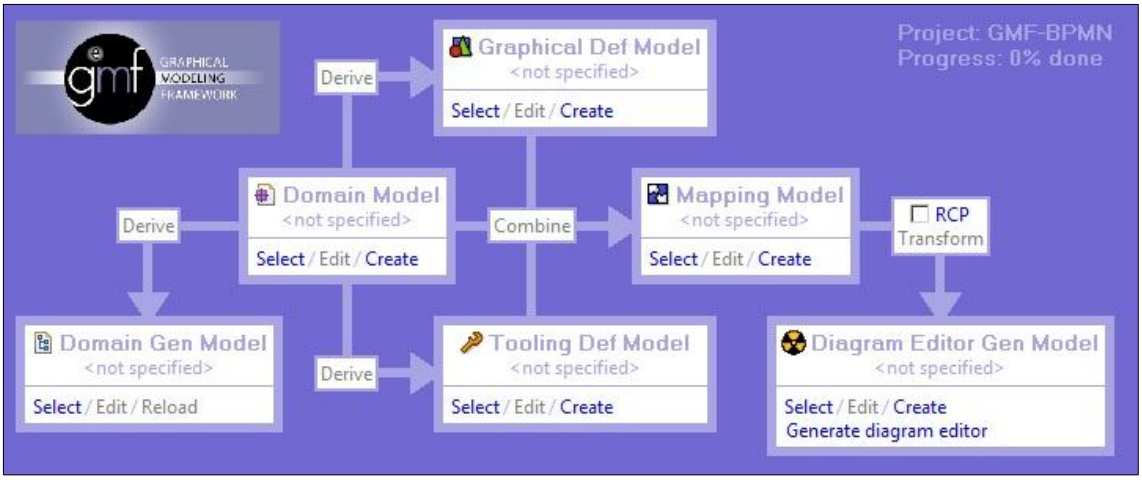


Figura 3.14. Pasos para la creación de un proyecto GMF

3.6.4. Medini QVT

Medini QVT (ikv++, 2010) es un conjunto de herramientas para el implementar las transformaciones entre modelos. Se basa en el estándar QVTr de la OMG (descrito en el apartado 3.3.4). Incluye un motor de transformaciones QVTr, un editor para asistir en la implementación de transformaciones QVTr así como un módulo de ejecución depuración. Se encuentra bajo la licencia pública EPL (*Eclipse Public License*) y entre sus características principales se encuentran:

- Ejecuta las transformaciones QVT expresadas en la sintaxis concreta del lenguaje Relations.
- Permite la integración con Eclipse.
- Proporciona un editor de código con un asistente.
- Proporciona un depurador.
- Permite la actualización incremental.
- Permite la creación de transformaciones bidireccionales.

4. MÉTODO DE TRABAJO

En este capítulo se describe el método de trabajo utilizado para llevar a cabo el desarrollo de la herramienta MARBLE Tool y la planificación que se ha seguido para su consecución.

4.1. Proceso Unificado de Desarrollo (PUD)

El PUD (Jacobson, Booch et al., 2000) es un marco de desarrollo de software extensible que puede ser adaptado a organizaciones o proyectos específicos. El PUD define un "*conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema software*".

El PUD surge como una evolución del Proceso Unificado de Rational (RUP) y posee como principales características el estar basado en componentes software conectados a través de interfaces y el uso del lenguaje UML (*Unified Modelling Language*, Lenguaje Unificado de Modelado) para especificar los diagramas y demás artefactos junto con el código ejecutable.

El objetivo principal del PUD es facilitar la creación de software de calidad que satisfaga las necesidades de los usuarios.

El PUD presenta las siguientes características:

- **Dirigido por casos de uso:** A la hora de desarrollar un sistema es necesario identificar las funcionalidades que los usuarios necesitan ejecutar cuando van a interaccionar con el sistema. Estas funcionalidades se denominan requisitos funcionales y habitualmente se representan mediante casos de uso. Cada caso de uso representa un escenario de uso. Se pueden usar los casos de uso como guía durante el proceso de desarrollo desde la especificación de requisitos hasta las pruebas. Al conjunto de todos los casos de uso se le denomina modelo de casos de uso. La elección de los casos de uso que deben desarrollarse primero condiciona notablemente la arquitectura del sistema y la propia evolución del desarrollo.

- **Centrado en la arquitectura:** La arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema y debe estar relacionada con los casos de uso para permitir el desarrollo de los mismos. Tanto la arquitectura como los casos de uso deben desarrollarse en paralelo.
- **Iterativo e incremental:** El PUD está compuesto de una serie de iteraciones (o miniproyectos). Cada iteración se centra en un conjunto de casos de uso elegidos convenientemente (por ejemplo desarrollando primero aquellos que condicionen más la arquitectura, o que tengan menos dependencias), y su finalización supone un incremento de la versión anterior del sistema, donde se han añadido nuevas funcionalidades o mejorado algunas existentes. De esta forma, cada iteración supone un refinamiento progresivo del sistema, que va además aumentando de tamaño. Cada una de estas iteraciones sigue el flujo de trabajo:
    - **Requisitos**: Se recolectan todos los requisitos funcionales del sistema.
    - **Análisis**: Se identifican y especifican los casos de uso.
    - **Diseño**: Se crea un diseño utilizando la arquitectura seleccionada como guía para tratar de dotar al sistema de las funcionalidades representadas por los casos de uso identificados en la etapa anterior.
    - **Implementación**: Se traduce a código las decisiones tomadas en la etapa de diseño.
    - **Pruebas**: Se verifica que el sistema cumple los casos de uso.

La relación entre estas tres características es que la arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración.

La Figura 4.1 muestra un resumen del Proceso Unificado de Desarrollo.

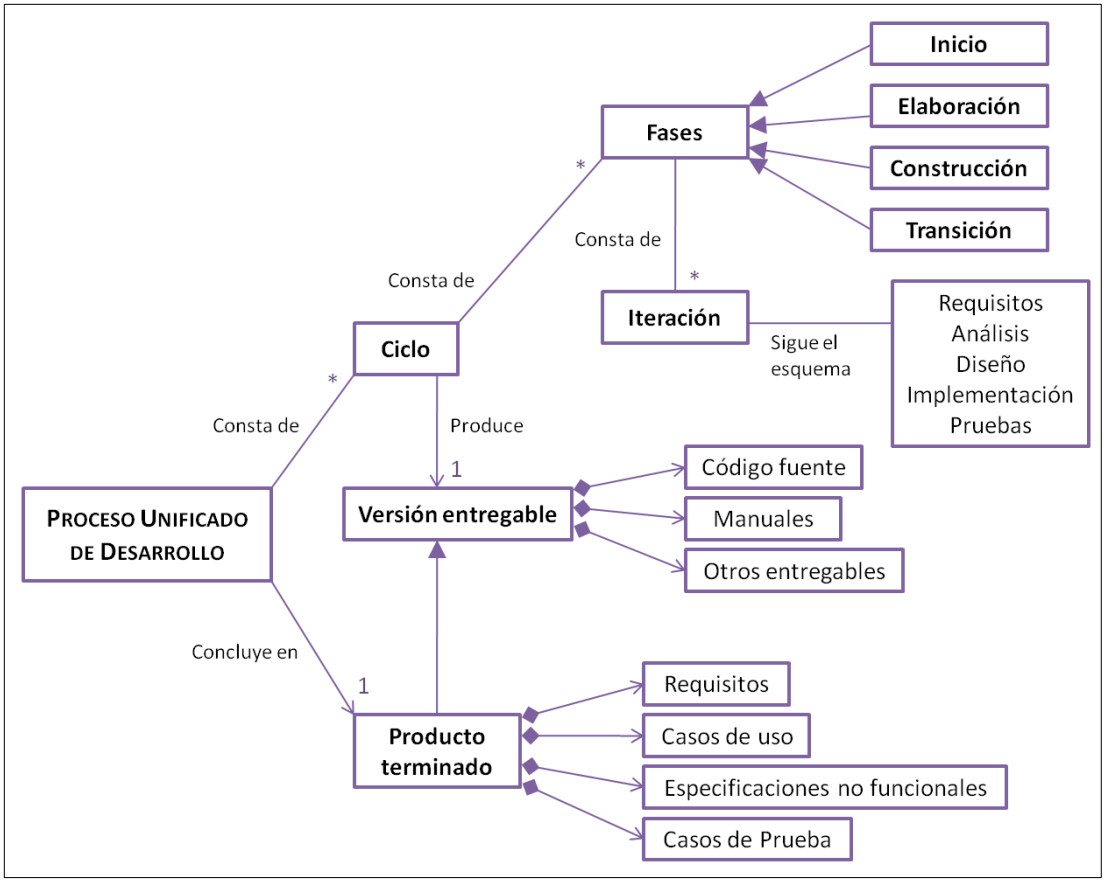


Figura 4.1. Mapa conceptual del PUD

4.1.1. Fases del Proceso Unificado de Desarrollo

El PUD divide el proceso de desarrollo de un proyecto en cuatro fases, como muestra la Figura 4.2, que a su vez se dividen en iteraciones. Cada una de estas iteraciones contempla algunos de los flujos de trabajo y cada fase termina en un hito donde se cumplen una serie de objetivos y se producen una serie de artefactos. Las fases del PUD son:

1. **Inicio o Comienzo:** En la primera fase de obtiene el modelo de casos de uso simplificado, se identifican los riesgos potenciales y se realiza una estimación aproximada del proyecto.
2. **Elaboración:** Se mejora el modelo de casos de uso de la fase anterior y se diseña la arquitectura del sistema. Se desarrollan los casos de uso más críticos que se identificaron en la fase de inicio.
3. **Construcción**: Se desarrollan todos los casos de uso y se prueba el software.

4. **Transición:** Se produce la entrega e instalación del software a los usuarios.

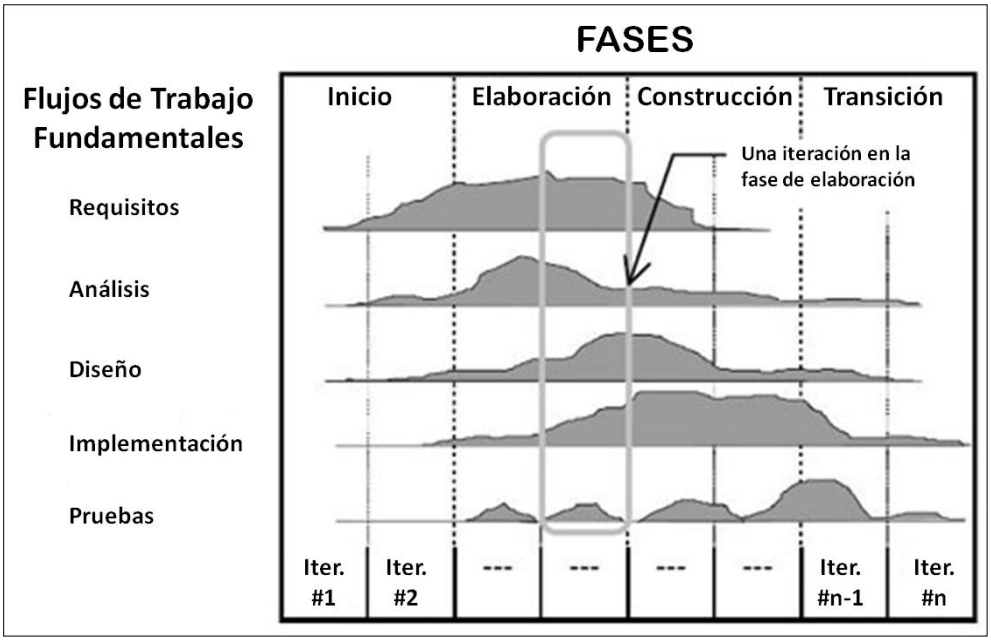


Figura 4.2. Fases del PUD

Durante el desarrollo del producto se construyen diferentes artefactos, los cuales representan el mismo sistema pero desde diferentes puntos de vista. Estos puntos de vista son:

- **Modelo de Análisis**: Explica los casos de uso con mayor detalle y asigna cada una de las funcionalidades a sus posibles clases.
- **Modelo de Diseño**: Constituye el diagrama de clases y de paquetes en la notación UML.
- **Modelo de Implementación:** Incluye los componentes y la relación entre clases y componentes.
- **Modelo de Despliegue**: Realiza una representación a través de nodos y las relaciones entre éstos.
- **Modelo de Prueba**: Se especifican los casos de prueba.

4.2. Evolución del Proyecto

Siguiendo la metodología PUD, en esta sección se muestran la planificación de las iteraciones que se pretenden llevar a cabo para la realización de la herramienta

propuesta en este Proyecto Fin de Carrera, así como los diferentes flujos de trabajo que se desarrollan en cada una de ellas. Posteriormente, los productos de salida de cada una de las iteraciones se mostrarán en el capítulo 5. El proyecto se ha estructurado en 10 iteraciones que se describen en los subapartados siguientes.

4.2.1. Iteración 1

Esta iteración sirve como base para el resto de iteraciones. En ella se obtiene la lista de requisitos del usuario y se realiza un análisis de los mismos mediante la identificación de los casos de uso, realizando una primera aproximación del diagrama de casos de uso de la herramienta. Una vez identificados los casos de uso se priorizan con el fin de establecer una cierta ordenación a la hora de realizar el análisis, diseño, implementación y pruebas de los mismos. Tras esto, se realiza la planificación del Proyecto Fin de Carrera indicando qué caso de uso se desarrolla en qué iteración. La Tabla 4.1 muestra de manera resumida la iteración 1.

Iteración 1	
Fase en la que se enmarca:	Inicio
Flujos de trabajo que se realizan:	Requisitos, Análisis
Objetivos	Productos de Salida
▪ **OB. 1.1.** Realizar el Análisis de los requisitos mediante la identificación de casos de uso (CdU) ▪ **OB. 1.2.** Priorización de los casos de uso (CdU) ▪ **OB. 1.3.** Realizar la Planificación del Proyecto Fin de Carrera	▪ **PS. 1.1.** Lista de requisitos del usuario. ▪ **PS. 1.2.** Primera versión del diagrama de casos de uso. ▪ **PS. 1.3.** Priorización de los casos de uso. ▪ **PS. 1.4.** Esquema de la descripción de la arquitectura candidata. ▪ **PS. 1.5.** Planificación del Proyecto Fin de Carrera.

Tabla 4.1. Resumen Iteración 1

Para facilitar su entendimiento se muestra por conveniencia aquí el diagrama de casos de uso (Figura 4.3), que puede encontrarse totalmente explicado en el capítulo 5.

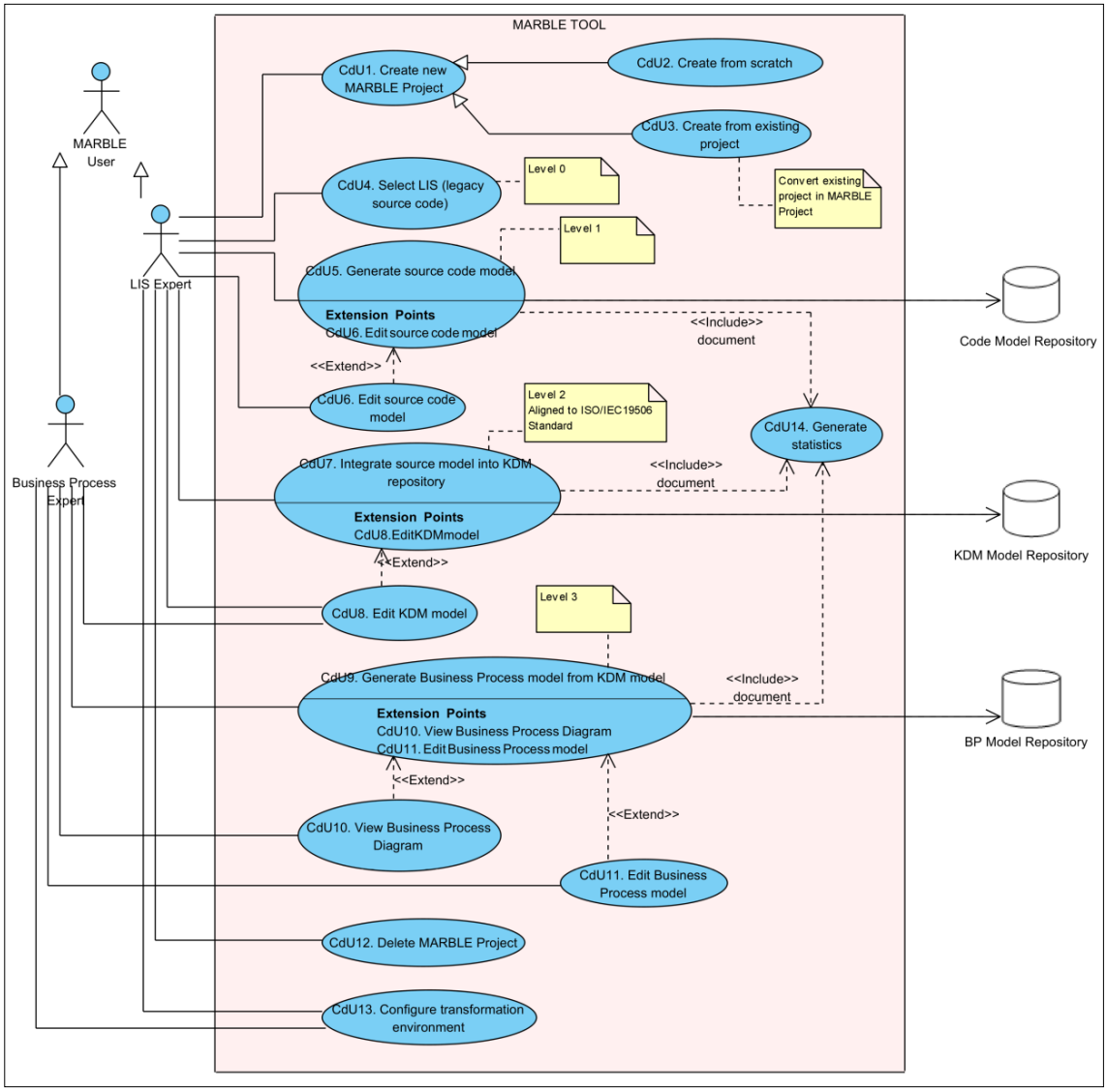


Figura 4.3. Diagrama de Casos de Uso

Con los productos de salida de esta iteración (que son mostrados en el capítulo siguiente correspondiente a los resultados: Tabla 5.1. Requisitos Funcionales, Figura 5.1. Diagrama de Casos de Uso, Tabla 5.16. Priorización de los casos de uso, Figura 5.3. Planificación del Proyecto Fin de Carrera) se ha confeccionado la Tabla 4.2, que resume la evolución del proyecto, mostrando las iteraciones realizadas, así como los diferentes flujos de trabajo que son desarrollados en cada una de ellas.

	Iteración	Casos de Uso	A	D	I	P
INICIO	1	Diagrama de CdU				
ELABORACIÓN	2	CdU1	X	X		
		CdU2	X	X		
	3	CdU1			X	
		CdU2			X	
		CdU3	X	X		
		CdU4	X	X		
		CdU5	X	X		
		CdU6	X	X		
		CdU14	X	X		
	4	CdU3			X	
		CdU4			X	
		CdU5			X	
		CdU6			X	
		CdU14			X	
		CdU7	X	X		
		CdU8	X	X		
	5	CdU7			X	
		CdU8			X	
		CdU9	X	X		
		CdU11	X	X		
	6	CdU9			X	
		CdU10	X	X		
		CdU11			X	
CONSTRUCCIÓN	7	CdU1				X
		CdU2				X
		CdU3				X
		CdU4				X
		CdU5				X
		CdU6				X
		CdU10			X	
		CdU12	X	X		
		CdU13	X	X		
		CdU14				X
	8	CdU7				X
		CdU8				X
		CdU9				X
		CdU10				X
		CdU11				X
		CdU12			X	
		CdU13			X	
	9	CdU12				X
		CdU13				X
TRANSICIÓN	10	Documentación				

Tabla 4.2. Evolución del Proyecto Fin de Carrera

4.2.2. Iteración 2

Una vez realizada la planificación del Proyecto Fin de Carrera se comienza la ejecución del mismo de acuerdo al plan resultante. Dicho plan lleva inicialmente a la realización del análisis y diseño de los primeros casos de uso (véase Tabla 5.16 con la priorización de los casos de uso realizada de acuerdo a su importancia y dependencias funcionales con otros casos de uso). En esta iteración se trata de proporcionar la opción de crear un nuevo tipo de proyecto (Proyecto MARBLE) que contendrá todos los entregables. Esta acción se podrá hacer desde cero o a partir de un proyecto existente. En esta iteración sólo se contemplará el caso de crear un proyecto desde cero. Se realiza un análisis del problema, viendo las posibles opciones para abordar el problema, y se propone un diseño de la solución. La Tabla 4.3 muestra de manera resumida la iteración 2.

Iteración 2	
Fase en la que se enmarca:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño
Objetivos	Productos de Salida
▪ **OB. 2.1.** Análisis y Diseño de los casos de uso: o CdU1. Crear nuevo Proyecto MARBLE o CdU2. Crear desde cero	▪ **PS. 2.1.** Diagramas de secuencia de análisis de los casos de uso CdU1 y CdU2 ▪ **PS. 2.2.** Diagramas de comunicación de los casos de uso CdU1 y CdU2 ▪ **PS. 2.3.** Modelo arquitectónico del proyecto ▪ **PS. 2.4.** Diagrama de clases de los casos de uso CdU1 y CdU2 ▪ **PS. 2.5.** Diagrama de secuencia de diseño de los casos de uso CdU1 y CdU2 ▪ **PS. 2.6.** Prototipos de las GUIs relativas a los casos de uso CdU1 y CdU2.

Tabla 4.3. Resumen Iteración 2

4.2.3. Iteración 3

En esta iteración se continúa con el análisis y diseño de los siguientes casos de uso atendiendo a la priorización realizada (ver Tabla 5.16 con la priorización de los casos de uso). De forma concurrente a este análisis y diseño se comienza a realizar la implementación de los casos de uso anteriores. De esta forma, se obtiene el diseño de los casos de uso siguientes y la implementación de los casos de uso estudiados en la iteración anterior. La Tabla 4.4 muestra de manera resumida la iteración 3.

Iteración 3	
Fase en la que se enmarca:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación
Objetivos	Productos de Salida
▪ **OB. 3.1**. Análisis y Diseño de los casos de uso: o CdU3. Crear a partir de un proyecto existente. o CdU4. Seleccionar LIS (código fuente heredado) o CdU5. Generar modelo de código fuente o CdU6. Editar modelo de código fuente o CdU14.Generar estadísticas ▪ **OB. 3.2.** Implementación de los casos de uso CdU1 y CdU2.	▪ **PS. 3.1.** Diagramas de secuencia de análisis de los casos de uso CdU3, CdU4, CdU5, CdU6 y CdU14 ▪ **PS. 3.2.** Diagramas de comunicación de los casos de uso CdU3, CdU4, CdU5, CdU6 y CdU14 ▪ **PS. 3.3.** Diagrama de clases de los casos de uso CdU3, CdU4, CdU5, CdU6 y CdU14 ▪ **PS. 3.4.** Diagrama de secuencia de diseño de los casos de uso CdU3, CdU4, CdU5, CdU6 y CdU14 ▪ **PS. 3.5.** Prototipos de las GUIs relativas a los casos de uso CdU3, CdU4, CdU5, CdU6 y CdU14. ▪ **PS. 3.6.** Plug-in de Eclipse (MARBLE Tool) con la opción de crear un proyecto personalizado, desde cero con su respectiva perspectiva MARBLE.

Tabla 4.4. Resumen Iteración 3

4.2.4. Iteración 4

En esta iteración, como en la iteración anterior (Iteración 3), se realiza de forma paralela el análisis y diseño de los siguientes casos de uso (Integrar modelos de código en repositorios KDM y Editar modelo KDM), atendiendo a la priorización realizada (ver Tabla 5.16 con la priorización de los casos de uso), junto a la implementación del caso de uso estudiado en la anterior iteración. Esta implementación permite a la herramienta poder seleccionar los LIS de los cuales se quieren obtener los procesos de negocio y también permite generar el modelo de código fuente a partir de ese LIS seleccionado. La Tabla 4.5 muestra de manera resumida la iteración 4.

4.2.5. Iteración 5

En esta iteración, como se viene haciendo en las 2 iteraciones anteriores, se continúa analizando y diseñando nuevos casos de uso atendiendo a la priorización realizada (ver Tabla 5.16 con la priorización de los casos de uso) mientras se implementan los casos de uso estudiados en la iteración anterior. Tras esta iteración, la herramienta dispone de la implementación de un parser de Java que transforma el modelo de código de la iteración anterior en modelo de código KDM y su

correspondiente opción en el Plug-in de Eclipse para llevar a cabo dicha transformación. La Tabla 4.6 muestra de manera resumida la iteración 5.

Iteración 4	
Fase en la que se enmarca:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación
Objetivos	Productos de Salida
▪ **OB. 4.1.** Análisis y Diseño de los casos de uso: o CdU7. Integrar modelos de código en repositorios KDM o CdU8. Editar modelo KDM ▪ **OB. 4.2.** Implementación de los casos de uso CdU3, CdU4, CdU5, CdU6 y CdU14.	▪ **PS. 4.1.** Diagramas de secuencia de análisis de los casos de uso CdU7 y CdU8 ▪ **PS. 4.2.** Diagramas de comunicación de los casos de uso CdU7 y CdU8 ▪ **PS. 4.3.** Diagrama de clases de los casos de uso CdU7 y CdU8 ▪ **PS. 4.4.** Diagrama de secuencia de diseño de los casos de uso CdU7 y CdU8 ▪ **PS. 4.5.** Prototipos de la GUI relativas a los caso de uso CdU7 y CdU8. ▪ **PS. 4.6.** Plug-in de Eclipse (MARBLE Tool) con la incorporación de las opciones de "Seleccionar LIS", "Convertir un proyecto Java existente en un proyecto MARBLE", "Generar modelo de código fuente" y la opción de editar el modelo.

Tabla 4.5. Resumen Iteración 4

Iteración 5	
Fase en la que se enmarca:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación
Objetivos	Productos de Salida
▪ **OB. 5.1.** Análisis y Diseño de los casos de uso: o CdU9. Generar modelo de Procesos de Negocio a partir de modelo KDM o CdU11. Editar modelo de Procesos de negocio. ▪ **OB. 5.2.** Implementación del caso de uso CdU7 y CdU8.	▪ **PS. 5.1.** Diagramas de secuencia de análisis de los casos de uso CdU9 y CdU11 ▪ **PS. 5.2.** Diagramas de comunicación de los casos de uso CdU9 y CdU11 ▪ **PS. 5.3.** Diagrama de clases de los casos de uso CdU9 y CdU11 ▪ **PS. 5.4.** Diagrama de secuencia de diseño de los casos de uso CdU9 y CdU11 ▪ **PS. 5.5.** Prototipos de las GUIs relativas a los casos de uso CdU9 y CdU11. ▪ **PS. 5.6.** Parser de Java para la conversión a modelo KDM. ▪ **PS. 5.7.** Plug-in de Eclipse (MARBLE Tool) con la incorporación de la opción para "Integrar modelos de código en repositorios KDM" y "Editar modelo KDM".

Tabla 4.6. Resumen Iteración 5

4.2.6. Iteración 6

En esta sexta iteración se aborda el análisis y diseño del caso de uso Visualizar diagramas de procesos de negocio. Este caso de uso conlleva una gran envergadura ya que es necesario crear un metamodelo bajo el marco de trabajo EMF/GMF. Este marco de trabajo propone una metodología con la cual obtener un editor integrable en el entorno de desarrollo Eclipse (véase Figura 3.14). En esta iteración se desarrollan los pasos relativos al análisis y diseño de esa metodología. De forma paralela se lleva a cabo la implementación de los patrones de negocio QVT necesarios para realizar la última transformación y su correspondiente opción en el Plug-in de Eclipse. Esta es, por tanto, la iteración más densa del proyecto. La Tabla 4.7 muestra de manera resumida la iteración 6.

Iteración 6	
Fase en la que se enmarca:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación
Objetivos	Productos de Salida
▪ **OB. 6.1**. Análisis y Diseño de los casos de uso: o CdU10. Visualizar diagramas de procesos de negocio ▪ **OB. 6.2.** Implementación de los casos de uso CdU9 y CdU11.	▪ **PS. 6.1.** Diagrama de secuencia de análisis del caso de uso CdU10 ▪ **PS. 6.2.** Prototipo de la GUI relativa al caso de uso CdU10. ▪ **PS. 6.3.** Desarrollo del modelo de dominio (.ecore). ▪ **PS. 6.4.** Desarrollo del modelo de definición gráfica (.gmfgraph) ▪ **PS. 6.5.** Desarrollo del modelo de definición de la herramienta (.gmftool) ▪ **PS. 6.6.** Implementación de patrones de negocio en QVT. ▪ **PS. 6.7.** Plug-in de Eclipse (MARBLE Tool) con la incorporación de la opción para “Generar modelo de Procesos de Negocio a partir de modelo KDM”, con la opción de modificarlo.

Tabla 4.7. Resumen Iteración 6

4.2.7. Iteración 7

En esta iteración se continúa con la metodología propuesta por EMF/GMF siguiendo los pasos relativos a la implementación, se obtiene el modelo y el editor generado y se integra después con la herramienta MARBLE Tool. En esta iteración, de forma concurrente, se realiza también el análisis y diseño de los últimos casos de uso de la herramienta, atendiendo a la priorización realizada (ver Tabla 5.16 con la priorización

de los casos de uso). En esta iteración también se comienza a realizar las pruebas unitarias de todos los casos de uso implementados hasta el momento. La Tabla 4.8 muestra de manera resumida la iteración 7.

Iteración 7	
Fase en la que se enmarca:	Construcción
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación, Pruebas
Objetivos	Productos de Salida
▪ **OB. 7.1.** Análisis y Diseño de los casos de uso: o CdU12. Borrar proyecto MARBLE o CdU13. Configurar transformaciones ▪ **OB. 7.2.** Implementación del caso de uso CdU10. ▪ **OB. 7.3.** Pruebas unitarias de CdU1, CdU2, CdU3, CdU4, CdU5, CdU6 y CdU14.	▪ **PS. 7.1.** Diagramas de secuencia de análisis de los casos de uso CdU12 y CdU13 ▪ **PS. 7.2.** Diagramas de comunicación de los casos de uso CdU12 y CdU13 ▪ **PS. 7.3.** Diagrama de clases de los casos de uso CdU12 y CdU13 ▪ **PS. 7.4.** Diagrama de secuencia de diseño de los casos de uso CdU12 y CdU13 ▪ **PS. 7.5.** Prototipo de las GUIs relativas a los casos de uso CdU12 y CdU13. ▪ **PS. 7.6.** Generación de código del modelo EMF (.genmodel) ▪ **PS. 7.7.** Especificación de la correspondencia (.gmfmap) ▪ **PS. 7.8.** Generación del modelo generador (.gmfgen) ▪ **PS. 7.9.** Plug-in con el editor EMF/GMF (GMF-BPMN) para visualizar y editar diagramas de procesos de negocio, que será integrado con el anterior plug-in (MARBLE Tool) ▪ **PS. 7.10.** Resultados de las pruebas unitarias.

Tabla 4.8. Resumen Iteración 7

4.2.8. Iteración 8

En esta iteración se realiza la implementación de los últimos casos de uso de forma que quede terminada por completo la implementación de la herramienta. Adicionalmente, se continúa con la realización de las pruebas unitarias de los casos de uso restantes. La Tabla 4.9 muestra de manera resumida la iteración 8.

Iteración 8	
Fase en la que se enmarca:	Construcción
Flujos de trabajo que se realizan:	Diseño, Implementación, Pruebas
Objetivos	Productos de Salida
▪ **OB. 8.1.** Implementación de los casos de uso CdU12 y CdU13. ▪ **OB. 8.2.** Pruebas unitarias de CdU7 CdU8, CdU9, CdU10 y CdU11.	▪ **PS. 8.1.** Incorporación de la opción en el plug-in MARBLE Tool para eliminar un proyecto. ▪ **PS. 8.2.** Incorporación de la opción en el plug-in MARBLE Tool para "Configurar las transformaciones" ▪ **PS. 8.3.** Resultados de las pruebas unitarias

Tabla 4.9. Resumen Iteración 8

4.2.9. Iteración 9

Una vez terminada la implementación completa del plug-in de Eclipse perteneciente a la herramienta MARBLE Tool se procede a terminar de realizar las últimas pruebas unitarias y a crear las distintas pruebas de integración y aceptación. La Tabla 4.10 muestra de manera resumida la iteración 9.

Iteración 9	
Fase en la que se enmarca:	Construcción
Flujos de trabajo que se realizan:	Diseño, Implementación, Pruebas
Objetivos	Productos de Salida
▪ **OB. 9.1.** Pruebas unitarias de CdU12 y CdU13. ▪ **OB. 9.2.** Creación de pruebas de integración. ▪ **OB. 9.3.** Creación de pruebas de aceptación	▪ **PS. 9.1.** Resultados de las pruebas unitarias. ▪ **PS. 9.2.** Resultados de las pruebas de integración. ▪ **PS. 9.3.** Resultados de las pruebas de aceptación.

Tabla 4.10. Resumen Iteración 9

4.2.10. Iteración 10

Esta iteración pone fin al proceso de desarrollo de la herramienta propuesta. En esta iteración se prepara el producto final que ha sido realizado durante todo el proceso. También se confecciona la documentación de la herramienta junto a un manual de usuario y los resultados de los casos de estudio. La Tabla 4.11 muestra de manera resumida la iteración 10.

Iteración 10	
Fase en la que se enmarca:	Transición
Flujos de trabajo que se realizan:	Pruebas
Objetivos	Productos de Salida
▪ **OB. 10.1.** Realización de las últimas pruebas a la herramienta. ▪ **OB. 10.2.** Realización de la documentación de la herramienta. ▪ **OB. 10.3.** Realización del manual de usuario de la herramienta ▪ **OB. 10.4.** Entrega y distribución de la herramienta	▪ **PS.10.1.** Memoria del Proyecto Fin de Carrera. ▪ **PS 10.2.** Manual de usuario de la herramienta. ▪ **PS. 10.3.** Plug-in resultante. ▪ **PS. 10.4.** Informe sobre la distribución de la herramienta.

Tabla 4.11. Resumen Iteración 10

4.2.11. Resumen iteraciones

A continuación se muestra una tabla que resume el desarrollo de cada caso de uso por cada una de las iteraciones, mostrando en qué iteración se realiza qué flujos de trabajo (véase Tabla 4.12). No obstante, hay que indicar que no se incluyen aspectos temporales (no todas las iteraciones tuvieron la misma duración).

Caso de Uso	Análisis	Diseño	Implementación	Pruebas
CdU1	Iteración 2	Iteración 2	Iteración 3	Iteración7
CdU2	Iteración 2	Iteración 2	Iteración 3	Iteración7
CdU3	Iteración 3	Iteración 3	Iteración 4	Iteración7
CdU4	Iteración 3	Iteración 3	Iteración 4	Iteración7
CdU5	Iteración 3	Iteración 3	Iteración 4	Iteración7
CdU6	Iteración 3	Iteración 3	Iteración 4	Iteración7
CdU7	Iteración 4	Iteración 4	Iteración 5	Iteración8
CdU8	Iteración 4	Iteración 4	Iteración 5	Iteración8
CdU9	Iteración 5	Iteración 5	Iteración 6	Iteración8
CdU10	Iteración 6	Iteración 6	Iteración 7	Iteración8
CdU11	Iteración 5	Iteración 5	Iteración 6	Iteración8
CdU12	Iteración 7	Iteración 7	Iteración 8	Iteración9
CdU13	Iteración 7	Iteración 7	Iteración 8	Iteración9
CdU14	Iteración 3	Iteración 3	Iteración 4	Iteración7

Tabla 4.12. Resumen de casos de uso

En el gráfico siguiente (Figura 4.4) se muestran la distribución de esfuerzos de las iteraciones del proyecto, mostrando la cantidad de análisis, diseño, implementación y pruebas que contiene, en cuanto al número de casos de uso con los que se trabaja. En el gráfico mostrado en la Figura 4.5 se puede apreciar cómo varía la cantidad de cada flujo de trabajo a lo largo de las iteraciones del proyecto.

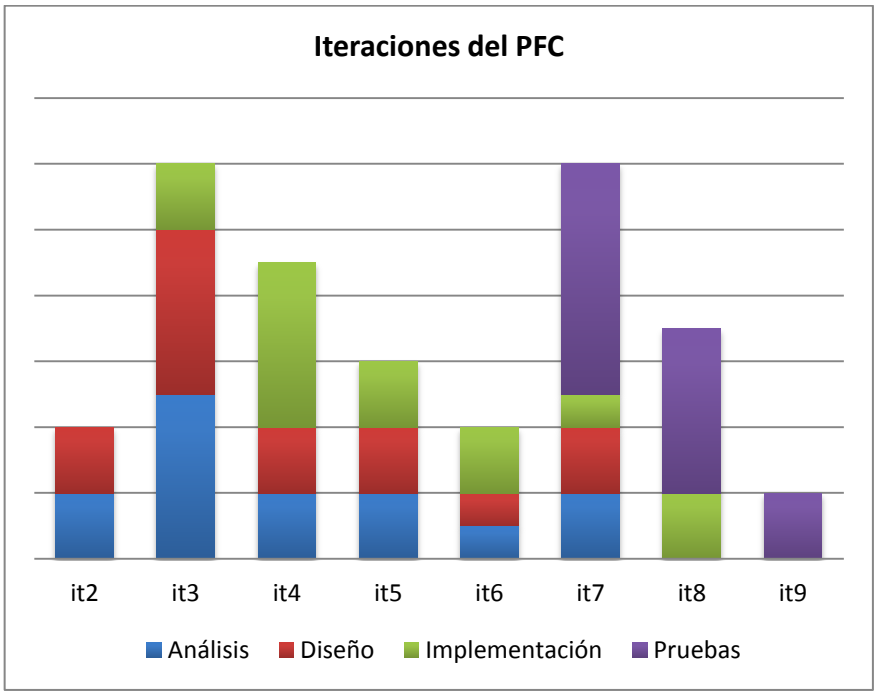


Figura 4.4. Gráfica iteraciones del Proyecto Fin de Carrera (I)

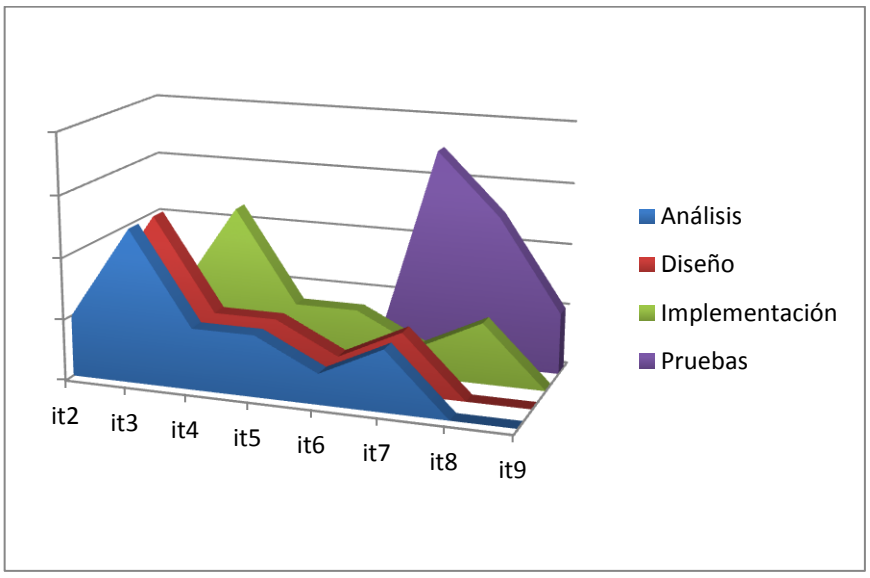


Figura 4.5. Gráfica iteraciones del Proyecto Fin de Carrera (II)

5. RESULTADOS

En este capítulo se presentan los resultados obtenidos según la planificación descrita en el capítulo 4. Las siguientes subsecciones muestran los resultados de cada iteración agrupados según el flujo de trabajo al que pertenecen.

5.1. Iteración 1

Como se indica en la Tabla 4.1, en esta iteración se han generado los productos de salida que se detallan a continuación, agrupados según el flujo de trabajo al que pertenecen.

5.1.1. Especificación de Requisitos

La especificación de requisitos es una tarea importante a la hora de llevar a cabo un proceso de desarrollo siguiendo la metodología PUD, ya que describe el comportamiento del sistema que se va a desarrollar. A continuación se muestran los requisitos funcionales y no funcionales, que corresponden al producto de salida PS.1.1 (véase Tabla 4.1).

5.1.1.1. Requisitos Funcionales

A continuación se presentan los requisitos funcionales identificados, mostrados en la Tabla 5.1, que guiarán todo el desarrollo del proceso. Esta lista se ha ido incrementando conforme ha ido avanzando la realización del proyecto. El rango del valor de “prioridad” oscilará en torno a 1 y 5, siendo 1 la prioridad más baja y 5 la prioridad más alta. Cada requisito funcional dispone de un código para su posterior trazabilidad durante el análisis, diseño e implementación.

Código	Descripción	Prioridad
RF.01	Permitir la creación de una estructura que albergue los modelos que se van a generar, siguiendo el enfoque de MARBLE, a partir del sistema de información heredado.	4
RF.02	Permitir que la información generada sea persistente, es decir, que se aloje en algún mecanismo para su posterior recuperación.	5
RF.03	Permitir incluir fácilmente los sistemas de información heredados en la estructura creada.	3
RF.04	Permitir el análisis de código fuente de un sistema de información existente a fin de reconocer ciertos elementos y estructuras del código que serán representadas en un modelo específico de plataforma (PSM) mediante las transformaciones propuestas por MARBLE (transformación entre el nivel 0 y nivel 1).	4
RF.05	Permitir la integración del modelo de código en un repositorio KDM siguiendo el estándar ISO/IEC 19506 mediante las transformaciones propuestas por MARBLE (transformación entre nivel 1 y nivel 2).	4
RF.06	Diseñar patrones de reconocimiento de las estructuras de código que permitan el descubrimiento de actividades de negocio.	4
RF.07	Implementación declarativa de los patrones de reconocimiento mediante transformaciones de modelos a través del lenguaje QVT.	4
RF.08	Permitir descubrir los procesos de negocio mediante los patrones de reconocimiento a fin de obtener un modelo independiente de la plataforma (PIM).	5
RF.09	Permitir la representación gráfica de los procesos de negocio descubiertos a fin de que estos puedan ser usados en fases posteriores de proyectos de reingeniería y migración. Los procesos de negocio serán representados de acuerdo a la notación estándar BPMN 2.0.	3
RF.10	Permitir editar los modelos generados a fin de corregirlos o ampliarlos.	4
RF.11	Permitir la eliminación de los modelos generados, así como toda la estructura creada.	4
RF.12	Permitir generar informes sobre las estadísticas de las transformaciones realizadas.	3
RF.13	Permitir configurar las transformaciones entre modelos.	3

Tabla 5.1. Requisitos Funcionales

5.1.1.2. Requisitos No Funcionales

A continuación se describen los requisitos no funcionales que cumple el presente Proyecto Fin de Carrera.

Tecnologías Empleadas

- **RNF.01.** La herramienta será implementada como un plug-in de Eclipse.

- **RNF.02.** Se generará un parser de Java con el objetivo de analizar una secuencia de símbolos a fin de determinar su estructura gramatical con respecto a la gramática formal de Java 1.6.
- **RNF.03.** Se empleará Medini QVT como motor de transformaciones QVTr.
- **RNF.04.** Se empleará EMF/GMF para el desarrollo de los editores gráficos.

Estándares aplicados

Al ser un proyecto de ingeniería informática, se han aplicado estándares internacionales para la perfecta interoperabilidad con herramientas similares y la fácil integración en la industria del software. Los estándares utilizados, por tanto, son:

- **RNF.05.** Estándar KDM versión 1.1. (descrito en la sección 3.3.3).
- **RNF.06.** Estándar BPMM versión 2.0 (descrito en la sección 3.4.1).

Requisitos de Desarrollo

En cuanto el entorno de desarrollo utilizado para la implementación del Proyecto Fin de Carrera se ha tenido en cuenta los siguientes requisitos:

- **RNF.07.** Plataforma de desarrollo: “Eclipse Helios”.
- **RNF.08.** Lenguaje de programación. “Java 1.6”.

5.1.2. Análisis

Una vez identificados los requisitos de la herramienta a desarrollar se procede a realizar un análisis preliminar de los mismos del que se desprende una primera versión del diagrama de casos de uso. Tras la definición del diagrama de casos de uso y la descripción de todos sus elementos se procede a realizar la priorización de los mismos teniendo en cuenta las precondiciones y postcondiciones de cada caso de uso.

Tras lo anterior, se puede proceder a describir la arquitectura de la herramienta que se pretende desarrollar. Finalmente, se describe la planificación del proyecto realizada teniendo en cuenta la priorización y se realiza una estimación del trabajo a realizar.

5.1.2.1. Diagrama de Casos de Uso

Los casos de uso representan la vista funcional del sistema mostrando las relaciones entre los actores y los distintos casos de uso (funcionalidades). Por tanto, los casos de uso tienen la tarea de especificar el comportamiento y la comunicación del sistema mediante su interacción con los usuarios y sistemas externos. En la Figura 5.1 se muestra el diagrama de casos de uso confeccionado y en los subapartados siguientes se describen cada uno de sus componentes. Esta parte corresponde al producto de salida PS. 1.2 (véase Tabla 4.1).

5.1.2.1.1. Roles

En el diagrama de casos de uso se pueden apreciar tres roles diferentes, dos de ellos que heredan de un tercero. Los roles pueden verse en el margen izquierdo de la Figura 5.1 y se describen en las siguientes líneas.

- **MARBLE User (Usuario MARBLE)**
  Es el usuario que accederá al sistema y tiene la necesidad de descubrir los procesos de negocio asociados/soportados por un sistema de información heredado. Este usuario conoce el método MARBLE y tiene conocimiento sobre lo descrito en el Estado del Arte (sección 3).
- **LIS Expert (Experto en Sistemas de Información Heredados)**
  Es un usuario MARBLE que es experto en el sistema de información heredado de entrada y es conocedor del estándar KDM.
- **Business Process Expert (Experto en Procesos de Negocio)**
  Es un usuario MARBLE que es experto en procesos de negocio y conoce la notación BPMN 2.0. También tiene conocimiento sobre el estándar KDM.

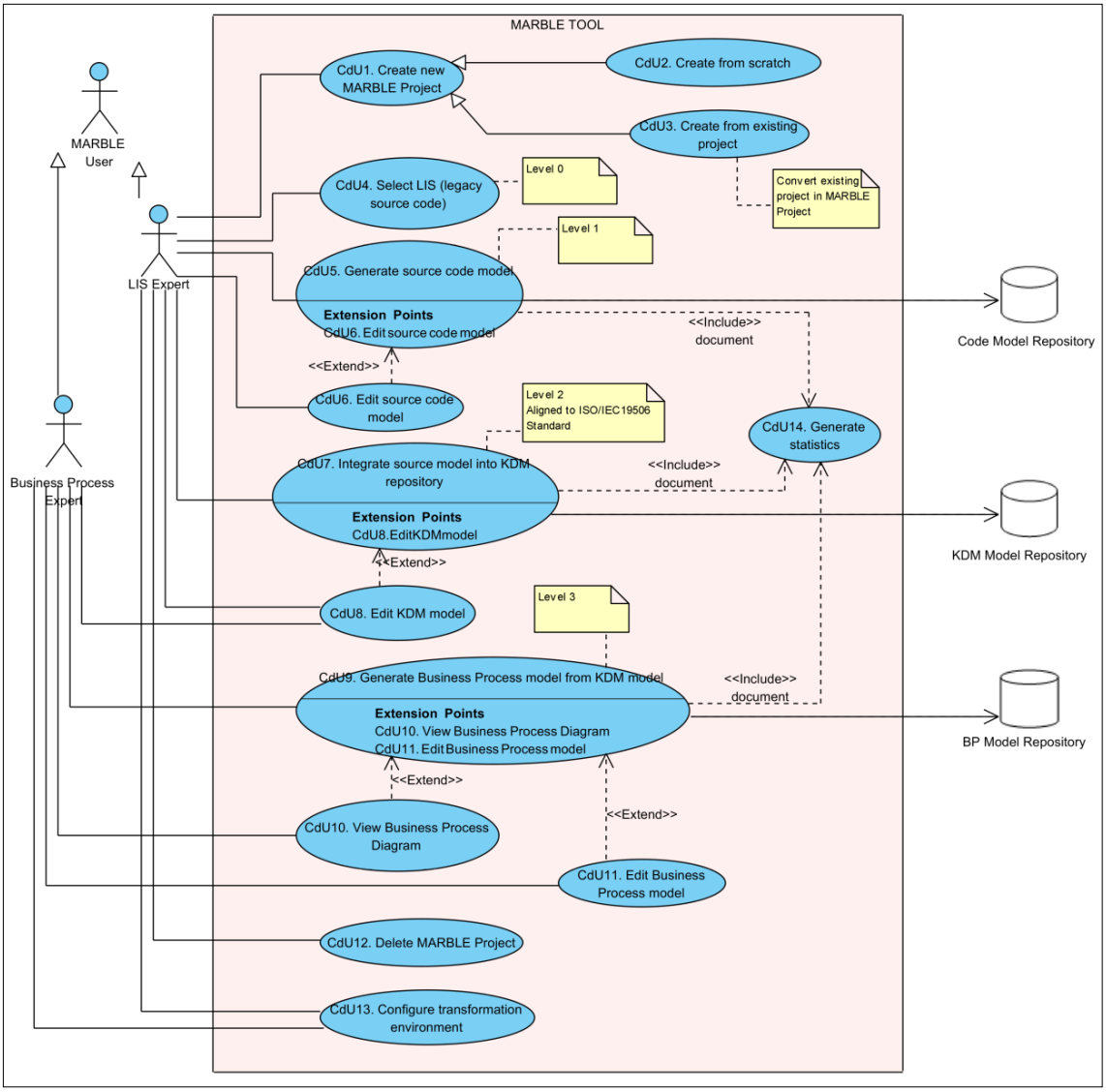


Figura 5.1. Diagrama de Casos de Uso

5.1.2.1.2. Casos de Uso

1. CdU1. Create new MARBLE Project (Crear Nuevo Proyecto MARBLE)

CdU1. Create new MARBLE Project	
Descripción	Se encarga de realizar la tarea de crear un nuevo tipo de proyecto con una estructura específica. Este tipo de proyecto dispone de cuatro carpetas que simbolizan cada uno de los niveles que propone MARBLE.
Requisito funcional asociado	RF.01 y RF.02
Actores involucrados	LIS Expert
Precondiciones	No existen precondiciones
Postcondiciones	El nuevo proyecto se guardará automáticamente en nuestro equipo

Tabla 5.2. Detalles de CdU1

2. CdU2. Create from scratch (Crear desde cero)

CdU2. Create from scratch	
Descripción	Se encarga de realizar la tarea de crear un nuevo tipo de proyecto desde cero con una estructura específica. Este tipo de proyecto dispone de cuatro carpetas que simbolizan cada uno de los niveles que propone MARBLE. Inicialmente estas carpetas estarán vacías.
Requisito funcional asociado	RF.01 y RF.02
Actores involucrados	LIS Expert
Precondiciones	No existen precondiciones
Postcondiciones	El nuevo proyecto se guardará automáticamente en nuestro equipo

Tabla 5.3. Detalles de CdU2

3. CdU3. Create from existing project (Crear desde proyecto existente)

CdU3. Create from existing project	
Descripción	Se encarga de realizar la tarea de crear un nuevo tipo de proyecto a partir de un proyecto existente con una estructura específica. Este tipo de proyecto dispone de cuatro carpetas que simbolizan cada uno de los niveles que propone MARBLE. Se inicializará una de las carpetas, la correspondiente al nivel L0, con los LIS que se encuentren en el proyecto existente.
Requisito funcional asociado	RF.01 y RF.02
Actores involucrados	LIS Expert
Precondiciones	El proyecto existente debe de ser un proyecto Java
Postcondiciones	El nuevo proyecto se guardará automáticamente en nuestro equipo. En el nivel L0 se añadirán los elementos que contenga el proyecto seleccionado

Tabla 5.4. Detalles de CdU3

4. CdU4. Select LIS (Legacy source code) (Seleccionar Sistema de Información heredado)

CdU4. Select LIS (Legacy source code)	
Descripción	Se encarga de realizar la tarea de añadir elementos al nivel L0 de la estructura del proyecto MARBLE. Se puede añadir tanto un único elemento como un directorio de elementos. Estos elementos serán código fuente Java.
Requisito funcional asociado	RF.03 y RF.02
Actores involucrados	LIS Expert
Precondiciones	Debe existir un proyecto MARBLE creado anteriormente
Postcondiciones	En el nivel L0 se añadirán los elementos seleccionados. Los cambios se guardarán automáticamente en nuestro equipo

Tabla 5.5. Detalles de CdU4

5. CdU5. Generate source code model (Generar modelo de código fuente)

CdU5. Generate source code model	
Descripción	Se encarga de realizar la tarea de transformar el código fuente Java en un modelo de código mediante el reconocimiento de la gramática Java a través de un parser.
Requisito funcional asociado	RF.04 y RF.02
Actores involucrados	LIS Expert
Precondiciones	Debe existir un proyecto MARBLE creado anteriormente que contenga elementos añadidos previamente al nivel L0
Postcondiciones	En el nivel L1 se añadirán los nuevos elementos tras la transformación. Los nuevos elementos se guardarán automáticamente en nuestro equipo. Se generará un informe con los detalles de dicha transformación.

Tabla 5.6. Detalles de CdU5

6. CdU6. Edit source code model (Editar modelo de código fuente)

CdU6. Edit source code model	
Descripción	Se encarga de editar el modelo de código a fin de mejorarlo o corregir posibles errores.
Requisito funcional asociado	RF.10 y RF.02
Actores involucrados	LIS Expert
Precondiciones	Debe existir un modelo de código en el nivel L1
Postcondiciones	Los cambios se guardarán automáticamente en nuestro equipo

Tabla 5.7. Detalles de CdU6

7. CdU7. Integrate source model into KDM repository (Integrar modelo de código en repositorio KDM)

CdU7. Integrate source model into KDM repository	
Descripción	Se encarga de realizar la tarea de integrar el modelo de código en un repositorio KDM, de tal forma que cumpla el estándar ISO/IEC 19506.
Requisito funcional asociado	RF.07 y RF.02
Actores involucrados	LIS Expert
Precondiciones	Debe existir un proyecto MARBLE creado anteriormente al que ya se le haya aplicado la anterior transformación y, por tanto, contenga elementos en el nivel L1
Postcondiciones	En el nivel L2 se añadirán los nuevos elementos tras la transformación. Los nuevos elementos se guardarán automáticamente en nuestro equipo. Se generará un informe con los detalles de dicha transformación.

Tabla 5.8. Detalles de CdU7

8. CdU8. Edit KDM model (Editar modelo KDM)

CdU8. Edit KDM model	
Descripción	Se encarga de editar el modelo KDM a fin de mejorarlo o corregir posibles errores.
Requisito funcional asociado	RF.10 y RF.02
Actores involucrados	LIS Expert y Business Process Expert
Precondiciones	Debe existir un modelo KDM en el nivel L2
Postcondiciones	Los cambios se guardarán automáticamente en nuestro equipo

Tabla 5.9. Detalles de CdU8

9. CdU9. Generate Business Process model from KDM model (Generar modelo de Procesos de Negocio a partir de modelo KDM)

CdU9. Generate Business Process model from KDM model	
Descripción	Se encarga de realizar la tarea de descubrir los procesos de negocio a partir de los modelos KDM mediante el reconocimiento de patrones de reconocimiento de estructuras de código implementadas mediante transformaciones QVT.
Requisito funcional asociado	RF.06, RF.07, RF.08 y RF.02
Actores involucrados	Business Process Expert
Precondiciones	Deben existir modelos KDM en el nivel L2 generados previamente
Postcondiciones	En el nivel L3 se añadirán los nuevos elementos tras la transformación. Los nuevos elementos se guardarán automáticamente en nuestro equipo. Se generará un informe con los detalles de dicha transformación.

Tabla 5.10. Detalles de CdU9

10. CdU10. View Business Process Diagram (Visualizar Diagrama de Procesos de Negocio)

CdU10. Business Process Diagram	
Descripción	Se encarga de representar los procesos de negocio mediante un editor gráfico, desarrollado con EMF/GMF, siguiendo la notación estándar BPMN.
Requisito funcional asociado	RF.09
Actores involucrados	Business Process Expert
Precondiciones	Debe existir un modelo de procesos de negocio en el nivel L3
Postcondiciones	En el nivel L3 se incluirá el diagrama generado. Los nuevos elementos se guardarán automáticamente en nuestro equipo

Tabla 5.11. Detalles de CdU10

11. CdU11. Edit Business Process model (Editar modelo de Procesos de Negocio)

CdU11. Edit Business Process model	
Descripción	Se encarga de editar el modelo de procesos de negocio a fin de mejorarlo, corregir posibles errores, aumentarlo, etc.
Requisito funcional asociado	RF.10 y RF.02
Actores involucrados	Business Process Expert
Precondiciones	Debe existir un modelo de procesos de negocio en el nivel L3
Postcondiciones	Los cambios se guardarán automáticamente en nuestro equipo

Tabla 5.12. Detalles de CdU11

12. CdU12. Delete MARBLE Project (Borrar proyecto MARBLE)

CdU12. Delete MARBLE Project	
Descripción	Se encarga de realizar la tarea de borrar un proyecto MARBLE.
Requisito funcional asociado	RF.11
Actores involucrados	LIS Expert
Precondiciones	Debe existir un proyecto MARBLE creado anteriormente
Postcondiciones	El proyecto será eliminado de nuestro equipo

Tabla 5.13. Detalles de CdU12

13. CdU13. Configure transformation environment (Configurar entorno de las transformaciones)

CdU13. Configure transformation environment	
Descripción	Se encarga de realizar la tarea de configurar algunos aspectos de las transformaciones entre modelos
Requisito funcional asociado	RF.13
Actores involucrados	LIS Expert y Business Process Expert
Precondiciones	No hay
Postcondiciones	La configuración seleccionada se guardará para la siguiente sesión

Tabla 5.14. Detalles de CdU13

14. CdU14. Generate statistics (Generar estadísticas)

CdU14. Generate statistics	
Descripción	Se encarga de realizar la tarea de generar informes sobre las transformaciones, aportando estadísticas sobre las mismas.
Requisito funcional asociado	RF.12
Actores involucrados	
Precondiciones	Realizar alguna de las transformaciones
Postcondiciones	Se generará un informe con los detalles de una transformación concreta.

Tabla 5.15. Detalles de CdU14

5.1.2.2. Priorización de Casos de Uso

La Tabla 5.16 muestra los casos de uso con su correspondiente priorización y la iteración en la que se comienza su desarrollo atendiendo a dicha priorización. El rango del valor de "prioridad" oscilará en torno a 1 y 5, siendo 1 la prioridad más baja y 5 la prioridad más alta. Esta priorización se ha realizado teniendo en cuenta que se deben desarrollar antes los casos de uso que pueden condicionar la arquitectura. Además, se ha tenido en cuenta que para desarrollar algunos casos de uso era necesario haber desarrollado otros con anterioridad, por lo que todo esto queda reflejado en la priorización. Esta parte corresponde al producto de salida PS. 1.3 (véase Tabla 4.1).

Caso de Uso	Prioridad	Iteración
CdU1	5	2
CdU2	5	2
CdU3	5	3
CdU4	3	3
CdU5	4	3
CdU6	2	3
CdU7	3	4
CdU8	2	4
CdU9	4	5
CdU10	3	6
CdU11	2	5
CdU12	3	7
CdU13	2	7
CdU14	3	7

Tabla 5.16. Priorización de los casos de uso

5.1.2.3. Arquitectura

La arquitectura de este proyecto está orientada a componentes, de tal forma que el desarrollo de la herramienta consiste en varios plug-ins, apoyados por una serie de librerías, que se integrarán para conseguir el objetivo final.

Estos plug-ins son: la herramienta MARBLE Tool, donde se encuentran las funcionalidades principales, y los editores. Entre los editores se encuentran el editor (gráfico) GMF-BPMN creado para la representación de los diagramas de procesos de negocio obtenidos siguiendo la notación BPMN, el editor para la visualización de modelos KDM y el editor para la visualización de modelos de código CodeEditor.

Las librerías mencionadas en las que se apoya la herramienta son: la librería JDom para la creación de archivos XML, las librerías de KDM y mediniQVT, y la librería iText para la creación de archivos PDF. Todo lo anterior puede verse reflejado en la Figura 5.2. En cuanto al diagrama de componentes, puede verse más adelante en la Figura 5.108.

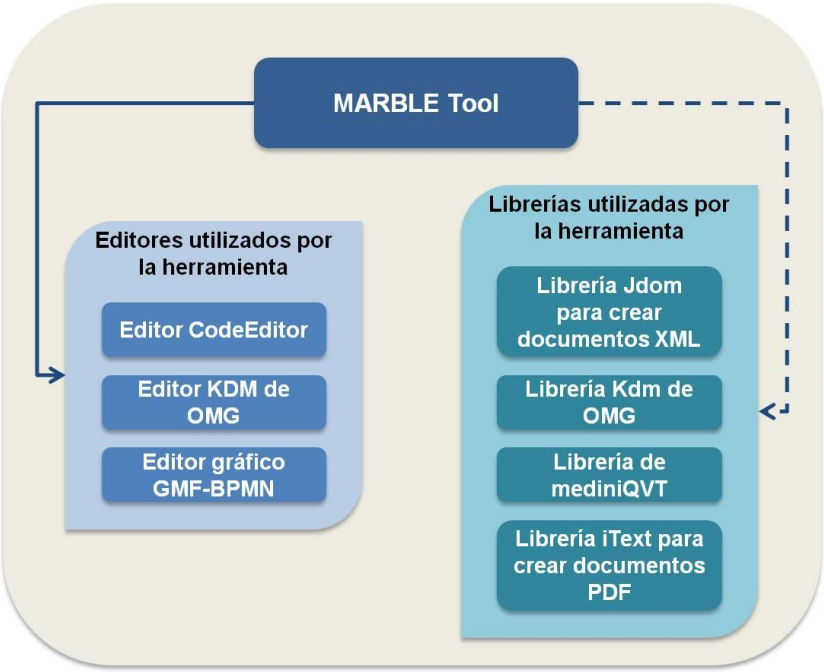


Figura 5.2. Estructura de componentes del PFC

En esta documentación se muestra el desarrollo del plug-in principal y de los editores GMF-BPMN y CodeEditor. Para el caso del editor KDM se ha utilizado el plug-in existente propuesto en (Analytics, 2010).

Esta parte corresponde al producto de salida PS. 1.4 (véase Tabla 4.1).

5.1.2.4. Planificación del Proyecto Fin de Carrera

Atendiendo a la priorización de casos de uso descrita en la Tabla 5.16 se realiza una planificación del proyecto mediante la herramienta *MS Project 2007*. Esta planificación es una estimación en cuanto a esfuerzo necesario para realizar las tareas. De forma colapsada se muestra en la Figura 5.3, donde se puede apreciar el diagrama de *Gantt* relativo a la planificación realizada.

Esta planificación corresponde al producto de salida PS. 1.5 (véase Tabla 4.1).

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto Fin de Carrera: MARBLE-Tool**	**214,5 días**	**lun 06/09/10**	**vie 01/07/11**
2	**Iteración 1**	**46 días**	**lun 06/09/10**	**lun 08/11/10**
17	**Iteración 2**	**12,25 días**	**mar 09/11/10**	**jue 25/11/10**
28	**Iteración 3**	**21 días**	**jue 25/11/10**	**vie 24/12/10**
39	**Iteración 4**	**23 días**	**vie 24/12/10**	**mié 26/01/11**
50	**Iteración 5**	**19 días**	**mié 26/01/11**	**mar 22/02/11**
61	**Iteración 6**	**24,75 días**	**mar 22/02/11**	**lun 28/03/11**
72	**Iteración 7**	**10,25 días**	**mar 29/03/11**	**mar 12/04/11**
73	**Análisis**	**1,38 días**	**mar 29/03/11**	**mié 30/03/11**
74	Diagramas de secuencia de análisis CdU12 y CdU13	5 horas	mar 29/03/11	mar 29/03/11
75	Diagramas de comunicación de CdU12 y CdU13	6 horas	mar 29/03/11	mié 30/03/11
76	**Diseño**	**7,88 días**	**mié 30/03/11**	**lun 11/04/11**
77	Diagramas de clases CdU12 y CdU13	5 días	mié 30/03/11	mié 06/04/11
78	Diagramas de secuencia de diseño CdU12 y CdU13	2 días	mié 06/04/11	vie 08/04/11
79	Prototipo de GUIs de CdU12 y CdU13	7 horas	vie 08/04/11	lun 11/04/11
80	**Implementación**	**8 días**	**mar 29/03/11**	**jue 07/04/11**
81	Generación de código del modelo EMF	2 días	mar 29/03/11	mié 30/03/11
82	Especificación de la correspondencia	4 días	jue 31/03/11	mar 05/04/11
83	Generación del modelo generador	2 días	mié 06/04/11	jue 07/04/11
84	**Pruebas**	**6 días**	**mar 29/03/11**	**mar 05/04/11**
85	Pruebas unitarias de CdU1, CdU2, CdU3, CdU4, CdU5, C	6 días	mar 29/03/11	mar 05/04/11
86	Entregables Iteración 7	1 día	lun 11/04/11	mar 12/04/11
87	**Iteración 8**	**15 días**	**mar 12/04/11**	**mar 03/05/11**
93	**Iteración 9**	**11,25 días**	**mar 03/05/11**	**mié 18/05/11**
99	**Iteración 10**	**31 días**	**mié 18/05/11**	**jue 30/06/11**
104	Finalización del Proyecto	1 día	jue 30/06/11	vie 01/07/11

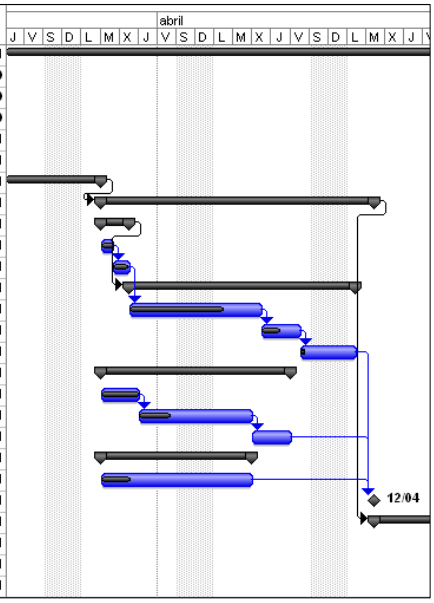


Figura 5.3. Planificación del Proyecto Fin de Carrera

5.2. Iteración 2

Como se indica en la Tabla 4.3, en esta iteración se han generado los productos de salida que se detallan a continuación agrupados según los flujos de trabajo de análisis y diseño.

5.2.1. Análisis

Para realizar el análisis de los casos de uso de la iteración 2 se ha optado por la utilización de diagramas de secuencia y diagramas de comunicación.

5.2.1.1. Diagramas de Secuencia de Análisis en la iteración 2

Para comprender en mayor medida la interacción de cada caso de uso con los roles del sistema se muestran a continuación los diagramas de secuencia para cada uno de los casos de uso contemplados en esta iteración (CdU1 y CdU2). Este tipo de diagrama permite mostrar la interacción a lo largo del tiempo de los usuarios con el sistema. Para cada uno de los casos de uso se muestran dos tipos de escenarios: escenario principal y escenario alternativo. Esta parte corresponde al producto de salida PS. 2.1 (véase Tabla 4.3).

1. **CdU1. Create new MARBLE Project**

   a) **Escenario principal**

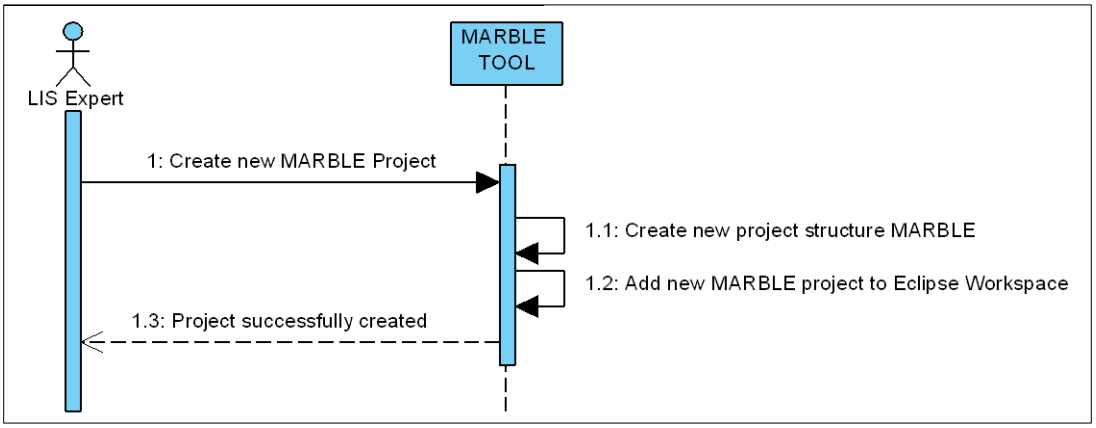


Figura 5.4. Diagrama de secuencia CdU1. Escenario Principal

   b) **Escenario alternativo 1 de error**

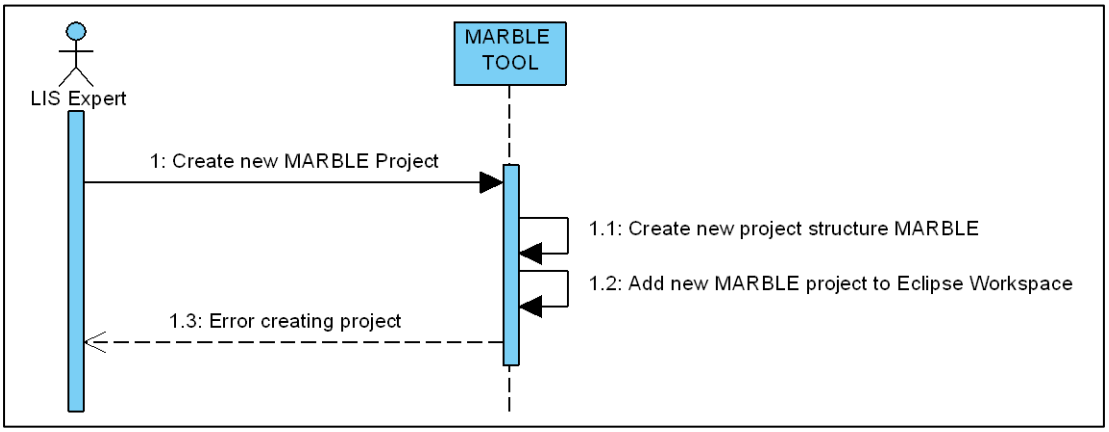


Figura 5.5. Diagrama de secuencia CdU1. Escenario de Error

2. CdU2. Create from scratch

a) Escenario principal

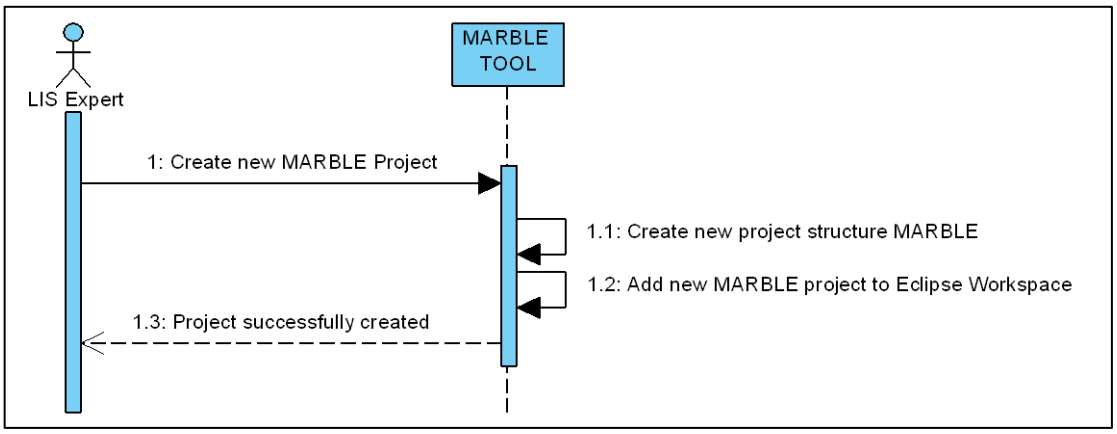


Figura 5.6. Diagrama de secuencia CdU2. Escenario Principal

b) Escenario alternativo 1 de error

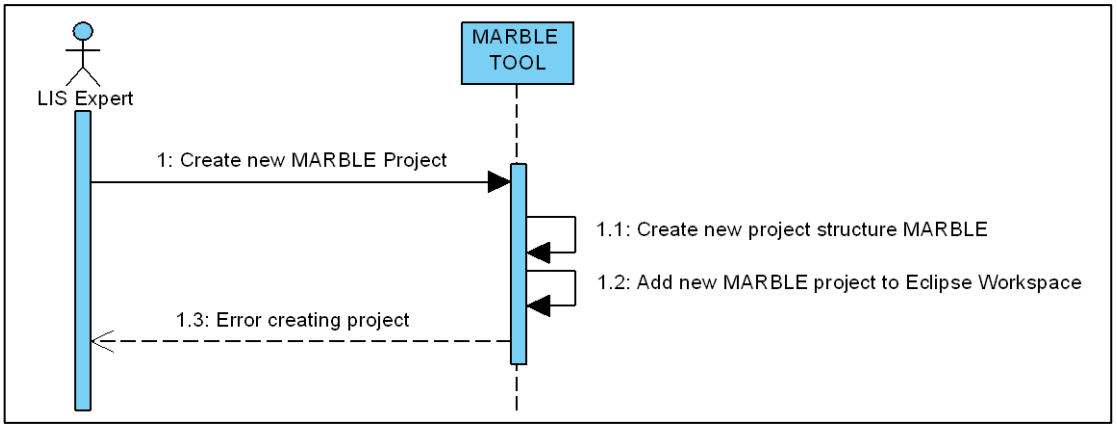


Figura 5.7. Diagrama de secuencia CdU2. Escenario de Error

5.2.1.2. Diagramas de Comunicación en la iteración 2

La Figura 5.8 muestra el diagrama realizado para el caso de uso CdU2. Este tipo de diagramas facilita al usuario la comprensión del sistema que finalmente será implementado, mediante la representación de las clases de análisis estereotipadas como *Boundaries* (que representan las clases relativas a la interfaz), *Controls* (que representan las clases relativas al control) y *Entities* (que representan las clases relativas a entidades). Esta parte corresponde al producto de salida PS. 2.2 (véase Tabla 4.3).

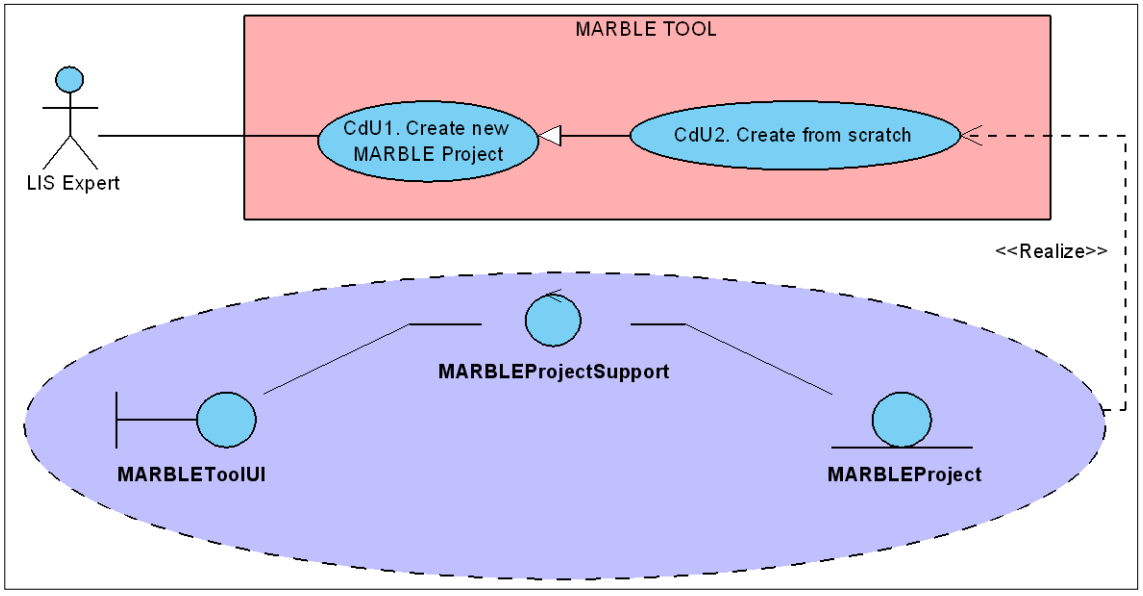


Figura 5.8. Diagrama de comunicación de CdU2

5.2.2. Diseño

En este apartado se describe el diseño realizado en la iteración 2. En él se describe el modelo arquitectónico que sigue el diseño del plug-in MARBLE Tool y el diseño de los casos de uso seleccionados en esta iteración, mediante el uso de diagramas de clases y diagramas de secuencia de diseño que muestran la interacción entre las distintas clases.

5.2.2.1. Modelo arquitectónico

A la hora de realizar el diseño de la herramienta propuesta por el Proyecto Fin de Carrera se ha optado por la aplicación de un modelo en tres capas que organiza la arquitectura en paquetes, permitiendo separar el comportamiento entre capas de tal forma que la modificación en una de las capas no influye en las restantes. Es decir, se maximizan dos de los principios más importantes del diseño de sistemas orientados a objetos: máxima cohesión y mínimo acoplamiento. Las tres capas de la arquitectura son: presentación (*presentation*), lógica de dominio (*domain*) y persistencia (*persistence*). Esta arquitectura multicapa se optimizará mediante la aplicación de patrones de diseño que se muestran en la sección 5.2.2.1.1. Esta parte corresponde al producto de salida PS. 2.3 (véase Tabla 4.3).

El diagrama de paquetes de la herramienta queda de la forma reflejada en la Figura 5.9, una vez que se ha aplicado en modelo de tres capas, y en ella se muestran las dependencias que existen entre las capas.

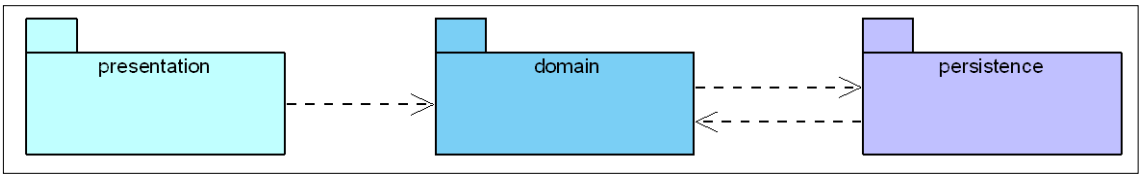


Figura 5.9. Diagrama de Paquetes representando una arquitectura multicapa.

- **Capa de presentación**: Esta clase alberga todas las clases relativas a la interfaz de usuario de la aplicación. Su función es la de recibir la información del exterior por parte del usuario y transmitirla a la capa de dominio.
- **Capa de lógica de dominio**: Esta capa alberga todas las clases específicas del dominio de negocio que se pretende modelar. Es decir, contiene todas aquellas clases que especifican la lógica de funcionamiento de la aplicación a partir de la información recibida por parte de la capa de presentación.
- **Capa de persistencia**: Esta capa alberga todas aquellas clases que permiten la gestión de los datos de forma persistente a lo largo de las diferentes ejecuciones de la herramienta. Las clases contenidas en esta capa hacen posible el cumplimiento del requisito funcional RF.2. descrito en la especificación de requisitos funcionales de la sección 5.1.1.1.

  La herramienta maneja la entidad 'Proyecto' como la unidad que almacena y gestiona toda la información relativa a la extracción de procesos de negocio desde un sistema heredado. Es esta entidad la que se almacena de forma persistente en todo momento.

5.2.2.1.1. Patrones de diseño

Los patrones son buenas soluciones a problemas conocidos y frecuentes. Dichos patrones proponen soluciones genéricas muy sencillas de adaptar a problemas de diseño concretos. Dentro de estos patrones se encuentran los denominados patrones de diseño,

que son utilizados principalmente en el flujo de trabajo de diseño del software para establecer la estructura del sistema a desarrollar (Gamma, Helm et al., 1995).

Los patrones de diseño utilizados en este Proyecto Fin de Carrera se encuentran los siguientes:

- **Patrón Fachada**: El patrón Fachada es un patrón de propósito estructural que sigue la estructura mostrada en la Figura 5.10. Este patrón permite centralizar el punto de acceso a las clases de un subsistema. Concretamente en este Proyecto Fin de Carrera se utiliza para desarrollar una fachada en toda la capa de dominio. De esta forma, la capa de presentación accede a todas las funcionalidades necesarias de la capa de dominio a fin de simplificar la lógica de acceso a la misma. Es mucho más sencillo desacoplar un subsistema de otros que lo necesiten y además se hace mucho más portable. Este patrón permite reducir considerablemente el acoplamiento entre las clases de los distintos subsistemas.

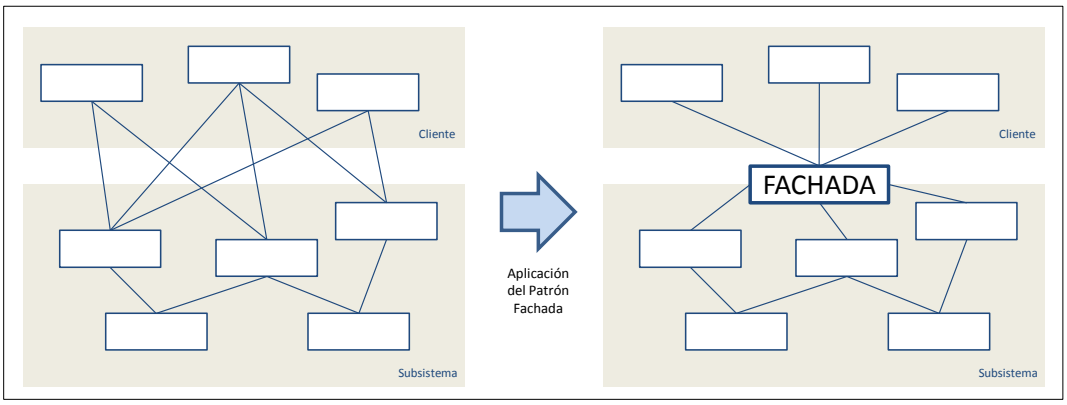


Figura 5.10. Patrón Fachada (Gamma, Helm et al., 1995)

- **Patrón Singleton**: El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. Su estructura es la mostrada en la Figura 5.11.

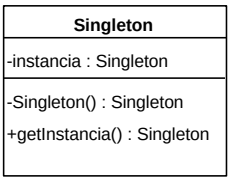


Figura 5.11. Patrón Singleton (Gamma, Helm et al., 1995)

- **Patrón Fábrica Abstracta**: El patrón de diseño fábrica abstracta se utiliza para crear familias de objetos que guardan cierta relación entre sí. Este patrón está aconsejado cuando se prevé la inclusión de nuevas familias de productos. En este Proyecto Fin de Carrera se utiliza a la hora de implementar el parser de Java del que se hace uso. En la Figura 5.12 se muestra el aspecto de este patrón.

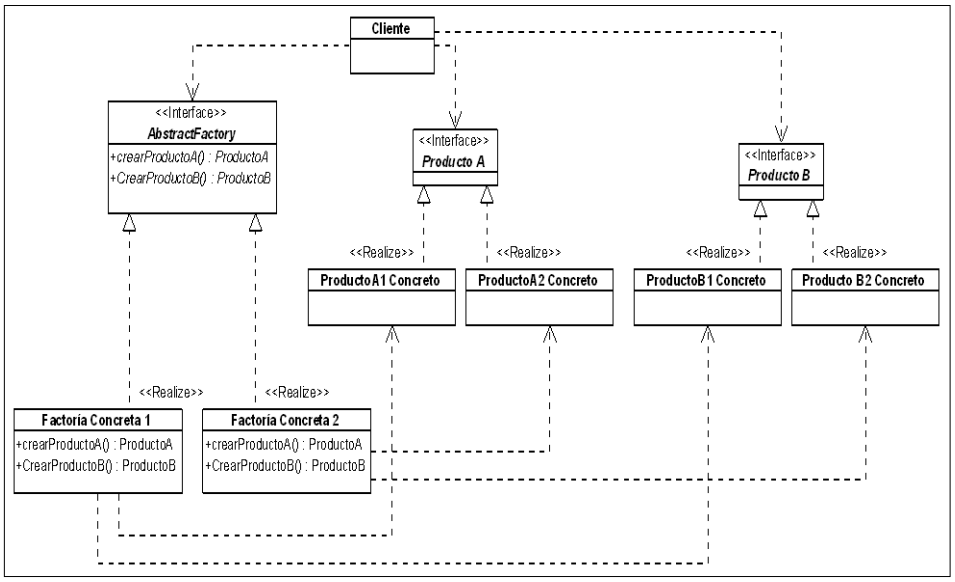


Figura 5.12. Patrón Fábrica abstracta (Gamma, Helm et al., 1995)

En los sucesivos subapartados se indicará en que clase/clases se aplican los patrones comentados.

5.2.2.2. Diagrama de Clases del sistema en la iteración 2

Una vez establecida la arquitectura de la herramienta a desarrollar se procede a realizar el diagrama de clases correspondiente al diseño de los primeros casos de uso (CdU1 y CdU2), el cual se muestra de forma colapsada en la Figura 5.13. A continuación se detallan las clases involucradas en dicho diagrama. Esta parte corresponde al producto de salida PS. 2.4 (véase Tabla 4.3).

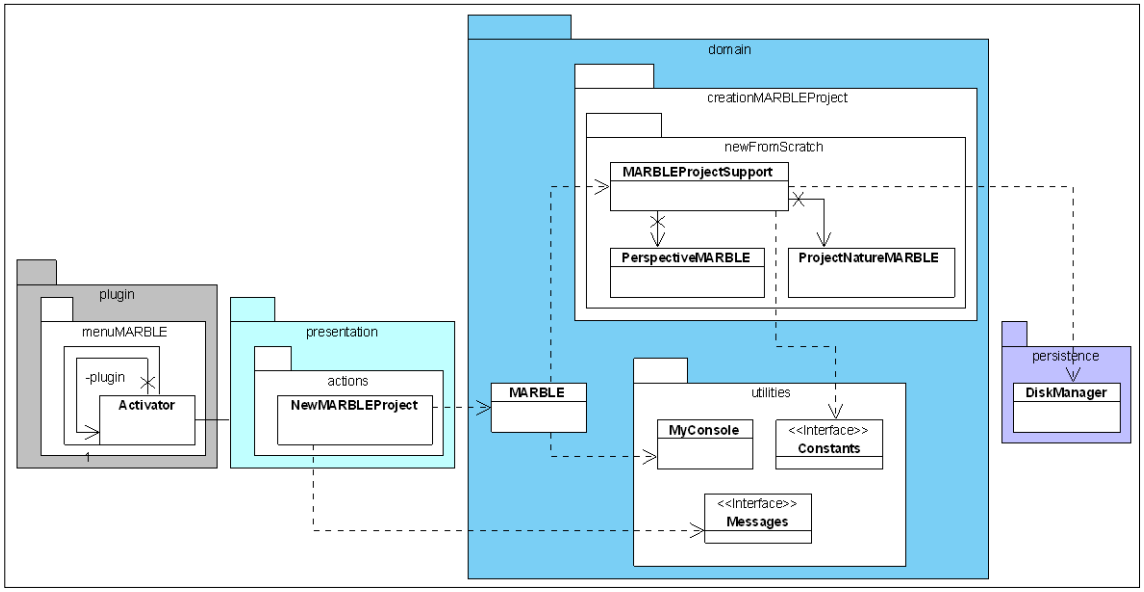


Figura 5.13. Diagrama de clases iteración 2

Activator

La Figura 5.14 muestra el diagrama UML correspondiente a la clase Activator. Esta clase es la encargada de activar el ciclo de vida del plug-in creado. Teniendo en cuenta la naturaleza y las responsabilidades de esta clase, de cara a mejorar la eficiencia del sistema, se ha decidido mejorar su diseño mediante la aplicación del patrón singleton descrito en la sección 5.2.2.1.1.

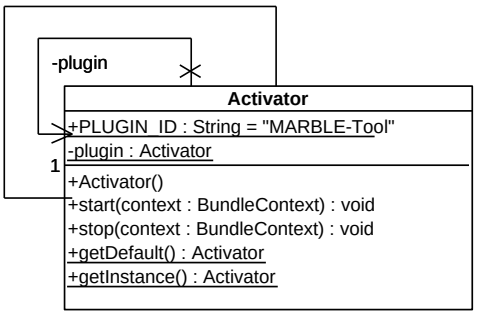


Figura 5.14. Diagrama de Clases iteración 2: Clase Activator

NewMARBLEProject

La Figura 5.15 muestra el diagrama UML correspondiente a la clase NewMARBLEProject. Esta clase está contenida en la capa de presentación, en el subconjunto de acciones. La responsabilidad de esta clase será la de proporcionar una interfaz de usuario para la creación de un nuevo proyecto MARBLE vacío.

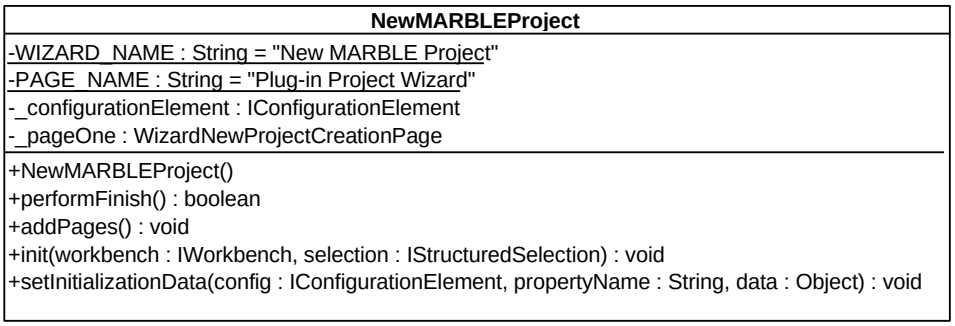


Figura 5.15. Diagrama de Clases iteración 2: Clase NewMARBLEProject

MARBLE

La Figura 5.16 muestra el diagrama UML correspondiente a la clase MARBLE. Esta clase está mejorada con el patrón Fachada (véase sección 5.2.2.1.1).

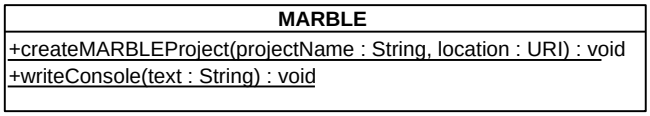


Figura 5.16. Diagrama de Clases iteración 2: Clase MARBLE

MARBLEProjectSupport

La Figura 5.17 muestra el diagrama UML correspondiente a la clase MARBLEProjectSupport. Esta clase está contenida en la capa de dominio, en el paquete de crear un nuevo proyecto (*creationMARBLEProject*) y dentro de éste en el paquete de un proyecto nuevo desde cero (*newFromScratch*). La responsabilidad de esta clase será la de proporcionar la funcionalidad para la creación de un nuevo proyecto MARBLE.

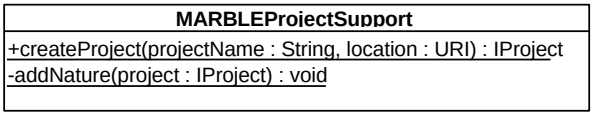


Figura 5.17. Diagrama de Clases iteración 2: Clase MARBLEProjectSupport

PerspectiveMARBLE

La Figura 5.18 muestra el diagrama UML correspondiente a la clase PerspectiveMARBLE. Esta clase está contenida en la capa de dominio, en el paquete de crear un nuevo proyecto (*creationMARBLEProject*) y dentro de éste en el paquete de un proyecto nuevo desde cero (*newFromScratch*). La responsabilidad de esta clase será la

de definir una nueva perspectiva de Eclipse relativa a un proyecto MARBLE, en la que se muestren aquellas vistas que son de interés.

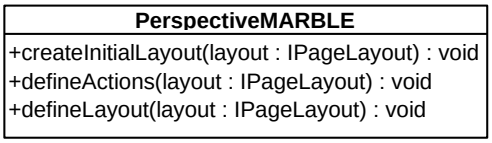


Figura 5.18. Diagrama de Clases iteración 2: Clase PerspectiveMARBLE

ProjectNatureMARBLE

La Figura 5.19 muestra el diagrama UML correspondiente a la clase ProjectNatureMARBLE. Esta clase está contenida en la capa de dominio, en el paquete de crear un nuevo proyecto (*creationMARBLEProject*) y dentro de éste en el paquete de un proyecto nuevo desde cero (*newFromScratch*). La responsabilidad de esta clase será la de definir las imágenes que se utilizarán para representar al nuevo proyecto MARBLE en el workspace de Eclipse.

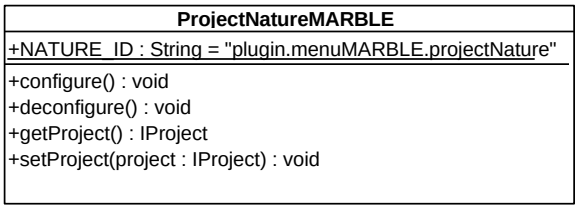


Figura 5.19. Diagrama de Clases iteración 2: Clase ProjectNatureMARBLE

MyConsole

La Figura 5.20 muestra el diagrama UML correspondiente a la clase MyConsole. Esta clase está contenida en la capa de dominio, en el paquete utilidades (*Utilities*). La responsabilidad de esta clase será la de mostrar un determinado texto en la consola del workspace de Eclipse.

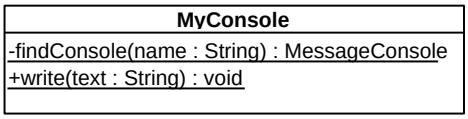


Figura 5.20. Diagrama de Clases iteración 2: Clase MyConsole

Interface Messages

La Figura 5.21 muestra el diagrama UML correspondiente a la interfaz Messages. Esta interfaz está contenida en la capa de dominio, en el subconjunto utilidades (*Utilities*). Esta interfaz será implementada por las clases de la capa de persistencia y contiene el texto a mostrar para cada una de las acciones que se van realizando. Estos textos serán los que se mostrarán por la consola.

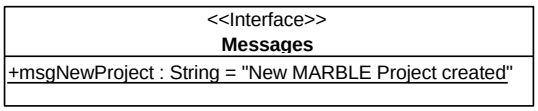


Figura 5.21. Diagrama de Clases iteración 2: Interfaz Messages

Interface Constants

La Figura 5.22 muestra el diagrama UML correspondiente a la interfaz Constants. Esta interfaz está contenida en la capa de dominio, en el subconjunto utilidades (*Utilities*). Esta interfaz será implementada por las clases de la capa de dominio y contiene las constantes que son utilizadas.

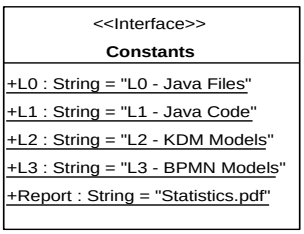


Figura 5.22. Diagrama de Clases iteración 2: Interfaz Constants

DiskManager

La Figura 5.23 muestra el diagrama UML correspondiente a la clase DiskManager. Esta clase está contenida en la capa de persistencia. Esta clase será la encargada de agregar, modificar y borrar elementos en el disco. Es decir, se encarga del manejo de aquellos datos que son persistentes. De cara a mejorar su eficiencia se ha optado por aplicar el patrón singleton descrito en la sección 5.2.2.1.1.

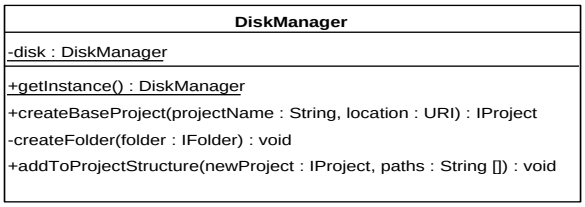


Figura 5.23. Diagrama de Clases iteración 2: Clase DiskManager

5.2.2.3. Diagramas de Secuencia de Diseño en la iteración 2

La Figura 5.24 muestra el diagrama de secuencia realizado en la iteración 2, correspondiente al caso de uso CdU2. Esta parte corresponde al producto de salida PS. 2.5 (véase Tabla 4.3).

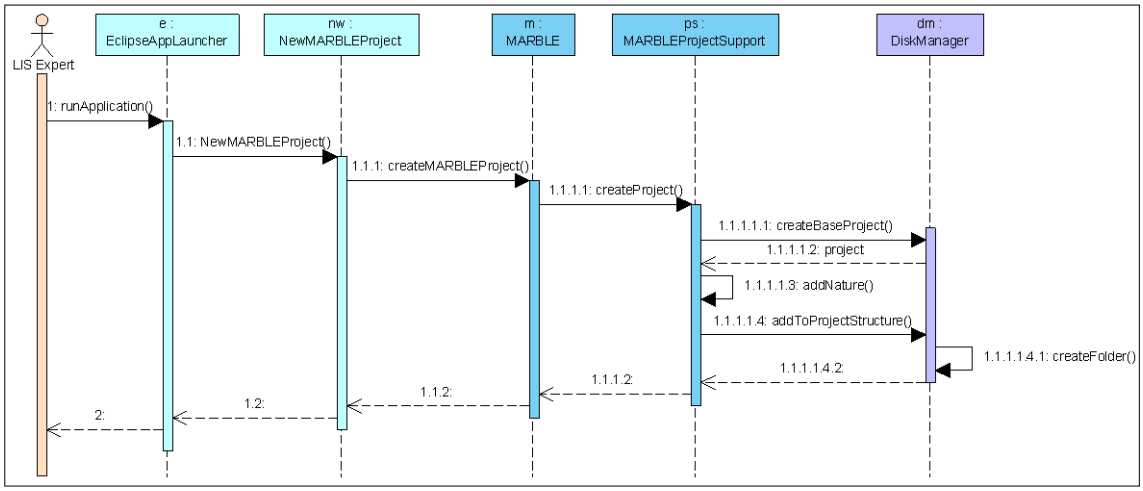


Figura 5.24. Diagrama de secuencia iteración 2 (CdU2)

5.2.2.4. Prototipos de la GUIs en la iteración 2

El diseño de la interfaz de usuario es una tarea que ha adquirido relevancia a la hora de desarrollar un sistema, ya que se trata de la información que se le muestra al usuario para que pueda interactuar con el sistema. Por ello, a continuación se muestra el prototipo de interfaz de usuario perteneciente a la funcionalidad del caso de uso CdU2 (Figura 5.25). Esta parte corresponde al producto de salida PS. 2.6. Para realizar los prototipos iniciales de las interfaces de usuario (GUIs) se ha utilizado la herramienta *Balsamiq Mockups* (Balsamiq, 2011).

La Figura 5.26 muestra el prototipo de perspectiva asociada a un proyecto MARBLE.

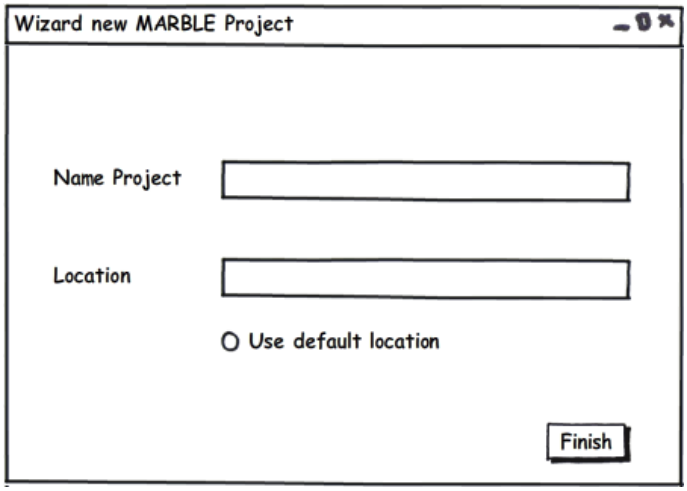


Figura 5.25. Prototipo de la GUI de CdU2

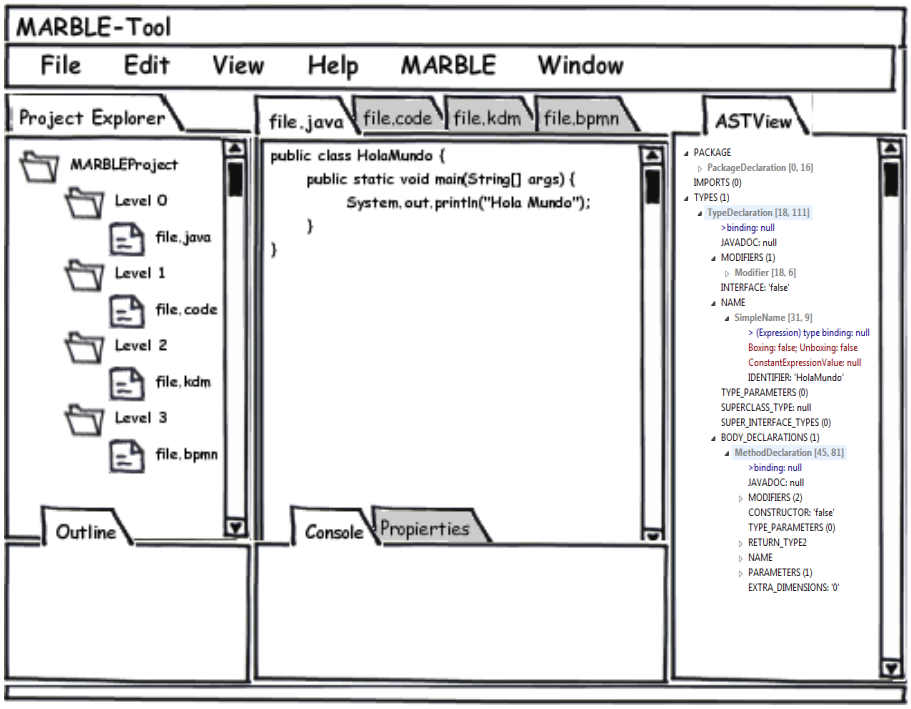


Figura 5.26. Prototipo de la perspectiva MARBLE

5.3. Iteración 3

Una vez finalizada la iteración 2 se comienza a realizar las acciones de la iteración 3. Como se indica en la Tabla 4.4, en esta iteración se han generado los productos de salida siguientes agrupados en los flujos de trabajo de análisis, diseño e implementación.

5.3.1. Análisis

Para realizar el análisis de los casos de uso de la iteración 3 se ha optado por la utilización de diagramas de secuencia y diagramas de comunicación.

5.3.1.1. Diagramas de Secuencia de Análisis en la iteración 3

Para comprender en mayor medida la interacción de cada caso de uso con los roles del sistema se muestran a continuación los diagramas de secuencia para cada uno de los casos de uso contemplados en esta iteración (CdU3, CdU4, CdU5, CdU6 y CdU14). Este tipo de diagrama permite mostrar la interacción a lo largo del tiempo de los usuarios con el sistema. Para cada uno de los casos de uso se muestran al menos dos tipos de escenarios: escenario principal y escenario alternativo.

Esta parte corresponde al producto de salida PS. 3.1 (véase Tabla 4.4).

1. **CdU3. Create from existing project**

   a) **Escenario principal**

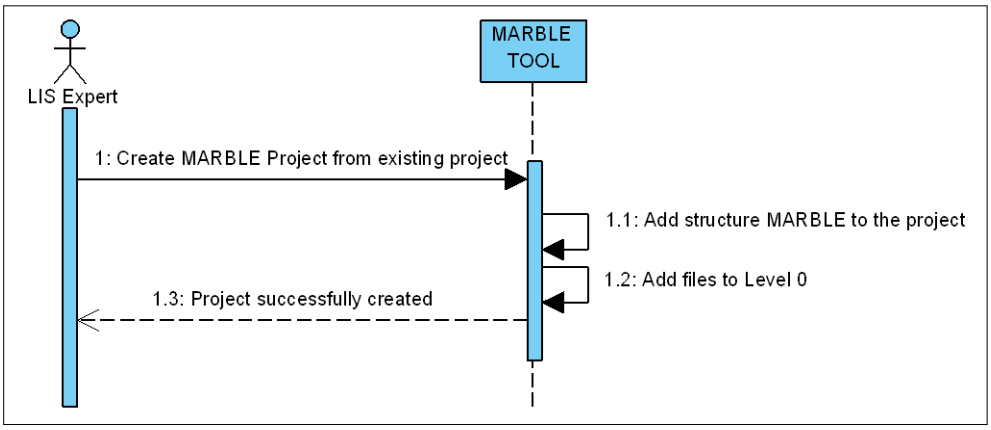


Figura 5.27. Diagrama de secuencia CdU3. Escenario Principal

b) Escenario de alternativo de error

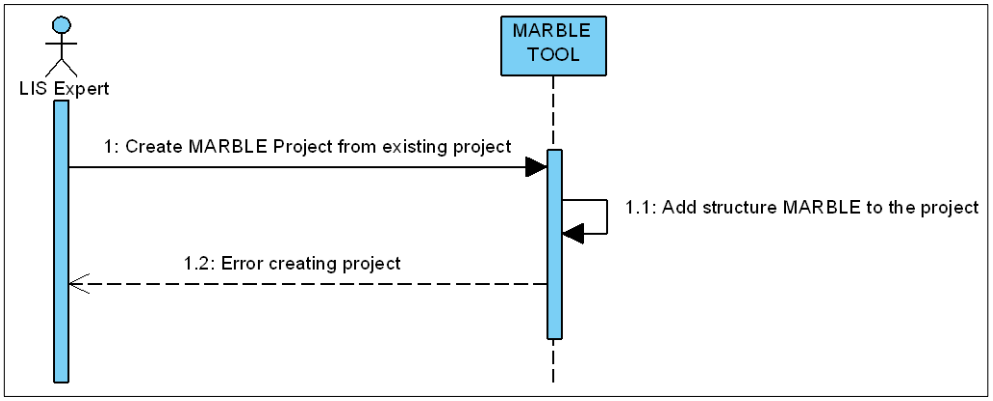


Figura 5.28. Diagrama de secuencia CdU3. Escenario de Error

2. CdU4. Select LIS (Legacy source code)

Este caso de uso se podrá hacer de dos formas: seleccionando un único LIS (caso 1) o seleccionando un directorio que contenga los LISs (caso 2).

a) Escenario principal

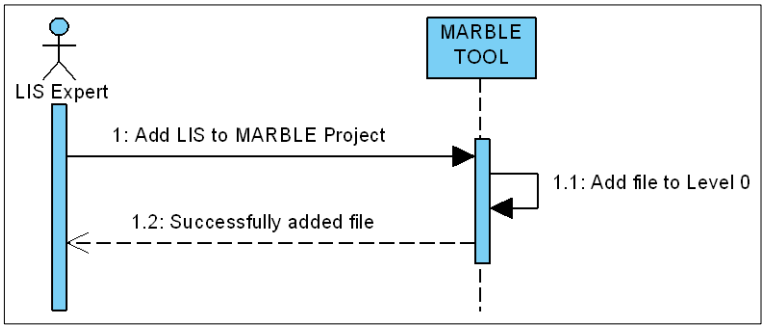


Figura 5.29. Diagrama de secuencia CdU4. Escenario Principal

b) Escenario alternativo 1

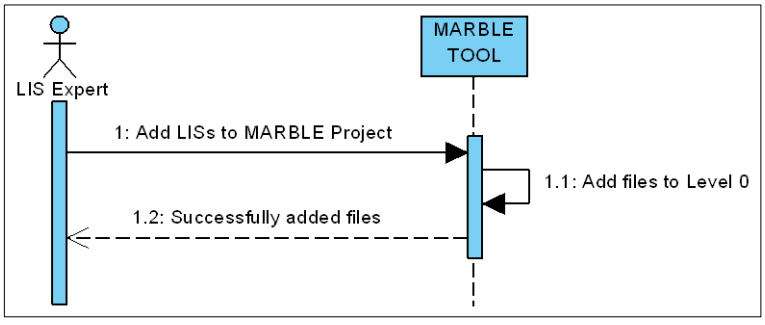


Figura 5.30. Diagrama de secuencia CdU4. Escenario Alternativo

c) Escenario alternativo 2 de error

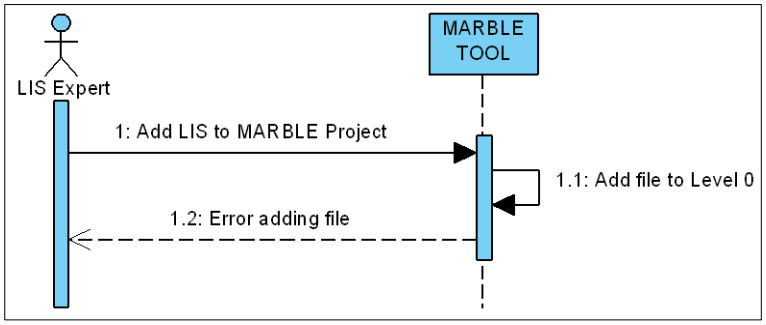


Figura 5.31. Diagrama de secuencia CdU4. Escenario de Error (I)

d) Escenario alternativo 3 de error

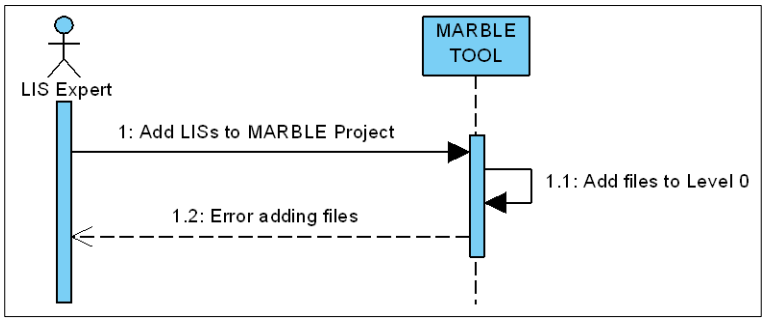


Figura 5.32. Diagrama de secuencia CdU4. Escenario de Error (II)

3. CdU5. Generate source code model

a) Escenario principal

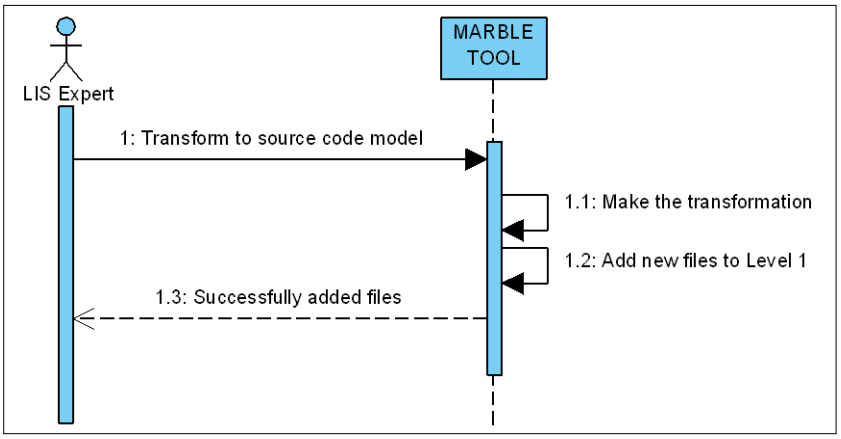


Figura 5.33. Diagrama de secuencia CdU5. Escenario de Principal

b) Escenario alternativo de error

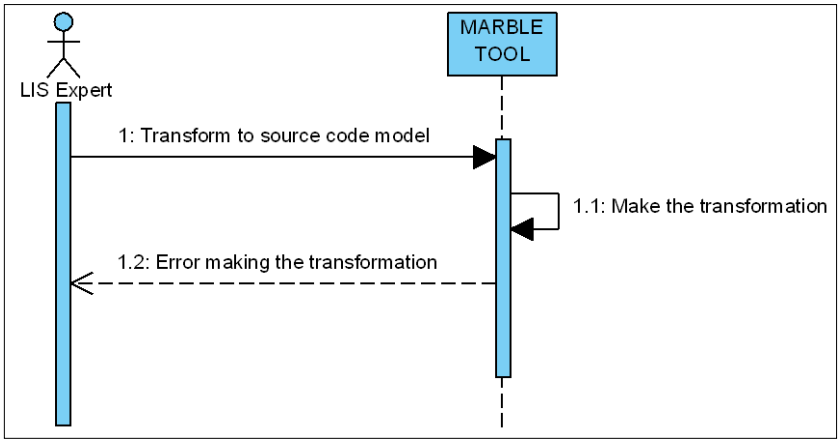


Figura 5.34. Diagrama de secuencia CdU5. Escenario de Error

4. CdU6. Edit source code model

a) Escenario principal

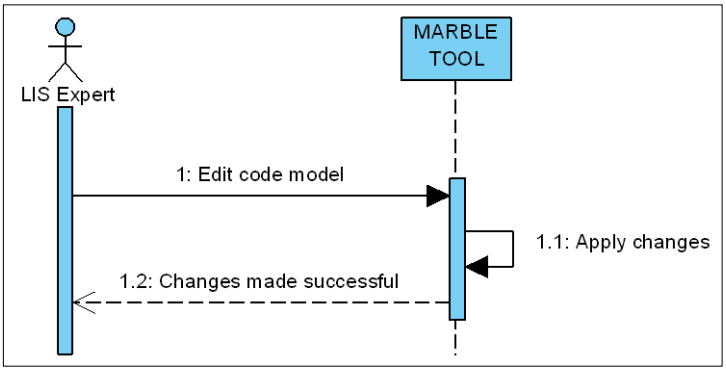


Figura 5.35. Diagrama de secuencia CdU6. Escenario de Principal

b) Escenario alternativo de error

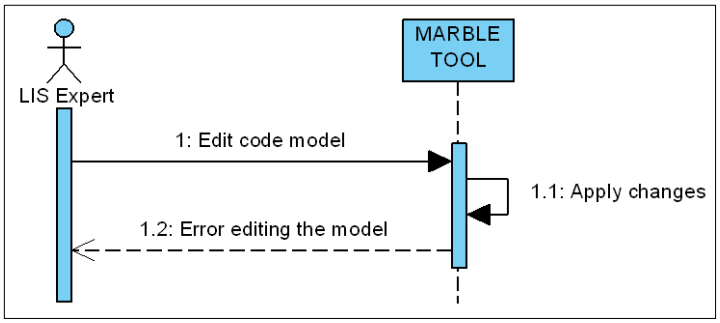


Figura 5.36. Diagrama de secuencia CdU6. Escenario de Error

5. CdU14. Generate statistics

a) Escenario principal

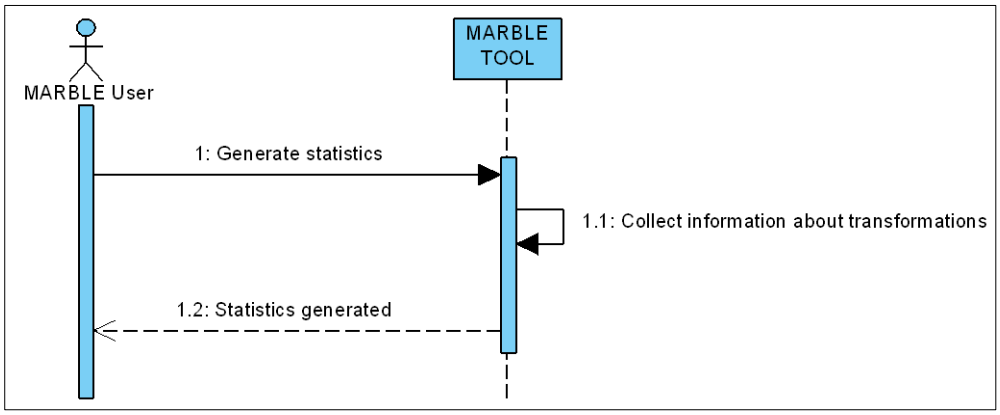


Figura 5.37. Diagrama de secuencia CdU14. Escenario Principal

b) Escenario alternativo de error

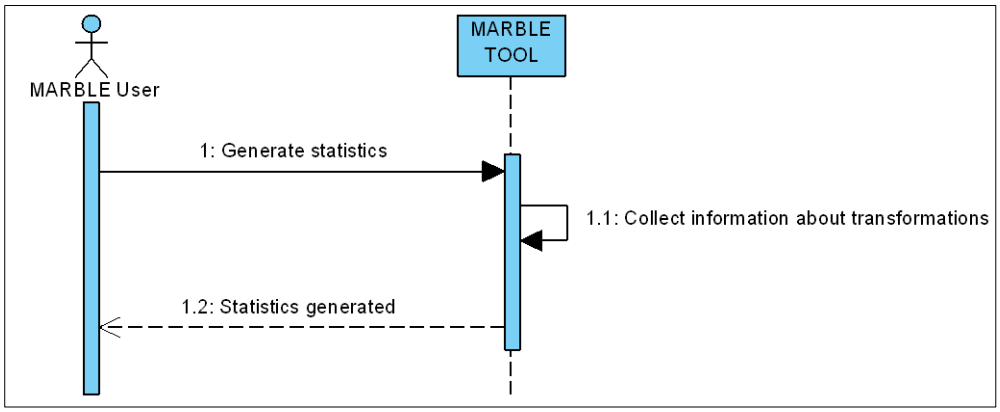


Figura 5.38. Diagrama de secuencia CdU14. Escenario de Error

5.3.1.2. Diagramas de Comunicación en la iteración 3

Lo mostrado en la Figura 5.39, en la Figura 5.40 y en la Figura 5.41 representa diagramas de comunicación realizados para los casos de uso CdU3, CdU4 y CdU5, respectivamente. Esta parte corresponde al producto de salida PS. 3.2.

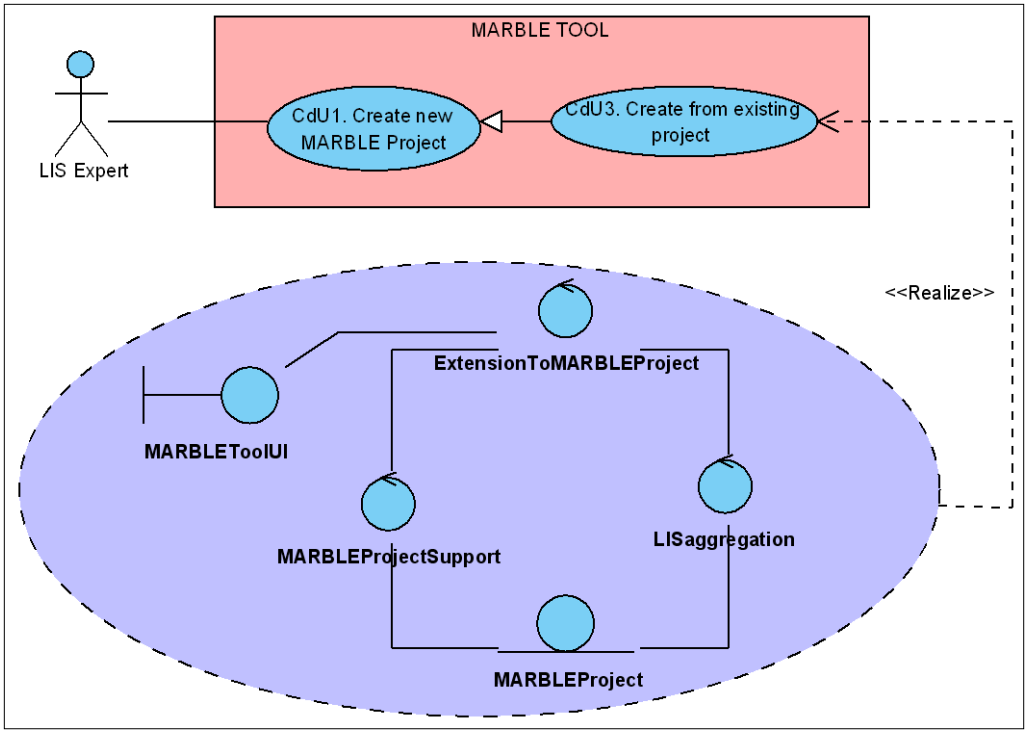


Figura 5.39. Diagrama de comunicación de CdU3

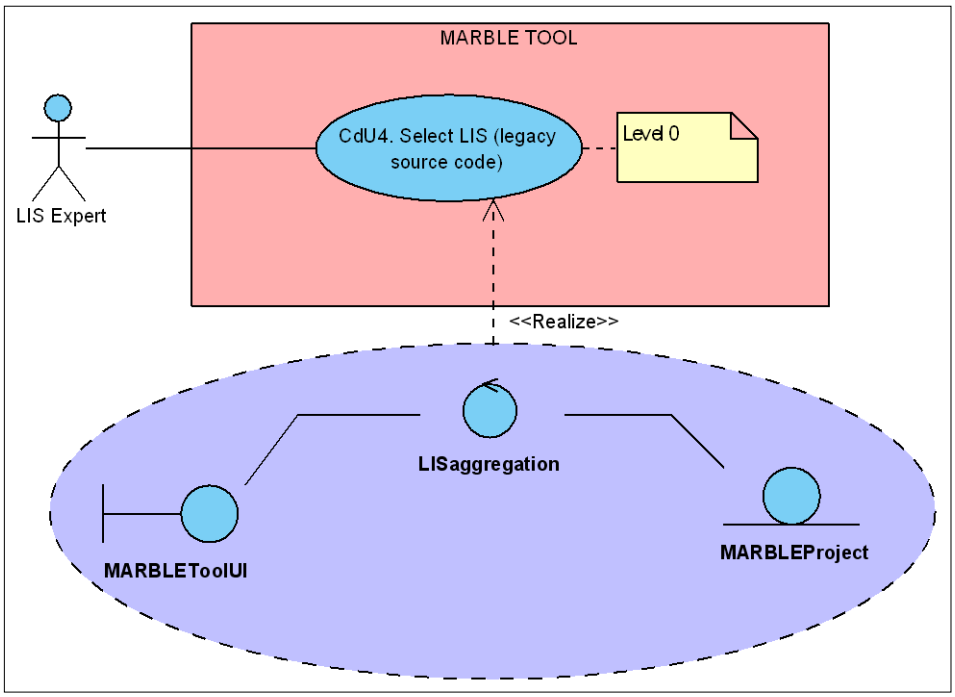


Figura 5.40. Diagrama de comunicación de CdU4

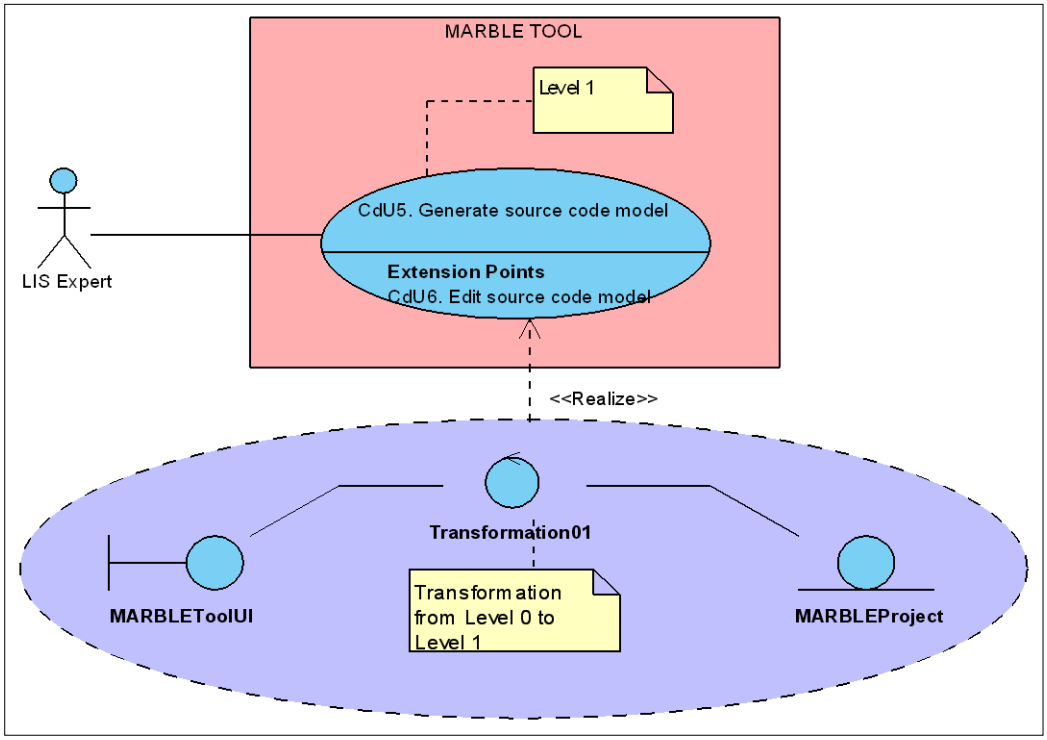


Figura 5.41. Diagrama de comunicación de CdU5

5.3.2. Diseño

Para realizar el diseño de los casos de uso seleccionados en esta iteración se van a utilizar los diagramas de clases y los diagramas de secuencia que muestren la interacción entre las distintas clases.

5.3.2.1. Diagrama de Clases del sistema en la iteración 3

El diagrama de clases correspondiente al diseño de los casos de uso CdU3, CdU4, CdU5, CdU6 y CdU14 se muestra de forma colapsada en la Figura 5.42. A continuación, se detallan las clases involucradas en el diagrama. Sólo se detallan las clases nuevas o aquellas que han sufrido cambios en esta iteración respecto a la iteración anterior.

Esta parte corresponde al producto de salida PS. 3.3 (véase Tabla 4.4).

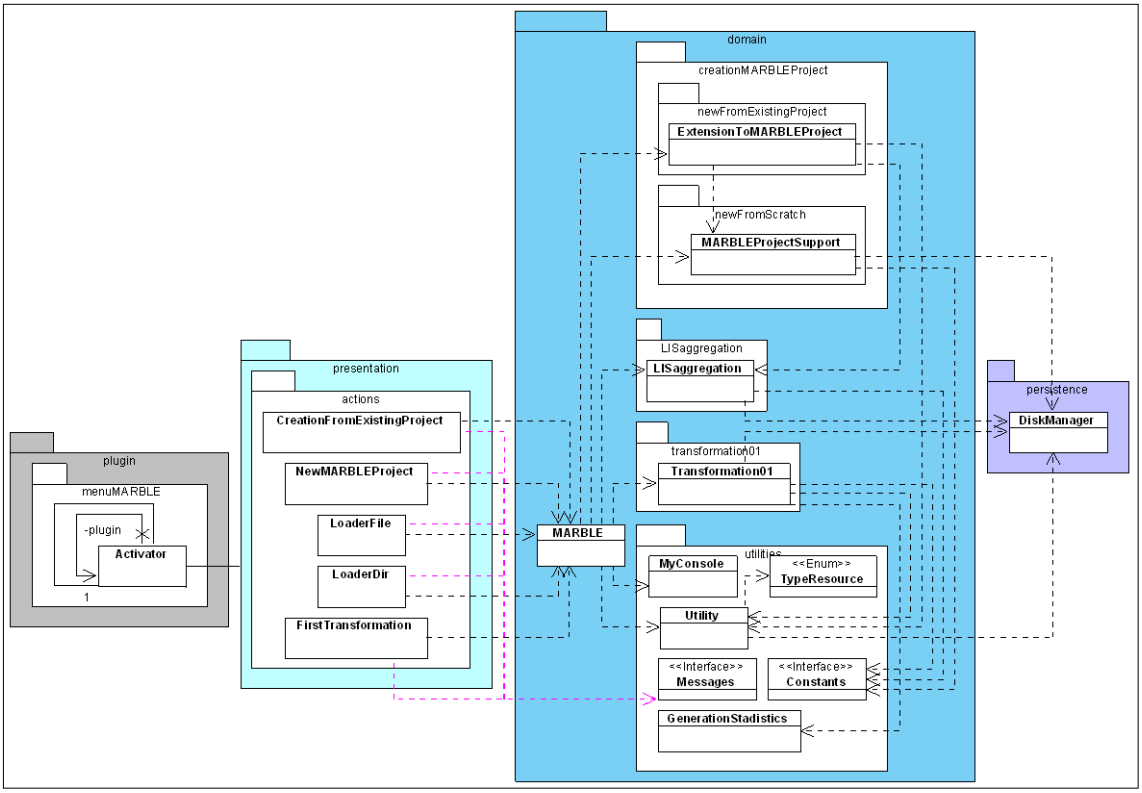


Figura 5.42. Diagrama de Clases iteración 3

CreationFromExistingProject

La Figura 5.43 muestra el diagrama UML correspondiente a la clase CreationFromExistingProject. Esta clase está contenida en la capa de presentación, en el paquete de acciones (*actions*). La responsabilidad de esta clase será la de proporcionar una interfaz de usuario para la creación de un proyecto MARBLE a partir de un proyecto Java existente.

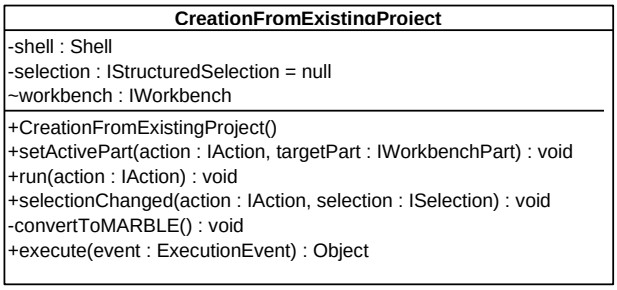


Figura 5.43. Diagrama de Clases iteración 3: Clase CreationFromExistingProject

LoaderFile

La Figura 5.44 muestra el diagrama UML correspondiente a la clase LoaderFile. Esta clase está contenida en la capa de presentación, en el paquete de acciones (*actions*). La responsabilidad de esta clase será la de proporcionar una interfaz de usuario para añadir un archivo Java correspondiente a un LIS a un proyecto MARBLE existente.

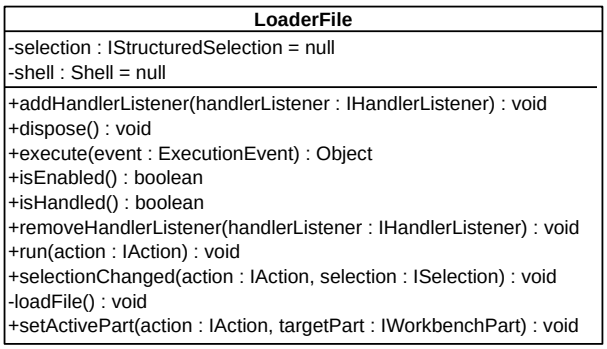


Figura 5.44. Diagrama de Clases iteración 3: Clase LoaderFile

LoaderDir

La Figura 5.45 muestra el diagrama UML correspondiente a la clase LoaderDir. Esta clase está contenida en la capa de presentación, en el paquete de acciones (*actions*). La responsabilidad de esta clase será la de proporcionar una interfaz de usuario para añadir un directorio que contiene archivos Java, correspondientes a LISs, a un proyecto MARBLE existente.

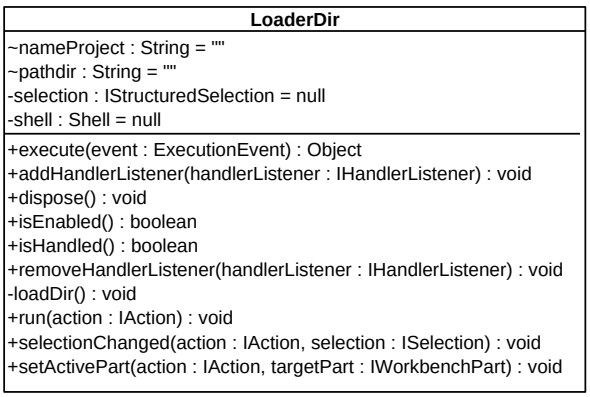


Figura 5.45. Diagrama de Clases iteración 3: Clase LoaderDir

FirstTransformation

La Figura 5.46 muestra el diagrama UML correspondiente a la clase FirstTransformation. Esta clase está contenida en la capa de presentación, en el paquete de acciones (*actions*). La responsabilidad de esta clase será la de proporcionar una interfaz de usuario para realizar la primera transformación propuesta por MARBLE a partir de unos determinados LIS(s) seleccionados de un proyecto MARBLE existente.

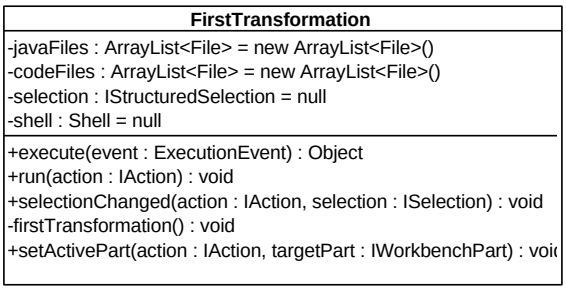


Figura 5.46. Diagrama de Clases iteración 3: Clase FirstTransformation

MARBLE

La Figura 5.47 muestra el diagrama UML correspondiente a la clase MARBLE. En esta iteración se han añadido nuevos elementos a la clase MARBLE para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

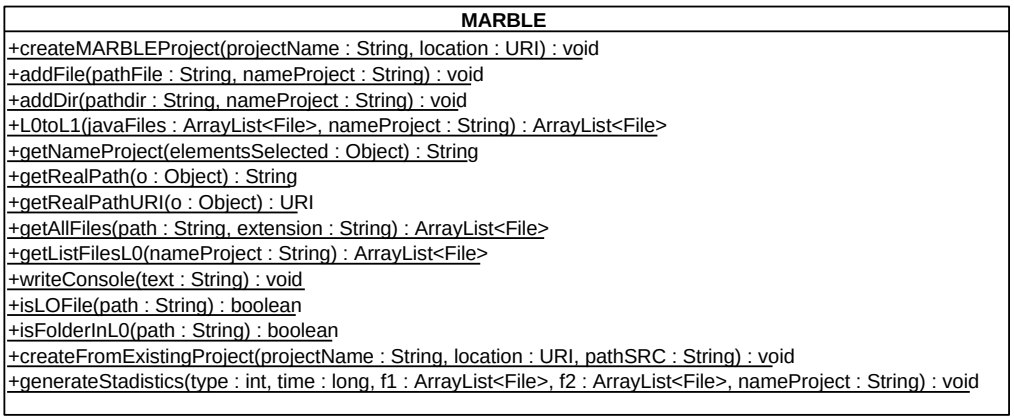


Figura 5.47. Diagrama de Clases iteración 3: Clase MARBLE

ExtensionToMARBLEProject

La Figura 5.48 muestra el diagrama UML correspondiente a la clase ExtensionToMARBLEProject. Esta clase está contenida en la capa de dominio, en el paquete de crear un nuevo proyecto (*creationMARBLEProject*) y dentro de éste en el

paquete de un proyecto nuevo desde uno existente (*newFromExistingProject*). La responsabilidad de esta clase será la de proporcionar la funcionalidad para la extensión de un proyecto Java existente a un proyecto MARBLE.

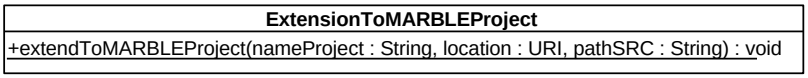


Figura 5.48. Diagrama de Clases iteración 3: Clase ExtensionToMARBLEProject

LISaggregation

La Figura 5.49 muestra el diagrama UML correspondiente a la clase LISaggregation. Esta clase está contenida en la capa de dominio, en el paquete de crear agregar LIS (*LISaggregation*). La responsabilidad de esta clase será la de proporcionar la funcionalidad para añadir LIS(s) a un proyecto MARBLE existente.

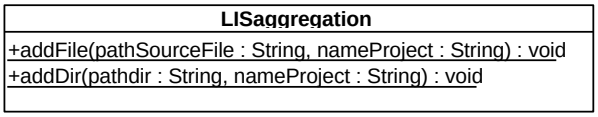


Figura 5.49. Diagrama de Clases iteración 3: Clase LISAggregation

Transformation01

La Figura 5.50 muestra el diagrama UML correspondiente a la clase Transformation01. Esta clase está contenida en la capa de dominio, en el paquete de realizar la primera transformación (*transformation01*). La responsabilidad de esta clase será la de proporcionar la funcionalidad para realizar la primera transformación propuesta por MARBLE a partir de unos determinados LIS(s) seleccionados de un proyecto MARBLE existente.

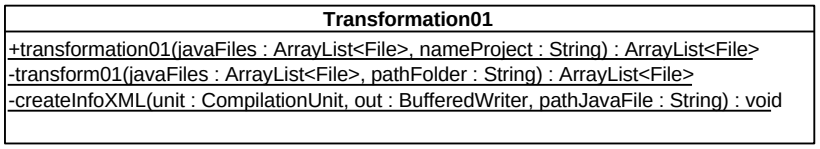


Figura 5.50. Diagrama de Clases iteración 3: Clase Tranformation01

Utility

La Figura 5.51 muestra el diagrama UML correspondiente a la clase Utility. Esta clase está contenida en la capa de dominio, en el paquete de utilidades (*utilities*). La

responsabilidad de esta clase será la de proporcionar funcionalidades auxiliares que serán utilizadas por las clases de la capa de dominio. También sirve como comunicación entre la capa de presentación y la capa de persistencia.

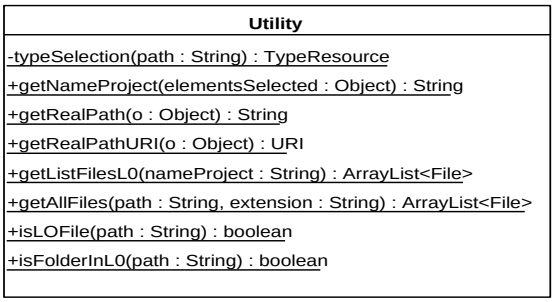


Figura 5.51. Diagrama de Clases iteración 3: Clase Utility

Interface Messages

La Figura 5.52 muestra el diagrama UML correspondiente a la interfaz Messages. Esta interfaz será implementada por las clases de la capa de persistencia y contiene el texto a mostrar para cada una de las acciones que se van realizando. Estos textos serán los que se mostrarán por la consola.

En esta iteración se han añadido nuevos elementos a la interfaz para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

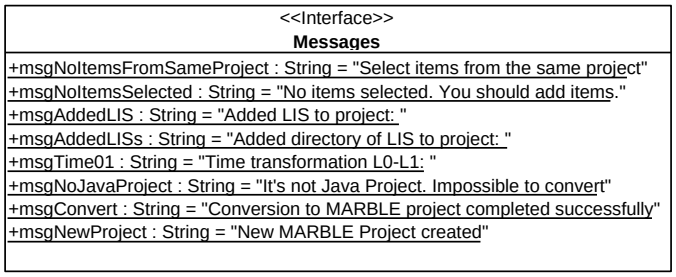


Figura 5.52. Diagrama de Clases iteración 3: Interfaz Messages

Enum TypeResource

La Figura 5.53 muestra el diagrama UML correspondiente a la enumeración TypeResource. Esta enumeración está contenida en la capa de dominio, en el paquete utilidades (*Utilities*). Esta enumeración contiene todos los tipos de archivos que puede contener un proyecto MARBLE: archivos ".java", archivos ".code", archivos ".kdm", archivos ".bpmn", carpetas, archivos con otra extensión, etc.

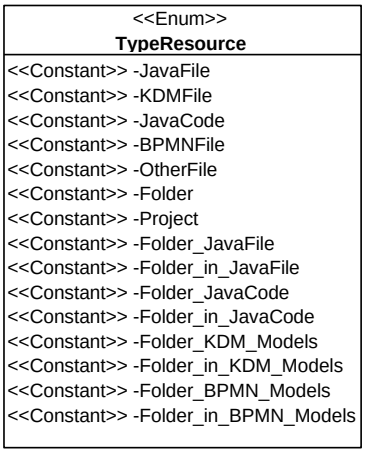


Figura 5.53. Diagrama de Clases iteración 3: Enumeración TypeResource

GenerationStatistics

La Figura 5.54 muestra el diagrama UML correspondiente a la clase GenerationStatistics. Esta clase está contenida en la capa de dominio, en el paquete de utilidades (*utilities*). La responsabilidad de esta clase será la de proporcionar funcionalidades para generar un informe de las transformaciones que son realizadas.

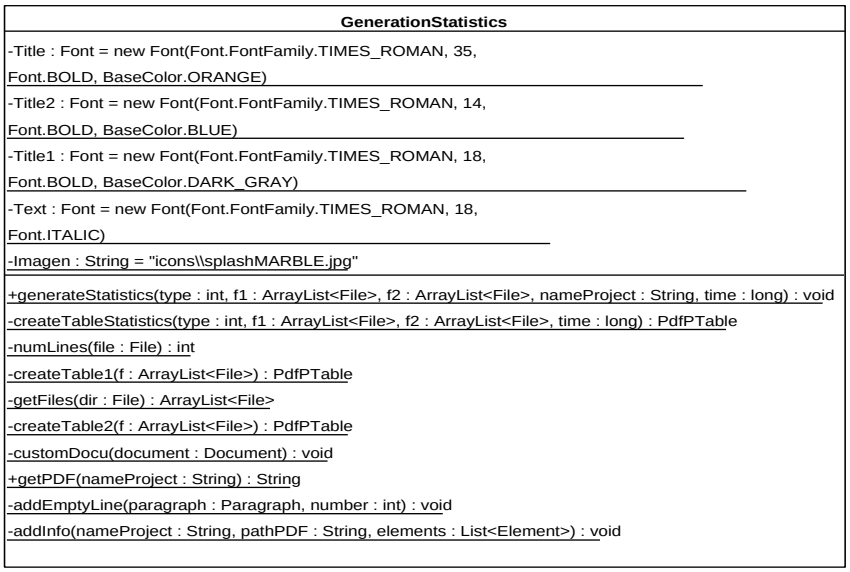


Figura 5.54. Diagrama de Clases iteración 3: Clase GenerationStatistics

DiskManager

La Figura 5.55 muestra el diagrama UML correspondiente a la clase. Esta clase es la encargada de agregar, modificar y borrar elementos en el disco, es decir, se encarga del manejo de aquellos datos que son persistentes.

En esta iteración se han añadido nuevos elementos respecto al diagrama mostrado en la Figura 5.23 con el fin de satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

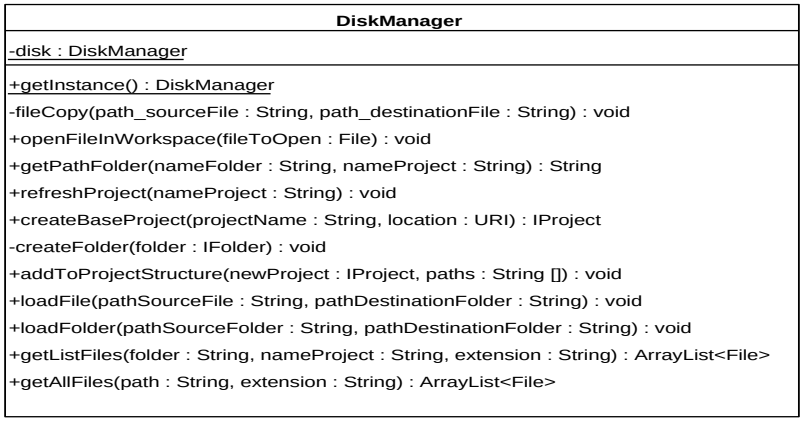


Figura 5.55. Diagrama de Clases iteración 3: Clase DiskManager

5.3.2.2. Diagramas de Secuencia de Diseño en la iteración 3

Las figuras Figura 5.56 y Figura 5.57 corresponden al diagrama de secuencia del caso de uso CdU4 para sus dos opciones: añadir un archivo y añadir un directorio.

La Figura 5.58 muestra el diagrama de secuencia correspondiente al diseño del caso de uso CdU3. Este caso de uso hace uso de las funcionalidades de los caso de uso CdU2 y CdU4. Figura 5.59 muestra el diagrama de secuencia del caso de uso CdU5.

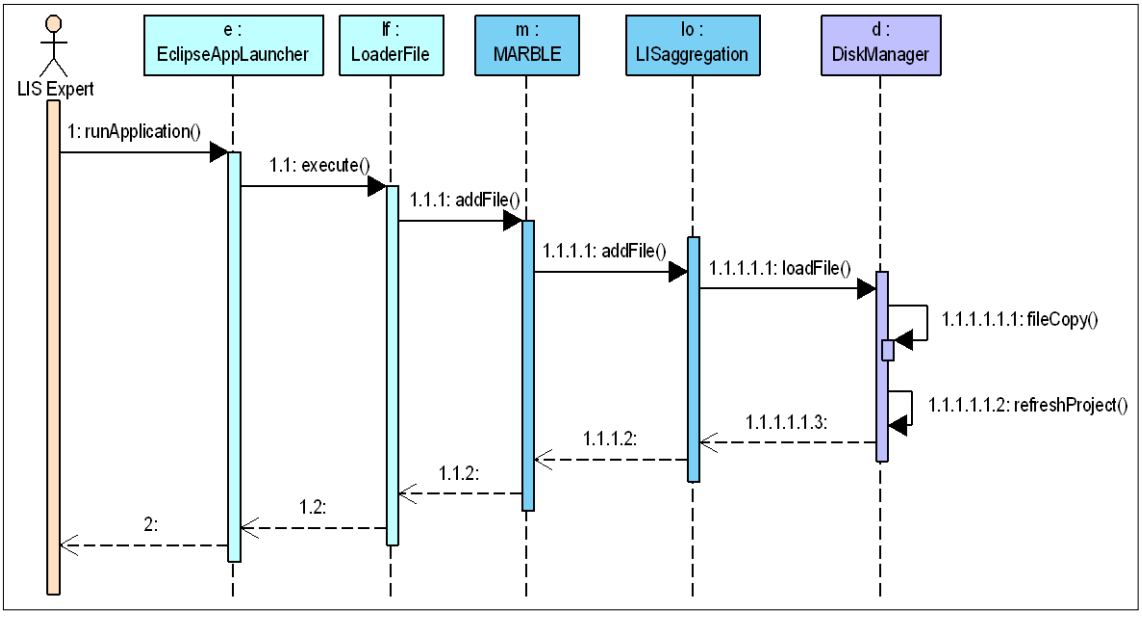


Figura 5.56. Diagrama de secuencia iteración 3 (CdU4: File)

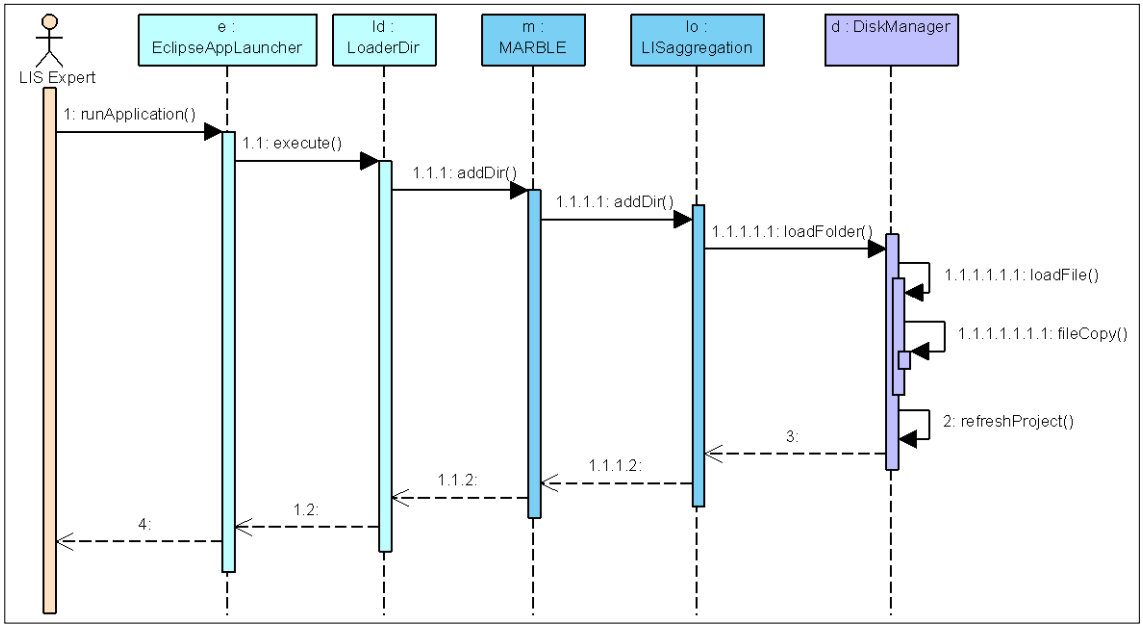


Figura 5.57. Diagrama de secuencia iteración 3 (CdU4: Directory)

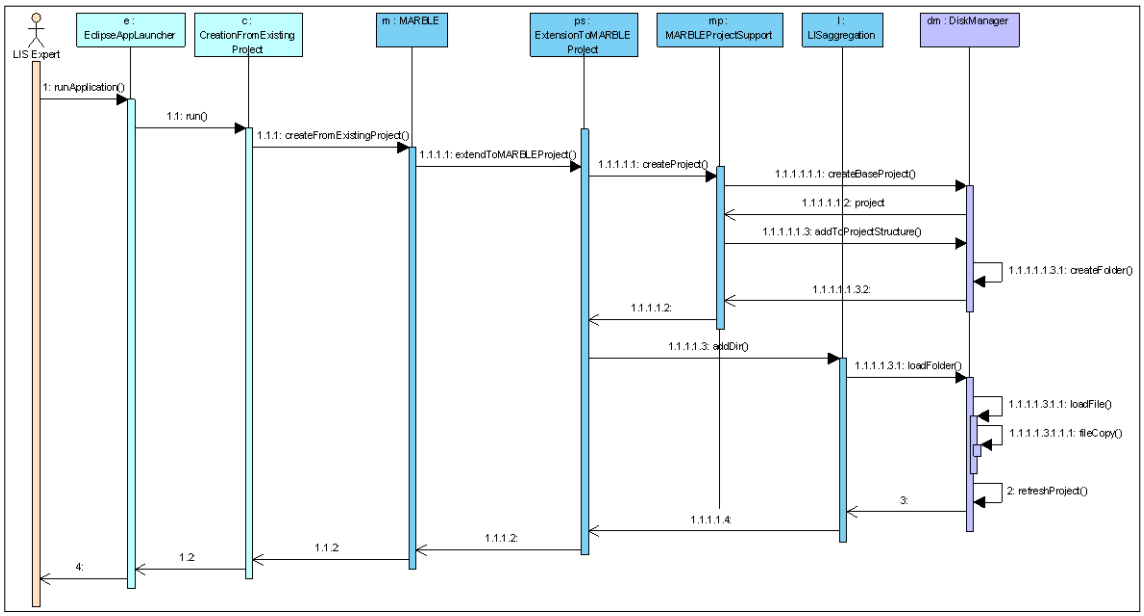


Figura 5.58. Diagrama de secuencia iteración 3 (CdU3)

La Figura 5.60 corresponde al diagrama de la Figura 5.59 extendido con las funcionalidades de generar informes del caso de uso CdU14.

Esta parte corresponde al producto de salida PS. 3.4 (véase Tabla 4.4).

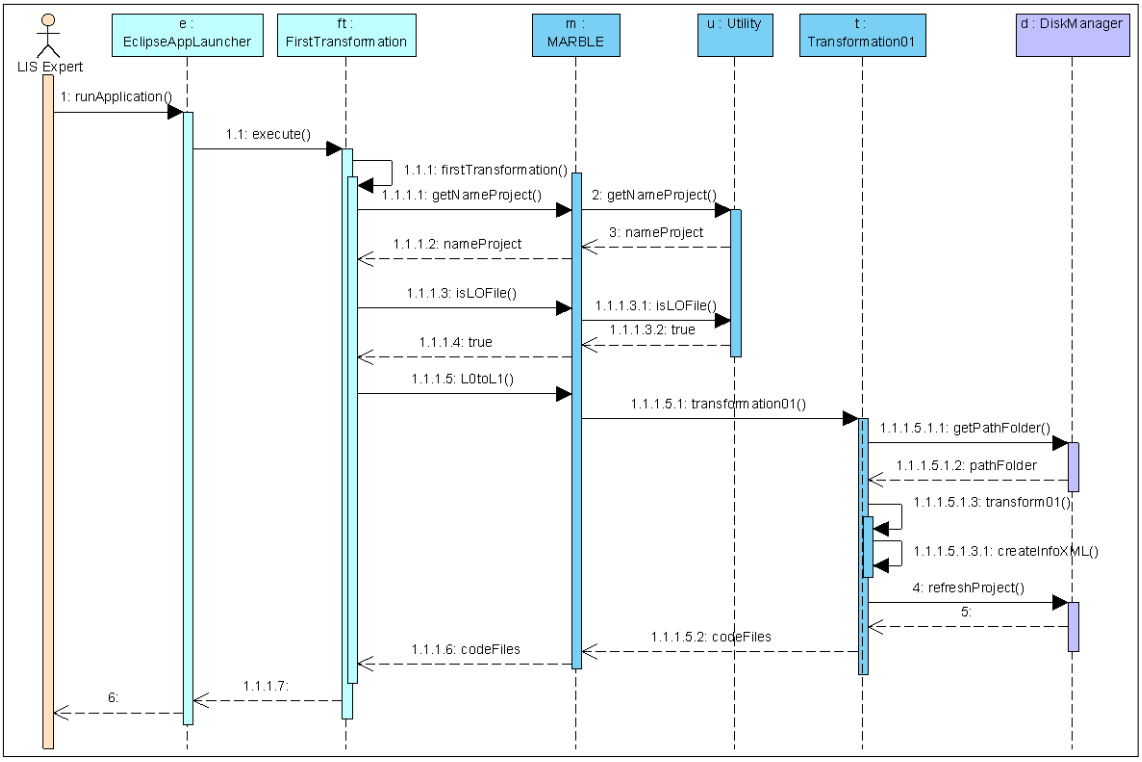


Figura 5.59. Diagrama de secuencia iteración 3 (CdU5)

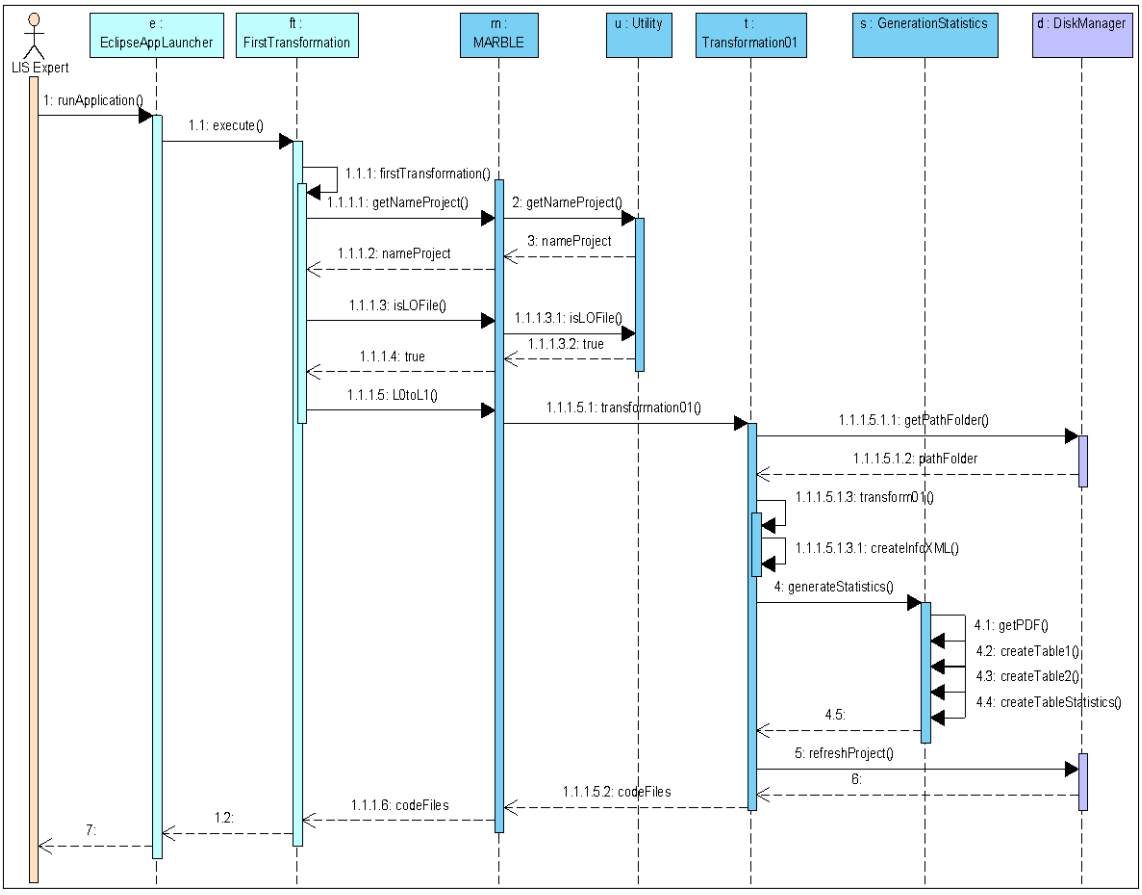


Figura 5.60. Diagrama de secuencia iteración 3 (CdU14)

5.3.2.3. Prototipos de las GUIs en la iteración 3

A continuación se muestra el prototipo de interfaz de usuario perteneciente a las funcionalidades de los casos de uso CdU3, CdU4, CdU5 y CdU6 (Figura 5.61, Figura 5.62, Figura 5.63, Figura 5.64, Figura 5.65, Figura 5.67, Figura 5.66). El caso de uso CdU14 no dispone de interfaz de usuario. Esta parte corresponde al producto de salida PS. 3.5.

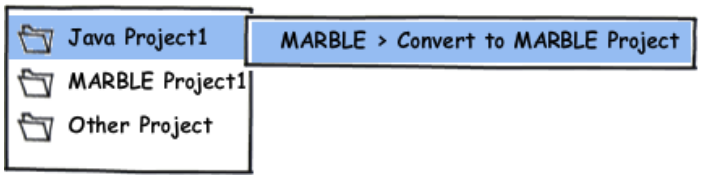


Figura 5.61. Prototipo de la GUI del CdU3. Menú contextual

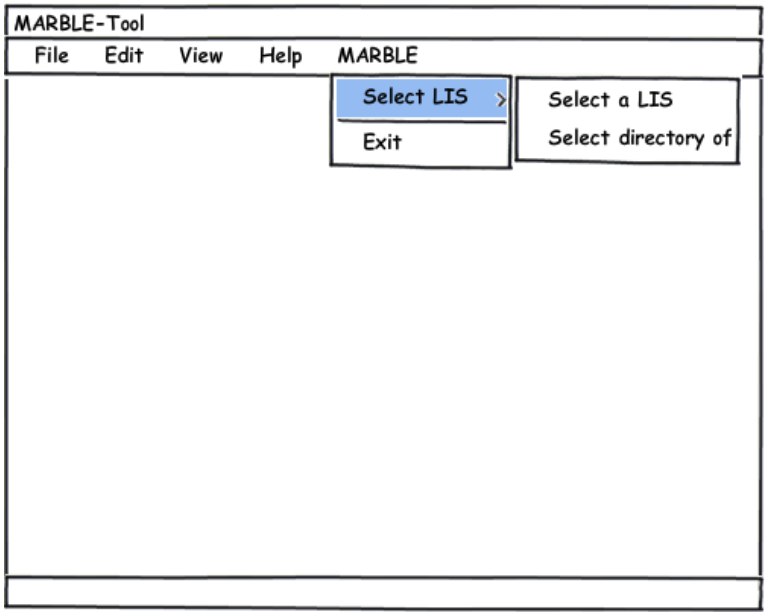


Figura 5.62. Prototipo de la GUI del CdU4. Barra de menú

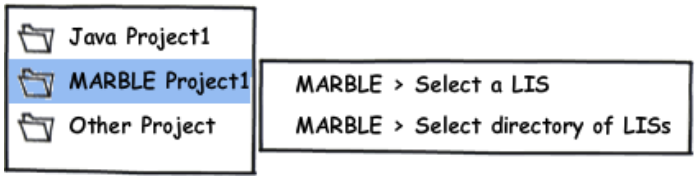


Figura 5.63. Prototipo de la GUI del CdU4. Menú contextual

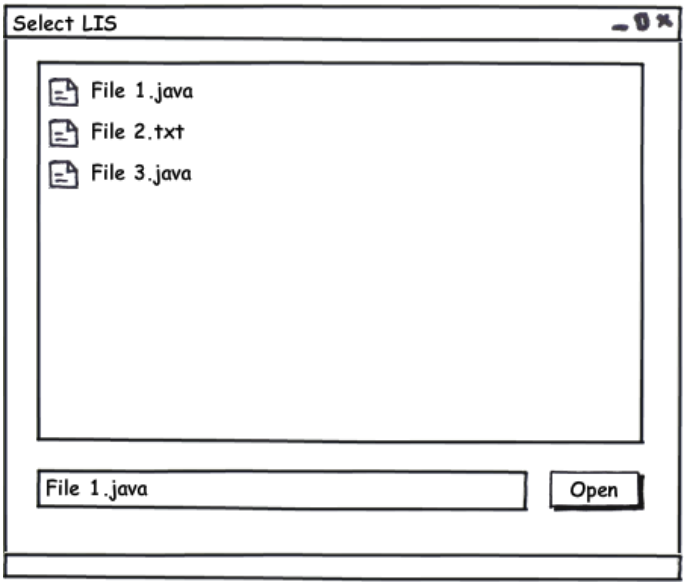


Figura 5.64. Prototipo de la GUI del CdU4 (Select a LIS)

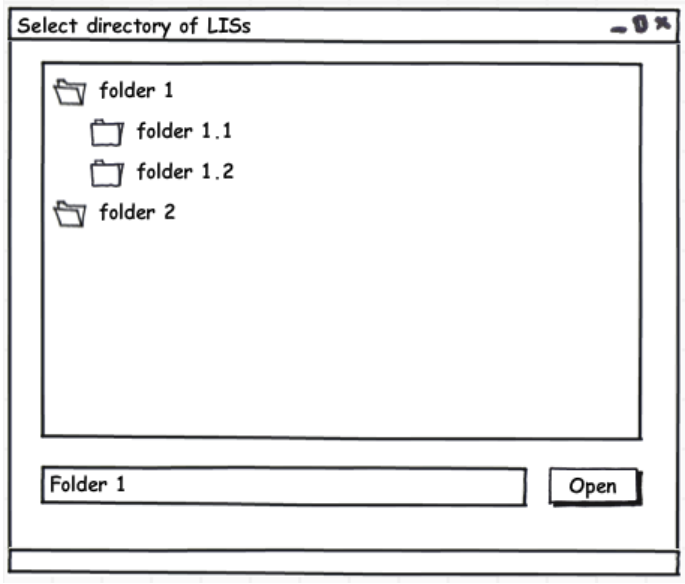


Figura 5.65. Prototipo de la GUI del CdU4 (Select directory of LISs)

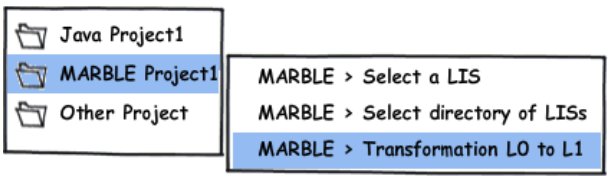


Figura 5.66. Prototipo de la GUI del CdU5. Menú contextual

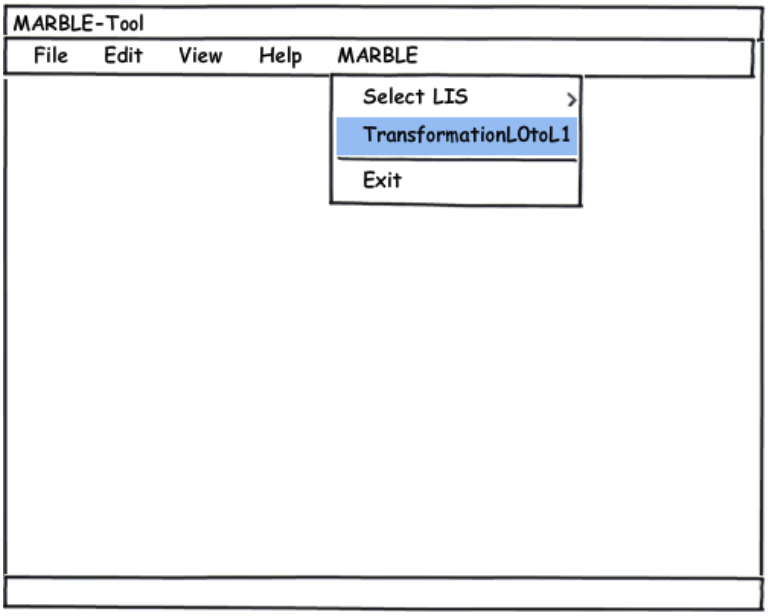


Figura 5.67. Prototipo de la GUI del CdU5. Barra de menú

5.3.3. Implementación

En esta iteración se comienza a realizar la implementación de los primeros casos de uso de la herramienta (CdU1 y CdU2), que serán el punto de partida para el resto de implementaciones. En esta parte sólo se comentarán los aspectos más destacables de la implementación y los problemas encontrados durante su desarrollo. La totalidad de la implementación de la herramienta se encuentra en el CD adjunto a esta memoria.

Al tratarse de un plug-in de Eclipse la forma de implementar los casos de uso es a través de extensiones, las cuales son vinculadas a las clases que han sido definidas en la etapa de diseño y aportan toda la funcionalidad a dicha extensión.

Para realizar la implementación de los casos de uso de esta iteración (CdU1 y CdU2) para proporcionar la funcionalidad de crear un nuevo tipo de proyecto MARBLE asociado a una perspectiva específica se han realizado las siguientes acciones:

a) Proporcionar un asistente (*wizard*) para la creación de un nuevo tipo de proyecto MARBLE mediante el punto de extensión de la manera mostrada en Fragmento de código 5.1.

b) Proporcionar una perspectiva MARBLE que muestre aquellas vistas que son de interés. Estas vistas son:

- Vista del explorador de proyectos
- Vista del editor
- Vista de outline
- Vista de la consola
- Vista de propiedades
- Vista del árbol sintáctico abstracto (ASTView)

Esta perspectiva se crea mediante un punto de extensión de la manera mostrada en Fragmento de código 5.2.

La vista del árbol sintáctico abstracto se ha creado como una extensión de la manera mostrada en Fragmento de código 5.3.

El plug-in resultante después de esta iteración corresponde al producto de salida PS. 3.6.

```
<extension
      point="org.eclipse.ui.newWizards">
   <category
         id="plugin.menuMARBLE.category"
         name="MARBLE">
   </category>
   <wizard
         category="plugin.menuMARBLE.category"
         class="presentation.actions.NewMARBLEProject"
         finalPerspective="plugin.menuMARBLE.perspective"
         icon="/icons/marbleCubo.gif"
         id="plugin.menuMARBLE.wizard"
         name="MARBLE Project">
   </wizard>
</extension>
<extension
      id="plugin.menuMARBLE.projectNature"
      point="org.eclipse.core.resources.natures">
   <runtime>
      <run

    class="domain.creationMARBLEProject.newFromScratch.ProjectNatureMARBLE
    ">
      </run>
   </runtime>
</extension>
<extension
      point="org.eclipse.ui.ide.projectNatureImages">
   <image
         icon="icons/marbleCubo.gif"
         id="plugin.menuMARBLE.imageNature"
         natureId="plugin.menuMARBLE.projectNature">
   </image>
</extension>
```

Fragmento de código 5.1. Extensión org.eclipse.ui.newWizards

```
<extension
      point="org.eclipse.ui.perspectives">
   <perspective
         class="domain.creationMARBLEProject.newFromScratch.PerspectiveM
          ARBLE"
         fixed="true"
         icon="icons/marbleCubo.gif"
         id="plugin.menuMARBLE.perspective"
         name="MARBLE">
   </perspective>
</extension>
```

Fragmento de código 5.2. Extensión org.eclipse.ui.perspectives

```
<extension
      point="org.eclipse.ui.views">
   <view
         allowMultiple="true"
         class="org.eclipse.jdt.astview.views.ASTView"
         icon="icons/ast.gif"
         id="plugin.menuMARBLE.ASTView"
         name="Abstract Syntax Tree"
         restorable="true">
   </view>
</extension>
```

Fragmento de código 5.3. Extensión org.eclipse.ui.views

5.4. Iteración 4

Como se indica en la Tabla 4.5, en esta iteración se han generado los productos de salida que se detallan a continuación agrupados según los flujos de trabajo de análisis, diseño e implementación.

5.4.1. Análisis

Para realizar el análisis de los casos de uso de la iteración 4 se ha optado por la utilización de diagramas de secuencia de análisis y diagramas de comunicación.

5.4.1.1. Diagramas de Secuencia de Análisis en la iteración 4

Para comprender en mayor medida la interacción de cada caso de uso con los roles del sistema se muestran a continuación los diagramas de secuencia para cada uno de los casos de uso contemplados en esta iteración (CdU7 y CdU8).

Esta parte corresponde al producto de salida PS. 4.1 (véase Tabla 4.5).

1. CdU7. Integrate source model into KDM repository

a) Escenario principal

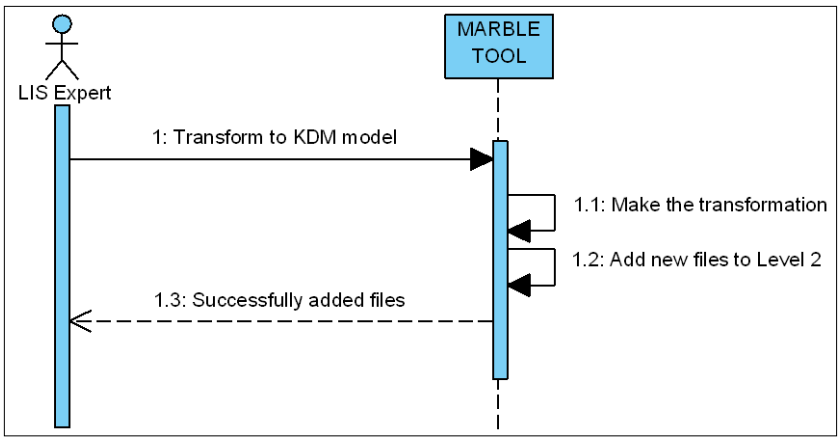


Figura 5.68. Diagrama de secuencia CdU7. Escenario Principal

b) Escenario alternativo 1 de error

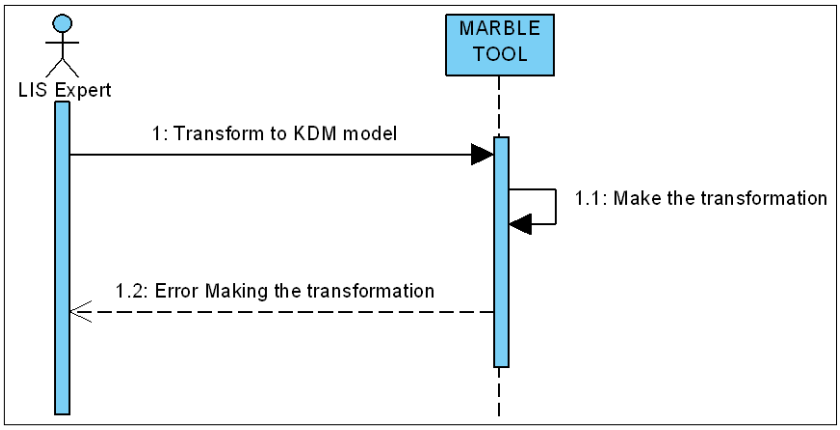


Figura 5.69. Diagrama de secuencia CdU7. Escenario de Error

2. CdU8. Edit KDM model

a) Escenario principal

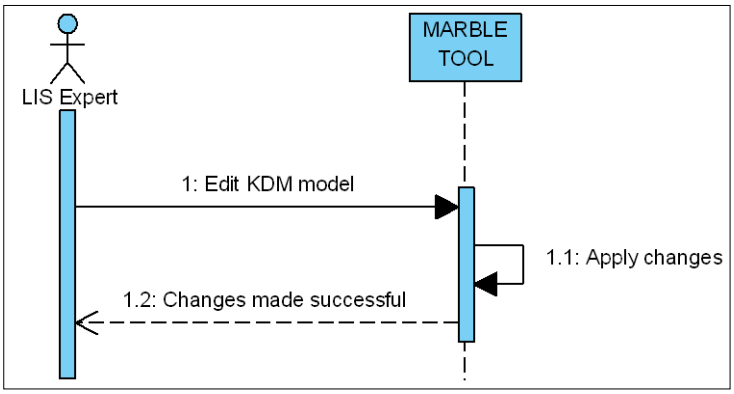


Figura 5.70. Diagrama de secuencia CdU8. Escenario Principal

b) Escenario alternativo 1

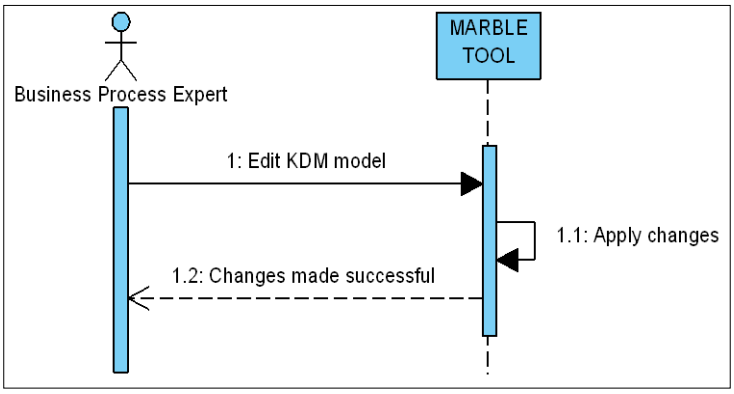


Figura 5.71. Diagrama de secuencia CdU8. Escenario alternativo

c) Escenario alternativo 2 de error

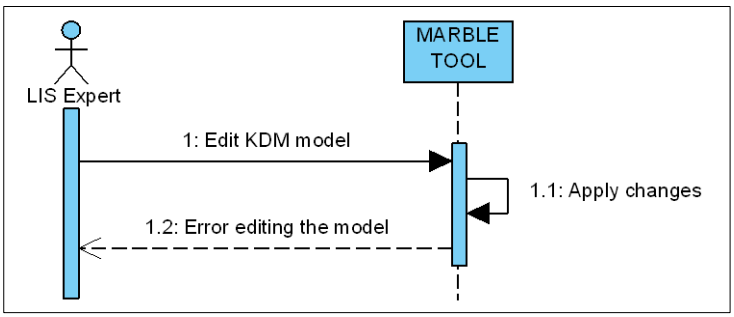


Figura 5.72. Diagrama de secuencia CdU8. Escenario de Error (I)

d) Escenario alternativo 3 de error

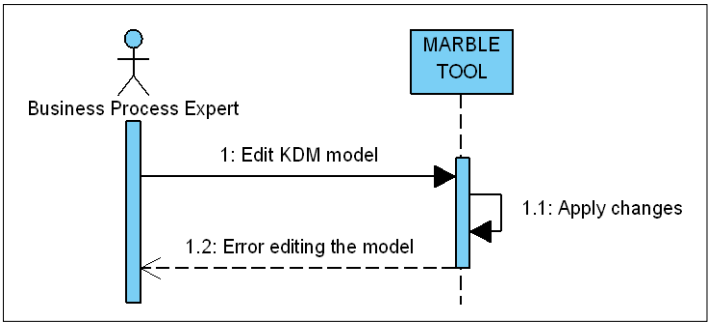


Figura 5.73. Diagrama de secuencia CdU8. Escenario de Error (II)

5.4.1.2. Diagramas de Comunicación en la iteración 4

La Figura 5.74 muestra el diagrama de comunicación correspondiente al caso de uso CdU7. Esta parte corresponde al producto de salida PS. 4.2.

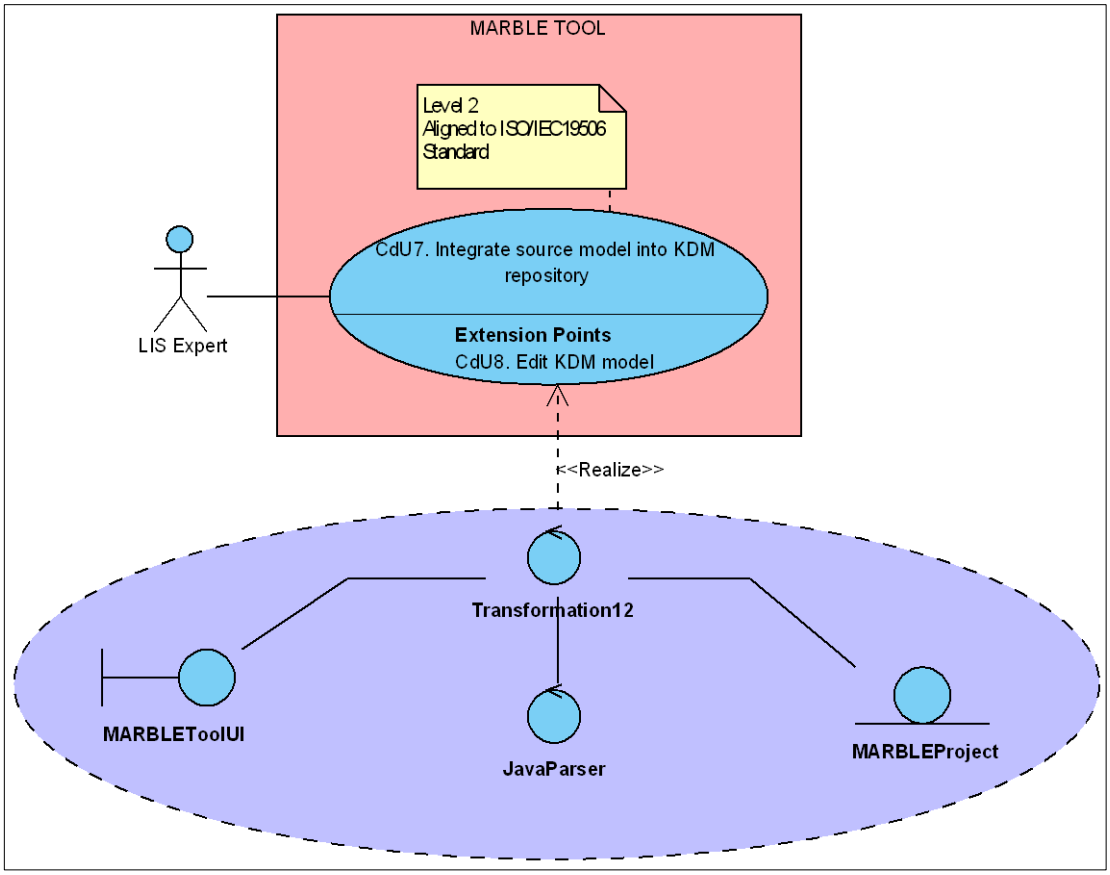


Figura 5.74. Diagrama de comunicación de CdU7

5.4.2. Diseño

Para realizar el diseño de los casos de uso seleccionados en esta iteración se van a utilizar los diagramas de clases y los diagramas de secuencia que muestren la interacción entre las distintas clases.

5.4.2.1. Diagrama de Clases del sistema en la iteración 4

El diagrama de clases correspondiente al diseño de los casos de uso CdU7 y CdU8 se muestra de forma colapsada en la Figura 5.75. A continuación, se detallan las clases involucradas en el diagrama. Sólo se detallan las clases nuevas o aquellas que han sufrido cambios en esta iteración respecto a la iteración anterior. Esta parte corresponde al producto de salida PS. 4.3.

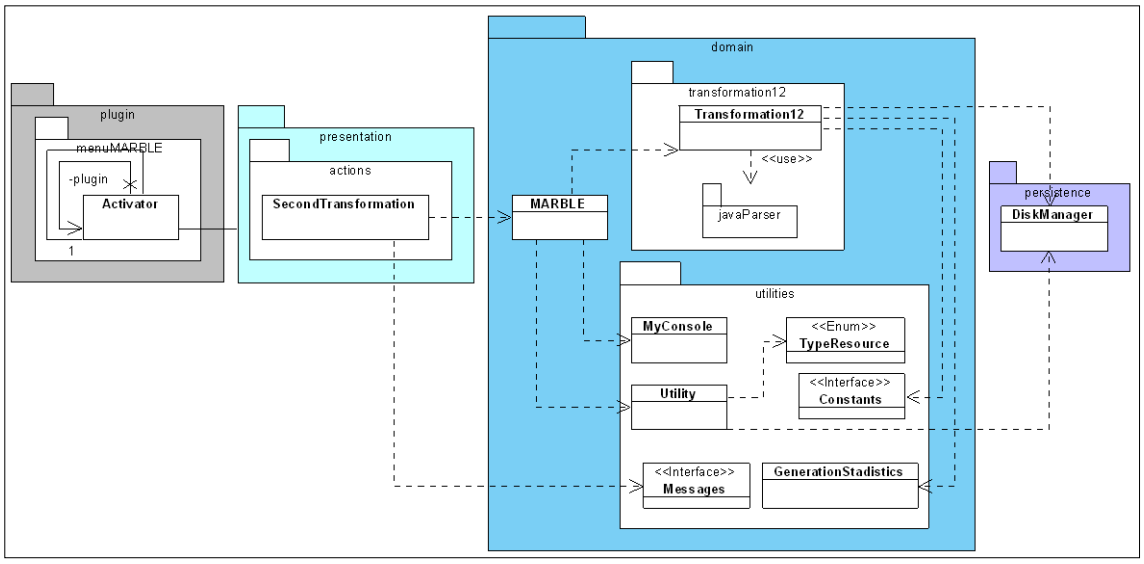


Figura 5.75. Diagrama de clases iteración 4

SecondTransformation

La Figura 5.76 muestra el diagrama UML correspondiente a la clase SecondTransformation. La responsabilidad de esta clase será la de proporcionar una interfaz de usuario para realizar la segunda transformación propuesta por MARBLE a partir de unos determinados modelos de código seleccionados de un proyecto MARBLE existente.

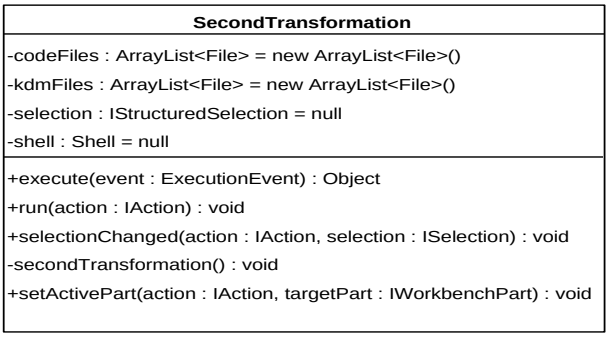


Figura 5.76. Diagrama de Clases iteración 4: Clase SecondTransformation

MARBLE

La Figura 5.47 muestra el diagrama UML correspondiente a la clase MARBLE. En esta iteración se han añadido nuevos métodos a la clase MARBLE para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

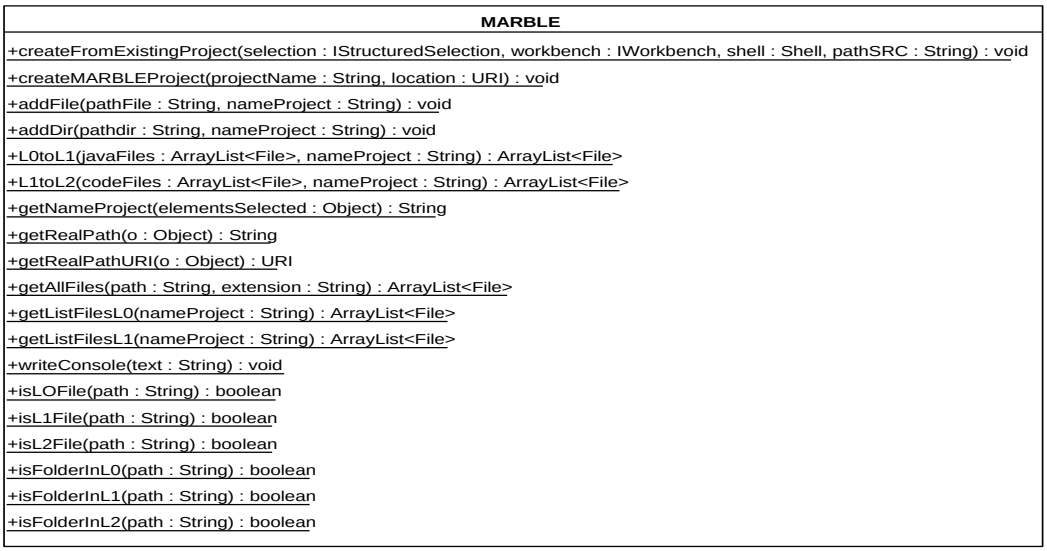


Figura 5.77. Diagrama de Clases iteración 4: Clase MARBLE

Transformation12

La Figura 5.78 muestra el diagrama UML correspondiente a la clase Transformation12. Esta clase está contenida en la capa de dominio, en el paquete de realizar la segunda transformación (*transformation12*). La responsabilidad de esta clase será la de proporcionar la funcionalidad para realizar la segunda transformación propuesta por MARBLE a partir de unos determinados modelos de código seleccionados de un proyecto MARBLE existente.

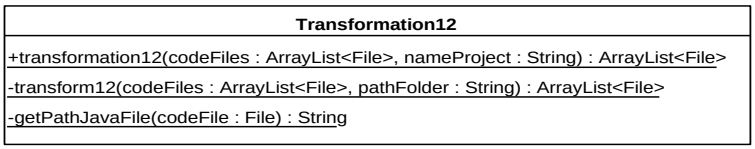


Figura 5.78. Diagrama de Clases iteración 4: Clase Transformation12

JavaParser (paquete)

Este paquete contiene todas las clases necesarias para realizar un parser de Java que permiten el análisis de código fuente de un sistema de información existente a fin de reconocer ciertos elementos y estructuras del código que serán representadas en un modelo KDM.

Las clases contenidas en este paquete han seguido el patrón fábrica abstracta para facilitar la futura incorporación de nuevos parsers (véase sección 5.2.2.1.1). Un fragmento de la aplicación de este patrón puede verse en la Figura 5.79.

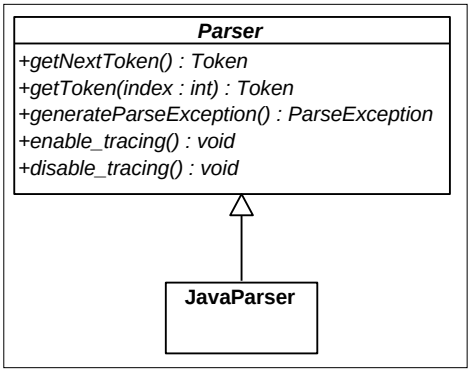


Figura 5.79. Aplicación de patrón fábrica abstracta

Utility

La Figura 5.80 muestra el diagrama UML correspondiente a la clase Utility. En esta iteración se han añadido nuevos elementos a la clase Utility para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

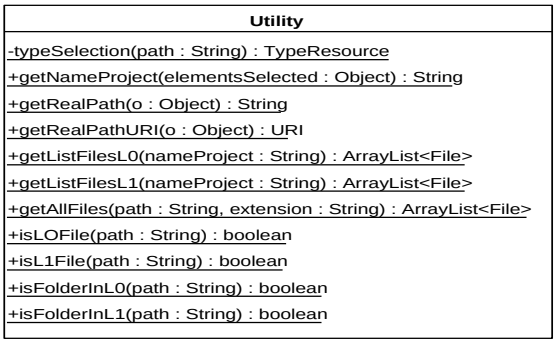


Figura 5.80. Diagrama de Clases iteración 4: Clase Utility

Interface Messages

La Figura 5.81 muestra el diagrama UML correspondiente a la interfaz Messages. En esta iteración se han añadido nuevos elementos a la interfaz para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

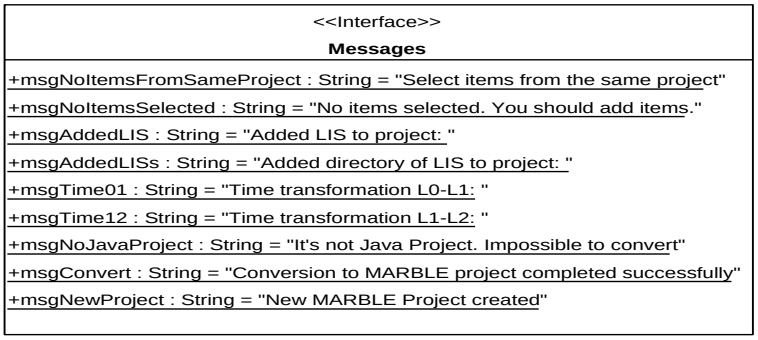


Figura 5.81. Diagrama de Clases iteración 4: Interfaz Messages

5.4.2.2. Diagramas de Secuencia de Diseño en la iteración 4

La Figura 5.82 muestra el diagrama de secuencia para el caso de uso CdU7. Esto corresponde al producto de salida PS. 4.4.

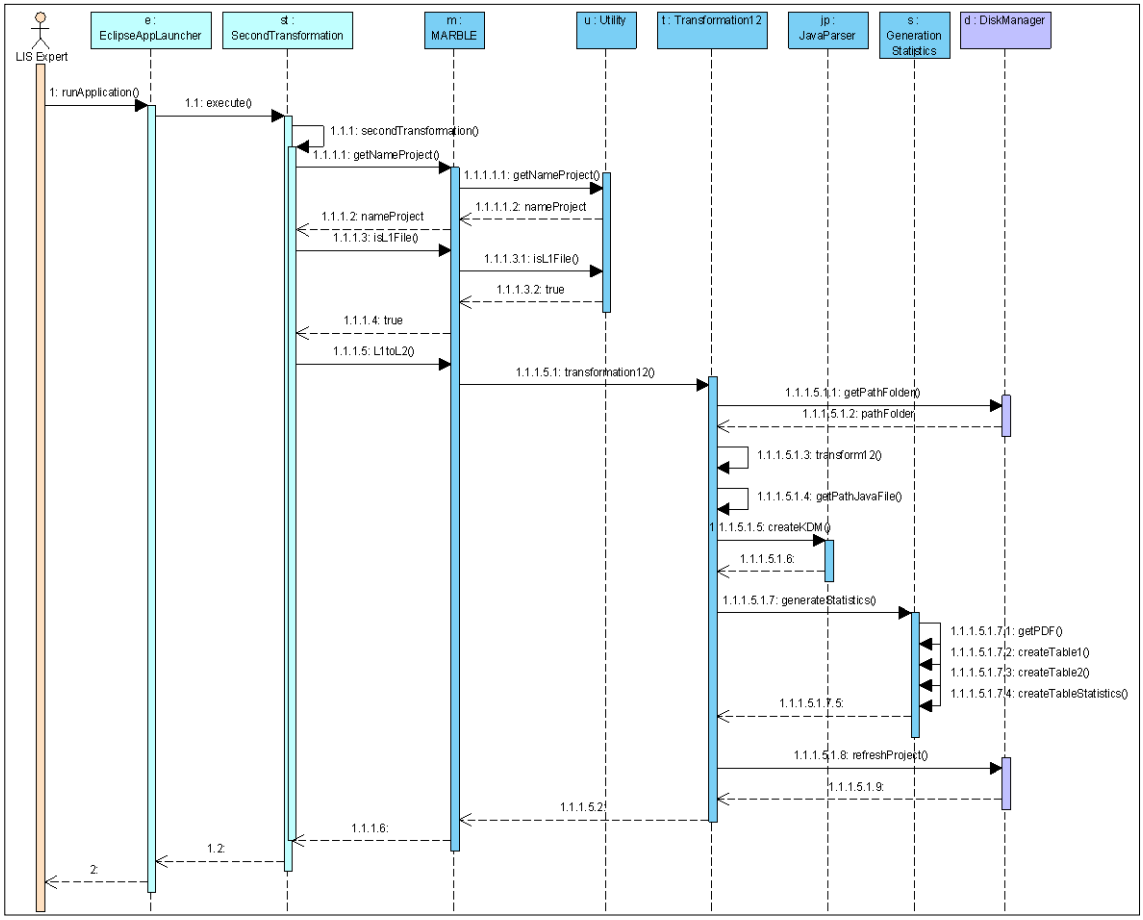


Figura 5.82. Diagrama de secuencia iteración 4 (CdU7)

5.4.2.3. Prototipos de las GUIs en la iteración 4

A continuación se muestra el prototipo de interfaz de usuario perteneciente a la funcionalidad del caso de uso CdU7 (Figura 5.83 y Figura 5.84). Esta parte corresponde al producto de salida PS. 4.5.

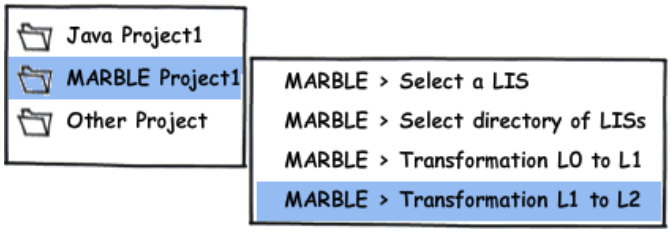


Figura 5.83. Prototipo de GUI de CdU7. Menú contextual

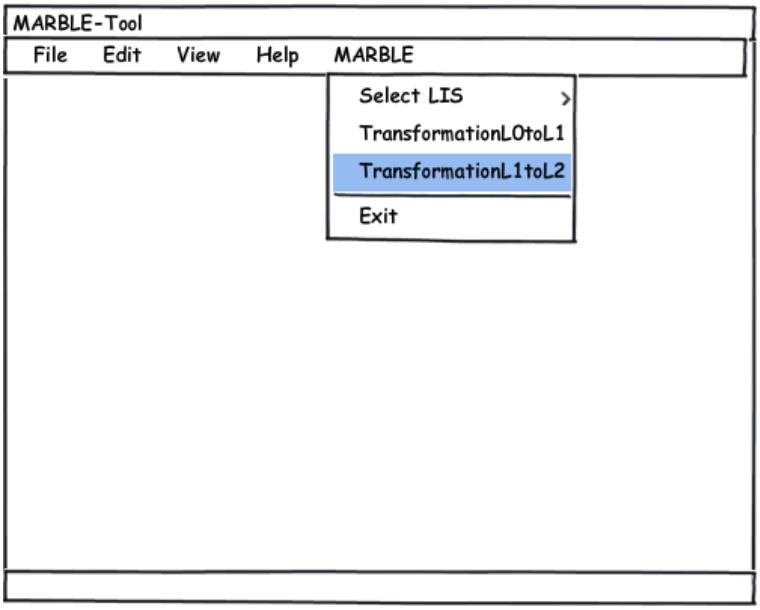


Figura 5.84. Prototipo de GUI de CdU7. Barra de menú

5.4.3. Implementación

En esta parte sólo se comentarán los aspectos más destacables de la implementación y los problemas encontrados durante el desarrollo de los casos de uso de esta iteración (CdU3, CdU4, CdU5, CdU6 y CdU14). Las acciones que se han realizado son las siguientes:

a) Proporcionar un menú MARBLE en el que añadir las opciones que se van a implementar mediante un punto de extensión de la manera mostrada en Fragmento de código 5.4.

b) Crear un menú contextual MARBLE en el que añadir las opciones mediante un punto de extensión de la manera mostrada en Fragmento de código 5.5.

c) Crear un comando para seleccionar LIS(s) y añadir la opción al menú MARBLE de la forma mostrada en Fragmento de código 5.6 y al menú contextual MARBLE de la forma mostrada en Fragmento de código 5.7.

```
<extension
      point="org.eclipse.ui.menus">
   <menuContribution
         locationURI="menu:org.eclipse.ui.main.menu">
      <menu
            id="menuMARBLE"
            label="MARBLE">

      </menu>
   </menuContribution>
</extension>
```

Fragmento de código 5.4. Extensión org.eclipse.ui.menus

```
<extension
      point="org.eclipse.ui.popupMenus">
   <objectContribution
         id="plugin.menuMARBLE.contribution"
         objectClass="org.eclipse.core.resources.IResource">
      <menu
            icon="icons/marbleCubo.gif"
            id="plugin.menuMARBLE.menu1"
            label="MARBLE">
      </menu>
   </objectContribution>
</extension>
```

Fragmento de código 5.5. Extensión org.eclipse.ui.popupMenus

d) Crear un comando para realizar la primera transformación y añadir la opción al menú MARBLE y al menú contextual MARBLE de la misma forma que la opción anterior.

e) El CdU6 no es necesario implementarlo ya que se trata de una funcionalidad (editar un proyecto) que heredan todos los plug-in de Eclipse.

Para implementar la primera transformación (de código fuente heredado a un modelo de código) se ha utilizado un parser de Java. Este parser ha sido una extensión del propuesto por `org.eclipse.jdt.core.dom.ASTParser`.

```
<extension
      point="org.eclipse.ui.commands">
   <command
         defaultHandler="presentation.actions.LoaderDirectory"
         id="plugin.menuMARBLE.LoadDir"
         name="LoadDir">
   </command>
   <command
         defaultHandler="presentation.actions.LoaderFile"
         id="plugin.menuMARBLE.LoadFile"
         name="LoadFile">
   </command>
</extension>

<extension
      point="org.eclipse.ui.menus">
   <menuContribution
         locationURI="menu:org.eclipse.ui.main.menu">
      <menu
            id="menuMARBLE"
            label="MARBLE">

         <menu
               icon="icons/folder.gif"
               label="Select LIS">
            <command
                  commandId="plugin.menuMARBLE.LoadFile"
                  icon="icons/java.gif"
                  label="Select a LIS"
                  style="push">
            </command>
            <command
                  commandId="plugin.menuMARBLE.LoadDir"
                  icon="icons/project.gif"
                  label="Select directory of LISs"
                  style="push">
            </command>
         </menu>

      </menu>
   </menuContribution>
</extension>
```

Fragmento de código 5.6. Extensión org.eclipse.ui.commands y comando en menú

```
<extension
      point="org.eclipse.ui.popupMenus">
   <objectContribution
         id="plugin.menuMARBLE.contribution"
         objectClass="org.eclipse.core.resources.IResource">
      <menu
            icon="icons/marbleCubo.gif"
            id="plugin.menuMARBLE.menu1"
            label="MARBLE">
          <separator
               name="group3">
         </separator>

      </menu>

      <action
            class="presentation.actions.LoaderDirectory"
            enablesFor="1"
            icon="icons/project.gif"
            id="plugin.menuMARBLE.loaddir"
            label="Select directory of LISs"
            menubarPath="plugin.menuMARBLE.menu1/group3">
      </action>
      <action
            class="presentation.actions.LoaderFile"
            enablesFor="1"
            icon="icons/java.gif"
            id="plugin.menuMARBLE.selectfile"
            label="Select a LIS"
            menubarPath="plugin.menuMARBLE.menu1/group3">
      </action>
   </objectContribution>
</extension>
```

Fragmento de código 5.7. Extensión org.eclipse.ui.popupMenus y comando en menú contextual

Como apoyo a la implementación en esta iteración se ha utilizado la librería existente *JDom* y se ha desarrollado el editor CodeEditor.

Para realizar la implementación del caso de uso CdU14 se ha utilizado la librería *iText* para generar las estadísticas en un documento PDF. Esto se muestra en el diagrama de componentes de la Figura 5.85 y se describe a continuación:

- **libs_jdom**: Corresponde al conjunto de librerías de JDOM que se encarga de la manipulación de datos XML a través de código Java.
- **libs_pdf**: Corresponde con el conjunto de librerías de iText para permitir la creación de documentos PDF a través de código Java.
- **CodeEditor**: Editor desarrollado como un plug-in de Eclipse para la visualización de los modelos de código generados en la primera transformación.

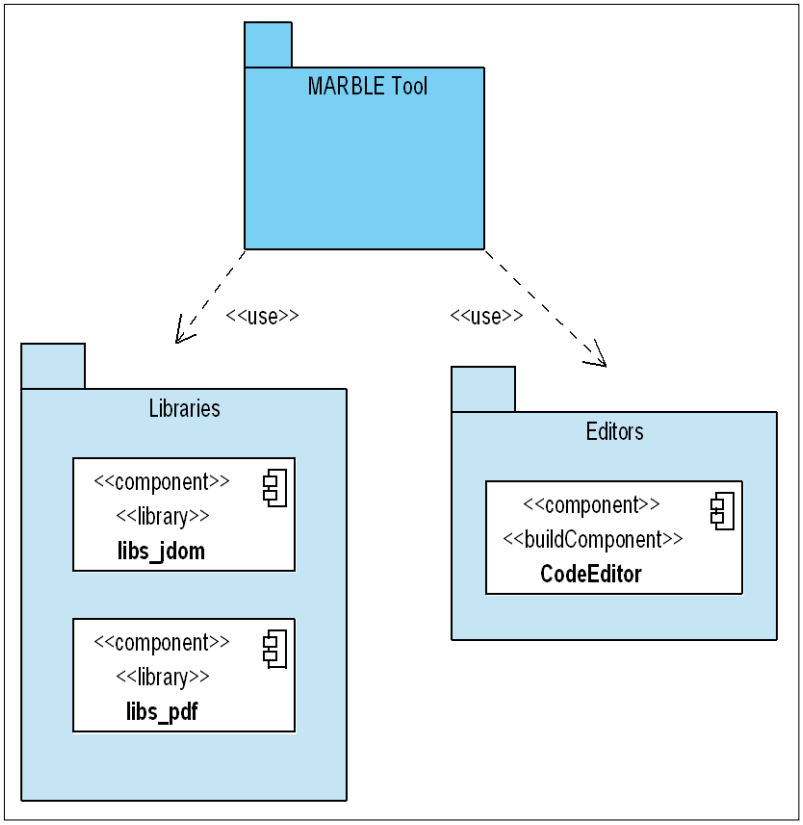


Figura 5.85. Diagrama de componentes (I)

El plug-in resultante de añadir las anteriores extensiones corresponde al producto de salida PS. 4.6.

5.5. Iteración 5

Como se indica en la Tabla 4.6, en esta iteración se han generado los productos de salida que se detallan a continuación agrupados según el flujo de trabajo al que pertenecen.

5.5.1. Análisis

Para realizar el análisis de los casos de uso de la iteración 5 se ha optado por la utilización de diagramas de secuencia y diagramas de comunicación.

5.5.1.1. Diagramas de Secuencia de Análisis en la iteración 5

Para comprender en mayor medida la interacción de cada caso de uso con los roles del sistema se muestran a continuación los diagramas de secuencia para cada uno de los casos de uso contemplados en esta iteración (CdU9 y CdU11). Para cada uno de los casos de uso se muestran al menos dos tipos de escenarios: escenario principal y escenario alternativo o de error. Esta parte corresponde al producto de salida PS. 5.1 (véase Tabla 4.6).

1. CdU9. Generate Business Process (model) from KDM model

a) Escenario Principal

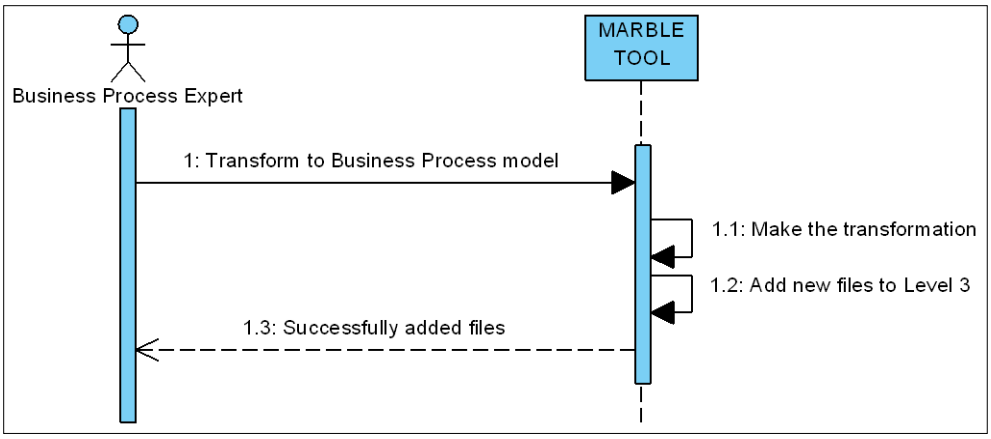


Figura 5.86. Diagrama de secuencia CdU9. Escenario Principal

b) Escenario alternativo de Error

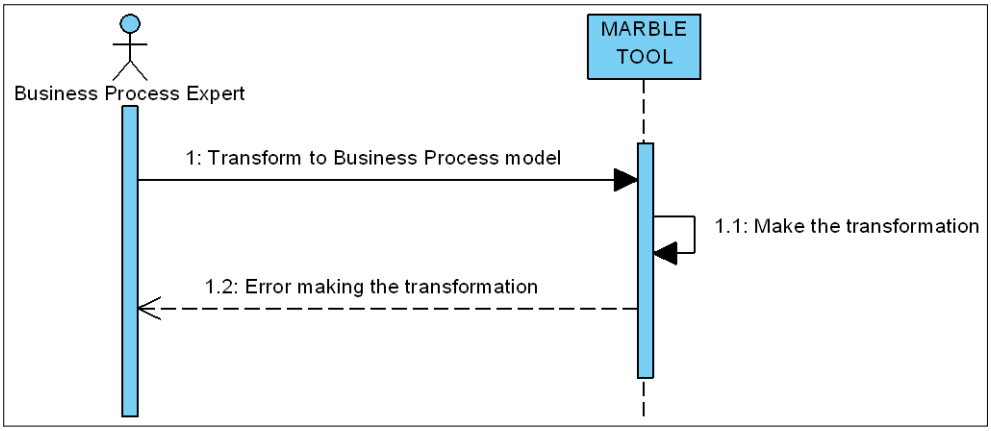


Figura 5.87. Diagrama de secuencia CdU9. Escenario de Error

2. CdU11. Edit Business Process model

a) Escenario Principal

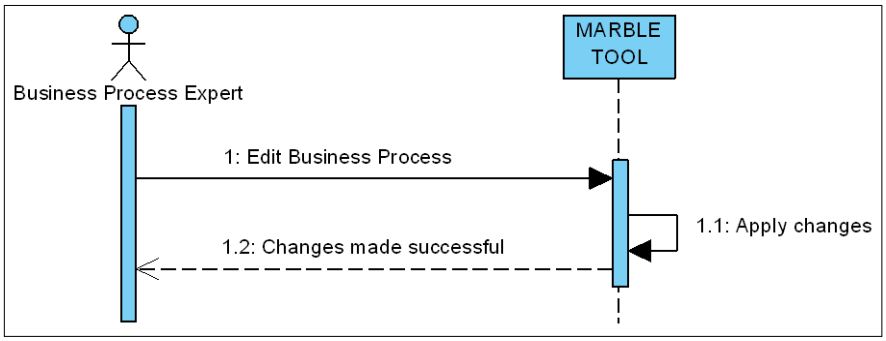


Figura 5.88. Diagrama de secuencia CdU11. Escenario Principal

b) Escenario alternativo de Error

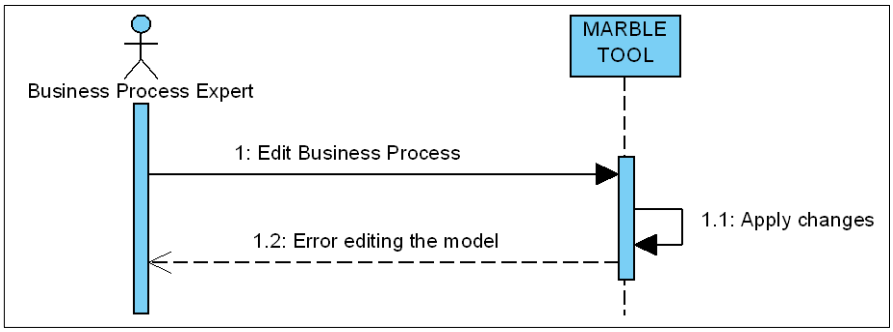


Figura 5.89. Diagrama de secuencia CdU11. Escenario de Error

5.5.1.2. Diagramas de Comunicación en la iteración 5

La Figura 5.90 muestra el diagrama de comunicación correspondiente al caso de uso CdU9. Esta parte corresponde al producto de salida PS. 5.2.

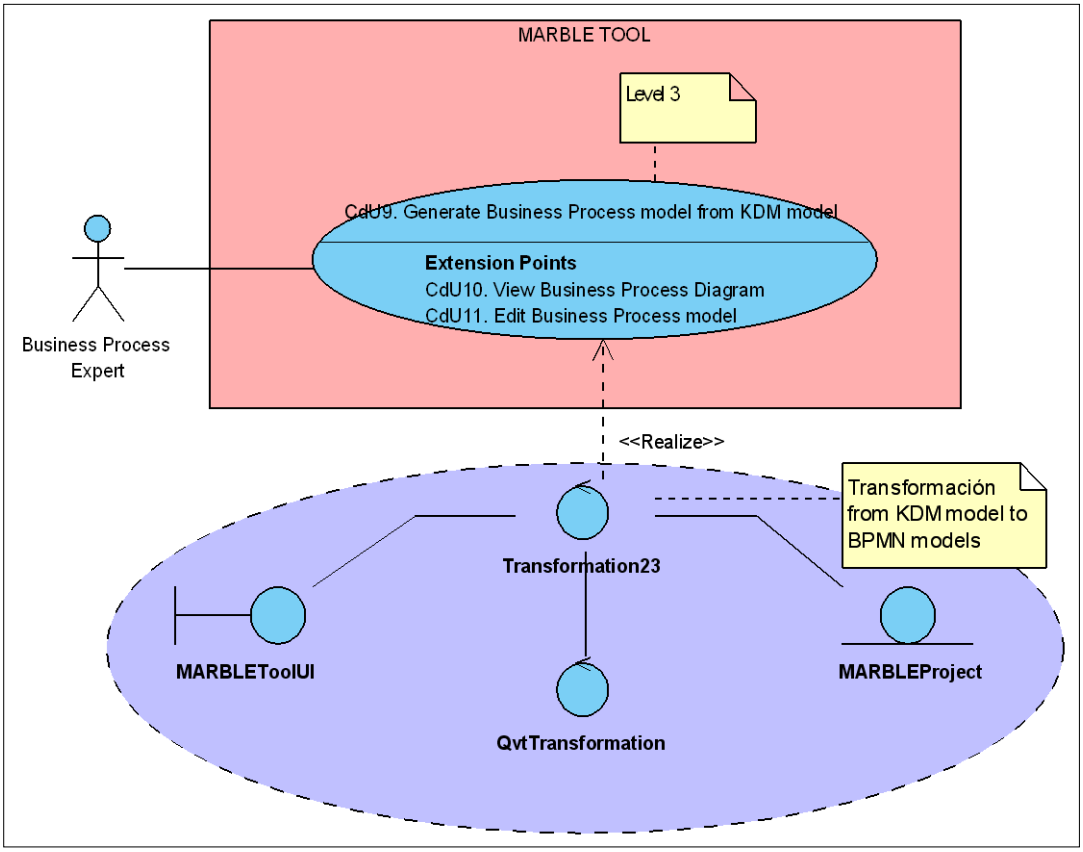


Figura 5.90. Diagrama de comunicación de CdU9

5.5.2. Diseño

Para realizar el diseño de los casos de uso seleccionados en esta iteración se van a utilizar los diagramas de clases y los diagramas de secuencia que muestren la interacción entre las distintas clases.

5.5.2.1. Diagrama de Clases del sistema en la iteración 5

El diagrama de clases correspondiente al diseño de los casos de uso CdU9 y CdU11 se muestra de forma colapsada en la Figura 5.91. A continuación, se detallan las clases involucradas en el diagrama. Nuevamente, sólo se detallan las clases nuevas o aquellas que han sufrido cambios en esta iteración respecto a la iteración anterior.

Esta parte corresponde al producto de salida PS. 5.3.

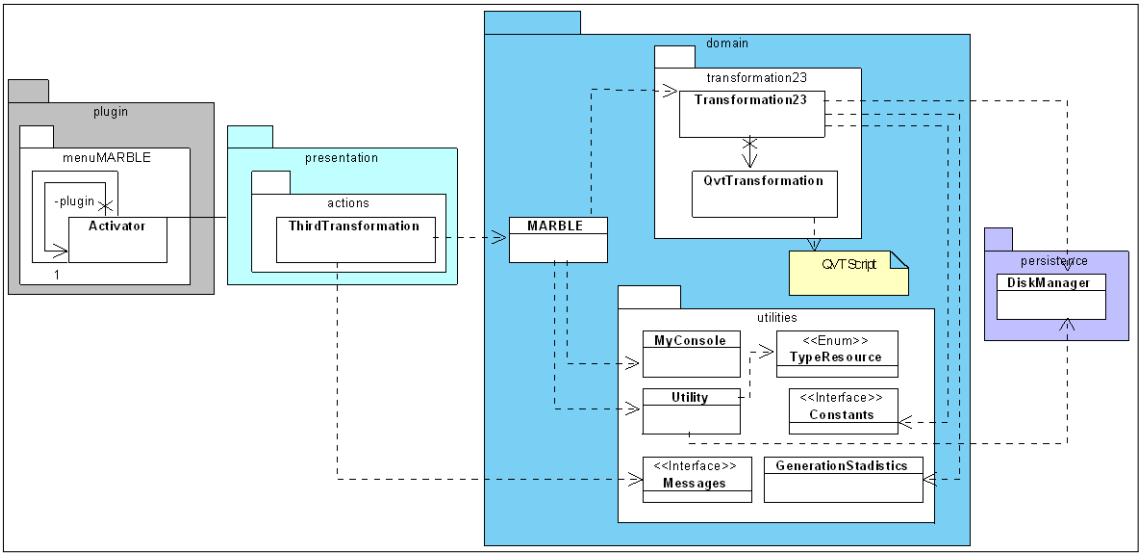


Figura 5.91. Diagrama de clases iteración 5

ThirdTransformation

La Figura 5.92 muestra el diagrama UML correspondiente a la clase ThirdTransformation. Esta clase está contenida en la capa de presentación, en el paquete de acciones (*actions*). La responsabilidad de esta clase será la de proporcionar una interfaz de usuario para realizar la tercera transformación propuesta por MARBLE a partir de unos determinados modelos kdm seleccionados de un proyecto MARBLE existente.

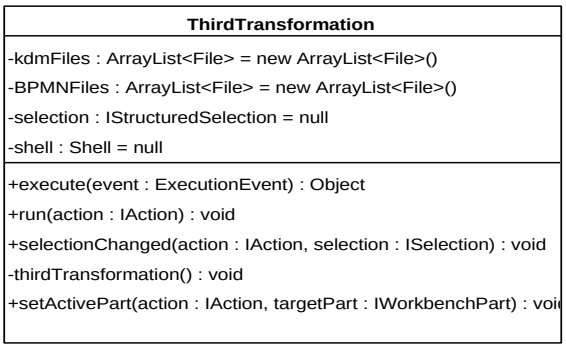


Figura 5.92. Diagrama de Clases iteración 5: Clase ThirdTransformation

MARBLE

La Figura 5.93 muestra el diagrama UML correspondiente a la clase MARBLE. En esta iteración se han añadido nuevos elementos a la clase MARBLE para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

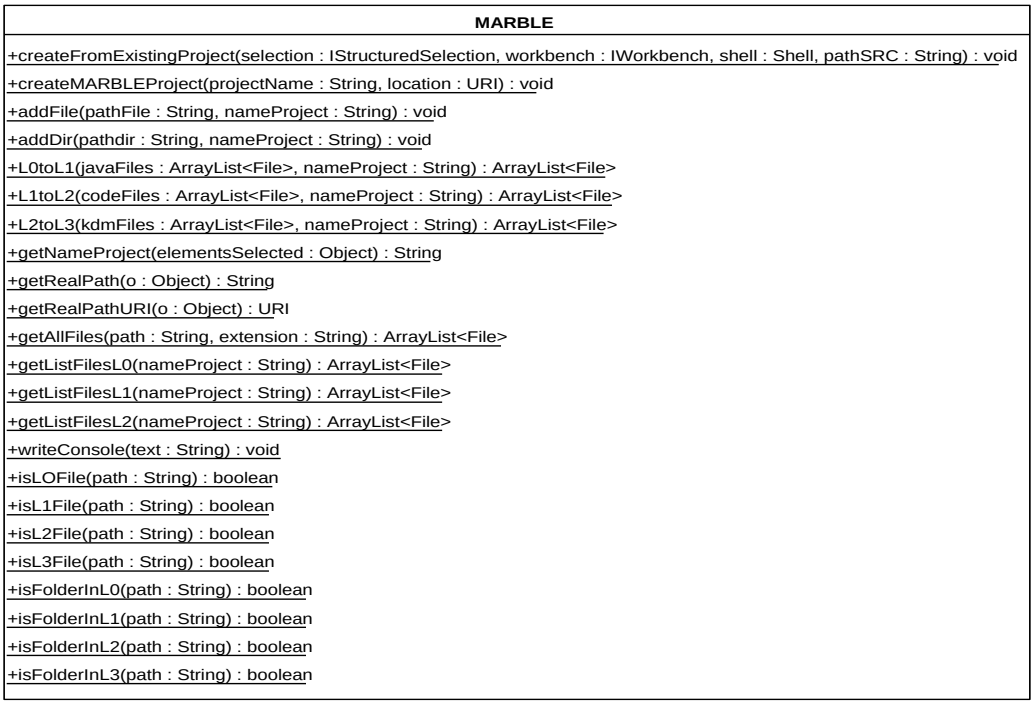


Figura 5.93. Diagrama de Clases iteración 5: Clase MARBLE

Transformation23

La Figura 5.94 muestra el diagrama UML correspondiente a la clase Transformation23. Esta clase está contenida en la capa de dominio, en el paquete de realizar la tercera transformación (*transformation23*). La responsabilidad de esta clase será la de proporcionar la funcionalidad para realizar la tercera transformación propuesta por MARBLE a partir de unos determinados modelos kdm seleccionados de un proyecto MARBLE existente.

Figura 5.94. Diagrama de Clases iteración 5: Clase Transformation23

QvtTransformacion

La Figura 5.95 muestra el diagrama UML correspondiente a la clase QvtTransformation. Esta clase está contenida en la capa de dominio, en el paquete de realizar la tercera transformación (*transformation23*). Se trata de la implementación de patrones de reconocimiento de estructuras de código mediante transformaciones QVT.

Esta clase hace uso de un archivo “patterns.qvt” que implementa todas las transformaciones de acuerdo al lenguaje QVTr.

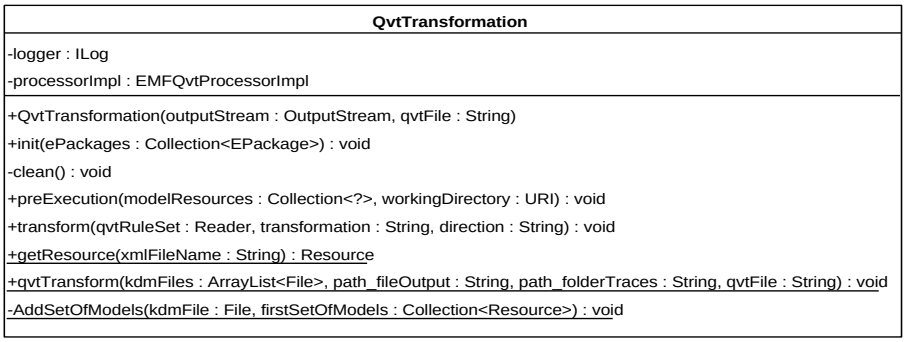


Figura 5.95. Diagrama de Clases iteración 5: Clase QvtTransformation

Utility

La Figura 5.96 muestra el diagrama UML correspondiente a la clase Utility. En esta iteración se han añadido nuevos elementos a la clase Utility para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

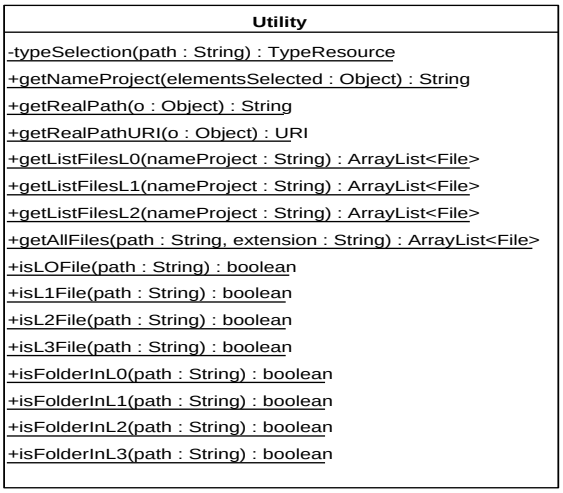


Figura 5.96. Diagrama de Clases iteración 4: Clase Utility

Interface Messages

La Figura 5.97 muestra el diagrama UML correspondiente a la interfaz Messages. En esta iteración se han añadido nuevos elementos a la interfaz para satisfacer las necesidades de los casos de uso que se tratan en esta iteración.

<<Interface>> **Messages**
+msgNoItemsFromSameProject : String = "Select items from the same project"
+msgNoItemsSelected : String = "No items selected. You should add items."
+msgAddedLIS : String = "Added LIS to project: "
+msgAddedLISs : String = "Added directory of LIS to project: "
+msgTime01 : String = "Time transformation L0-L1: "
+msgTime12 : String = "Time transformation L1-L2: "
+msgTime23 : String = "Time transformation L2-L3: "
+msgNoJavaProject : String = "It's not Java Project. Impossible to convert"
+msgConvert : String = "Conversion to MARBLE project completed successfully"
+msgNewProject : String = "New MARBLE Project created"

Figura 5.97. Diagrama de Clases iteración 4: Interfaz Messages

5.5.2.2. Diagramas de Secuencia de Diseño en la iteración 5

La Figura 5.98 muestra el diagrama de secuencia para el caso de uso CdU9. Esta parte corresponde al producto de salida PS. 5.4.

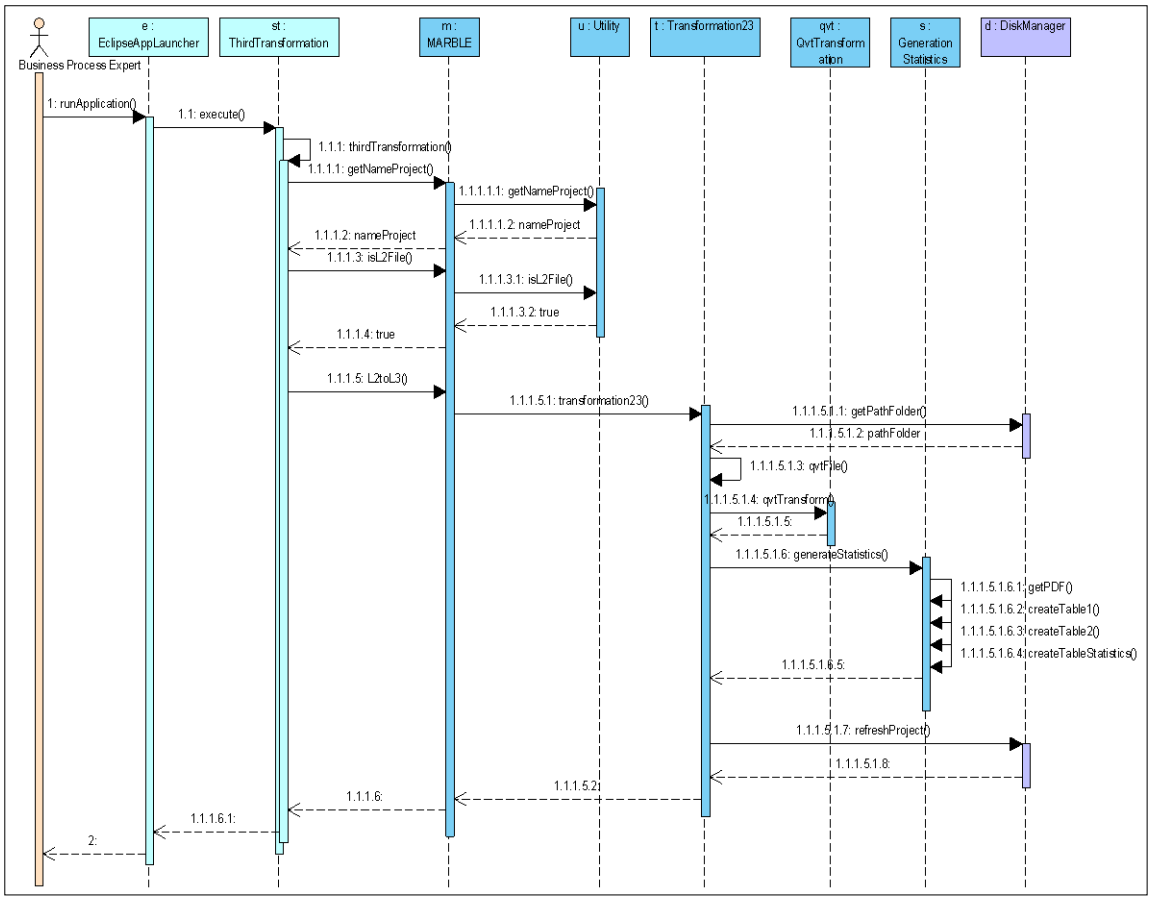


Figura 5.98. Diagrama de secuencia iteración 5 (CdU9)

5.5.2.3. Prototipos de las GUIs en la iteración 5

A continuación se muestra el prototipo de interfaz de usuario perteneciente a la funcionalidad del caso de uso CdU9 (Figura 5.99 y Figura 5.100). Esta parte corresponde al producto de salida PS. 5.5.

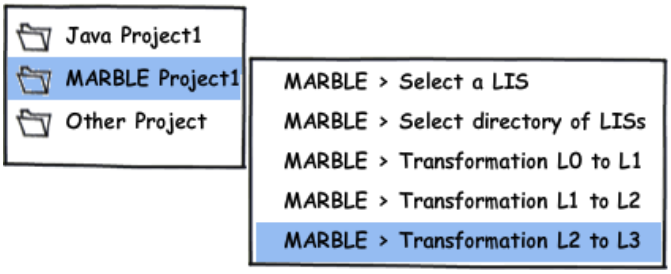


Figura 5.99. Prototipo de GUI de CdU9. Menú contextual

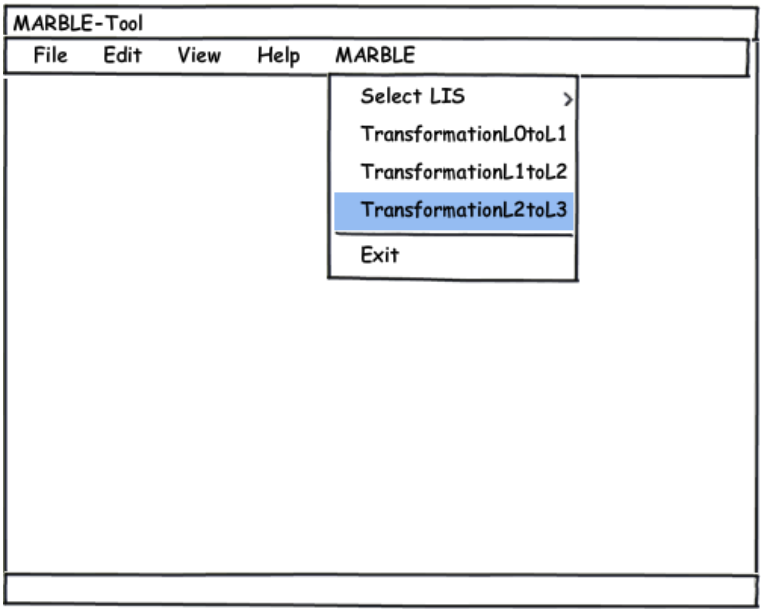


Figura 5.100. Prototipo de GUI de CdU9. Barra de menú

5.5.3. Implementación

En esta parte sólo se comentarán los aspectos más destacables de la implementación y los problemas encontrados durante su desarrollo.

Por tanto, para realizar la implementación de los casos de uso de esta iteración (CdU7 y CdU8) se han realizado las siguientes acciones:

a) Crear un comando para realizar la segunda transformación y añadir la opción al menú MARBLE y al menú contextual MARBLE de la misma forma que las opciones anteriores.

b) El CdU8 no es necesario implementarlo ya que se trata de una funcionalidad (editar un proyecto) que heredan todos los plug-in de Eclipse. Por este motivo, se asume la integración de estas características en el diseño del caso de uso CdU8.

Para implementar la segunda transformación (de modelo de código a modelo KDM) se ha utilizado un parser de Java generado mediante JavaCC (correspondiente al producto de salida PS. 5.6) y un conjunto de transformaciones QVTr.

Como apoyo a la implementación en esta iteración se han utilizado una librería existente y un editor existente, de forma que el diagrama de componentes queda de la forma mostrada en Figura 5.101, donde se han añadido los siguientes componentes:

- **org.omg.kdm**: Corresponde al conjunto de librerías de KDM.
- **org.omg.kdm.editor**: Editor utilizado como un plug-in de Eclipse para la visualización de los modelos KDM generados en la segunda transformación.

El plug-in resultante de añadir la extensión anterior corresponde al producto de salida PS. 5.7.

5.6. Iteración 6

Como se indica en la Tabla 4.7, en esta iteración se han generado los productos de salida que se detallan a continuación agrupados según el flujo de trabajo al que pertenecen.

5.6.1. Análisis

Como en iteraciones anteriores, para realizar el análisis de los casos de uso de la iteración 6 se ha optado por la utilización de diagramas de secuencia de análisis.

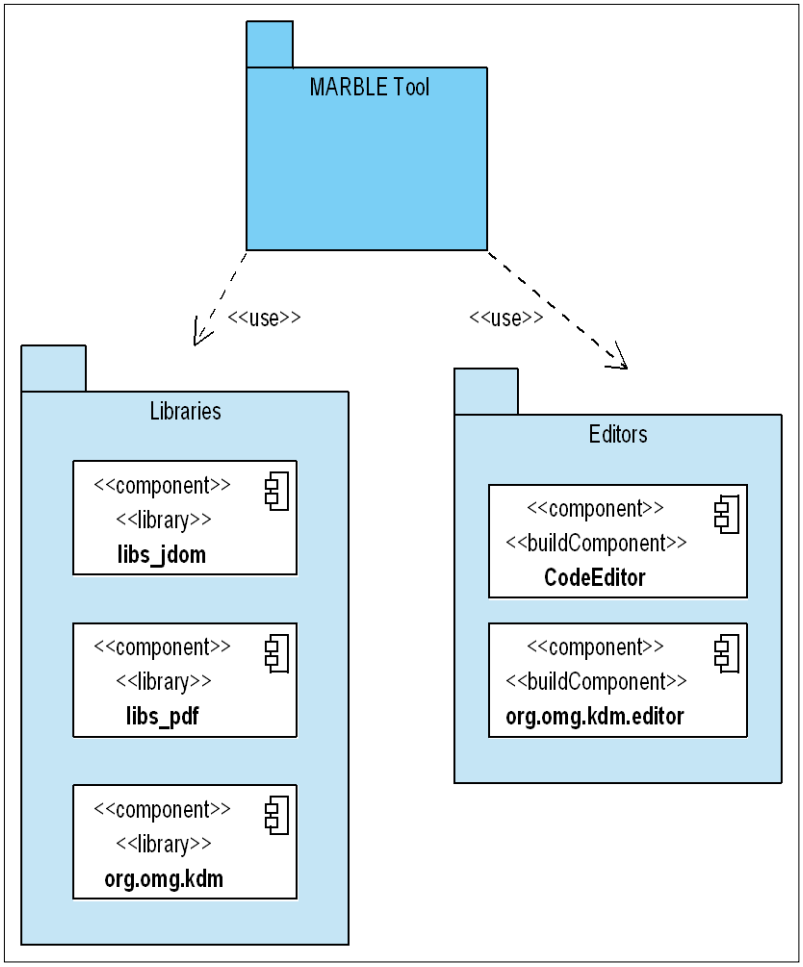


Figura 5.101. Diagrama de componentes (II)

5.6.1.1. Diagramas de Secuencia de Análisis en la iteración 6

Para comprender en mayor medida la interacción del caso de uso con los roles del sistema se muestran a continuación los diagramas de secuencia para el caso de uso contemplado en esta iteración. Este tipo de diagrama permite mostrar la interacción a lo largo del tiempo de los usuarios con el sistema. Se muestran al menos dos tipos de escenarios: escenario principal y escenario alternativo. Esta parte corresponde al producto de salida PS. 6.1 (véase Tabla 4.7).

1. CdU10. View Business Process Diagram

a) Escenario Principal

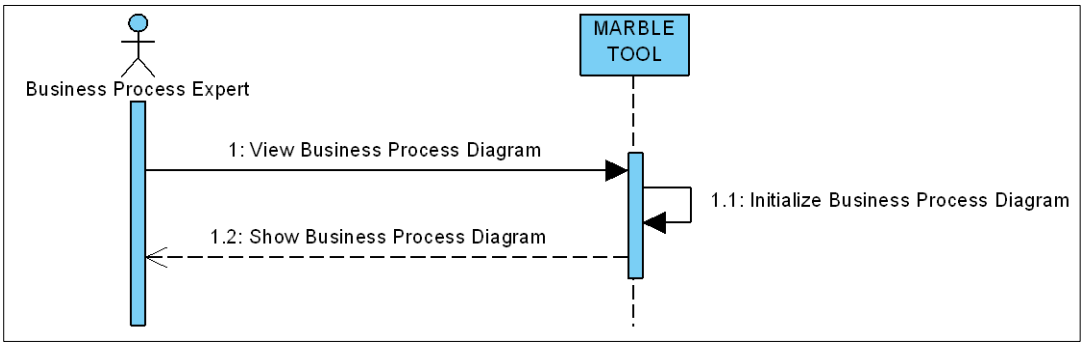


Figura 5.102. Diagrama de secuencia CdU10. Escenario Principal

b) Escenario alternativo de Error

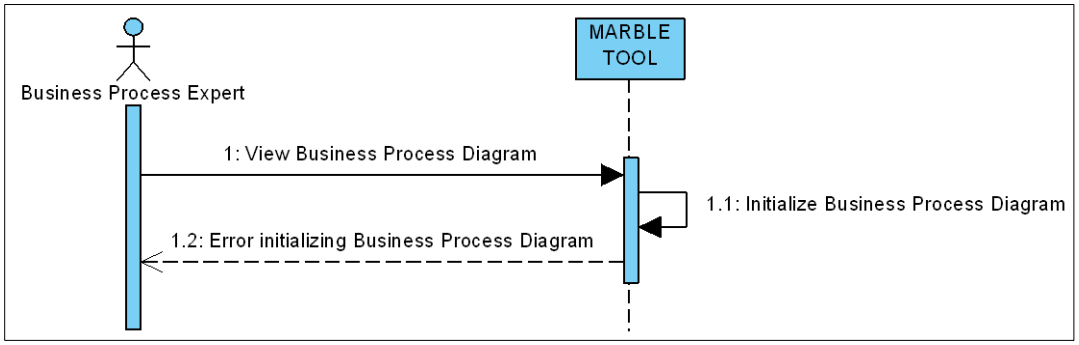


Figura 5.103. Diagrama de secuencia CdU10. Escenario de Error

5.6.2. Diseño

Para realizar el diseño del caso de uso seleccionado no se han seguido los pasos de PUD sino que se ha seguido los pasos propuestos por EMF/GMF (véase Figura 3.14) para el diseño de un editor gráfico. En esta iteración se obtienen los siguientes productos de salida:

- **PS. 6.3. Desarrollo del modelo de dominio (.ecore).**

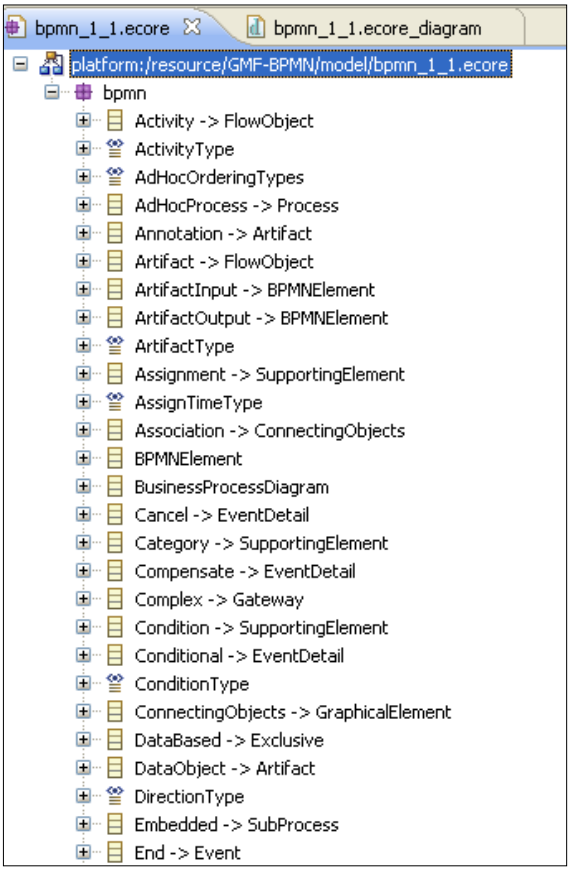


Figura 5.104. Fragmento del modelo de dominio (bpmn_1_1.ecore)

- **PS. 6.4. Desarrollo del modelo de definición gráfica (.gmfgraph)**

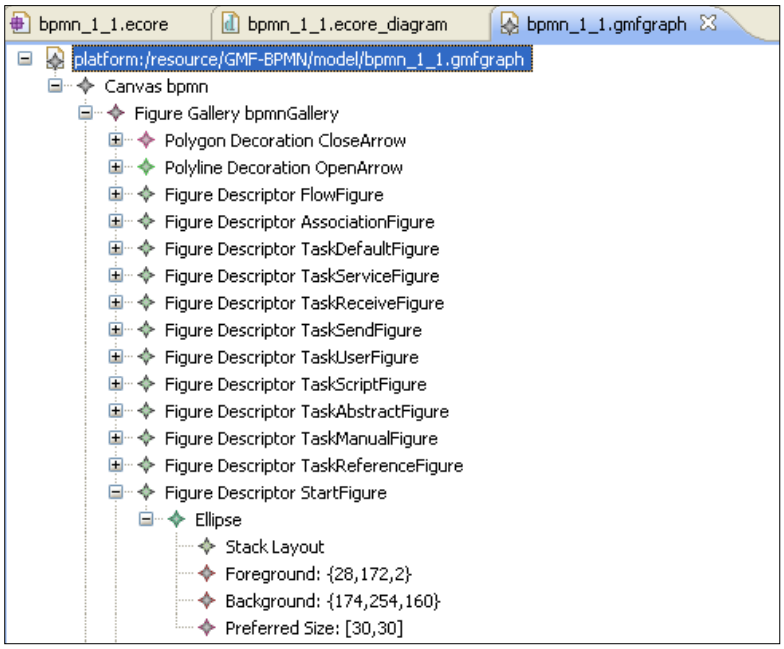


Figura 5.105. Fragmento del modelo de definición gráfica (bpmn_1_1.gmfgraph)

- **PS. 6.5. Desarrollo del modelo de definición de la herramienta (.gmftool)**

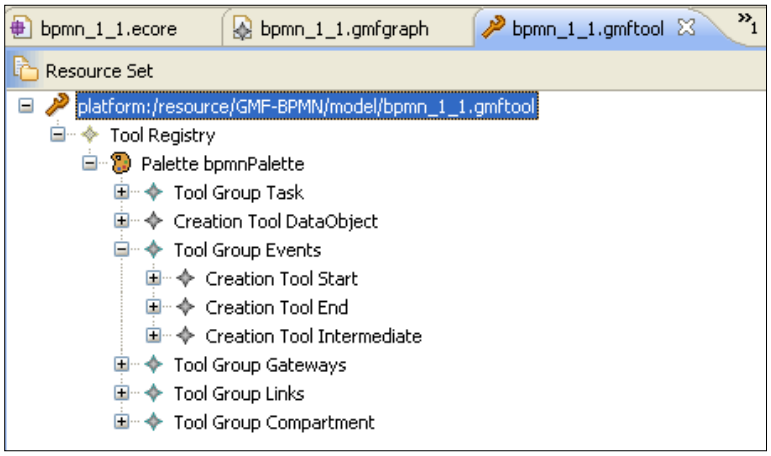


Figura 5.106. Modelo de definición de la herramienta (bpmn_1_1.gmftool)

5.6.2.1. Prototipos de las GUIs en la iteración 6

A continuación se muestra el prototipo de interfaz de usuario perteneciente a la funcionalidad del caso de uso CdU10 (Figura 5.107). Esta parte corresponde al producto de salida PS. 6.2 (véase Tabla 4.7).

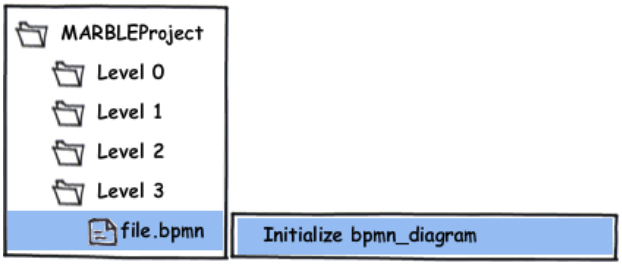


Figura 5.107. Prototipo de GUI de CdU10. Menú contextual

5.6.3. Implementación

En esta parte sólo se comentarán los aspectos más destacables de la implementación y los problemas encontrados durante su desarrollo.

Al tratarse de un plug-in de Eclipse la forma de implementar los casos de uso es a través de extensiones, las cuales son vinculadas a las clases que han sido definidas en la etapa de diseño y aportan toda la funcionalidad a dicha extensión.

Por tanto, para realizar la implementación de los casos de uso de esta iteración (CdU9 y CdU11) se han realizado las siguientes acciones:

a) Crear un comando para realizar la tercera transformación y añadir la opción al menú MARBLE y al menú contextual MARBLE de la misma forma que las opciones anteriores.

b) El CdU11 no es necesario implementarlo ya que se trata de una funcionalidad (editar un proyecto) que heredan todos los plug-in de Eclipse.

Para implementar la tercera transformación (de modelo KDM a modelo de procesos de negocio) se han utilizado un conjunto de transformaciones QVTr que representan patrones de negocio que se han creado (correspondientes al PS. 6.6). El código del script QVTr de la transformación entre modelos KDM y BPMN aplicando una serie de patrones de negocio pude consultarse en el Anexo IV. Script QVT.

Como apoyo a la implementación en esta iteración se han utilizado una librería existente y el editor desarrollado en el CdU10, de forma que el diagrama de componentes quedaría de la forma mostrada en Figura 5.108, donde se han añadido los siguientes componentes:

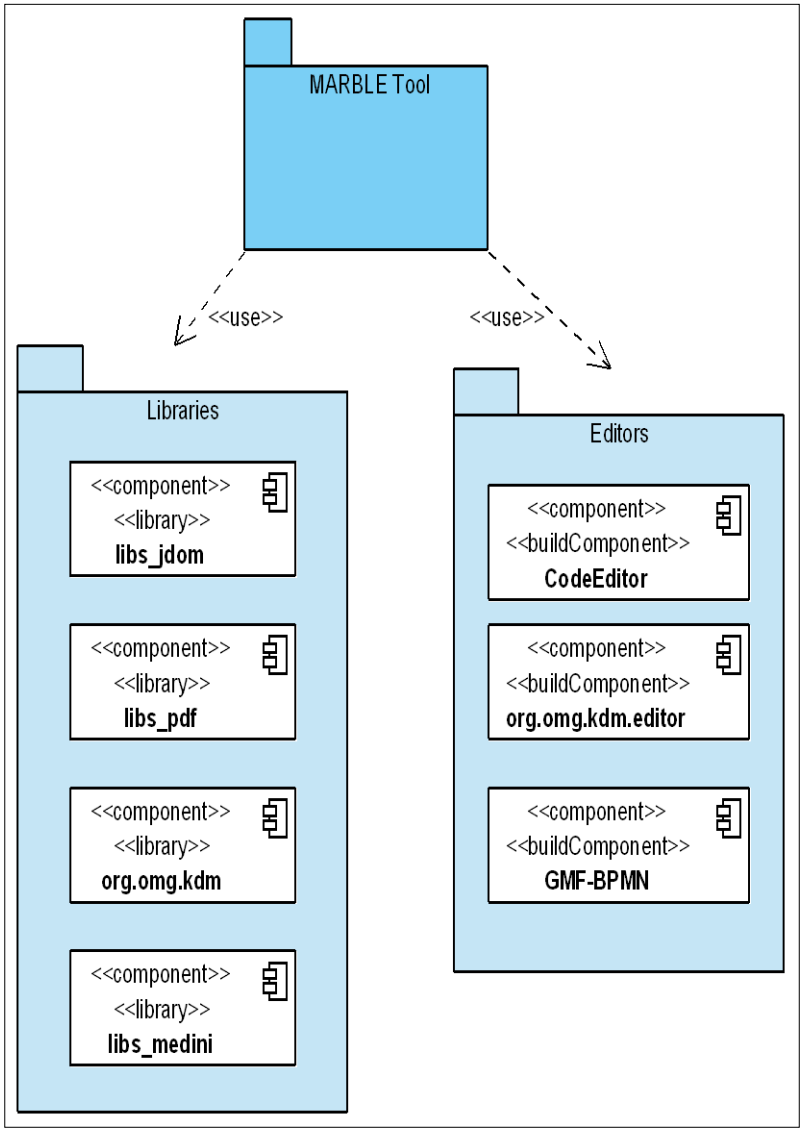


Figura 5.108. Diagrama de componentes (final)

- **libs_medini**: Corresponde al conjunto de librerías proporcionadas por medini QVT.
- **GMF-BPMN**: Editor desarrollado como parte del CdU10.

El plug-in resultante de añadir la extensión anterior corresponde al producto de salida PS. 6.7.

5.7. Iteración 7

Como se indica en la Tabla 4.8, en esta iteración se han generado los productos de salida que se detallan a continuación, agrupados según el flujo de trabajo al que pertenecen.

5.7.1. Análisis

Para realizar el análisis de los casos de uso de la iteración 7 se ha optado por la utilización de diagramas de secuencia y diagramas de comunicación.

5.7.1.1. Diagramas de Secuencia de Análisis en la iteración 7

Para comprender en mayor medida la interacción de los casos de uso con los roles del sistema se muestran a continuación los diagramas de secuencia para los caso de uso contemplados en esta iteración (CdU12 y CdU13). Para cada uno de los casos de uso se muestran al menos dos tipos de escenarios: escenario principal y escenario alternativo. Esta parte corresponde al producto de salida PS. 7.1 (véase Tabla 4.8).

1. **CdU12. Delete MARBLE Project**

   a) **Escenario principal**

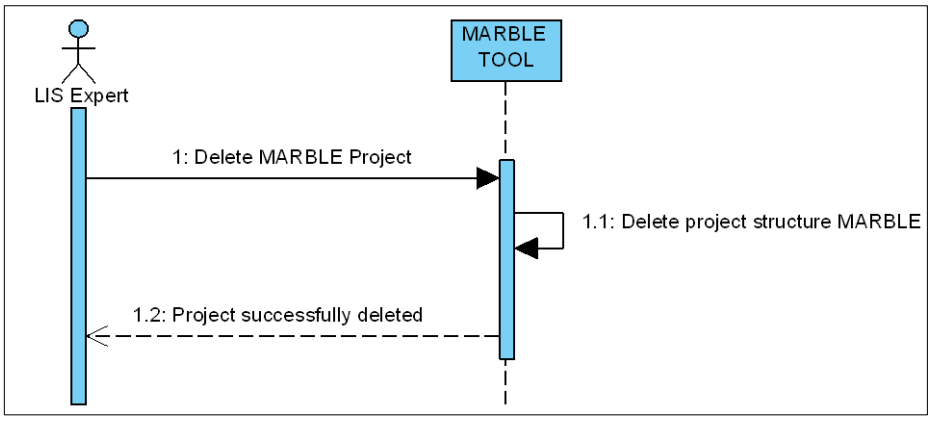


Figura 5.109. Diagrama de secuencia CdU12. Escenario Principal

b) Escenario alternativo de error

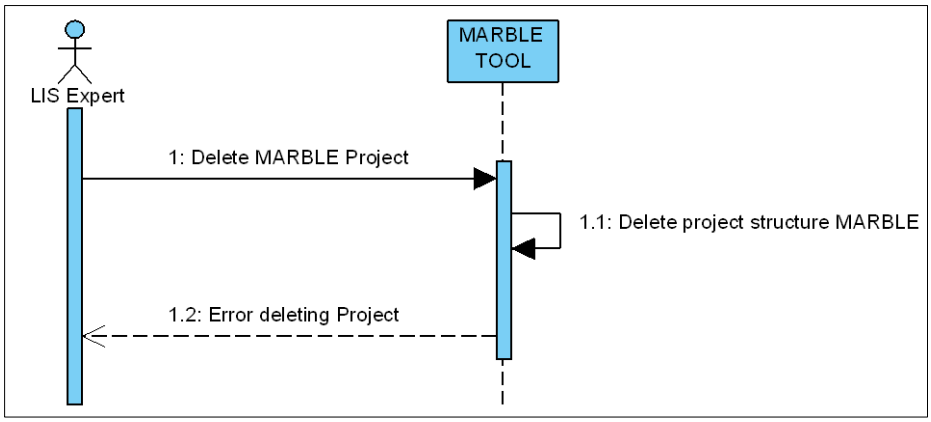


Figura 5.110. Diagrama de secuencia CdU12. Escenario de Error

2. CdU13. Configure transformation environment

a) Escenario principal

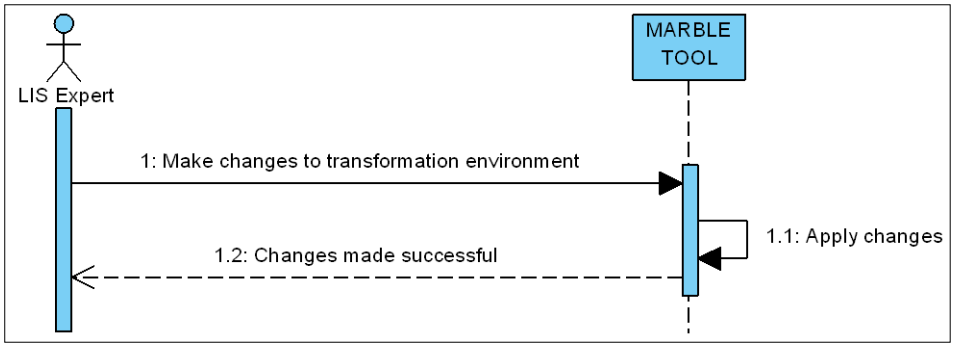


Figura 5.111. Diagrama de secuencia CdU13. Escenario Principal

b) Escenario alternativo 1

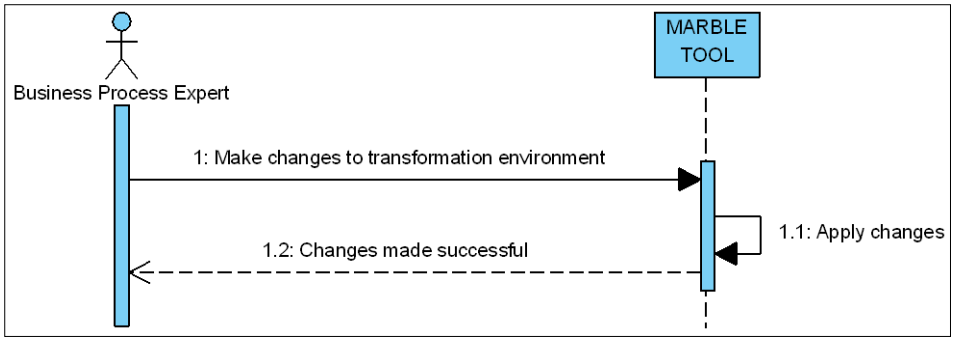


Figura 5.112. Diagrama de secuencia CdU13. Escenario Alternativo

c) Escenario alternativo 2 de error

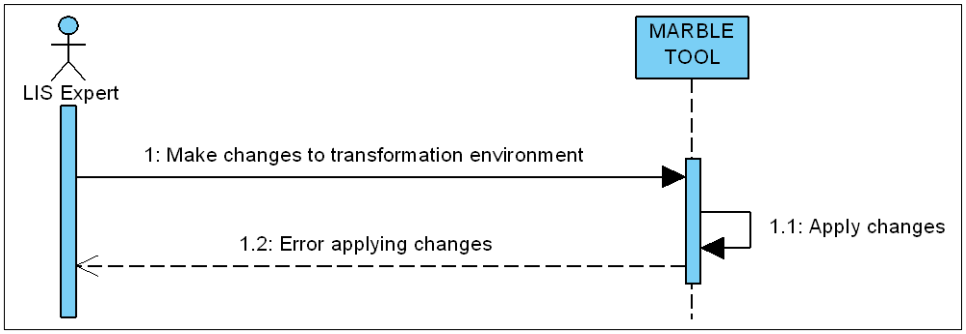


Figura 5.113. Diagrama de secuencia CdU13. Escenario de Error (I)

d) Escenario alternativo 3 de error

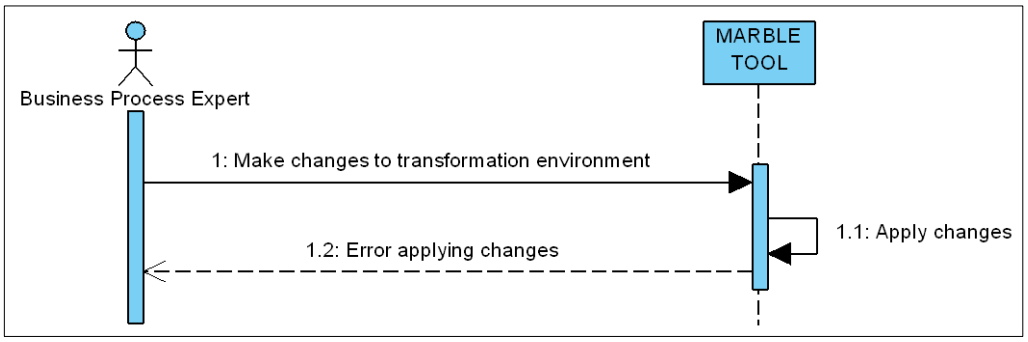


Figura 5.114. Diagrama de secuencia CdU13. Escenario de Error (II)

5.7.1.2. Diagramas de Comunicación en la iteración 7

Las figuras Figura 5.115 y Figura 5.116 muestran los diagramas de comunicación correspondientes a los casos de uso CdU12 y CdU13. Esta parte corresponde al producto de salida PS. 7.2.

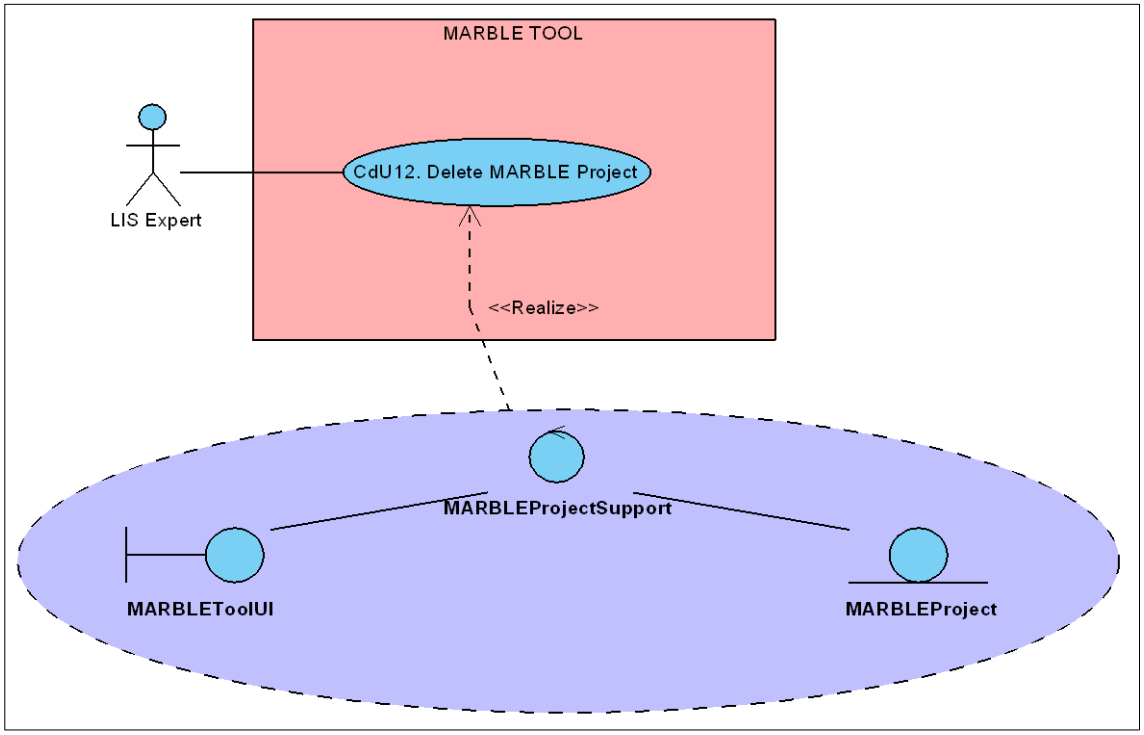


Figura 5.115. Diagrama de comunicación de CdU12

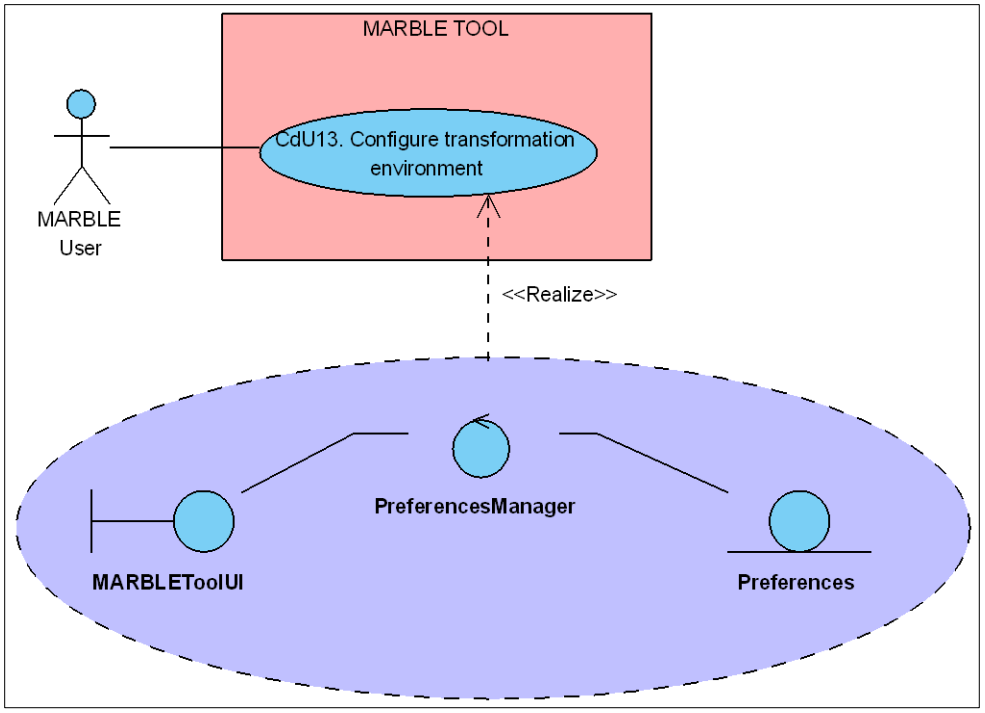


Figura 5.116. Diagrama de comunicación de CdU13

5.7.2. Diseño

Al realizar la herramienta como un plug-in de Eclipse, se heredan las funciones comunes de Eclipse, entre ellas la de borrar un proyecto independientemente del tipo que sea. Por este motivo, se asume la integración de estas características en el diseño del caso de uso CdU12, y en cuanto a su implementación y posteriores pruebas es sólo necesario realizar su integración.

Respecto al caso de uso CdU13, su funcionalidad será diseñada como una extensión del plug-in relativa a las preferencias. Los plug-in de Eclipse permiten extender la funcionalidad propia de Eclipse (en este caso la ventana de preferencias) con nuevas opciones. Por esta razón, no se muestra el diagrama de secuencia pero sí el diagrama de clases con las clases añadidas para dar soporte a la funcionalidad del CdU13.

5.7.2.1. Diagrama de Clases del sistema en la iteración 7

El diagrama de clases correspondiente al diseño de los casos de uso CdU13 se muestra de forma colapsada en la Figura 5.117. A continuación, se detallan las clases involucradas en el diagrama. Sólo se detallan las clases nuevas o aquellas que han sufrido cambios en esta iteración respecto a la iteración anterior. Esta parte corresponde al producto de salida PS. 7.3.

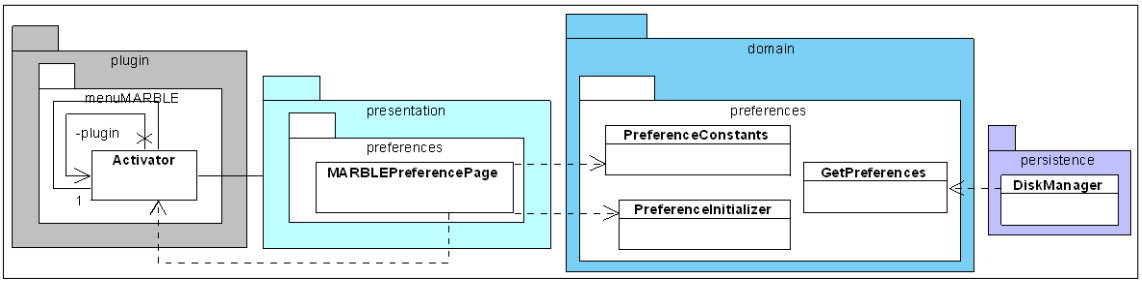


Figura 5.117. Diagrama de clases iteración 7

MARBLEPreferencePage

La Figura 5.118 muestra el diagrama UML correspondiente a la clase MARBLEPreferencePage. Esta clase está contenida en la capa de presentación, en el paquete de preferencias (*preferences*). La responsabilidad de esta clase será la de

proporcionar una interfaz de usuario para realizar la configuración de aspectos de las transformaciones.

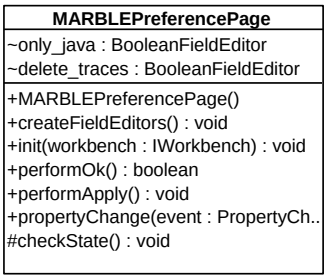


Figura 5.118. Diagrama de clases iteración 7: Clase MARBLEPreferencePage

PreferenceConstants

La Figura 5.119 muestra el diagrama UML correspondiente a la clase PreferenceConstants. Esta clase está contenida en la capa de dominio, en el paquete de preferencias (*preferences*). La responsabilidad de esta clase será la de proporcionar las constantes relativas a las preferencias.

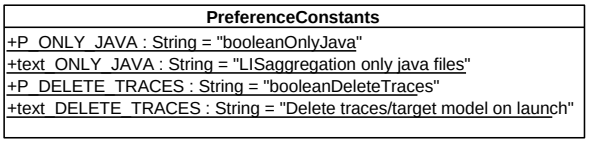


Figura 5.119. Diagrama de clases iteración 7: Clase PreferenceConstants

PreferenceInicializer

La Figura 5.120 muestra el diagrama UML correspondiente a la clase PreferenceInicializer. Esta clase está contenida en la capa de dominio, en el paquete de preferencias (*preferences*). La responsabilidad de esta clase será la de inicializar los valores de las preferencias.

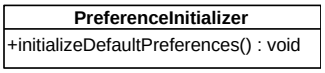


Figura 5.120. Diagrama de clases iteración 7: Clase PreferenceInicializer

GetPreferences

La Figura 5.121 muestra el diagrama UML correspondiente a la clase GetPreferences. Esta clase está contenida en la capa de dominio, en el paquete de

preferencias (*preferences*). La responsabilidad de esta clase será la de proporcionar el valor de las preferencias.

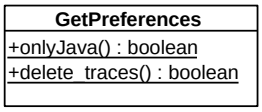


Figura 5.121. Diagrama de clases iteración 7: Clase GetPreferences

5.7.2.2. Prototipos de las GUIs en la iteración 7

A continuación se muestra los prototipos de interfaz de usuario perteneciente a la funcionalidad del caso de uso CdU12 (Figura 5.122), CdU13 (Figura 5.123). Esta parte corresponde al producto de salida PS. 7.5.

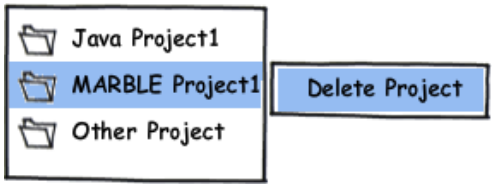


Figura 5.122. Prototipo de GUI de CdU12

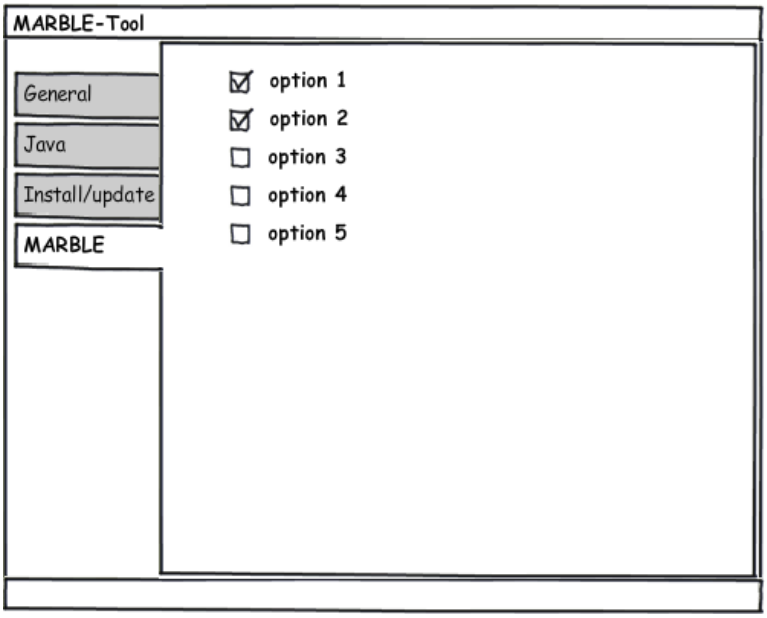


Figura 5.123. Prototipo de GUI de CdU13

5.7.3. Implementación

Para realizar la implementación del caso de uso seleccionado no se han seguido los pasos de PUD sino que se ha seguido los pasos propuestos por EMF/GMF (véase Figura 3.14) para la implementación de un editor gráfico. En esta iteración se obtienen los siguientes productos de salida:

- **PS. 7.6. Generación de código del modelo EMF (.genmodel)**

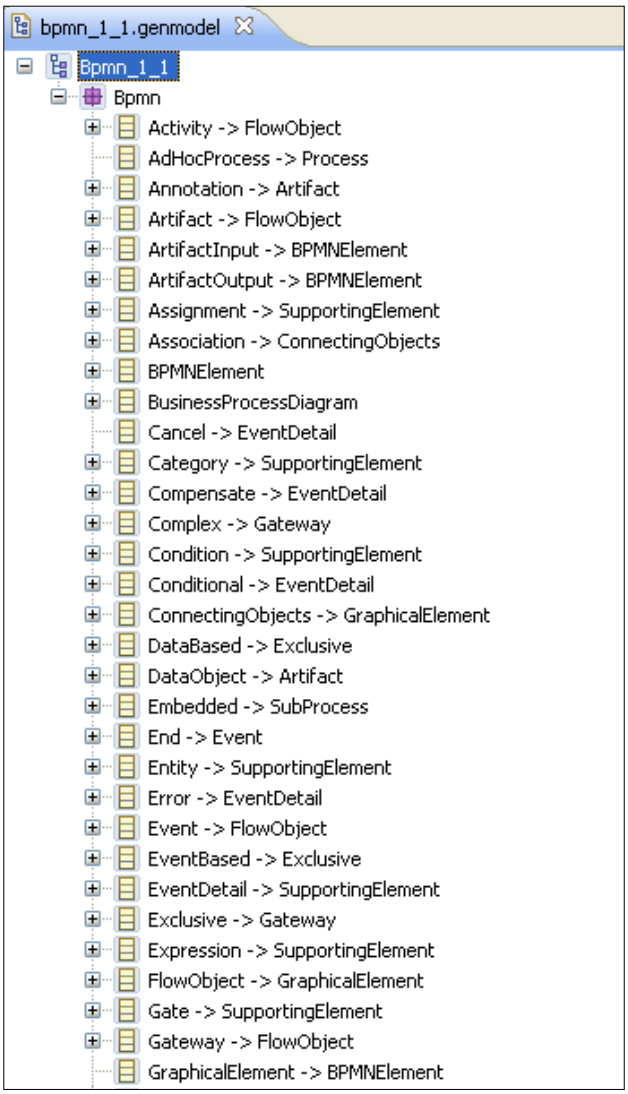


Figura 5.124. Fragmento del generador de código bpmn_1_1.genmodel

- **PS. 7.7. Especificación de la correspondencia (.gmfmap)**

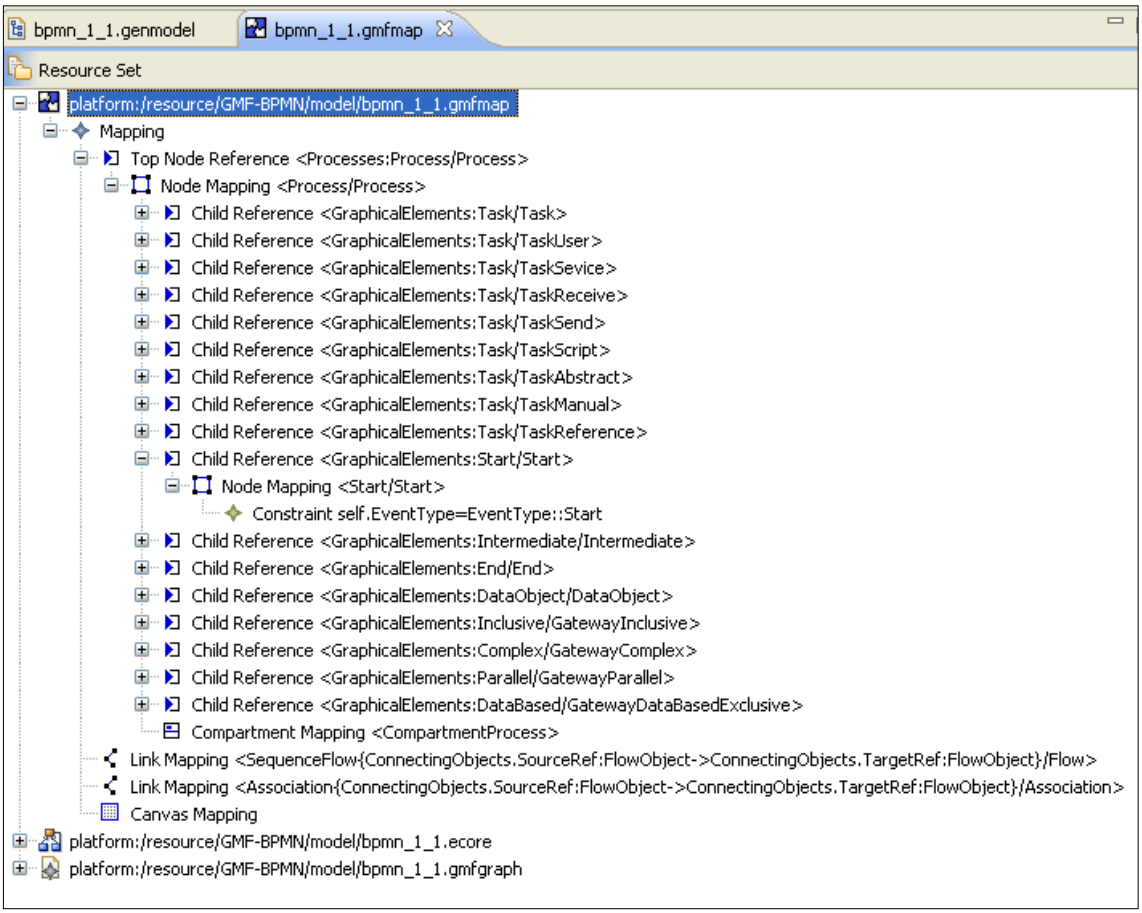


Figura 5.125. Fragmento de la especificación de correspondencia

- **PS. 7.8. Generación del modelo generador (.gmfgen)**

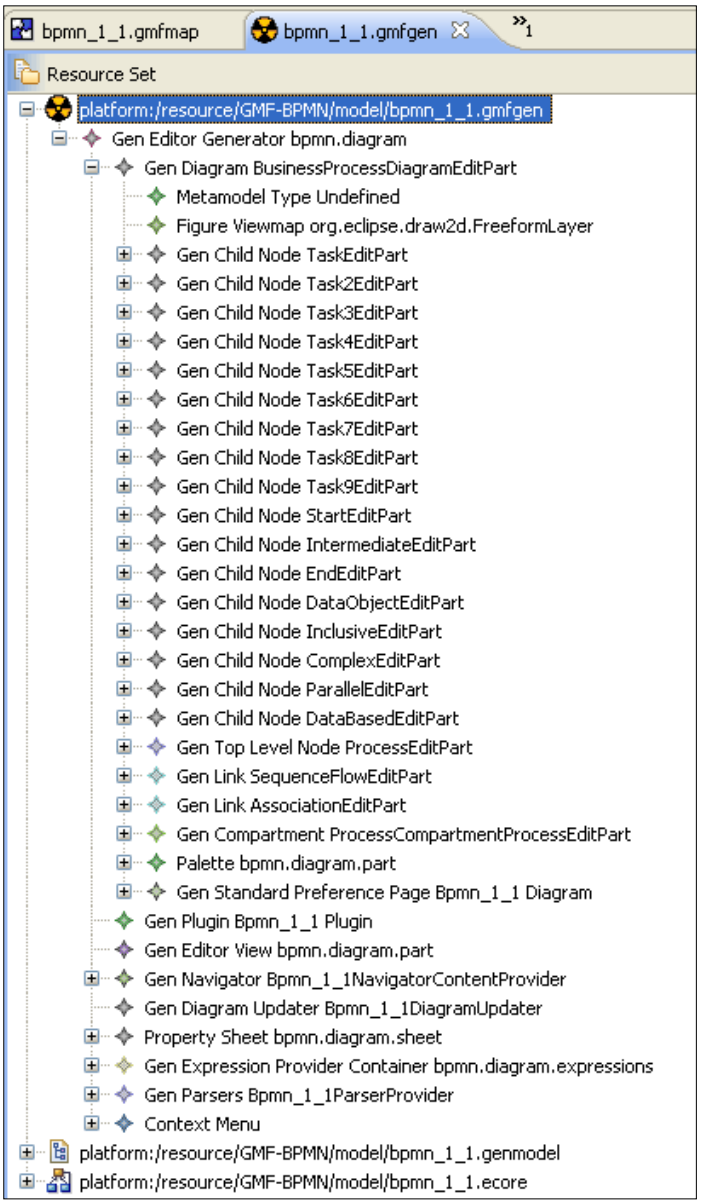


Figura 5.126. Fragmento del modelo generador

Una vez que creado el metamodelo con EMF en la iteración anterior, se pasa al siguiente paso de desarrollo del editor utilizando GMF que es la generación del código del modelo. Tras la generación de código, automática, se genera la estructura de directorios mostrada en la Figura 5.127.

Este editor corresponde al producto de salida PS. 7.9 y sirve para visualizar los modelos generados por el CdU9 y mostrar la información de forma gráfica.

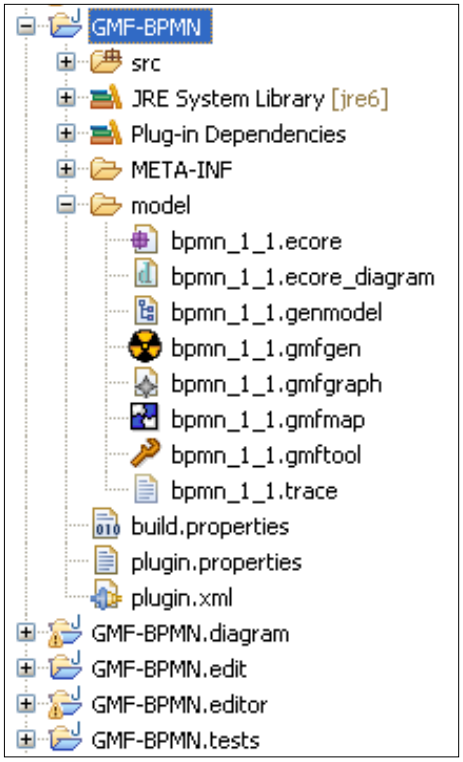


Figura 5.127. Estructura de directorios de editor construido con EMF/GMF

5.7.4. Pruebas

La etapa de pruebas consiste en el diseño y ejecución de las pruebas relativas a las partes construidas en el flujo de trabajo de implementación con el fin de verificar y validar el trabajo realizado. Las pruebas que son realizadas en este proyecto son tres tipos: pruebas unitarias, pruebas de integración y pruebas de aceptación.

En concreto en la iteración 7 se comienzan a realizar las pruebas unitarias de los primeros casos de uso, implementados en iteraciones anteriores. Los resultados de las pruebas realizadas en esta iteración se muestran a continuación.

5.7.4.1. Pruebas unitarias en la iteración 7

Las pruebas unitarias han consistido en la implementación de casos de prueba siguiendo un enfoque de *"caja negra"*. Cada uno de ellos prueba una parte o módulo del sistema implementado por separado. En sucesivas iteraciones del proyecto se añaden nuevos casos de prueba unitarios que prueban una parte de la herramienta implementada. A la vez se sigue comprobando los test previos para observar si las

nuevas funcionalidades implementadas introdujeron errores colaterales. De esta forma se realizaron pruebas unitarias de forma incremental agrandando el *test suite*.

Para realizar la ejecución de las pruebas unitarias se ha utilizado el plug-in de Eclipse *JUnit*, específico para el lenguaje de programación Java. Por otra parte, para evaluar la cobertura de las pruebas realizadas se ha utilizado el plug-in de Eclipse *Eclemma*, el cual soporta las ejecuciones de *JUnit*.

Sólo se han realizado pruebas unitarias de la capa de dominio y persistencia ya que son las más susceptibles a errores. Los resultados de las pruebas se muestran a continuación y corresponden al producto de salida PS. 7.10 (véase Tabla 4.8). La realización de las pruebas ha seguido el siguiente orden:

1. **CdU2. Create from scratch (Crear desde cero)**

Para probar este caso de uso se ha creado un *test case* que prueba su funcionalidad. Los resultados de las pruebas han sido los mostrados en la Figura 5.132. La cobertura de las pruebas para este caso de uso se muestra en la Figura 5.128, la cual indica que se alcanza una cobertura del 92,3% en el paquete domain/creationMARBLEProject/newFromScratch, el cual contiene las clases que se han pretendido probar.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
domain.creationMARBLEProject.newFromScratch	92,3 %	169	14	183
MARBLEProjectSupport.java	88,7 %	86	11	97
ProjectNatureMARBLE.java	62,5 %	5	3	8
PerspectiveMARBLE.java	100,0 %	78	0	78

Figura 5.128. Cobertura de las pruebas unitarias de CdU2

2. **CdU4. Select LIS (Legacy source code) (Seleccionar Sistema de Información heredado)**

Para probar este caso de uso se ha creado un *test case* que prueba su funcionalidad. Los resultados de las pruebas han sido los mostrados en la Figura 5.132. La cobertura de las pruebas para este caso de uso se muestra en la Figura 5.129, la cual indica que se alcanza una cobertura del 93,3% en el paquete domain/LISaggregation, el cual contiene las clases que se han pretendido probar.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
domain.LISaggregation	93,3 %	42	3	45
LISaggregation.java	93,3 %	42	3	45

Figura 5.129. Cobertura de las pruebas unitarias de CdU4

El motivo por el que las pruebas del CdU4 son realizadas antes que las pruebas CdU3 es que el CdU3 hace uso de los casos de uso CdU2 y CdU4 y, por tanto, estos dos casos de uso deben ser implementados y probados antes.

3. CdU3. Create from existing project (Crear desde proyecto existente)

Para probar este caso de uso se ha creado un *test case* que prueba su funcionalidad. Los resultados de las pruebas han sido los mostrados en la Figura 5.132. La cobertura de las pruebas para este caso de uso se muestra en la Figura 5.130, la cual indica que se alcanza una cobertura del 80,6% en el paquete domain/creationMARBLEProject/newFromExistingProject, el cual contiene las clases que se han pretendido probar.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
domain.creationMARBLEProject.newFromExistingProject	80,6 %	25	6	31
ExtensionToMARBLEProject.java	80,6 %	25	6	31

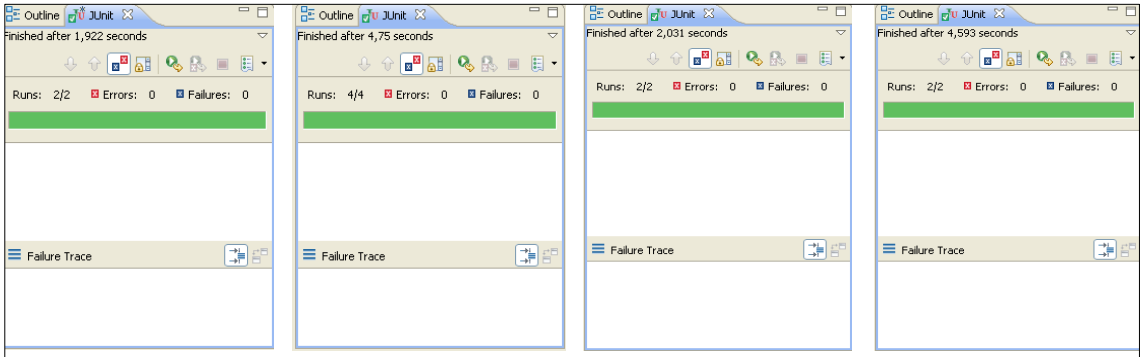
Figura 5.130. Cobertura de las pruebas unitarias de CdU3

4. CdU5. Generate source code model (Generar modelo de código fuente)

Para probar este caso de uso se ha creado un *test case* que prueba su funcionalidad. Los resultados de las pruebas han sido los mostrados en la Figura 5.132. La cobertura de las pruebas para este caso de uso se muestra en la Figura 5.131, la cual indica que se alcanza una cobertura del 99,6% en el paquete domain/transformation01, el cual contiene las clases que se han pretendido probar.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
domain.transformation01	99,6 %	721	3	724
Transformation01.java	99,6 %	721	3	724

Figura 5.131. Cobertura de las pruebas unitarias de CdU5



pruebas unitarias de CdU2 | pruebas unitarias de CdU4 | pruebas unitarias de CdU3 | pruebas unitarias de CdU5

Figura 5.132. Resultados de JUnit (I)

5.8. Iteración 8

Como se indica en la Tabla 4.9, en esta iteración se han generado los productos de salida que se detallan a continuación agrupados según el flujo de trabajo al que pertenecen, i.e., implementación o pruebas en este caso.

5.8.1. Implementación

Para realizar la implementación de los casos de uso de esta iteración (CdU12 y CdU13) se han realizado las siguientes acciones:

a) Crear una nueva página en la ventana de preferencias con las preferencias de las transformaciones de la forma mostrada en Fragmento de código 5.8.

```
<extension
      point="org.eclipse.ui.preferencePages">
   <page
         class="presentation.preferences.MARBLEPreferencePage"
         id="plugin.menumarble.preferences.MARBLEPreferencePage"
         name="MARBLE">
   </page>
</extension>
```

Fragmento de código 5.8. Extensión org.eclipse.ui.preferencePages

El CdU12 no es necesario implementarlo ya que se trata de una funcionalidad (borrar un proyecto) que heredan todos los plug-in de Eclipse, por lo que únicamente sería necesaria su integración con la herramienta.

El plug-in resultante de añadir la extensión anterior corresponde al producto de salida PS. 8.2 (véase Tabla 4.9).

5.8.2. Pruebas

En esta iteración se continúa con la realización de pruebas unitarias. Los resultados de las pruebas se muestran a continuación.

5.8.2.1. Pruebas unitarias en la iteración 8

La construcción de las pruebas unitarias ha seguido la misma estructura que en la iteración anterior. Sólo se han realizado pruebas unitarias de la capa de dominio y persistencia. Los resultados de las pruebas se muestran a continuación y corresponden al producto de salida PS. 8.3 (véase Tabla 4.9). La realización de las pruebas ha seguido el siguiente orden:

1. **CdU7. Integrate source model into KDM repository**

Para probar este caso de uso se ha creado un *test case* que prueba su funcionalidad. Los resultados de las pruebas han sido los mostrados en la Figura 5.135. La cobertura de las pruebas para este caso de uso se muestra en la Figura 5.133, la cual indica que se alcanza una cobertura del 96,5% en el paquete domain/transformation12, el cual contiene las clases que se han pretendido probar.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
domain.transformation12	96,5 %	164	6	170
Transformation12.java	96,5 %	164	6	170
domain.transformation12.javaParser	51,9 %	12720	11772	24492

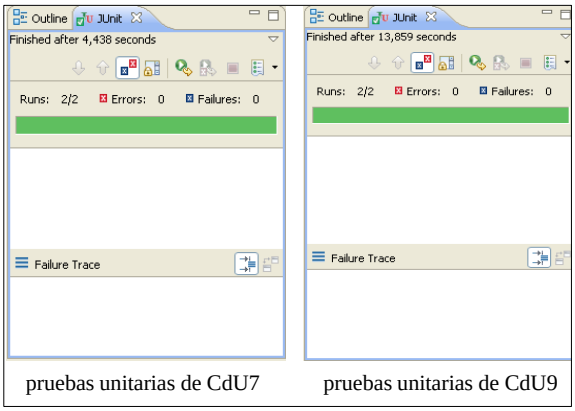
Figura 5.133. Cobertura de las pruebas unitarias de CdU7

2. **CdU9. Generate Business Process (model) from KDM model**

Para probar este caso de uso se ha creado un *test case* que prueba su funcionalidad. Los resultados de las pruebas han sido los mostrados en la Figura 5.135. La cobertura de las pruebas para este caso de uso se muestra en la Figura 5.134, la cual indica que se alcanza una cobertura del 96,7% en el paquete domain/transformation23, el cual contiene las clases que se han pretendido probar.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
domain.transformation23	96,7 %	384	13	397
QvtTransformation.java	96,4 %	269	10	279
Transformation23.java	97,5 %	115	3	118

Figura 5.134. Cobertura de las pruebas unitarias de CdU9



pruebas unitarias de CdU7 pruebas unitarias de CdU9

Figura 5.135. Resultados de JUnit (II)

5.9. Iteración 9

Como se indica en la Tabla 4.10, en esta iteración se han generado los productos de salida que se detallan a continuación.

5.9.1. Pruebas

En esta iteración se concluyen las pruebas unitarias y se da comienzo a las pruebas de integración y pruebas de aceptación del sistema implementado.

5.9.1.1. Pruebas unitarias en la iteración 9

La construcción de las pruebas unitarias ha seguido la misma estructura que en iteraciones anteriores. Sólo se han realizado pruebas unitarias de la capa de dominio y persistencia ya que son las más susceptibles a errores. Los resultados de las pruebas se muestran a continuación y corresponden al producto de salida PS. 9.1 (véase Tabla 4.10). El único caso de uso del cual se han realizado pruebas ha sido el CdU13. Esto es debido a que, como se dijo anteriormente, la funcionalidad del caso de uso CdU12 es heredada del propio Eclipse y, por tanto, no es necesario realizar pruebas de dicha funcionalidad.

5.9.1.2. Pruebas de integración en la iteración 9

Las pruebas de integración se encargan de verificar que todos los componentes construidos interaccionan de forma adecuada. Estas pruebas corresponden al producto de salida PS. 9.2 (véase Tabla 4.10).

Estas pruebas también son realizadas utilizando las herramientas utilizadas en las pruebas unitarias, es decir, *JUnit* y *Eclemma*. Los resultados de la cobertura se muestran en la Figura 5.136, en los cuales se puede apreciar que las clases relativas al dominio y persistencia tienen altos niveles en cuanto a la cobertura de sus pruebas y las clases relativas a la presentación tienen valores bajos, debido a que no se han realizado pruebas para dichas clases.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
MARBLE-Tool	52,3 %	15343	14010	29353
src	52,3 %	15343	14010	29353
domain.transformation12.javaParser	51,9 %	12720	11772	24492
presentation.actions	9,0 %	91	921	1012
test	41,5 %	384	541	925
domain.utilities	41,4 %	235	333	568
domain	0,0 %	0	246	246
persistence	82,5 %	348	74	422
presentation.preferences	0,0 %	0	65	65
domain.creationMARBLEProject.newFromScratch	92,3 %	169	14	183
domain.transformation23	96,7 %	384	13	397
domain.preferences	76,3 %	29	9	38
domain.creationMARBLEProject.newFromExistingProject	80,6 %	25	6	31
domain.transformation12	96,5 %	164	6	170
exceptions	75,0 %	12	4	16
domain.LISaggregation	93,3 %	42	3	45
domain.transformation01	99,6 %	721	3	724
plugin.menumarble	100,0 %	19	0	19

Figura 5.136. Cobertura de las pruebas de integración

5.9.1.3. Pruebas de aceptación en la iteración 9

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento. Esta parte corresponde al producto de salida PS. 9.3 (véase Tabla 4.10).

Como parte de las pruebas de aceptación se ha llevado a cabo un ejemplo de aplicación de la herramienta en un entorno empresarial. Dicho ejemplo de aplicación puede ser consultado en detalle en el Anexo III. Ejemplo de Aplicación en un Entorno Empresarial.

5.10. Iteración 10

Como se indica en la Tabla 4.11, en esta iteración se ha generado la presente memoria del Proyecto Fin de Carrera (correspondiente al producto de salida PS. 10.1) y se ha producido el plug-in resultante (correspondiente al producto de salida PS. 10.3) con la integración de todos los plug-ins y librerías anteriormente citadas. La herramienta se muestra en el CD que se adjunta para que pueda ser utilizada.

5.10.1. Manual de usuario

Tras el desarrollo de la herramienta se ha realizado un manual de usuario con las funciones principales de las que dispone. Esta parte corresponde al producto de salida PS 10.2 y es mostrado en detalle en el Anexo II. Manual de Usuario.

5.10.2. Distribución de la herramienta

Como último paso en la planificación realizada en el apartado 5.1.2.4 se muestra la distribución que se ha realizado de la herramienta, correspondiente al producto de salida PS. 10.4:

- MARBLE Tool se encuentra disponible para su descarga en la página web dedicada a la arqueología de procesos de negocio (Alarcos, 2011).
- Adicionalmente, MARBLE Tool se encuentra disponible para su descarga en el sitio oficial Eclipse Market Place (Eclipse, 2011b) en http://marketplace.eclipse.org/content/marble.
- Tras su distribución, MARBLE Tool se aplicó satisfactoriamente a diferentes sistemas de información heredados en producción. Entre esos sistemas destacan *CHES*, un sistema de evaluación oncológica implantado en hospitales de Austria; *Eadmin-Xunta*, un sistema de administración electrónica de Galicia; y *LabVillasante*, el cual se presenta a modo de ejemplo en el Anexo III.

6. CONCLUSIONES Y PROPUESTAS

Este capítulo discute las conclusiones obtenidas tras la realización del presente Proyecto Fin de Carrera y presenta el posible trabajo futuro que se plantea a partir de la finalización de este proyecto.

6.1. Conclusiones

El presente proyecto tenía como objetivo desarrollar una herramienta para la extracción y representación de los procesos de negocio a partir de los sistemas de información heredados siguiendo un enfoque dirigido por modelos. La técnica soportada ha sido concretamente MARBLE, la cual propone cuatro niveles de abstracción hasta llegar a los procesos de negocio y tres transformaciones entre los modelos de dichos niveles.

Como resultado, la herramienta desarrollada contribuye a la modernización del software al obtener los procesos de negocio embebidos ya que permite desarrollar nuevos sistemas de información alineados con los procesos de negocio que realmente son los que lleva a cabo una organización.

El hecho de que la herramienta haya sido desarrollada mediante un plug-in de Eclipse permite además su futura extensión con posibles mejoras y su integración con otras herramientas de modernización del software. Además, al aplicarse estándares, tales como KDM, BPMN y QVT, se facilita su adopción por parte de la industria de la ingeniería del software.

Tras la finalización del Proyecto Fin de Carrera **se puede considerar que se han satisfecho los objetivos marcados al inicio del mismo**. Como muestra de ello, a continuación se muestran los requisitos funcionales descritos al inicio del desarrollo junto a la justificación de que se han satisfecho cada uno de ellos.

Código	Descripción	OK	Justificación
RF.01	Permitir la creación de una estructura que albergue los modelos que se van a generar, siguiendo el enfoque de MARBLE, a partir del sistema de información heredado.	☑	Se ha creado una estructura "proyecto MARBLE" que alberga todos los modelos generados
RF.02	Permitir que la información generada sea persistente, es decir, que se aloje en algún mecanismo para su posterior recuperación.	☑	La información generada es alojada en el disco, al igual que cualquier proyecto de Eclipse
RF.03	Permitir incluir fácilmente los sistemas de información heredados en la estructura creada.	☑	La agregación de los sistemas de información heredados se realiza de manera cómoda mediante dos tipos de opciones: mediante la agregación de un único archivo y mediante la agregación de un directorio completo.
RF.04	Permitir el análisis de código fuente de un sistema de información existente a fin de reconocer ciertos elementos y estructuras del código que serán representadas en un modelo específico de plataforma (PSM) mediante las transformaciones propuestas por MARBLE (transformación entre el nivel 0 y nivel 1).	☑	La transformación entre dichos niveles se ha realizado con éxito, siendo capaz de obtener los modelos específicos de la plataforma.
RF.05	Permitir la integración del modelo de código en un repositorio KDM siguiendo el estándar ISO/IEC 19506 mediante las transformaciones propuestas por MARBLE (transformación entre nivel 1 y nivel 2).	☑	La transformación entre dichos niveles se ha realizado con éxito, siendo capaz de integrar los modelos anteriores en un repositorio KDM, alineados al estándar ISO/IEC 19506
RF.06	Diseñar patrones de reconocimiento de las estructuras de código que permitan el descubrimiento de actividades de negocio.	☑	Se han diseñado patrones que realizan el reconocimiento de las estructuras de código a fin de descubrir las actividades de negocio.
RF.07	Implementación declarativa de los patrones de reconocimiento mediante transformaciones de modelos a través del lenguaje QVT.	☑	Se han implementados dichos patrones a través del lenguaje declarativo QVTr (véase Anexo IV. Script QVT).
RF.08	Permitir descubrir los procesos de negocio mediante los patrones de reconocimiento a fin de obtener un modelo independiente de la plataforma (PIM).	☑	Se han obtenido los modelos independientes de la plataforma (modelos de procesos de negocio) a partir de los modelos anteriores.
RF.09	Permitir la representación gráfica de los procesos de negocio descubiertos a fin de que estos puedan ser usados en fases posteriores de proyectos de reingeniería y migración. Los procesos de negocio serán representados de acuerdo a la notación estándar BPMN 2.0.	☑	Se ha facilitado un editor gráfico para la representación de los procesos de negocio descubiertos.
RF.10	Permitir editar los modelos generados a fin de corregirlos o ampliarlos.	☑	Se ha permitido la edición de los modelos generados.
RF.11	Permitir la eliminación de los modelos generados, así como toda la estructura creada.	☑	Se ha permitido la eliminación de los modelos generados.
RF.12	Permitir generar informes sobre las estadísticas de las transformaciones realizadas.	☑	Se ha permitido la generación de informes.
RF.13	Permitir configurar las transformaciones entre modelos.	☑	Se ha permitido la opción de configurar las transformaciones anteriores.

Tabla 6.1. Cumplimiento de los requisitos funcionales iniciales tras la finalización del Proyecto Fin de Carrera

6.2. Contraste de resultados

El trabajo presentado en esta memoria ha sido parte de varias publicaciones científicas, en las que se pretendía presentar a la comunidad los avances tanto de la técnica MARBLE como la viabilidad a nivel empresarial de la herramienta desarrollada, como un medio de transferencia tecnológica. Los artículos a los que se hace referencia son los siguientes:

Título	**"MARBLE"**
Autores	María Fernández-Ropero, Ricardo Pérez-Castillo, Ignacio García-Rodríguez de Guzmán, Mario Piattini
Publicación	XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2011). Sesión temática: Desarrollo de Software Dirigido por Modelos (DSDM).
Fecha	5 – 7 de Septiembre, 2011
Lugar	A Coruña, España
Estado	Aceptado. Presentado.

Tabla 6.2. Artículo JISBD'11

Título	**"MARBLE. A Business Process Archeology Tool"**
Autores	Ricardo Pérez-Castillo, María Fernández-Ropero, Ignacio García-Rodríguez de Guzmán and Mario Piattini
Publicación	The 27th IEEE International Conference on Software Maintenance (ICSM 2011)
Fecha	September 25 – 30, 2011
Lugar	Williamsburg, Virginia, USA
Estado	Aceptado. Pendiente de presentación.

Tabla 6.3. Artículo ICSM'11

6.3. Líneas de trabajo futuras

Siguiendo la línea de desarrollo del proyecto surgen varias líneas de trabajo futuras por las que continuar trabajando en MARBLE Tool:

- Mejorar la técnica de extracción de procesos de negocio mediante la mejora de las técnicas utilizadas de tal forma que se reduzca la intervención a posteriori de los expertos de negocio y que los modelos obtenidos sean más precisos.
- Proporcionar el análisis de sistemas de información heredados basados o implementados en otras tecnologías y/o lenguajes de programación comunes en la industria.

- Obtener una versión de la herramienta ofrecida como aplicación web, ofreciendo además el almacenamiento de la información relativa a los proyectos MARBLE en un servidor. De esta forma, MARBLE se adaptaría a las nuevas necesidades que surgen desde el campo de *Cloud Computing* y la información de los proyectos MARBLE no dependería de un dispositivo físico de almacenamiento ni sería necesario tener la herramienta instalada previamente en un ordenador local.

María de la Sierra Fernández Ropero, Septiembre 2011

7. REFERENCIAS

[1] 1T3XT-BVBA. (2011). "iText." de http://itextpdf.com/.

[2] Aalst, W. M. P. V. D., Hofstede, A. H. M. T. y Weske, M. (2003). Business process management: a survey. Proceedings of the 2003 international conference on Business process management. Eindhoven, The Netherlands, Springer-Verlag: 1-12.

[3] Alarcos. (2011). "Business process archeology." de http://www.businessprocessarcheology.org/.

[4] Analytics, K. (2010). "Knowledge Discovery Metamodel SDK 2.0 Eclipse plugin ", de http://kdmanalytics.com/kdmsdk/index.php.

[5] Balsamiq (2011). Mockups.

[6] Canfora, G. y Di Penta, M. (2007). New Frontiers of Reverse Engineering. Future of Software Engineering (FOSE'07), IEEE Computer Society.

[7] Clayberg, E. y Rubel, D. (2008). Eclipse Plug-ins (3rd Edition), Addison-Wesley Professional.

[8] Chikofsky, E. J. y Cross, J. H. (1990). "Reverse Engineering and Design Recovery: A Taxonomy." IEEE Softw. 7(1): 13-17.

[9] Eclipse. (2010a). "Eclipse Graphical Modeling Framework (GMF)." de http://www.eclipse.org/modeling/gmf/.

[10] Eclipse. (2010b). "Eclipse Modeling Tools." de http://www.eclipse.org/downloads/packages/eclipse-modeling-tools-includes-incubating-components/heliossr2.

[11] Eclipse. (2011a). "Eclipse." de http://www.eclipse.org/.

[12] Eclipse. (2011b). "MARBLE 2.0." de http://marketplace.eclipse.org/content/marble.

[13] Gamma, E. y Beck, K. (2010). JUnit.

[14] Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Inc. Boston, MA, USA, Addison Wesley.

[15] Hammer, M. y Champy, J. (1994). Reengineering the Corporation: A Manifesto for Business Revolution, HarperBusiness.

[16] Heuvel, W.-J. v. d. (2006). Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems), The MIT Press.

[17] Hoffmann, M. R. y Janiczak, B. (2011). Java Code Coverage for Eclipse

[18] Hunter, J. (2009). JDOM.

[19] ikv++. (2010). "Medini QVT ", de http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77.

[20] ISO/IEC (2009). ISO/IEC DIS 19506. Knowledge Discovery Meta-model (KDM), v1.1 (Architecture-Driven Modernization). , ISO/IEC: 302.

[21] Jacobson, I., Booch, G. y Rumbaugh, J. (2000). El Proceso Unificado de Desarrollo de Software.

[22] Jeston, J., Nelis, J. y Davenport, T. (2008). Business Process Management: Practical Guidelines to Successful Implementations. NV, USA, Butterworth-Heinemann (Elsevier Ltd.).

[23] Kazman, R., Woods, S. G. y Carrière, S. J. (1998). Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. Proceedings of the Working Conference on Reverse Engineering (WCRE'98), IEEE Computer Society.

[24] Khusidman, V. (2008). ADM Transformation.

[25] Khusidman, V. y Ulrich, W. (2007). Architecture-Driven Modernization: Transforming the Enterprise. DRAFT V.5. , OMG: 7.

[26] Kleppe, A., Warmer, J. y Bast, W. (2003). MDA explained: the model driven architecture: practice and promise, Addison-Wesley.

[27] Koskinen, J., Ahonen, J. J., Sivula, H., Tilus, T., Lintinen, H. y Kankaanpää, I. (2005). Software Modernization Decision Criteria: An Empirical Study. European Conference on Software Maintenance and Reengineering, IEEE Computer Society.

[28] Loshin, D. (2010). The Practitioner's Guide to Data Quality Improvement Morgan Kaufmann.

[29] Mellor, S. J. (2003). Guest Editors' Introduction: Model-Driven Development. A. N. Clark y T. Futagami. 20: 14-18.

[30] Mens, T. y Demeyer, S. (2008). Software Evolution, Springer-Verlag Berlin Heidelberg.

[31] Müller, H. A., Jahnke, J. H., Smith, D. B., Storey, M.-A., Tilley, S. R. y Wong, K. (2000). Reverse engineering: a roadmap. Proceedings of the Conference on The Future of Software Engineering. Limerick, Ireland, ACM.

[32] OMG (2003a). MDA Guide Version 1.0.1. OMG: 62.

[33] OMG (2003b). Model Driven Architecture.

[34] OMG (2006). Object Constraint Language.

[35] OMG (2007). Architecture-Driven Modernization: Transforming the Enterprise, Object Management Group.

[36] OMG (2009). Architecture-Driven Modernization (ADM): Knowledge Discovery Meta-Model (KDM), OMG: 308.

[37] OMG (2011a). Business Process Model and Notation (BPMN) 2.0, Object Management Group: 538.

[38] OMG (2011b). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, OMG.

[39] Paradauskas, B. y Laurikaitis, A. (2006). "Business Knowledge Extraction from Legacy Informations Systems." Information Technology and Control 35(3): 214-221.

[40] Pérez-Castillo, R., García-Rodríguez de Guzmán, I. y Piattini, M. (2011a). Architecture-Driven Modernization. Modern Software Engineering Concepts and Practices: Advanced Approaches. A. H. Dogru y V. Bier, IGI Global: 75-103.

[41] Pérez-Castillo, R., García-Rodríguez de Guzmán, I. y Piattini, M. (2011b). "Business Process Archeology using MARBLE." Information and Software Technology In Press.

[42] Polo, M., Piattini, M. y Ruiz, F. (2003). Advances in software maintenance management: technologies and solutions, Idea Group Publishing.

[43] Redman, T. C. (2008). Data driven: profiting from your most important business asset, Harvard Business School Pr.

[44] Rodríguez Ríos, A., Fernández-Medina, E. y Piattini, M. (2007). Especificación y Diseño de Procesos de Negocio Seguros. Departamento de Tecnologías y Sistemas de Información. Ciudad Real, Universidad de Castilla-La Mancha.

[45] Sneed, H. M. (2005). Estimating the Costs of a Reengineering Project, IEEE Computer Society.

[46] Sommerville, I. (2006). Software Engineering, Addison Wesley.

[47] Steinberg, D., Budinsky, F., Paternostro, M. y Merks, E. (2008). EMF: Eclipse Modeling Framework, Addison-Wesley Professional,.

[48] Weske, M. (2007). Business Process Management: Concepts, Languages, Architectures. Leipzig, Alemania, Springer-Verlag Berlin Heidelberg.

[49] WfMC (1999). Workflow Management Coalition: Terminology and Glossary. Document Number WFMC-TC-1011.

[50] White, S. y Miers, D. (2008). BPMN Modeling and Reference Guide.

8. ANEXOS

En esta sección se muestran los anexos de los que se compone el Proyecto Fin de Carrera. El primer anexo presenta la notación BPMN 2.0, describiendo sus elementos más destacados. El segundo anexo presenta el manual de usuario de la herramienta MARBLE Tool, mostrando las funcionalidades de la misma. El tercer anexo presenta el ejemplo de aplicación realizado a fin de probar las funcionalidades de la herramienta. El último anexo presenta el script QVT realizado.

8.1. Anexo I. Notación BPMN

Este anexo presenta la notación BPMN con sus elementos más importantes y destacados. La Tabla 8.1 muestra cada elemento junto a su descripción y su notación gráfica.

Elemento		Descripción	Notación
Evento (*Event*)		Un evento es algo que ocurre durante el transcurso de un proceso de negocio. Estos eventos afectan al flujo del proceso y normalmente tienen una causa o un impacto. Los eventos son círculos con centros abiertos que permiten contener indicadores en su interior que diferencien entre diferentes causas (*triggers*) o impactos (*results*). Existen tres tipos de eventos según cuando ellos afecten al flujo: Comienzo (*Start*), Intermedio (*Intermediate*) y Final (*End*).	
Actividades (*Activities*)		Una actividad es un término genérico para el trabajo que una empresa u organización realiza. Representa el trabajo realizado en un Proceso de Negocio. Una actividad normalmente conlleva un tiempo en realizarse, implica la participación de uno o más recursos de la organización, requiere algún tipo de entrada y produce algún tipo de salida. Una actividad puede ser atómica o no-atómica (compuesta). Los tipos de actividades que son parte de un Modelo de Proceso son: Tarea, Transacción, Subproceso de evento y Actividad de llamado (*Call Activity*).	
Compuertas o puertas (*Gateway*)		Un ***Gateway*** es usado para controlar la divergencia y la convergencia de un Flujo de Secuencia. Así, él puede determinar la ramificación, la bifurcación, la fusión y la unión de caminos. Los marcadores internos indicaran el tipo de control de comportamiento.	
Tipos de Gateway	**Paralela (*Parallel*)**	En un punto de bifurcación, todos los caminos salientes serán activados simultáneamente. En un punto de convergencia, la compuerta espera a que todos los flujos incidentes completen antes de activar el flujo saliente.	
	Compleja (*Complex*)	Comportamiento complejo de convergencia o bifurcación no capturado por el resto de compuertas.	

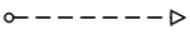
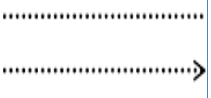
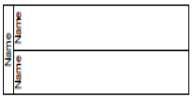
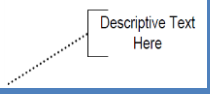
Elemento		Descripción	Notación
Tipos de Gateway	**Inclusiva (*Inclusive*)**	En un punto de bifurcación, al menos un flujo es activado. En un punto de convergencia, espera a todos los flujos que fueron activados para activar al saliente.	
	Exclusiva (*Exclusive*)	En un punto de bifurcación, selecciona exactamente un flujo de secuencia de entre las alternativas existentes. En un punto de convergencia, la compuerta espera a que un flujo incidente complete para activar el flujo saliente.	
Objeto de Datos (*Data Object*)		Un Objeto de Datos representa información que fluye a través del proceso tales como documentos, correos electrónicos o cartas.	
Flujo de Secuencia (*Sequence Flow*)		Define el orden de ejecución entre dos actividades.	
Flujo de Mensajes (*Message Flow*)		Simboliza la información que fluye a través de las organizaciones. Este flujo puede conectarse con compartimentos, actividades o eventos de mensaje.	
Asociación (*Association*)		Una asociación se utiliza para vincular la información y Artefactos.	
Contenedores y Compartimentos (*Pools and Lanes*)		Representan a las entidades responsables de las actividades en un proceso. Por ejemplo, una organización, un rol o un sistema.	
Contenedores (*Pools*)		Es la representación gráfica de un participante en una colaboración.	
Compartimentos (*Lanes*)		Un Lane es una sub-partición dentro de un Pool.	
Anotaciones de texto (*Text annotation*)		Proporciona información de texto adicional para el lector de un diagrama BPMN.	

Tabla 8.1. Elementos de la notación BPMN 2.0

8.2. Anexo II. Manual de Usuario

En este apartado se va a describir el funcionamiento de MARBLE Tool. La herramienta se distribuye en dos modalidades, por una parte como una aplicación independiente basada en Eclipse, y por otra como un plug-in, de forma que pueda ser integrada en el IDE de Eclipse. A continuación se muestra el procedimiento de uso para ambas distribuciones.

8.2.1. Manual de Usuario de la aplicación independiente

En el caso de trabajar con la aplicación independiente no es necesaria instalación previa. Al igual que ocurre con el IDE de Eclipse, simplemente se copiará la carpeta MARBLE_TOOL en el lugar del equipo que se desee, por ejemplo en el *Escritorio*, y se lanzará la aplicación haciendo clic en el ejecutable MARBLE.exe (Figura 8.1).

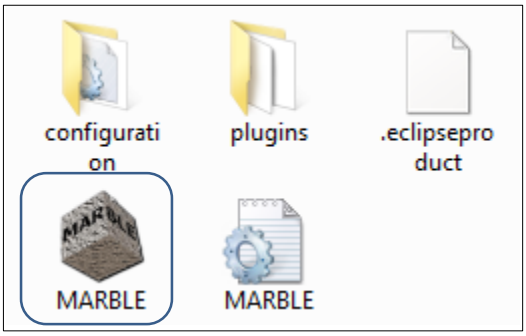


Figura 8.1. Contenido de la carpeta MARBLE_TOOL

Al ejecutar la aplicación aparecerá la imagen de la Figura 8.2 y el *workspace* mostrado en la Figura 8.3. Las opciones disponibles se describen a continuación en los sucesivos apartados.

8.2.1.1. Crear un nuevo proyecto MARBLE

Para crear un nuevo proyecto MARBLE pulsaremos el comandos *Ctrl+N* o bien haciendo clic en *File->New->Other*. Al hacer esto aparecerá la ventana mostrada en la Figura 8.4 donde aparece un nuevo tipo de proyecto que es el proyecto en cuestión. Al crear un nuevo proyecto MARBLE aparecerá la ventana para introducir el nombre del proyecto y su ubicación (Figura 8.5).

Figura 8.2. Imagen de carga de MARBLE Tool

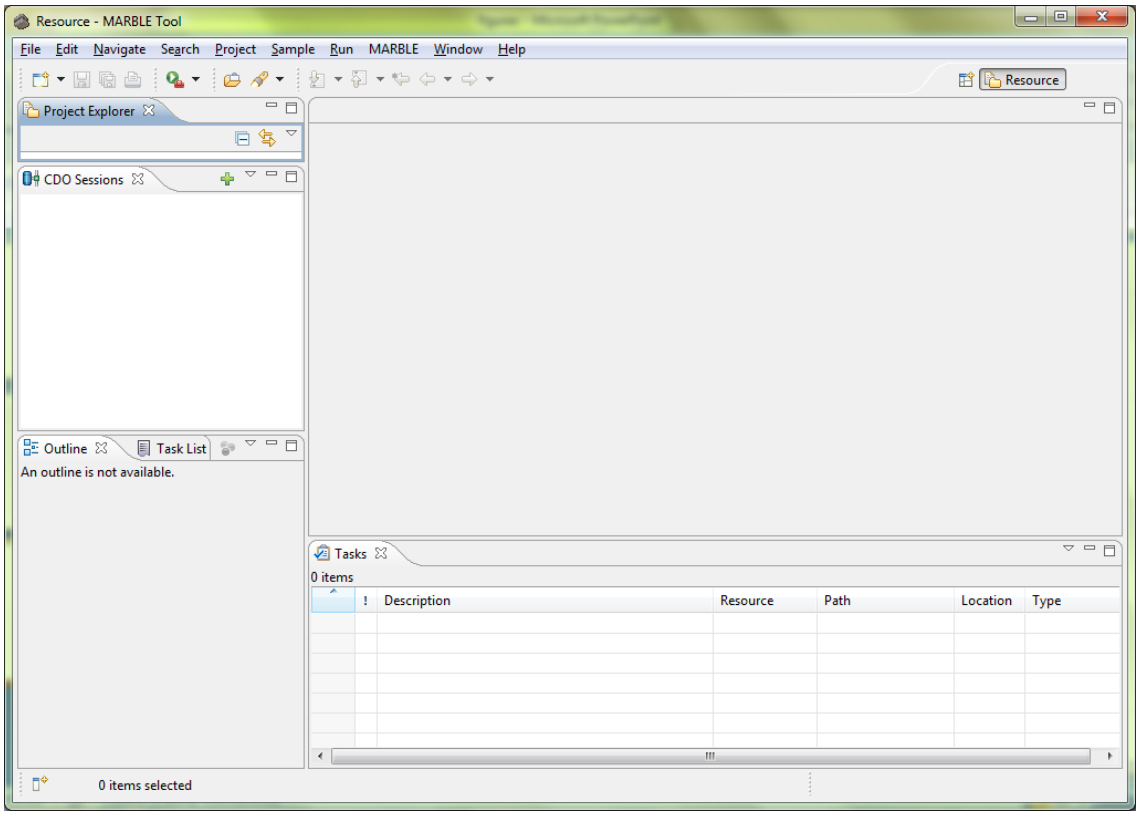


Figura 8.3. Workspace inicial de MARBLE Tool

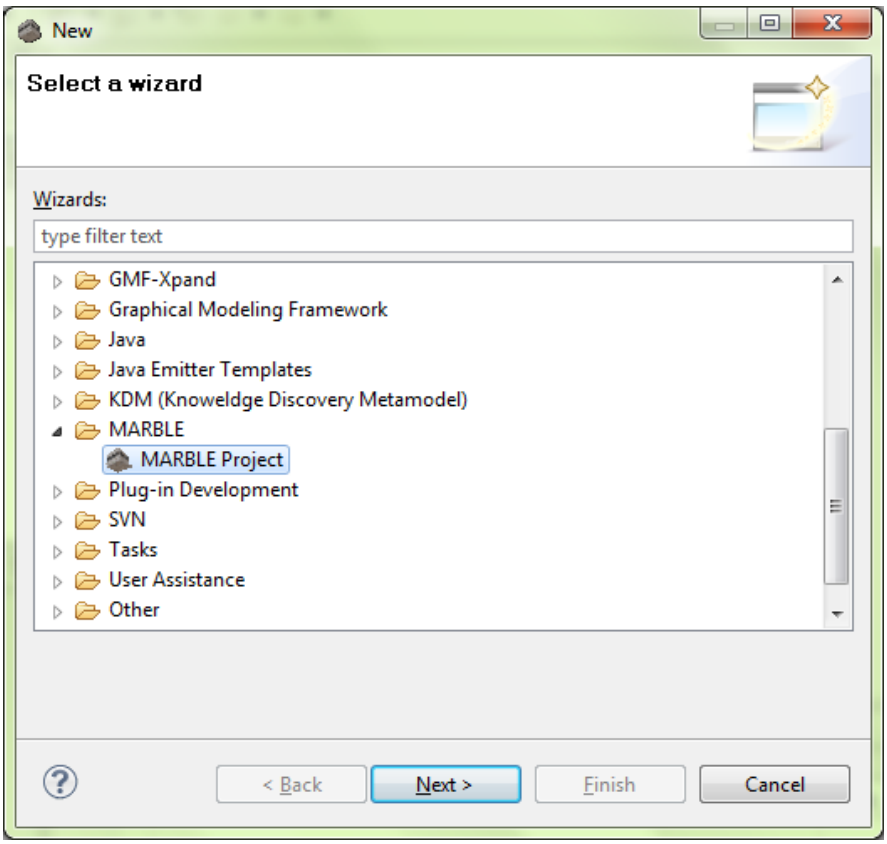


Figura 8.4. Ventana de nuevo proyecto MARBLE

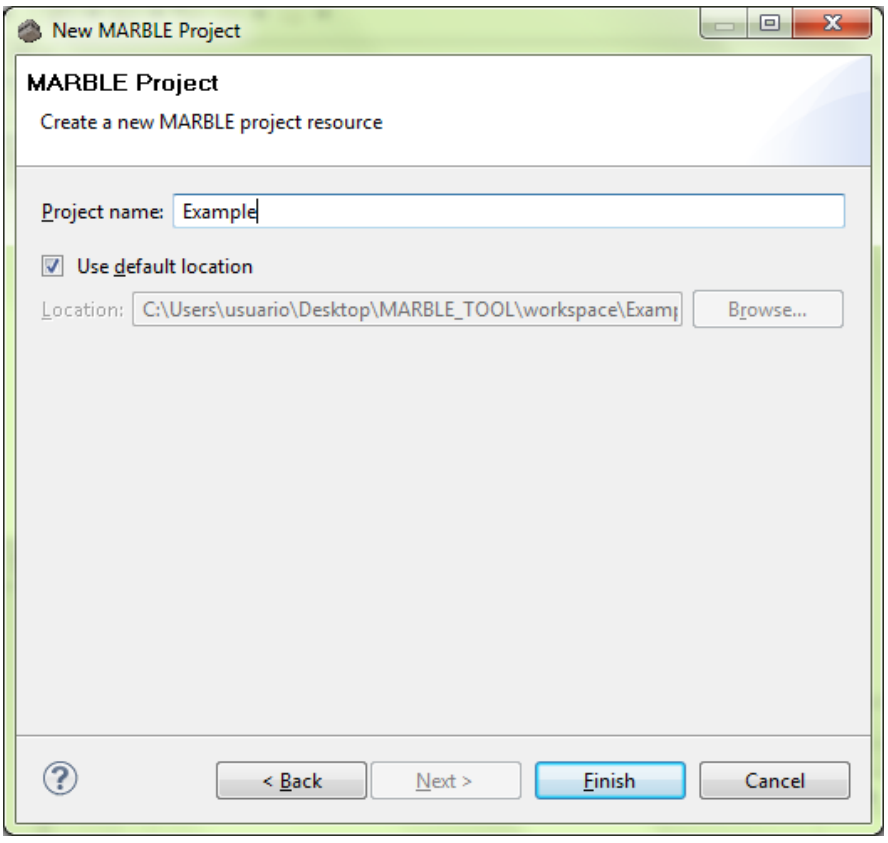


Figura 8.5. Introducir nombre y ubicación del proyecto MARBLE

Una vez creado el proyecto aparecerá el mensaje mostrado en la Figura 8.6 indicándonos que se abrirá la nueva perspectiva MARBLE asociada a ese proyecto. El aspecto de dicha perspectiva es el mostrado en la Figura 8.7, en la que vemos el nuevo proyecto creado con su respectiva jerarquía de carpetas y sus respectivas vistas.

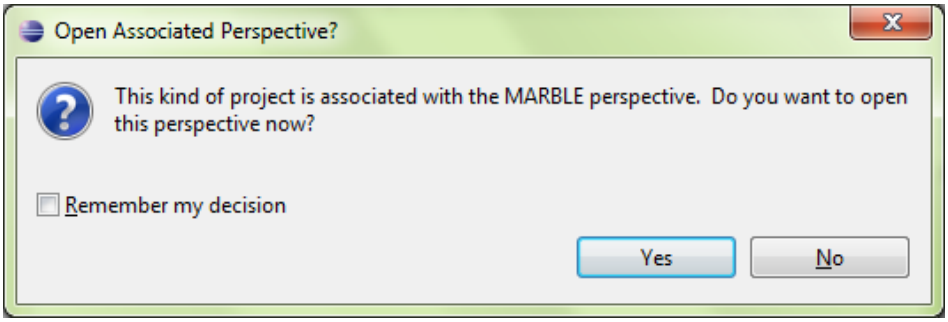


Figura 8.6. Aviso para abrir la perspectiva asociada

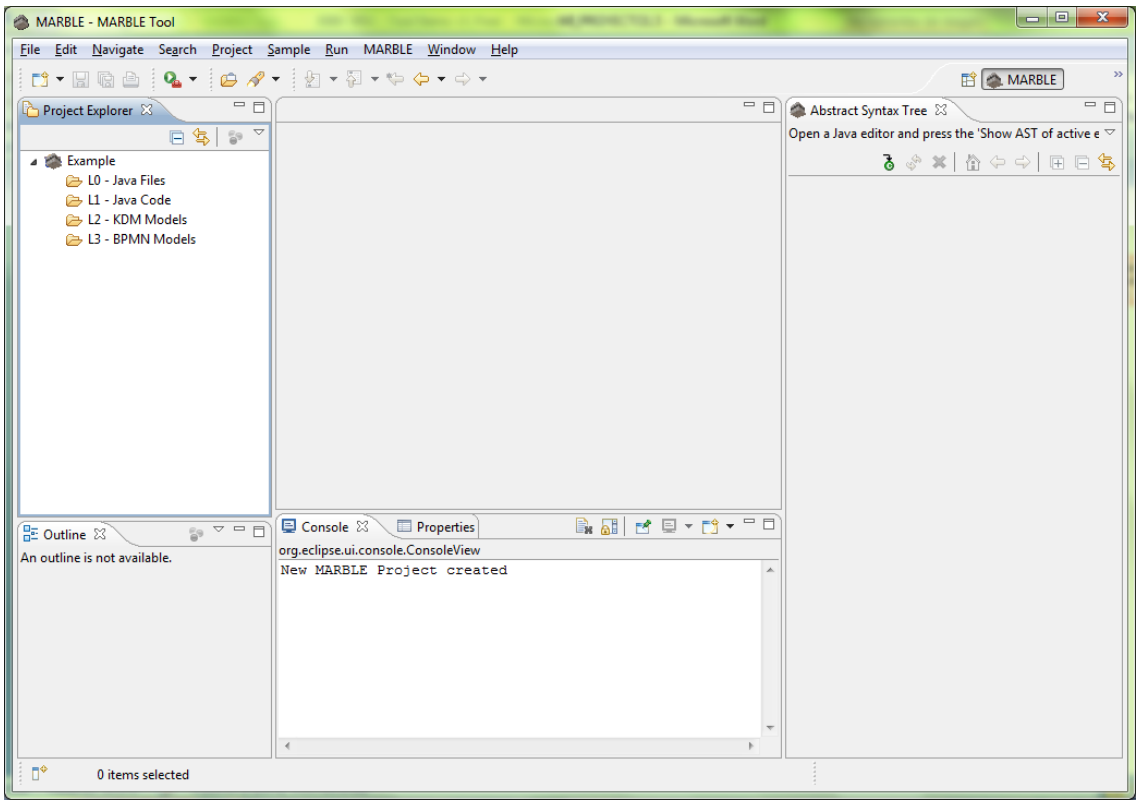


Figura 8.7. Perspectiva MARBLE

8.2.1.2. Agregar LIS

Para agregar el código fuente heredado se puede hacer de dos formas:

a) A través del menú de la **barra de herramientas** MARBLE (Figura 8.8) en las opciones de “Select a LIS” y “Select directory of LISs”.

b) A través del **menú contextual** MARBLE (Figura 8.9) en las opciones de "Select a LIS" y "Select directory of LISs".

En cualquiera de estas opciones aparecerá las siguientes ventanas: para el caso de agregar un único LIS (Figura 8.10) y para el caso de agregar un directorio que contiene los LISs (Figura 8.11).

Una vez seleccionados los archivos relativos a los LISs éstos se añadirán en la carpeta "L0 - Java Files"

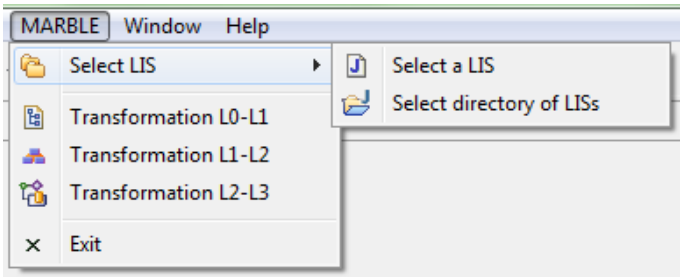


Figura 8.8. Menú MARBLE. Barra de herramientas

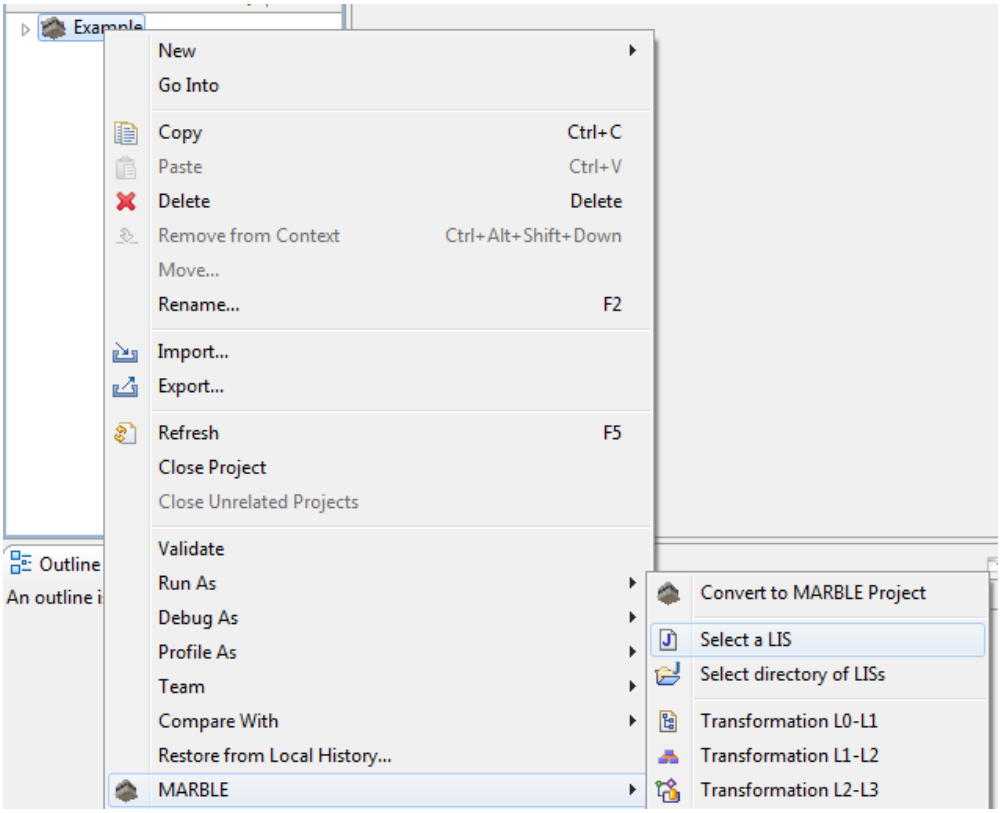


Figura 8.9. Menú contextual MARBLE

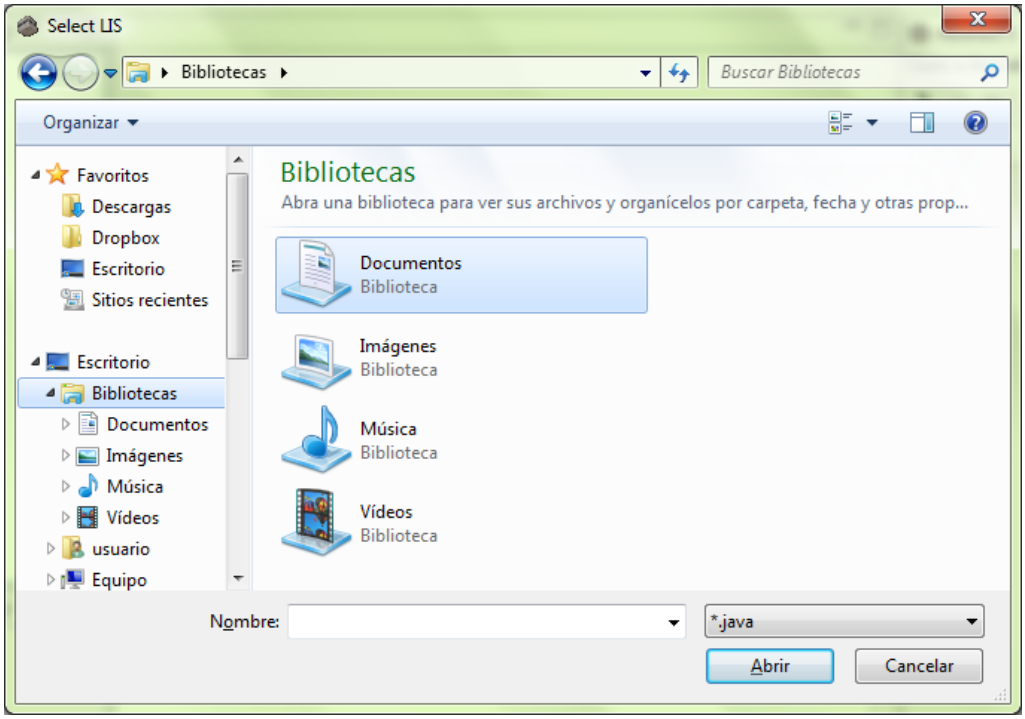


Figura 8.10. Seleccionar un único LIS

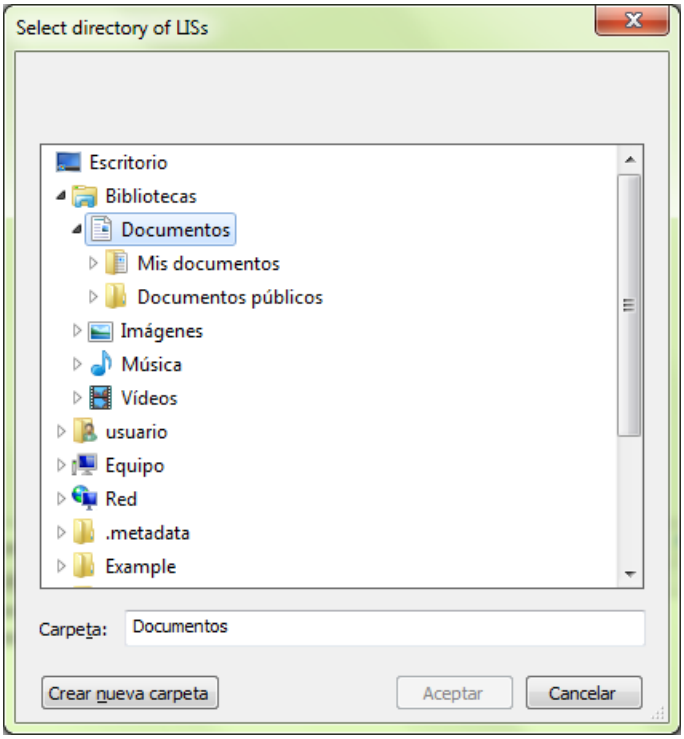


Figura 8.11. Seleccionar un directorio de LISs

8.2.1.3. Obtener el modelo de código del código fuente heredado

Para realizar la primera transformación se seleccionarán aquellos archivos de los que se desea obtener el modelo de código y se seleccionará el menú "Transformation L0-L1" mediante cualquiera de las dos opciones posibles (Figura 8.8 y Figura 8.9). Esta opción generará en la carpeta "L1 – Java Code" los respectivos modelos de los archivos seleccionados.

Además, mediante la vista *Abstract Syntax Tree* se mostrará el árbol sintáctico abstracto del archivo (Figura 8.12). Para ello, será necesario tener abierto en el editor un archivo Java y pulsar  para generar el árbol de ese archivo.

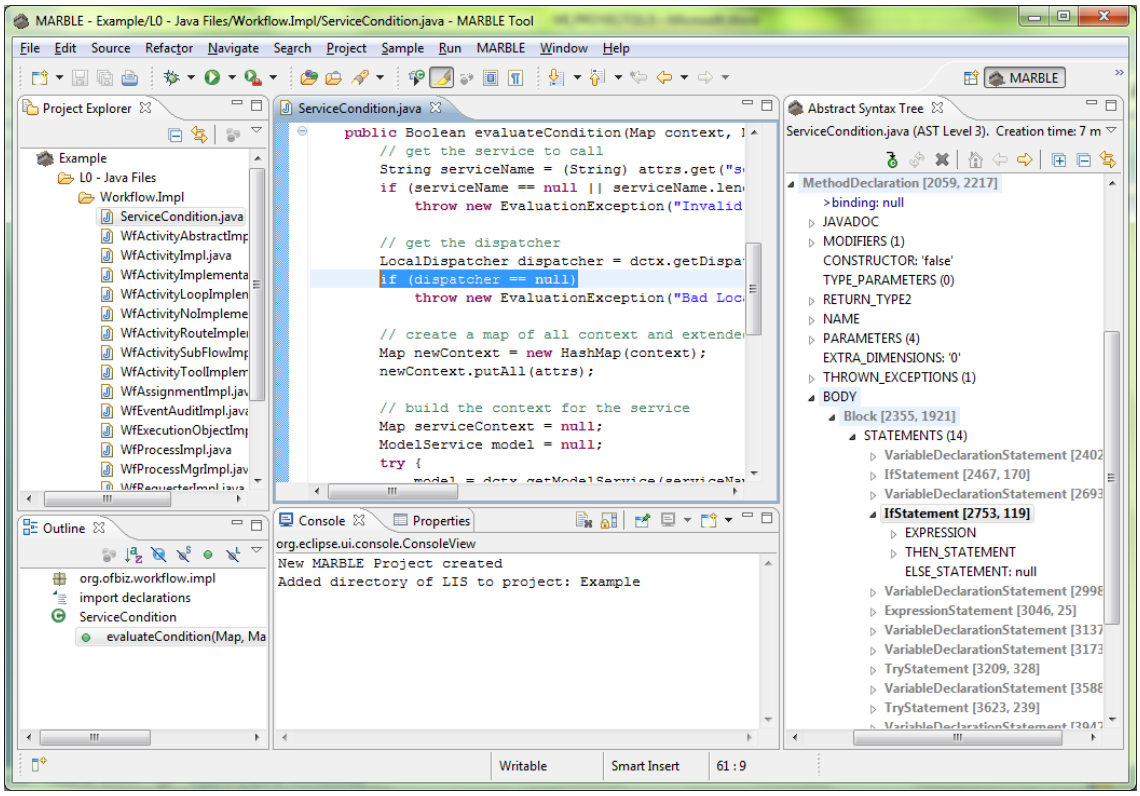


Figura 8.12. Vista del árbol sintáctico abstracto del archivo seleccionado

8.2.1.4. Obtener los modelos KDM a partir de los modelos de código

Para realizar la segunda transformación se seleccionarán aquellos archivos de los que se desea obtener el modelo KDM y se seleccionará el menú "Transformation L1-L2" mediante cualquiera de las dos opciones posibles (Figura 8.8 y Figura 8.9). Esta opción generará en la carpeta "L2 – KDM Models" los respectivos modelos KDM de los archivos seleccionados.

Un ejemplo de uno de los modelos generados es mostrado en la Figura 8.13, visualizado en el editor de KDM.

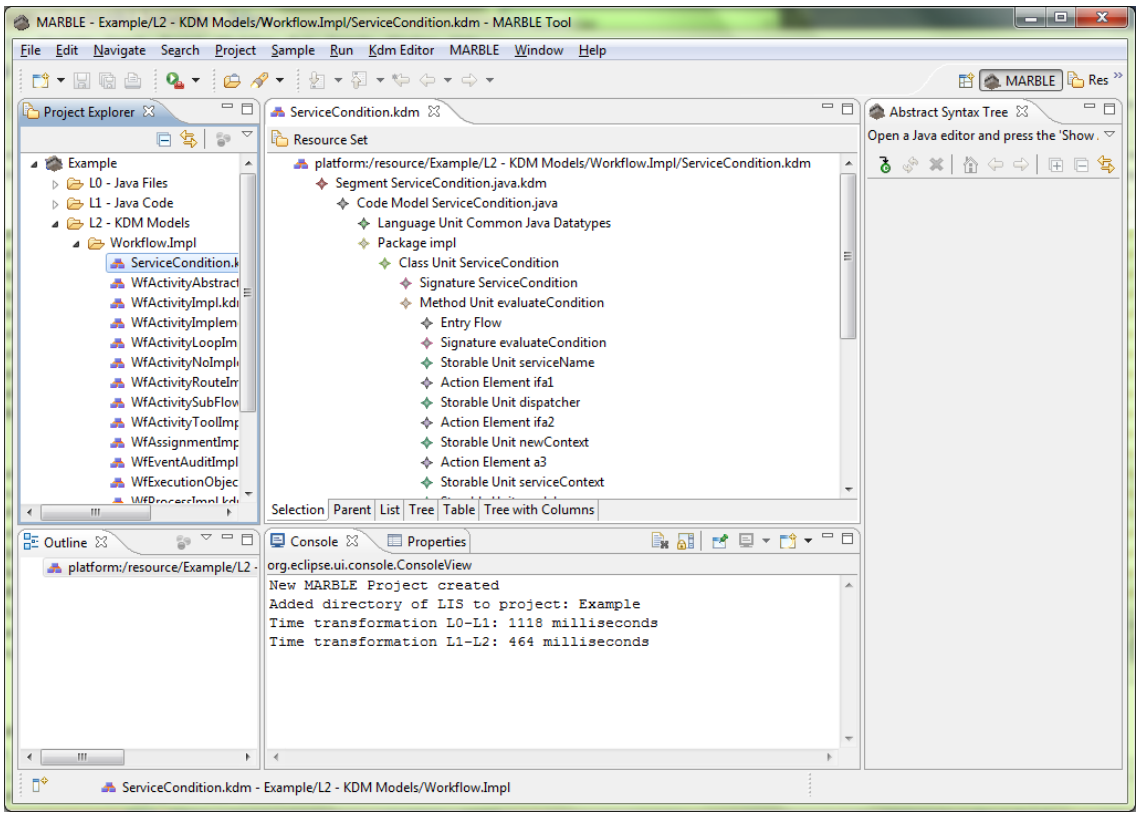


Figura 8.13. Ejemplo de modelo KDM generado

8.2.1.5. Obtener los modelos de Procesos de Negocio

Para realizar la tercera transformación se seleccionarán aquellos archivos de los que se desea obtener el modelo de procesos de negocio y se seleccionará el menú "Transformation L2-L3" mediante cualquiera de las dos opciones posibles (Figura 8.8 y Figura 8.9). Esta opción generará en la carpeta "L3 – BPMN Models" los respectivos modelos de los archivos seleccionados.

Un ejemplo de uno de los modelos generados es mostrado en la Figura 8.14, visualizado en el editor GMF-BPMN.

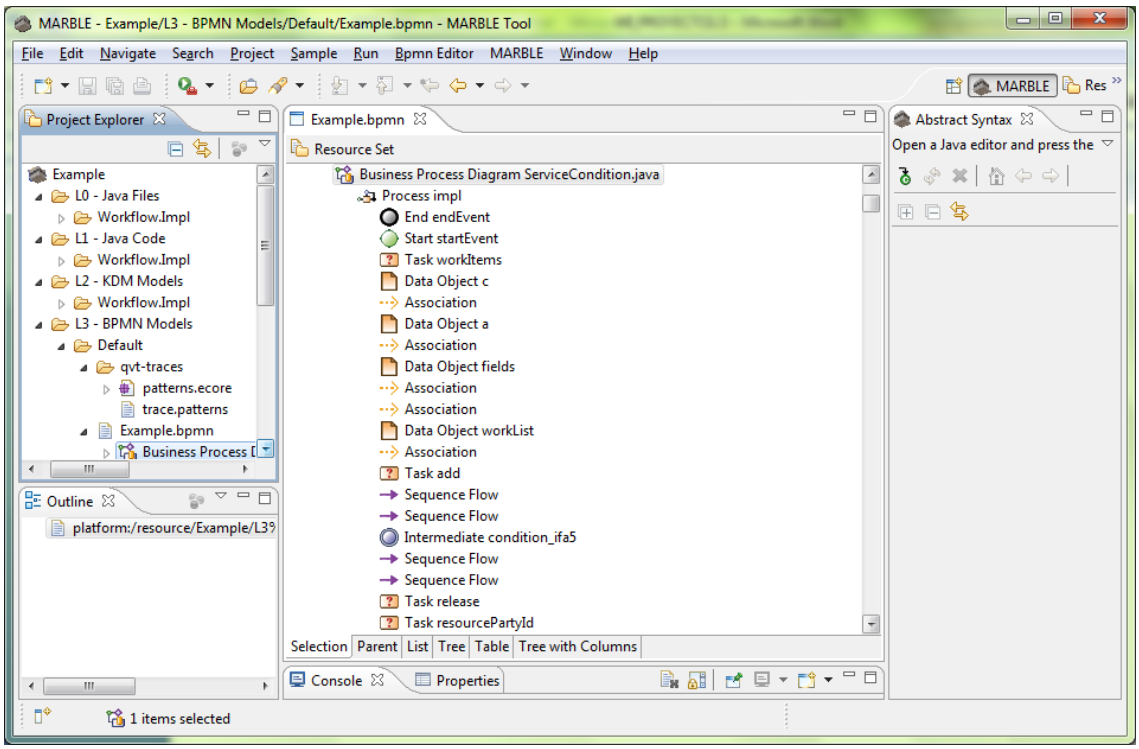


Figura 8.14. Ejemplo de modelo de procesos de negocio generado

8.2.1.6. Visualizar el diagrama de procesos de negocio

Para visualizar el diagrama de procesos de negocio pulsaremos el menú contextual en la opción "Initialize bpmn_diagram diagram file" mostrada en la Figura 8.15.

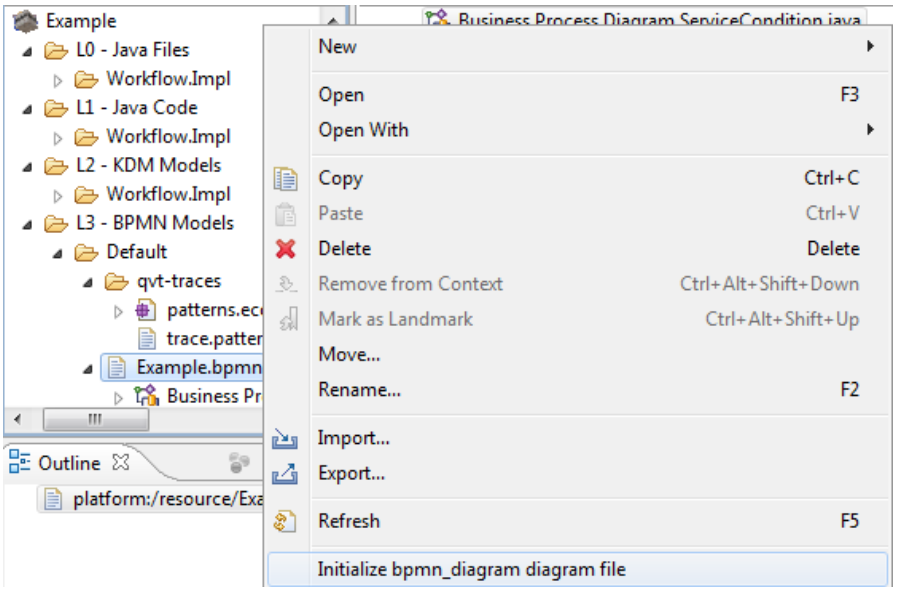


Figura 8.15. Visualizar el diagrama de procesos de negocio

Un ejemplo de diagrama resultado es el mostrado en la Figura 8.16.

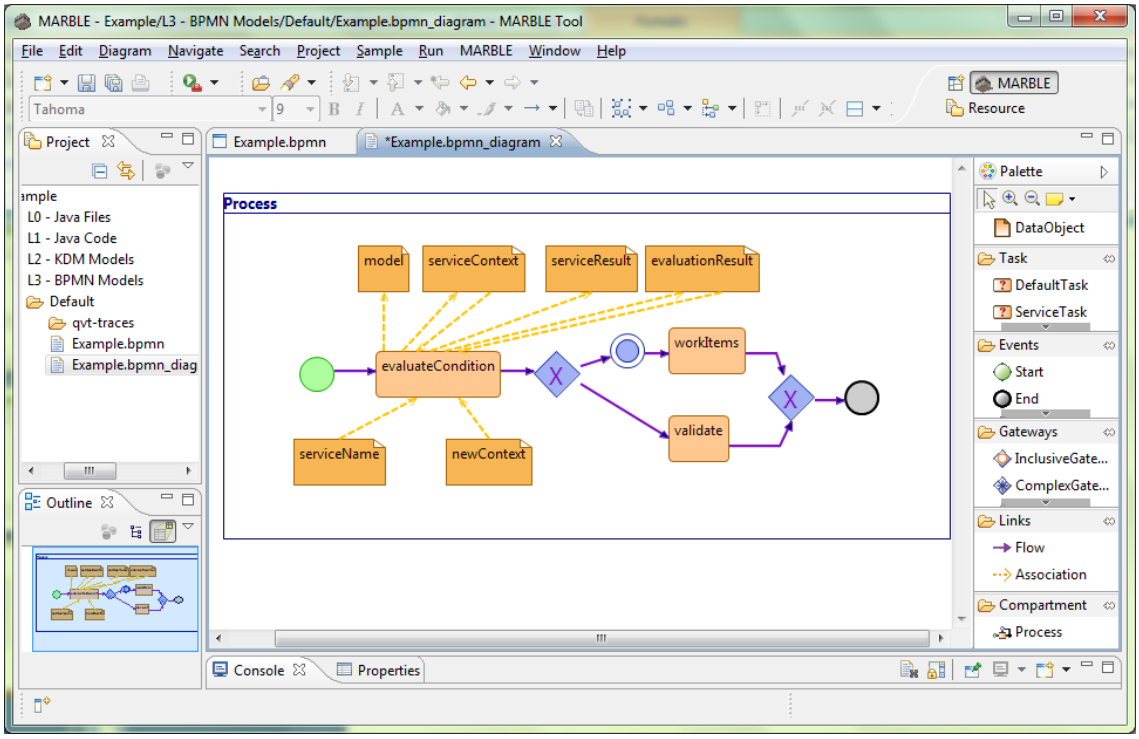


Figura 8.16. Ejemplo de diagrama de procesos de negocio

8.2.1.7. Generar estadísticas

A la hora de realizar cada una de las anteriores transformaciones se irá confeccionando automáticamente un documento PDF con toda la información relativa a las transformaciones. Entre dicha información se encuentra: nombres de archivos origen y destino en la transformación, tiempo utilizado en la transformación, etc. El documento llevará el nombre de *Statistics.pdf*. Un ejemplo es el mostrado en Figura 8.17.

Figura 8.17: Informe generado por MARBLE Tool en PDF

8.2.1.8. Convertir un proyecto Java existente en un proyecto MARBLE

Para convertir un proyecto existente en un proyecto MARBLE se seleccionará el proyecto a convertir y se seleccionará el menú contextual "Convert to MARBLE Project" mostrado en la Figura 8.9. Al proyecto existente se le añadirá la arquitectura de MARBLE y se abrirá la perspectiva MARBLE como se muestra en la Figura 8.18.

Una vez convertido se pueden realizar las opciones de Obtener el modelo de código del código fuente heredado, Obtener los modelos KDM a partir de los modelos de código, Obtener los modelos de Procesos de Negocio y Visualizar el diagrama de procesos de negocio.

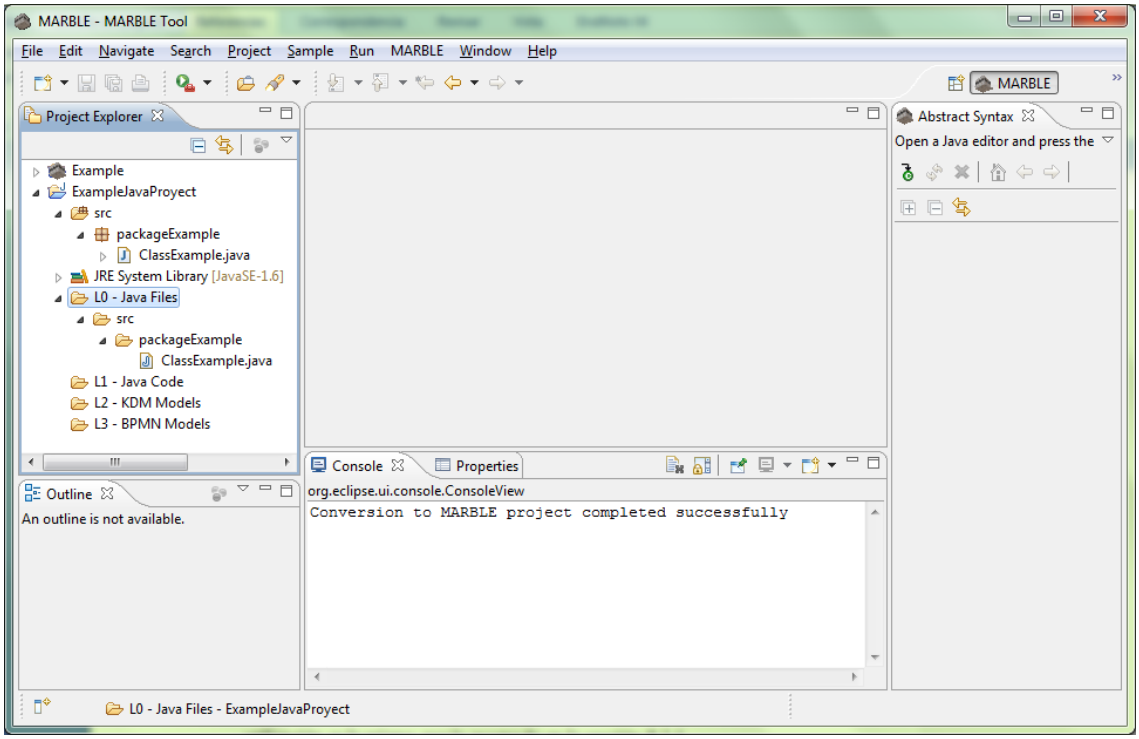


Figura 8.18. Ejemplo de conversión de un proyecto Java a un proyecto MARBLE

8.2.1.9. Configurar el entorno de las transformaciones

Para poder configurar el entorno de las transformaciones implementadas en la herramienta será necesario acceder al menú de Eclipse *Window->Preferences* y acceder a la página *MARBLE* de la forma que se muestra en la Figura 8.19.

Las opciones que son configurables son:

- "*LISAggregation only java files*": Si la opción está marcada sólo se agregan los archivos con extensión *.java* a la hora de seleccionar una carpeta de LISs. En el caso de no estar marcada, todos los archivos de esa carpeta son agregados a la carpeta L0, independientemente de su extensión.
- "*Delete traces/target model on launch*": Si la opción está marcada se eliminarán las trazas generadas tras las transformaciones QVT realizadas. En el caso de no estar marcada estas trazas no serán eliminadas.

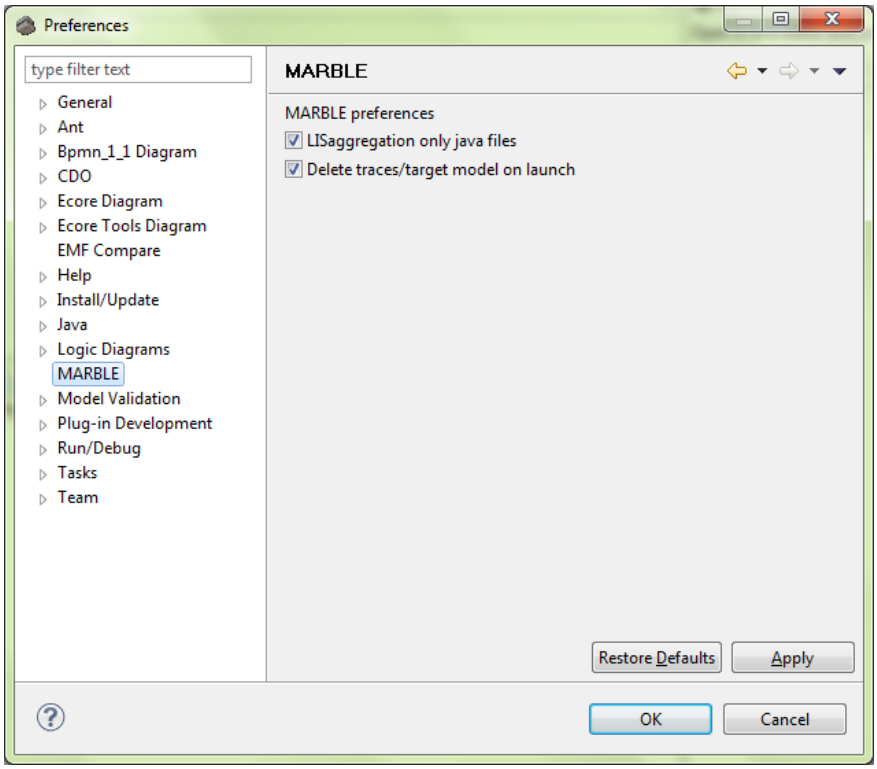


Figura 8.19. Ventana de Preferencias de MARBLE

8.2.2. Manual de Usuario del Plug-in desarrollado

Como es habitual, la forma de agregar plug-ins a nuestro IDE de Eclipse es copiar los plug-ins en cuestión en la carpeta *plugins* de Eclipse y lanzar el entorno de manera habitual.

El aspecto del entorno será el mostrado en la Figura 8.20 en la que vemos que se ha añadido el menú MARBLE y todas las opciones de MARBLE Tool. La forma de utilización es la misma que la mostrada en la sección 8.2.1.

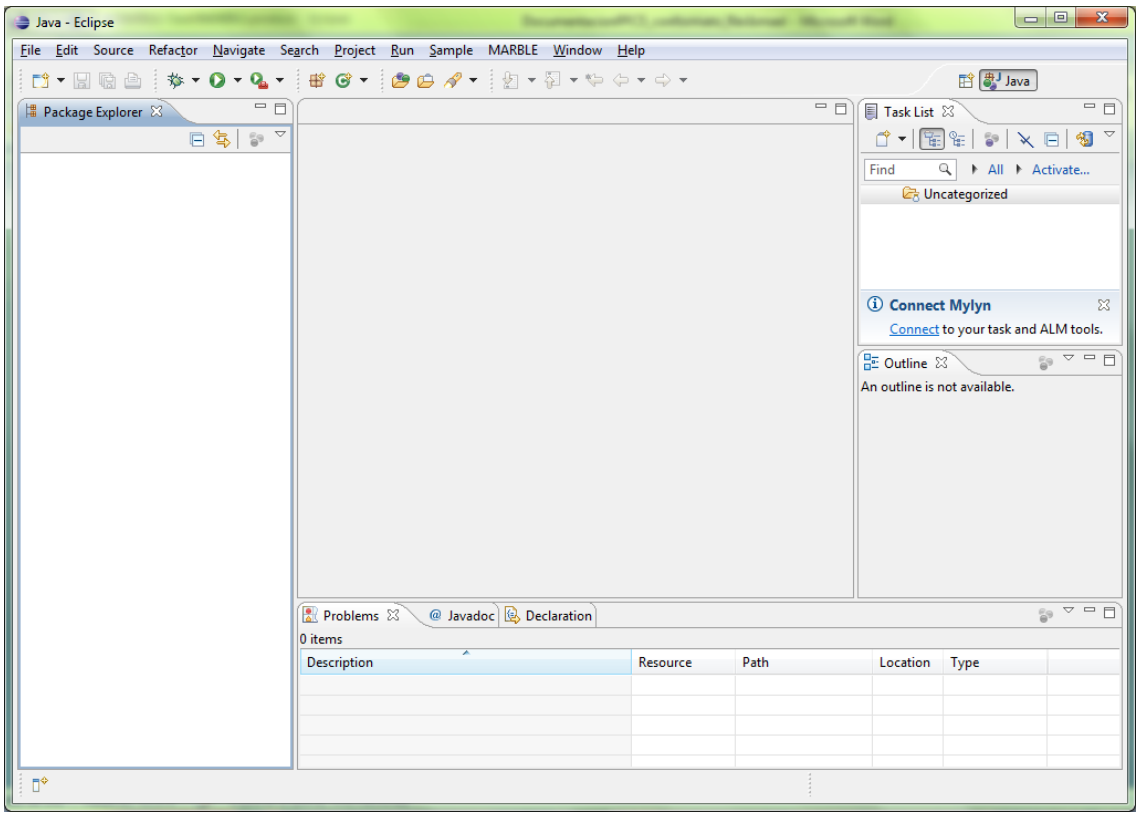


Figura 8.20. Eclipse con plug-ins de MARBLE Tool

8.3. Anexo III. Ejemplo de Aplicación en un Entorno Empresarial

Este anexo muestra un ejemplo de aplicación que aplica MARBLE Tool a un sistema de información heredado real (Pérez-Castillo, García-Rodríguez de Guzmán et al., 2011b). Este ejemplo de aplicación se encuadra dentro de las pruebas de aceptación del proyecto, que contribuye a probar todas las funciones de la aplicación así como analizar los resultados obtenidos.

8.3.1. Sistema de información heredado bajo estudio

El ejemplo de aplicación se ha realizado con el sistema *VillasanteLaboratory*. Este sistema es la aplicación de gestión de una compañía española dedicada al análisis químico de aguas potables y residuales. *VillasanteLaboratory* gestiona la información relativa a laboratorios químicos, clientes y productos, tales como análisis químicos, disoluciones y calibraciones químicas. Este análisis soporta un gran número de parámetros físicos, químicos y microbiológicos para el control y regulación de la calidad del agua. Además, el sistema dispone de gestión para usuarios y roles mediante un módulo de administración.

Desde un punto de vista tecnológico, *VillasanteLaboratory* consiste en una aplicación Web tradicional cuya arquitectura está separada en tres capas: presentación, negocio y persistencia. La tecnología utilizada para la capa de presentación es JSP (*Java Server Pages*), la capa de negocio utiliza el lenguaje Java y para la capa de persistencia utiliza SQL Server junto con JDBC-ODBC. El tamaño total de este sistema heredado es de 28.8 KLOC (*miles de líneas de código fuente*).

8.3.2. Ejecución del ejemplo de aplicación

Para realizar la ejecución del ejemplo de aplicación se han llevado a cabo una serie de pasos que se enumeran a continuación:

1. Realizar reuniones con los responsables de la empresa con el fin de establecer un acuerdo de confidencialidad. En este paso también se designará al experto en negocio.

2. Obtener el código fuente Java del sistema heredado a fin de poder utilizar la herramienta.
3. Extraer los modelos de procesos de negocio mediante la herramienta MARBLE Tool. Esta parte se describe en más detalle en el apartado 8.3.2.1.
4. Analizar los modelos obtenidos. Esta acción es llevada a cabo por el experto de negocio. Esta parte se describe en más detalle en el apartado 8.3.2.2.
5. Extracción de conclusiones a partir de dicho análisis. Esta parte se describe con mayor detalle en el apartado 8.3.2.3.
6. Informar a la empresa de los resultados del ejemplo de aplicación.

8.3.2.1. Extracción de los modelos de procesos de negocio

Para llevar a cabo la extracción de los modelos de procesos de negocio del sistema *VillasanteLaboratory* se han seguido los pasos propuestos por MARBLE a través de la herramienta. Estos pasos han sido los siguientes:

1. Creación de un nuevo proyecto MARBLE en la herramienta llamado "CaseStudy-VillasanteLab".
2. Añadir los archivos Java del sistema *VillasanteLaboratory* al proyecto creado.
3. Obtener los modelos de código de los archivos Java añadidos mediante la opción "Transformation L0-L1" de la herramienta. Uno de los modelos generados se muestra en el Fragmento de código 8.1.
4. Obtener los modelos KDM de los modelos de código generados anteriormente mediante la opción "Transformation L1-L2" de la herramienta. Uno de los modelos generados se muestra en el Fragmento de código 8.2 y en la Figura 8.21.
5. Obtener los modelos de procesos de negocio a partir de los modelos KDM generados anteriormente mediante la opción "Transformation L2-L3" de la herramienta. Una parte del modelo generado se muestra en la Figura 8.22, cuyo diagrama es el mostrado en la Figura 8.23.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<javaCode path="C:\Users\usuario\Desktop\MARBLE_Tool\workspace\CaseStudy-VillasanteLab\L0 - Java
Files\LabVillasante\src\com\snt\applab\utils\listeners\ListenerAplicacion.java">
  <package>
    <PackageDeclaration value="package com.snt.applab.utils.listeners;">
      <QualifiedName value="com.snt.applab.utils.listeners" />
    </PackageDeclaration>
  </package>
  <imports count="9">
    <ImportDeclaration value="import java.io.IOException;">
      <QualifiedName value="java.io.IOException" />
    </ImportDeclaration>
    <ImportDeclaration value="import java.io.InputStream;">
      <QualifiedName value="java.io.InputStream" />
    </ImportDeclaration>

      ...

  </imports>
  <types count="1">
    <TypeDeclaration>

      ...

      <Modifiers count="1">
        <Modifier value="public" />
      </Modifiers>
      <Name>
        <SimpleName value="ListenerAplicacion" />
      </Name>
      <BodyDeclaration count="3">
        <MethodDeclaration>
            ...
          <modifiers count="1">
            <Modifier value="public" />
          </modifiers>
        </MethodDeclaration>
        <MethodDeclaration>
          public void contextDestroyed(ServletContextEvent arg0){
          }
          <modifiers count="1">
            <Modifier value="public" />
          </modifiers>
        </MethodDeclaration>
        <MethodDeclaration>
            ...
          <modifiers count="1">
            <Modifier value="private" />
          </modifiers>
        </MethodDeclaration>
      </BodyDeclaration>
    </TypeDeclaration>
  </types>
</javaCode>
```

Fragmento de código 8.1. Modelo de código (adaptado) del ejemplo de aplicación

```
<?xml version="1.0" encoding="UTF-8"?>
<kdm:Segment xmlns:kdm="http://kdm.omg.org/kdm" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:action="http://kdm.omg.org/action"
xmlns:code="http://kdm.omg.org/code" xmi:version="2.0" xsi:schemaLocation="http://kdm.omg.org/action
http://alarcos.esi.uclm.es/per/rpdelcastillo/metamodels/kdm.ecore.xml#//action http://kdm.omg.org/code
http://alarcos.esi.uclm.es/per/rpdelcastillo/metamodels/kdm.ecore.xml#//code http://kdm.omg.org/kdm
http://alarcos.esi.uclm.es/per/rpdelcastillo/metamodels/kdm.ecore.xml#//kdm"
name="ListenerAplicacion.java.kdm">
 <model xsi:type="code:CodeModel" xmi:id="id.20901" name="ListenerAplicacion.java">
  <codeElement xsi:type="code:LanguageUnit" xmi:id="id.20902" name="Common Java Datatypes" />
  <codeElement xsi:type="code:Package" xmi:id="id.20903" name="listeners">
   <codeElement xsi:type="code:ClassUnit" xmi:id="id.20904" name="ListenerAplicacion">
    <codeElement xsi:type="code:Signature" xmi:id="id.20905" name="ListenerAplicacion" />
    <codeElement xsi:type="code:MethodUnit" xmi:id="id.20906" name="contextInitialized">
     <codeElement xsi:type="code:Signature" xmi:id="id.20907" name="contextInitialized">
      <parameterUnit xmi:id="id.20908" name="evt" />
     </codeElement>
     <codeElement xmi:id="id.20909" xsi:type="code:StorableUnit" name="sc" kind="local" />
     <entryFlow xmi:id="id.20910" from="id.20906" to="id.20911" />
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20911" name="a1" kind="Assign">
      <actionRelation xsi:type="action:Reads" xmi:id="id.20912" from="id.20911" to="id.20909" />
      <actionRelation xsi:type="action:Flow" xmi:id="id.20913" from="id.20911" to="id.20916" />
     </codeElement>
     <codeElement xmi:id="id.20914" xsi:type="code:StorableUnit" name="resources" kind="local" />
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20916" name="a2" kind="Assign">
      <actionRelation xsi:type="action:Reads" xmi:id="id.20917" from="id.20916" to="id.20914" />
      <actionRelation xsi:type="action:Flow" xmi:id="id.20918" from="id.20916" to="id.20920" />
     </codeElement>
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20920" name="a3" kind="Call">
      <attribute tag="api-call" value="setAttribute" />
     </codeElement>
    </codeElement>
    <codeElement xsi:type="code:MethodUnit" xmi:id="id.20923" name="contextDestroyed">
     <codeElement xsi:type="code:Signature" xmi:id="id.20924" name="contextDestroyed">
      <parameterUnit xmi:id="id.20925" name="arg0" />
     </codeElement>
    </codeElement>
    <codeElement xsi:type="code:MethodUnit" xmi:id="id.20926" name="loadResources">
     <codeElement xsi:type="code:Signature" xmi:id="id.20927" name="loadResources">
      <parameterUnit xmi:id="id.20928" name="sc" />
     </codeElement>
     <codeElement xmi:id="id.20929" xsi:type="code:StorableUnit" name="resources" kind="local" />
     <entryFlow xmi:id="id.20930" from="id.20926" to="id.20931" />
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20931" name="ifa1" kind="Condition">
      <actionRelation xsi:type="action:Flow" xmi:id="id.20932" from="id.20931" to="id.20936" />
     </codeElement>
     <codeElement xmi:id="id.20933" xsi:type="code:StorableUnit" name="resList" kind="local" />
     <codeElement xmi:id="id.20934" xsi:type="code:StorableUnit" name="al" kind="local" />
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20936" name="a2" kind="Assign">
      <actionRelation xsi:type="action:Reads" xmi:id="id.20937" from="id.20936" to="id.20933" />
      <actionRelation xsi:type="action:Flow" xmi:id="id.20938" from="id.20936" to="id.20941" />
     </codeElement>
     <codeElement xmi:id="id.20939" xsi:type="code:StorableUnit" name="is" kind="local" />
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20941" name="ifa3" kind="Condition">
      <actionRelation xsi:type="action:Flow" xmi:id="id.20942" from="id.20941" to="id.20944" />
     </codeElement>
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20944" name="a4" kind="Assign">
      <actionRelation xsi:type="action:Writes" xmi:id="id.20945" from="id.20944" to="id.20939" />
      <actionRelation xsi:type="action:Flow" xmi:id="id.20946" from="id.20944" to="id.20948" />
     </codeElement>
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20948" name="ifa5" kind="Condition">
      <actionRelation xsi:type="action:Flow" xmi:id="id.20949" from="id.20948" to="id.20951" />
      <actionRelation xsi:type="action:TrueFlow" xmi:id="id.20954" from="id.20948" to="id.20955" />
     </codeElement>
     <codeElement xsi:type="action:ActionElement" xmi:id="id.20951" name="a6" kind="Assign">
      <actionRelation xsi:type="action:Reads" xmi:id="id.20952" from="id.20951" to="id.20939" />
      <actionRelation xsi:type="action:Flow" xmi:id="id.20953" from="id.20951" to="id.20959" />
     </codeElement>

        ...
</codeElement>
    </codeElement>
   </codeElement>
 </model>
</kdm:Segment>
```

Fragmento de código 8.2. Modelo KDM (adaptado) del ejemplo de aplicación

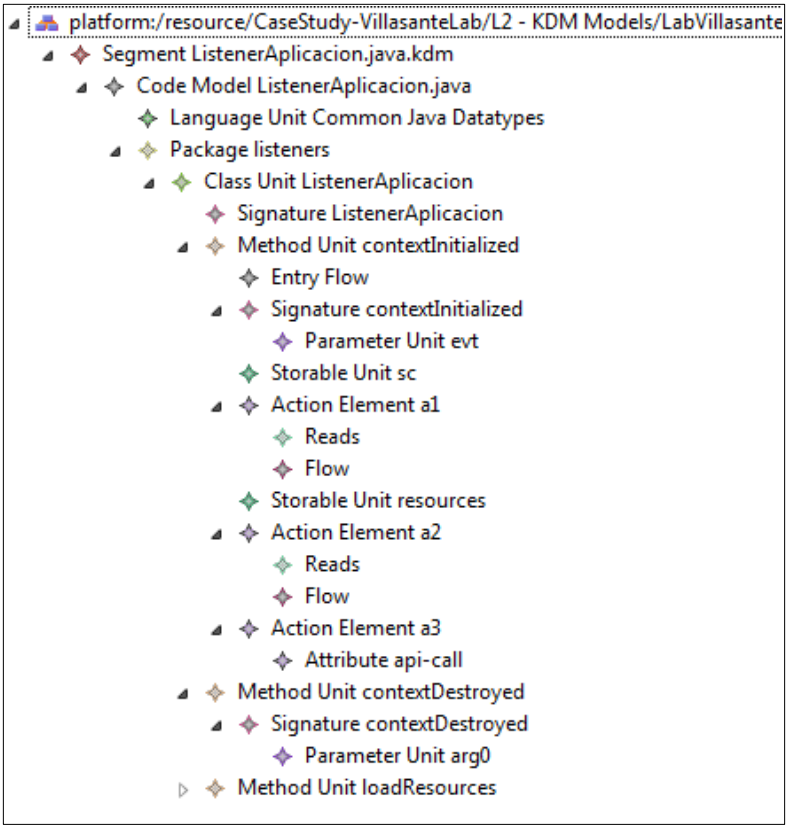


Figura 8.21. Parte del modelo KDM del ejemplo de aplicación en el editor KDM

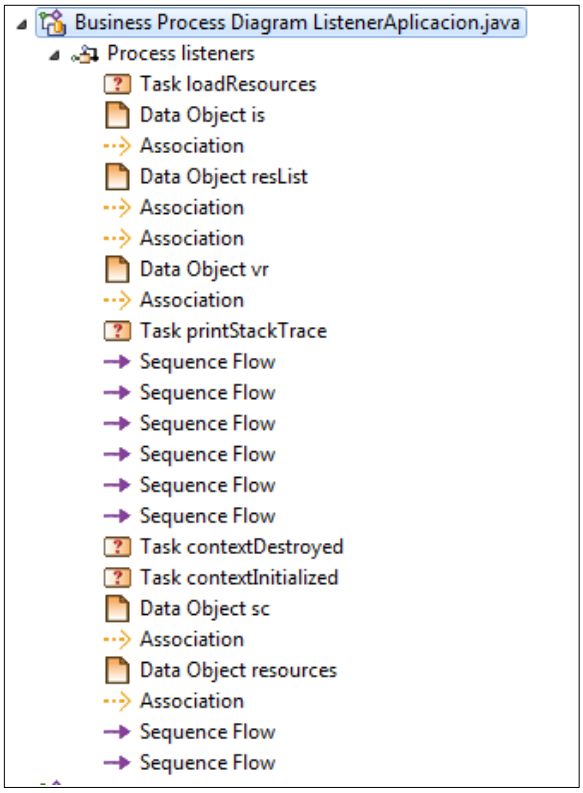


Figura 8.22. Parte del modelo BPMN del ejemplo de aplicación en el editor

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:bpmn="http://esi.uclm.es/metamodels/bpmn">

  ...
 <bpmn:BusinessProcessDiagram Name="ListenerAplicacion.java">
  <Processes Name="listeners">
   <GraphicalElements xsi:type="bpmn:Task" Name="loadResources" TaskType="None"/>
   <GraphicalElements xsi:type="bpmn:DataObject" Name="is"/>
   <GraphicalElements xsi:type="bpmn:Association" SourceRef="/10/@Processes.0/@GraphicalElements.0"
TargetRef="/10/@Processes.0/@GraphicalElements.1" Direction="One"/>
   <GraphicalElements xsi:type="bpmn:DataObject" Name="resList"/>
   <GraphicalElements xsi:type="bpmn:Association" SourceRef="/10/@Processes.0/@GraphicalElements.3"
TargetRef="/10/@Processes.0/@GraphicalElements.0" Direction="One"/>
   <GraphicalElements xsi:type="bpmn:Association" SourceRef="/10/@Processes.0/@GraphicalElements.1"
TargetRef="/10/@Processes.0/@GraphicalElements.0" Direction="One"/>
   <GraphicalElements xsi:type="bpmn:DataObject" Name="vr"/>
   <GraphicalElements xsi:type="bpmn:Association" SourceRef="/10/@Processes.0/@GraphicalElements.6"
TargetRef="/10/@Processes.0/@GraphicalElements.0" Direction="One"/>
   <GraphicalElements xsi:type="bpmn:Task" Name="printStackTrace"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/10/@Processes.0/@GraphicalElements.0"
TargetRef="/10/@Processes.0/@GraphicalElements.8"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/10/@Processes.0/@GraphicalElements.8"
TargetRef="/10/@Processes.0/@GraphicalElements.0"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/10/@Processes.0/@GraphicalElements.0"
TargetRef="/204/@Processes.0/@GraphicalElements.1"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/204/@Processes.0/@GraphicalElements.1"
TargetRef="/10/@Processes.0/@GraphicalElements.0"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/107/@Processes.0/@GraphicalElements.29"
TargetRef="/204/@Processes.0/@GraphicalElements.1"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/10/@Processes.0/@GraphicalElements.0"
TargetRef="/107/@Processes.0/@GraphicalElements.29"/>
   <GraphicalElements xsi:type="bpmn:Task" Name="contextDestroyed" TaskType="None"/>
   <GraphicalElements xsi:type="bpmn:Task" Name="contextInitialized" TaskType="None"/>
   <GraphicalElements xsi:type="bpmn:DataObject" Name="sc"/>
   <GraphicalElements xsi:type="bpmn:Association" SourceRef="/10/@Processes.0/@GraphicalElements.17"
TargetRef="/10/@Processes.0/@GraphicalElements.16" Direction="One"/>
   <GraphicalElements xsi:type="bpmn:DataObject" Name="resources"/>
   <GraphicalElements xsi:type="bpmn:Association" SourceRef="/10/@Processes.0/@GraphicalElements.19"
TargetRef="/10/@Processes.0/@GraphicalElements.16" Direction="One"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/10/@Processes.0/@GraphicalElements.16"
TargetRef="/107/@Processes.0/@GraphicalElements.144"/>
   <GraphicalElements xsi:type="bpmn:SequenceFlow" SourceRef="/107/@Processes.0/@GraphicalElements.144"
TargetRef="/10/@Processes.0/@GraphicalElements.16"/>
  </Processes>
 </bpmn:BusinessProcessDiagram>

  ...
</xmi:XMI>
```

Fragmento de código 8.3. Modelo BPMN (adaptado) del ejemplo de aplicación

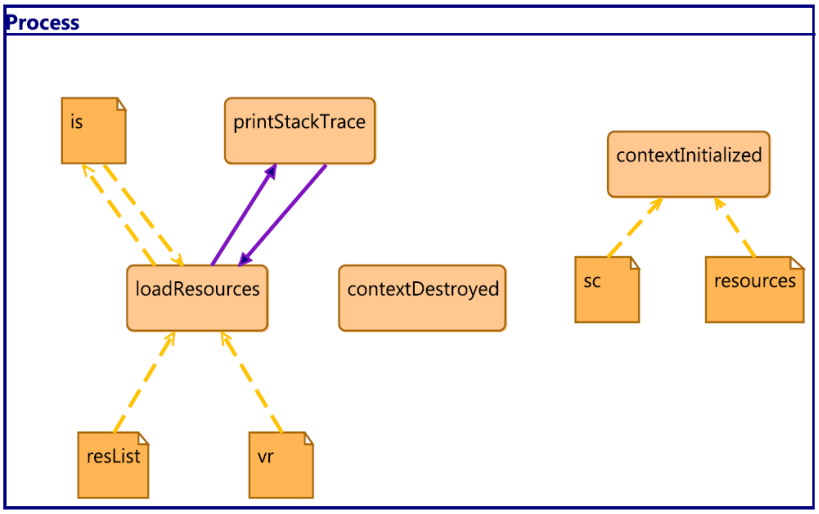


Figura 8.23. Parte de diagrama de procesos de negocio del ejemplo de aplicación

8.3.2.2. Resultados obtenidos

Una vez obtenidos los procesos de negocio, el experto de negocio puede proceder a realizar un análisis de los modelos y realizar las correcciones oportunas. Estas correcciones son recogidas en la Tabla 8.2. La tabla muestra que de 32 paquetes de los que dispone el sistema se han realizado correcciones y/o modificaciones a 14 de ellos, lo que supone que el 43,75% de los paquetes han necesitado intervención por parte del experto de negocio.

Paquete	Intervención manual en los modelos de procesos de negocio
src	R
security.manager	J[3,21], RN["Security Management"]
security.utils	J[2, 21], RN["Security Management"]
dao	R
manager	R
model	J[7], RN["Administration"]
web	J[6], RN["Administration"]
web.analysis	RN["Chemical Analysis Management"]
web.calibrations	RN["Chemical Calibration Management"]
web.customer	J[14], RN["User Management"]
web.dilution	RN["Chemical Dilution Management"]
web.bill	J[20,23], RN["Reporting"]
web.rol	R
web.user	J[10], RN["User Management"]
web.area	RN["District Management"]
exceptions	R
hibernate	R
listeners	R
mesages	R
pdf	J[12,23], RN["Reporting"]
validator	J[2,3], RN["Security Management"]
views	R
xml	J[12,20], RN["Reporting"]
servlet	R
servlet.analysis	R
servlet.calibration	R
servlet.customer	R
servlet.generic	R
servlet.tag	R
servlet.user	R
servlet.sitemesh	R
servlet.area	R

Tabla 8.2. Intervención del experto de negocio en los modelos obtenidos

8.3.2.3. Conclusiones del ejemplo de aplicación

Tras el análisis de los resultados anteriores y a través de una serie de encuestas a los empleados que utilizaron la herramienta MARBLE Tool se extrajeron las siguientes conclusiones:

- El equipo de desarrollo aprendió rápidamente el manejo de la herramienta.
- El grado de satisfacción de los empleados respecto a la herramienta fue óptimo, en cuanto a usabilidad.
- El experto en procesos de negocio realizó las correcciones de forma cómoda y sin grandes inconvenientes.
- Los modelos recuperados por la herramienta resultaron útiles para la empresa.

8.4. Anexo IV. Script QVT

```
transformation patterns (kdm:code, bpmn:bpmn){

   key BusinessProcessDiagram {Name};
   key Process {name};
   key bpmn::Task {Name};
   key Start {Name};
   key End {Name};
   key DataObject {Name};
   key bpmn::SequenceFlow {SourceRef, TargetRef};
   key Association {SourceRef, TargetRef};
   key Exclusive {Name};
   key Intermediate {Name};


       --################   P1. BPD SKELETON   ################

   top relation CodeModel2BPD {

      xName : String;

      checkonly domain kdm cm : code::CodeModel {
         name = xName
      };

      enforce domain bpmn bpd : BusinessProcessDiagram  {
         Name = xName
      };

      where {
         cm.codeElement->forAll (pk:code::AbstractCodeElement |
         pk.oclIsKindOf(code::Package) implies Package2Process(pk,bpd));
      }

   }

   relation Package2Process {

      xName : String;

      checkonly domain kdm pk : code::Package  {
         name = xName
      };

      enforce domain bpmn bpd: bpmn::BusinessProcessDiagram  {
         Name = xName,
         Processes = p : Process {
            Name = xName,
            GraphicalElements = e : bpmn::End {
               Name = getEndEventName (),
               EventType = bpmn::EventType::End
            },
            GraphicalElements = s : bpmn::Start {
               Name = getStartEventName(),
               EventType = bpmn::EventType::Start
            }
         }

      };

      where {
         pk.codeElement->forAll (c:code::AbstractCodeElement |
         c.oclAsType(code::ClassUnit).codeElement-
         >forAll(m:code::AbstractCodeElement | (m.oclIsKindOf(code::MethodUnit)
```

```
      and m.oclIsUndefined() and m.oclAsType(code::MethodUnit).name='main')
      implies    ClassUnitMain2Task (c, m,  p)));
      pk.codeElement->forAll (c:code::AbstractCodeElement |
      c.oclAsType(code::ClassUnit).codeElement-
      >forAll(m:code::AbstractCodeElement | (m.oclIsKindOf(code::MethodUnit)
      and m.oclIsUndefined() and m.oclAsType(code::MethodUnit).name<>'main')
      implies    MethodUnit2Task (m, p)));
   }

}


--###############   P2. SEQUENCE   ###############

relation MethodUnit2Task {

   xName : String;

   checkonly domain kdm m : code::MethodUnit {
      name = xName
   };

   enforce domain bpmn pr : bpmn::Process {
      GraphicalElements = t : bpmn::Task {
         Name = xName,
         Status = bpmn::StatusType::None,
         TaskType = bpmn::TaskType::None
      }
   };

   when {
      xName <>'main' and xName.substring(1, 3)<>'set' and xName.substring(1,
      3)<>'get';
   }

   where {
      m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclAsType(action::ActionElement).actionRelation->forAll
      (w:action::AbstractActionRelationship |
      (w.oclIsKindOf(action::Writes)) and
      w.oclAsType(action::Writes).to.oclIsKindOf(code::StorableUnit) implies
         WritesStorableUnit2DataObject (w, t, pr)));
      m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclAsType(action::ActionElement).actionRelation->forAll
      (r:action::AbstractActionRelationship | (r.oclIsKindOf(action::Reads))
      and r.oclAsType(action::Reads).to.oclIsKindOf(code::StorableUnit)
      implies ReadsStorableUnit2DataObject (r, t, pr)));

      m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclAsType(action::ActionElement).actionRelation->forAll
      (c:action::AbstractActionRelationship | (c.oclIsKindOf(action::Calls))
      and c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit)
      and (m.codeElement->forAll (a2:code::AbstractCodeElement |
      a2.oclAsType(action::ActionElement).actionRelation->forAll
      (tf:action::AbstractActionRelationship |
      tf.oclIsKindOf(action::TrueFlow) and tf.oclAsType(action::TrueFlow).to
      <> a) and a2.oclAsType(action::ActionElement).actionRelation->forAll
      (ff:action::AbstractActionRelationship |
      ff.oclIsKindOf(action::FalseFlow) and
      ff.oclAsType(action::FalseFlow).to <> a)))implies Calls2SequenceFlow
      (c, a, t, pr)));

      m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclAsType(action::ActionElement).actionRelation->exists
      (tf:action::AbstractActionRelationship |
      tf.oclIsKindOf(action::TrueFlow) and tf.oclIsUndefined() and
      tf.oclAsType(action::TrueFlow).to.oclIsKindOf(action::ActionElement))
      and    a.oclAsType(action::ActionElement).actionRelation->exists
```

```
      (ff:action::AbstractActionRelationship |
      ff.oclIsKindOf(action::FalseFlow) and ff.oclIsUndefined()   and
      ff.oclAsType(action::FalseFlow).to.oclIsKindOf(action::ActionElement))
      implies If2Gateway (a, t, pr));

      m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclAsType(action::ActionElement).actionRelation->exists
      (tf:action::AbstractActionRelationship |
      tf.oclIsKindOf(action::TrueFlow) and tf.oclIsUndefined() and
      tf.oclAsType(action::TrueFlow).to.oclIsKindOf(action::ActionElement))i
      mplies TrueFlow2ConditionalSequence (a, t, pr));

      m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclIsKindOf(action::ActionElement) and a.oclIsUndefined() implies
      ApiCall2CollaborationTask(a, t, pr));
   }

}

relation Calls2SequenceFlow {

   xName : String;

   checkonly domain kdm c : action::Calls {
      to = m : code::MethodUnit {
         name = xName
      }
   };

   checkonly domain kdm a : action::ActionElement{
      actionRelation = ac : action::Calls {
         from = c.from,
         to = c.to
      }
   };

   checkonly domain bpmn tk : bpmn::Task {
   };

   enforce domain bpmn pr : bpmn::Process {
      GraphicalElements = t : bpmn::Task {
         Name = tk.Name
      },
      GraphicalElements = t2 : bpmn::Task {
         Name = xName
      },
      GraphicalElements = sf : bpmn::SequenceFlow {
         SourceRef = t,
         TargetRef = t2
      },
      GraphicalElements = fe : bpmn::SequenceFlow {
         SourceRef = t2,
         TargetRef = e : bpmn::End {
            Name = getEndEventName (),
            EventType = bpmn::EventType::End
         }
      }
   };
   when {
      if (m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclAsType(action::ActionElement).actionRelation->forAll
      (c:action::AbstractActionRelationship | (c.oclIsKindOf(action::Calls))
      and c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit)
      and  not (m.codeElement->forAll (a2:code::AbstractCodeElement |
      a2.oclAsType(action::ActionElement).actionRelation->exists
      (tf:action::AbstractActionRelationship |
      tf.oclIsKindOf(action::TrueFlow) and
      tf.oclAsType(action::TrueFlow).to.name = a.name))))))
```

```
      then true
      else false
      endif;
      if (m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclAsType(action::ActionElement).actionRelation->forAll
      (c:action::AbstractActionRelationship | (c.oclIsKindOf(action::Calls))
      and c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit)
      and  not (m.codeElement->forAll (a2:code::AbstractCodeElement |
      a2.oclAsType(action::ActionElement).actionRelation->exists
      (ff:action::AbstractActionRelationship |
      ff.oclIsKindOf(action::FalseFlow) and
      ff.oclAsType(action::FalseFlow).to.name = a.name))))))
      then true
      else false
      endif;
   }
   where {
      a.actionRelation->forAll (f:action::AbstractActionRelationship |
      f.oclIsKindOf(action::Flow) and f.oclIsUndefined() and
      f.oclAsType(action::Flow).to.oclIsKindOf(action::ActionElement)
      implies Flow2SequenceFlow (f, t2, pr));
   }

}

relation Flow2SequenceFlow {

   checkonly domain kdm f : action::Flow {
      to = a : action::ActionElement {
      }
   };

   checkonly domain bpmn tk : bpmn::Task {
   };

   enforce domain bpmn pr : bpmn::Process {
      GraphicalElements = t : bpmn::Task {
         Name = tk.name
      }
   };

   where {
      a.actionRelation->forAll (c:action::AbstractActionRelationship |
      c.oclIsKindOf(action::Calls) and c.oclIsUndefined() and
      c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit) implies
      Calls2SequenceFlow (c, a, tk, pr));
      a.actionRelation->forAll (f:action::AbstractActionRelationship |
      (f.oclIsKindOf(action::Flow)) and f.oclIsUndefined() and
      f.oclAsType(action::Flow).to.oclIsKindOf(action::ActionElement)
      implies Flow2SequenceFlow (f, tk, pr));
   }

}

--###############   P3. BRANCHING   ###############

relation If2Gateway {

   checkonly domain kdm a: action::ActionElement {
      actionRelation = tf : action::TrueFlow {
         to = a1 : action::ActionElement {
         }
      },
      actionRelation = ff : action::FalseFlow {
         to = a2 : action::ActionElement {
         }
      }
   };
```

```
    checkonly domain bpmn tk : bpmn::Task {
    };

    enforce domain bpmn pr : bpmn::Process {
       GraphicalElements = t : bpmn::Task {
          Name = tk.name
       },
       GraphicalElements = g : bpmn::Exclusive {
          Name = a.name
       },
       GraphicalElements = sf : bpmn::SequenceFlow {
          SourceRef = t,
          TargetRef = g
       }
    };

    when {
    }

    where {
       a1.actionRelation->forAll (c:action::AbstractActionRelationship |
       c.oclIsKindOf(action::Calls) and c.oclIsUndefined() and
       c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit) implies
       If2SequenceFlow (c, a1, g, pr));
       a2.actionRelation->forAll (c:action::AbstractActionRelationship |
       c.oclIsKindOf(action::Calls) and c.oclIsUndefined() and
       c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit) implies
       If2SequenceFlow (c, a2, g, pr));
       a1.attribute->forAll(at:kdm::Attribute | at.tag='api-call' and
       at.oclIsUndefined() implies TrueFlow2SequenceFlowExt (at, a1, t, g,
       pr));
       a2.attribute->forAll(at:kdm::Attribute | at.tag='api-call' and
       at.oclIsUndefined() implies FalseFlow2SequenceFlowExt (at, a1, t, g,
       pr));
    }

}

relation If2SequenceFlow {

    xName : String;

    checkonly domain kdm c : action::Calls {
       to = m : code::MethodUnit {
          name = xName
       }
    };


    checkonly domain kdm a : action::ActionElement{
       actionRelation = ac : action::Calls {
          from = c.from,
          to = c.to
       }
    };

    checkonly domain bpmn g : bpmn::Exclusive {
    };

    enforce domain bpmn pr : bpmn::Process {
       GraphicalElements = g2 : bpmn::Exclusive {
          Name = g.Name
       },
       GraphicalElements = t2 : bpmn::Task {
          Name = xName
       },
       GraphicalElements = sf :bpmn::SequenceFlow {
```

```
            SourceRef = g,
            TargetRef = t2
         }
      };

      where {
         a.actionRelation->forAll (f:action::AbstractActionRelationship |
         f.oclIsKindOf(action::Flow) and f.oclIsUndefined() and
         f.oclIsUndefined() and
         f.oclAsType(action::Flow).to.oclIsKindOf(action::ActionElement)
         implies Flow2SequenceFlow (f, t2, pr));

      }

   }


      --###############   P4. COLLABORATION   ##############

   relation ApiCall2CollaborationTask {

      xValueName : String;

      checkonly domain kdm a : action::ActionElement{
         attribute = at : kdm::Attribute {
            value = xValueName
         }
      };

      checkonly domain kdm t : action::bpmn::Task {
      };

      enforce domain bpmn pr : bpmn::Process {
         GraphicalElements = t2 : bpmn::Task {
            Name = xValueName
         },
         GraphicalElements = sf :bpmn::SequenceFlow {
            SourceRef = t,
            TargetRef = t2
         },
         GraphicalElements = sf2 :bpmn::SequenceFlow {
            SourceRef = t2,
            TargetRef = t
         }
      };

      when {
         at.tag = 'api-call';
      }
      where {
      }

   }

   relation TrueFlow2SequenceFlowExt {

      xValueName : String;

      checkonly domain kdm at : kdm::Attribute {
         tag = 'api-call',
         value = xValueName
      };

      checkonly domain kdm a : action::ActionElement{
      };

      checkonly domain kdm t : action::bpmn::Task{
      };
```

```
    checkonly domain bpmn g : bpmn::Exclusive {
    };

    enforce domain bpmn pr : bpmn::Process {
       GraphicalElements = g2 : bpmn::Exclusive {
          Name = g.Name
       },
       GraphicalElements = t2 : bpmn::Task {
          Name = xValueName
       },
       GraphicalElements = sf :bpmn::SequenceFlow {
          SourceRef = g,
          TargetRef = t2
       },
       GraphicalElements = sf2 :bpmn::SequenceFlow {
          SourceRef = t2,
          TargetRef = t
       }
    };

    when {
    }

    where {
       a.actionRelation->forAll (f:action::AbstractActionRelationship |
       f.oclIsKindOf(action::Flow) and f.oclIsUndefined() and
       f.oclIsUndefined() and
       f.oclAsType(action::Flow).to.oclIsKindOf(action::ActionElement)
       implies Flow2SequenceFlow (f, t2, pr));
    }

}

relation FalseFlow2SequenceFlowExt {

    xValueName : String;

    checkonly domain kdm at : kdm::Attribute {
       tag = 'api-call',
       value = xValueName
    };

    checkonly domain kdm a : action::ActionElement{
    };

    checkonly domain kdm t : action::bpmn::Task{
    };

    checkonly domain bpmn g : bpmn::Exclusive {
    };

    enforce domain bpmn pr : bpmn::Process {
       GraphicalElements = g2 : bpmn::Exclusive {
          Name = g.Name
       },
       GraphicalElements = t2 : bpmn::Task {
          Name = xValueName
       },
       GraphicalElements = sf : bpmn::SequenceFlow {
          SourceRef = g,
          TargetRef = t2
       },
       GraphicalElements = sf2 : bpmn::SequenceFlow {
          SourceRef = t2,
          TargetRef = t
       }
    };
```

```
      when {
      }

      where {
         a.actionRelation->forAll (f:action::AbstractActionRelationship |
         f.oclIsKindOf(action::Flow) and f.oclIsUndefined() and
         f.oclIsUndefined() and
         f.oclAsType(action::Flow).to.oclIsKindOf(action::ActionElement)
         implies Flow2SequenceFlow (f, t2, pr));
      }

   }

   relation ConditionalCall2SequenceFlowExt {

      xValueName : String;

      checkonly domain kdm at : kdm::Attribute {
         tag = 'api-call',
         value = xValueName
      };

      checkonly domain kdm a : action::ActionElement{
      };

      checkonly domain kdm t : action::bpmn::Task{
      };

      checkonly domain bpmn i : bpmn::Intermediate {
      };

      enforce domain bpmn pr : bpmn::Process {
         GraphicalElements = i2 : bpmn::Intermediate {
            Name = i.Name
         },
         GraphicalElements = t2 : bpmn::Task {
            Name = xValueName
         },
         GraphicalElements = sf :bpmn::SequenceFlow {
            SourceRef = i,
            TargetRef = t2
         },
         GraphicalElements = sf2 :bpmn::SequenceFlow {
            SourceRef = t2,
            TargetRef = t
         }
      };

      where {
         a.actionRelation->forAll (f:action::AbstractActionRelationship |
         f.oclIsKindOf(action::Flow) and f.oclIsUndefined() and
         f.oclIsUndefined() and
         f.oclAsType(action::Flow).to.oclIsKindOf(action::ActionElement)
         implies Flow2SequenceFlow (f, t2, pr));
      }

   }


      --###############   P5. DATA INPUT   ###############

   relation ReadsStorableUnit2DataObject {

      xDataName : String;

      checkonly domain kdm r : action::Reads {
         to = su : code::StorableUnit {
```

```
            name = xDataName
         }
      };

      checkonly domain bpmn tk : bpmn::Task {
      };

      enforce domain bpmn pr : bpmn::Process {
         GraphicalElements = t : bpmn::Task {
            Name = tk.Name
         },
         GraphicalElements = do : bpmn::DataObject {
            Name = xDataName
         },
         GraphicalElements = as : bpmn::Association {
            SourceRef = do,
            TargetRef = t,
            Direction = bpmn::DirectionType::One
         }
      };

}


    --################   P6. DATA OUTPUT   ################

relation WritesStorableUnit2DataObject {

   xDataName : String;

   checkonly domain kdm w : action::Writes {
      to = su : code::StorableUnit {
         name = xDataName
      }
   };

   checkonly domain bpmn tk : bpmn::Task {
   };

   enforce domain bpmn pr : bpmn::Process {
      GraphicalElements = t : bpmn::Task {
         Name = tk.Name
      },
      GraphicalElements = do : bpmn::DataObject {
         Name = xDataName
      },
      GraphicalElements = as : bpmn::Association {
         SourceRef = t,
         TargetRef = do,
         Direction = bpmn::DirectionType::One
      }
   };

}


--################   P7. START   ################

relation ClassUnitMain2Task {

   xName : String;

   checkonly domain kdm c : code::ClassUnit {
      name = xName
   };

   enforce domain kdm m : code::MethodUnit {
      name = 'main'
```

```
};

enforce domain bpmn pr : bpmn::Process {
   GraphicalElements = t : bpmn::Task {
      Name = xName,
      Status = bpmn::StatusType::None,
      TaskType = bpmn::TaskType::None
   },
   GraphicalElements = s : Start {
      Name = getStartEventName(),
      EventType = bpmn::EventType::Start
   },
   GraphicalElements = sf :bpmn::SequenceFlow {
      SourceRef = s,
      TargetRef = t
   }
};

when {
   xName.substring(1, 3)<>'set' and xName.substring(1, 3)<>'get';
}

where {
   m.codeElement->forAll (a:code::AbstractCodeElement |
   a.oclAsType(action::ActionElement).actionRelation->forAll
   (w:action::AbstractActionRelationship |
   (w.oclIsKindOf(action::Writes)) and
   w.oclAsType(action::Writes).to.oclIsKindOf(code::StorableUnit) implies
      WritesStorableUnit2DataObject (w, t, pr)));
   m.codeElement->forAll (a:code::AbstractCodeElement |
   a.oclAsType(action::ActionElement).actionRelation->forAll
   (r:action::AbstractActionRelationship | (r.oclIsKindOf(action::Reads))
   and r.oclAsType(action::Reads).to.oclIsKindOf(code::StorableUnit)
   implies ReadsStorableUnit2DataObject (r, t, pr)))
   m.codeElement->forAll (a:code::AbstractCodeElement |
   a.oclAsType(action::ActionElement).actionRelation->exists
   (tf:action::AbstractActionRelationship |
   tf.oclIsKindOf(action::TrueFlow) and tf.oclIsUndefined() and
   tf.oclAsType(action::TrueFlow).to.oclIsKindOf(action::ActionElement))
   and a.oclAsType(action::ActionElement).actionRelation->exists
   (ff:action::AbstractActionRelationship |
   ff.oclIsKindOf(action::FalseFlow) and ff.oclIsUndefined() and
   ff.oclAsType(action::FalseFlow).to.oclIsKindOf(action::ActionElement))
   implies If2Gateway (a, t, pr));

   m.codeElement->forAll (a:code::AbstractCodeElement |
   a.oclAsType(action::ActionElement).actionRelation->exists
   (tf:action::AbstractActionRelationship |
   tf.oclIsKindOf(action::TrueFlow) and tf.oclIsUndefined() and
   tf.oclAsType(action::TrueFlow).to.oclIsKindOf(action::ActionElement))
   implies TrueFlow2ConditionalSequence (a, t, pr));

   m.codeElement->forAll (a:code::AbstractCodeElement |
   a.oclAsType(action::ActionElement).actionRelation->forAll
   (c:action::AbstractActionRelationship | (c.oclIsKindOf(action::Calls))
   and c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit)
   and (m.codeElement->forAll (a2:code::AbstractCodeElement |
   a2.oclAsType(action::ActionElement).actionRelation->forAll
   (tf:action::AbstractActionRelationship |
   tf.oclIsKindOf(action::TrueFlow) and tf.oclAsType(action::TrueFlow).to
   <> a) and a2.oclAsType(action::ActionElement).actionRelation->forAll
   (ff:action::AbstractActionRelationship |
   ff.oclIsKindOf(action::FalseFlow) and
   ff.oclAsType(action::FalseFlow).to <> a)))implies Calls2SequenceFlow
   (c, a, t, pr)));
```

```
      m.codeElement->forAll (a:code::AbstractCodeElement |
      a.oclIsKindOf(action::ActionElement) and a.oclIsUndefined() implies
      ApiCall2CollaborationTask(a, t, pr));


   }
}

query getStartEventName () : String {
   'startEvent'
}


--################   P8. IMPLICIT TERMINATION   ################

query getEndEventName () : String {
   'endEvent'
}


--################   P9. CONDITIONAL SEQUENCE   ################

relation TrueFlow2ConditionalSequence {

   checkonly domain kdm a: action::ActionElement {
      actionRelation = tf : action::TrueFlow {
         to = a1 : action::ActionElement {
         }
      }
   };

   checkonly domain bpmn tk : bpmn::Task {
   };

   enforce domain bpmn pr : bpmn::Process {
      GraphicalElements = t : bpmn::Task {
         Name = tk.name
      },
      GraphicalElements = i : bpmn::Intermediate {
         Name = 'condition_'+a.name,
         EventType = bpmn::EventType::Intermediate
      },
      GraphicalElements = sf : bpmn::SequenceFlow {
         SourceRef = t,
         TargetRef = i
      }
   };

   when {
      not a.actionRelation->exists (ff:action::AbstractActionRelationship |
      ff.oclIsKindOf(action::FalseFlow));
   }

   where {
      a1.actionRelation->forAll (c:action::AbstractActionRelationship |
      c.oclIsKindOf(action::Calls) and c.oclIsUndefined() and
      c.oclAsType(action::Calls).to.oclIsKindOf(code::MethodUnit) implies
      Then2SequenceFlow (c, a1, i, pr));

      a1.attribute->forAll(at:kdm::Attribute | at.tag='api-call' implies
      ConditionalCall2SequenceFlowExt (at, a1, t, i, pr));
   }
}

relation Then2SequenceFlow {

   xName : String;
```

```
      checkonly domain kdm c : action::Calls {
         to = m : code::MethodUnit {
            name = xName
         }
      };

      checkonly domain kdm a : action::ActionElement{
         actionRelation = ac : action::Calls {
            from = c.from,
            to = c.to
         }
      };

      checkonly domain bpmn i : bpmn::Intermediate {
      };

      enforce domain bpmn pr : bpmn::Process {
         GraphicalElements = i2 : bpmn::Intermediate {
            Name = i.Name
         },
         GraphicalElements = t2 : bpmn::Task {
            Name = xName
         },
         GraphicalElements = sf : bpmn::SequenceFlow {
            SourceRef = i,
            TargetRef = t2
         }
      };

      where {
         a.actionRelation->forAll (f:action::AbstractActionRelationship |
         f.oclIsKindOf(action::Flow) and f.oclIsUndefined() and
         f.oclAsType(action::Flow).to.oclIsKindOf(action::ActionElement)
         implies Flow2SequenceFlow (f, t2, pr));

      }

   }


   --###############   P10. EXCEPTION   ###############

}
```

ABREVIATURAS Y ACRÓNIMOS

A

ADM	Architecture Driven Modernization

B

BP	Business Process
BPD	Business Process Diagram
BPM	Business Process Management
BPMI	Business Process Management Initiative
BPMN	Business Process Management Notation

C

CASE	Computer Aided Software Engineering
CdU	Caso de Uso
CIM	Computation Independent Model
CWM	Common Warehouse Metamodel

E

EMF	Eclipse Modeling Framework

G

GEF	Graphical Editing Framework
GMF	Graphical Modeling Framework
GUI	Graphical User Interface

I

IDE	Integrated Development Environment

J

JDBC	Java Database Connectivity
JSP	Java Server Pages

K

KDM	Knowledge Discovery Metamodel
KLOC	thousands (Kilo) of Lines Of Code

L

LIS	Legacy Information System

M

MARBLE	Modernization Approach for Recovering Business process from Legacy systems
MDA	Model Driven Architecture
MDD	Model Driven Development
MOF	Meta Object Facility

O

OCL	Object Constraint Language
ODBC	Open Database Connectivity
OMG	Object Management Group

P

PDF	Portable Document Format
PFC	Proyecto Fin de Carrera
PIM	Platform Independent Model
PS	Producto de Salida
PSM	Platform Specific Model
PUD	Proceso Unificado de Desarrollo

Q

QVT	Query/View/Transformation

R

RUP	Rational Unified Process

S

SQL	Structured Query Language

U

UCLM	Universidad de Castilla-La Mancha
UML	Unified Modeling Language

W

W3C	World Wide Web Consortium

X

XMI	XML Metadata Interchange
XML	Extensible Markup Language