UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

AndrIU: Herramienta para la extracción, representación y migración de Interfaces Gráficas de Usuario de Sistemas de Información Heredados

Rafael Gómez-Cornejo Rubio

Septiembre, 2012

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

AndrIU: Herramienta para la extracción, representación y migración de Interfaces Gráficas de Usuario de Sistemas de Información Heredados

Autor: Rafael Gómez-Cornejo Rubio
Director: Dr. Ricardo Pérez del Castillo.
Tutor Académico: Dr. Ignacio García Rodríguez de Guzmán

Septiembre, 2012

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

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

RESUMEN

En la actualidad las nuevas plataformas y dispositivos móviles han ocupando un lugar muy importante en la sociedad, aportando nuevas soluciones y aplicaciones basadas en los principios de la movilidad y la ubicuidad. La industria del software está realizando grandes esfuerzos a la hora de facilitar y automatizar el desarrollo de aplicaciones para las plataformas móviles. Las nuevas plataformas móviles además fuerzan a los ingenieros de mantenimiento software a proveer soluciones más ágiles para migrar las aplicaciones existentes hacia estas nuevas plataformas móviles. Mientras la lógica de negocio de aplicaciones bien modularizadas puede ser reutilizada fácilmente, la migración de las interfaces de usuario entraña un gran desafío que normalmente es ignorado.

En el presente Proyecto de Fin de Carrera se presenta el desarrollo de la herramienta *AndrIU*, para automatizar la migración de interfaces gráficas de usuario hacia aplicaciones Android. La migración contempla la extracción y representación de información relacionada con las interfaces de usuario de sistemas de información heredados a través de modelos, y la posterior generación de interfaces gráficas de usuario para plataformas móviles. *AndrIU* se presenta como un plug-in para el entorno de desarrollo Eclipse, añadiendo una serie de herramientas y funcionalidades para realizar procesos típicos de la reingeniería y migración software. Los modelos gestionados por la herramienta se basan en especificaciones recogidas en estándares internacionales como ISO/IEC 19506. El uso de estos estándares junto con el hecho de que la herramienta *AndrIU* ha sido desarrollada como un plug-in para Eclipse hacer factible su aplicación en la industria de la ingeniería del software, así como su interoperabilidad con otras herramientas relacionadas.

ABSTRACT

Nowadays the new platforms and mobile devices have taken a very important place in our society, providing new solutions and applications based on the mobile and pervasive environment principles. The software industry is making great efforts providing and automatizing the application development for new mobile platforms. The new platforms force the software maintainers to provide agile solutions in order to migrate existing applications to these new mobile platforms. While business logic of tier-based modularized systems can be easily reused, software user interfaces migration entails a big challenge which is usually ignored.

This Grade Thesis presents the development of AndrIU, a tool for automatizing graphical user interfaces migration to Android applications. The migration process includes both data extraction and representation related to graphical user interfaces of legacy information systems by models, and graphical user interfaces generation for mobile platforms. AndrIU has been made as a plug-in for the integrated development environment Eclipse, adding a set of tools and performances to make typical software reengineering and migration processes. The models managed by this tool are based on specifications established by international standard ISO/IEC 19506. The use of these standards together with the fact that the AndrIU tool has been made as an Eclipse plug-in make possible its use in the software engineering industry, as well as its interoperatibily with other related tools.

DEDICATORIA

Dedicatoria

AGRADECIMIENTOS

Agradecimientos

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1

Introducción

En este primer capítulo se presenta una introducción del Proyecto de Fin de Carrera (PFC), así como la problemática en la que se centra. De este modo, se motivan y fundamentan las necesidades que han dado lugar a la realización del Proyecto, explicando el contexto donde se engloba el mismo. Este capítulo concluye con la descripción de la estructura que sigue el resto del documento.

1. INTRODUCCIÓN

Este capítulo presenta una introducción del Proyecto de Fin de Carrera (PFC), así como la problemática en la que se centra. Se motivan y fundamentan las necesidades que han dado lugar a la realización del PFC y se explica el contexto donde se engloba.

1.1. Introducción al tema

La evolución tecnológica que se está produciendo en los últimos años está provocando un cambio en la vida cotidiana, de manera que los entornos que nos rodean son cada vez más ubicuos, y permiten la interacción con una gran cantidad de servicios a través de una gran variedad de dispositivos móviles. Los nuevos dispositivos móviles como los *smartphones* o las *tablets* (véase Figura 1.1) son los que están tomando un papel más importante, al integrar de manera sencilla e integrada todo tipo de servicios (Schmidt, 2012), tales como los gestores de correo electrónico y agendas, sistemas de información geográficos o videojuegos; y diferentes servicios como mensajería instantánea, videoconferencias, redes sociales, etc.

Figura 1.1. Dispositivos móviles: *smartphones* y *tablets*

Es por ello por lo que estos entornos ubicuos están forzando a los desarrolladores de software a proveer aplicaciones y servicios móviles de manera ágil, para adaptarse de rápidamente a estas nuevas necesidades y a estos nuevos dispositivos. De hecho, el desarrollo de estas aplicaciones en tiendas virtuales como *App Store*,

Google Play o *Android Market* (véase Figura 1.2) está creciendo un 20% por mes y ofrecen ya más de 400.000 aplicaciones (Van Agten, 2012).

Figura 1.2. Sitio web de *Android Martket*

En las últimas décadas, investigadores e ingenieros han proporcionado diferentes métodos y técnicas ágiles para aliviar las dificultades que surgen en el desarrollo de software tradicional. Sin embargo, en numerosas ocasiones el objetivo de los desarrolladores no es sólo realizar nuevos desarrollos de aplicaciones desde cero, sino también migrar aplicaciones o sistemas existentes a nuevas plataformas. Esto es así porque muchas empresas y organizaciones llevan realizando sus actividades diarias de negocio mediante aplicaciones y Sistemas de Información (SI) existentes (Redman, 2008; Loshin, 2010). Con el paso del tiempo estos sistemas de información quedan tecnológicamente obsoletos, ya que fueron creados con tecnologías y/o métodos de desarrollo anteriores. De este modo surgen los llamados sistemas de información heredados (del inglés *Legacy Information System* (LIS)). Estos sistemas pueden llegar a convertirse en sistemas obsoletos tecnológicamente y quizás podrían no seguir soportando las reglas de negocio actuales que la empresa u organización llevan a cabo concretamente. De hecho, la calidad de los sistemas de información heredados puede llegar a degradarse por debajo de límites aceptables (Pérez-Castillo, García-Rodríguez

de Guzmán et al., 2011). La degradación de calidad junto con la rápida evolución de las Tecnologías de la Información y la Comunicación (TIC) hace que las empresas demanden cada vez más una rápida adaptación de sus SI, a fin de mantener su competitividad en el mercado.

En la actualidad existen varias estrategias para adaptar, evolucionar o modernizar los LIS. El principal problema de estas estrategias es que la mayoría descarta las reglas de negocio que se han embebido como consecuencia del mantenimiento evolutivo a lo largo del tiempo. Esas reglas de negocio se implementaron en los sistemas de información y por eso mismo podrían no estar presentes en ningún otro activo de las organizaciones. Por tanto (junto con la rápida evolución de los sistemas de información) las empresas están obligadas a preservar toda la lógica de negocio embebida en sus LIS. Una de las estrategias que se ha utilizado con más éxito para afrontar el mantenimiento evolutivo, y más concretamente la migración de LIS, es la reingeniería software. La reingeniería propone un proceso que consiste en tres etapas: ingeniería inversa, restructuración e ingeniería directa (Chikofsky and Cross, 1990). De acuerdo a (Arnold, 1993), la reingeniería consiste en la recuperación de representaciones abstractas de un LIS mediante ingeniería inversa para, posteriormente reestructurarlo y finalmente, mediante ingeniería directa generar una versión del sistema que, probablemente cuente con una mejora de la calidad, nuevas características, disponible para otras plataformas, etc. Los LIS son el punto de origen que se toma en la reingeniería para desarrollar nuevas versiones mejoradas.

A pesar de que la reingeniería es una buena técnica para el mantenimiento de sistemas software, adolece de estandarización y formalización, por lo que en muchos casos los proyectos de reingeniería pueden fracasar (Sneed, 2005). Debido a estas deficiencias, y como solución a los mismas, surge el concepto de Modernización del Software, que mejora la reingeniería tradicional mediante la aplicación de los principios del desarrollo dirigido por modelos (del inglés *Model Driven Architecture* (MDA)) (OMG, 2003a). De hecho, a fin de definir y estandarizar la modernización software, el Object Management Group (OMG) lanzó la iniciativa de la modernización dirigida por la arquitectura (del inglés *Architecture Driven Modernization* (ADM)) (OMG, 2006), que proporciona las especificaciones necesarias para llevar a cabo estos procesos de modernización.

A pesar de todos estos esfuerzos, las interfaces de usuario (IU o del inglés *User Interfaces* (UI)) son ignoradas por la mayoría de técnicas y métodos de modernización del software. No obstante, en los LIS las IU, que suelen ser normalmente gráficas (comúnmente nombradas como *Graphical User Interfaces* o GUI), juega un papel muy importante. Esto es debido a que la GUI soporta toda la interacción entre los usuarios y el sistema, y es la encargada de dar respuesta a dichas interacciones conectando con la lógica de negocio soportada por el sistema de información. Esto hace que se emplee gran parte del tiempo y de los recursos del desarrollo del software en el análisis, diseño e implementación de las GUI. Actualmente existen algunas representaciones estandarizadas de las GUI que son independientes de la plataforma y la tecnología. Además existen herramientas que generan de forma semi-automática GUI para plataformas concretas. Estos esfuerzos de formalización y automatización están encaminados a acortar de manera significativa los tiempos de desarrollo de las aplicaciones (E. Stroulia, 1999). Desafortunadamente, estos esfuerzos no se han realizado de forma significativa en ingeniería inversa, y en general en reingeniería, de GUI para facilitar su modernización y migración a nuevas plataformas móviles

Es por ello que surge la necesidad de realizar tareas de migración basadas en la modernización del software, centrándose especialmente en las GUI de los LIS. Este Proyecto Fin de Carrera aborda precisamente la reingeniería de las GUI de forma estándar y automática, para facilitar su migración a plataformas móviles. Más concretamente, el objetivo de este Proyecto de Fin de Carrera consiste en el análisis, diseño y construcción de AndrIU, una herramienta que permite la extracción y representación de información sobre las GUI de sistemas heredados, y su migración a otras plataformas, utilizando modelos estandarizados. Como resultado principal de este proyecto, se ha desarrollado AndrIU, una herramienta que consiste en un conjunto de plug-ins para el entorno de desarrollo software integrado (del inglés *Integrated Development Environment* (IDE)) Eclipse™. AndrIU facilita la implantación de la técnica de ingeniería inversa propuesta y se asegura su futura y fácil extensión e integración con otros plug-ins de Eclipse™. En concreto, la herramienta AndrIU toma como entrada sistemas heredados que utilizan la tecnología Java, pero resulta relativamente sencillo ampliar dicha herramienta con nuevos plug-ins que permitan otros lenguages y tenconogías. Del mismo modo, la herramienta pretende realizar la

migración de las GUI a plataformas móviles abiertas como Android (Android_Developers), pero con la posibilidad de ser ampliada a otras plataformas móviles en el futuro.

1.2. Estructura del documento

A continuación se describe de manera resumida la estructura de los contenidos de la memoria de este PFC. Dicha memoria consta de siete capítulos y una serie de anexos:

Capítulo 1. Introducción: en este primer capítulo se realiza una introducción de la problemática que ha dado lugar a la realización del presente PFC, así como la contextualización del mismo.

Capítulo 2. Objetivos: en este capítulo se detallan los principales objetivos del presente Proyecto Fin de Carrera.

Capítulo 3. Estado de la Cuestión: en este capítulo se resumen los temas relevantes que conciernen al presente PFC a través de la documentación consultada sobre dichos temas, haciendo mención a la bibliografía relevante.

Capítulo 4. Método de Trabajo: en este capítulo se describe el método de trabajo seguido para la consecución de los objetivos descritos en el capítulo 2. Se describen las iteraciones planificadas y se describe la metodología de desarrollo utilizada: el Proceso Unificado de Desarrollo (PUD).

Capítulo 5. Resultados: en este capítulo se exponen los resultados obtenidos en el desarrollo de este Proyecto Fin de Carrera tras la aplicación del PUD, según las iteraciones descritas en el capítulo 4.

Capítulo 6. Conclusiones y Propuestas: en este capítulo se presentan las conclusiones obtenidas tras la realización del presente proyecto y se establecen posibles líneas de trabajo futuras.

Capítulo 7. Referencias bibliográficas: en este último capítulo se proporciona una lista con las referencias bibliográficas consultadas para la realización de este PFC.

Anexo I. Manual de usuario: este primer anexo presenta el manual de usuario de la herramienta AndrIU, especificando aspectos como la instalación, configuración y uso de la misma.

Anexo II. Casos de Estudio: este segundo anexo describe los resultados obtenidos al utilizar la herramienta AndrIU con aplicaciones reales.

Capítulo 2

Objetivos del Proyecto

En este capítulo se describen los objetivos del presente Proyecto de Fin de Carrera, así como los medios técnicos y tecnológicos que emplean para llevar a cabo dichos objetivos.

2. OBJETIVOS DEL PROYECTO

Este capítulo describe los objetivos del presente PFC, así como los medios técnicos y tecnológicos que se emplean para llevar a cabo dichos objetivos.

2.1. Objetivos

El principal objetivo del Proyecto de Fin de Carrera consiste en el desarrollo de un entorno para la extracción y análisis de información sobre las interfaces gráficas de usuario de sistemas heredados con el fin de migrar estas interfaces a otras plataformas móviles. Con este proyecto se pretende extraer información de la capa de presentación de los sistemas heredados para generar modelos de IU e integrar estos con los modelos del resto de capas del sistema. Más concretamente, este PFC establece el siguiente objetivo principal (véase Figura 2.1.).

> *Desarrollar una herramienta software, consistente en un plug-in para Eclipse™, que permita la migración de GUIs de sistemas existentes a través de la extracción y representación de estas GUIs mediante modelos estandarizados.*

Figura 2.1. Objetivo Principal del Proyecto de Fin de Carrera

2.2. Sub-objetivos

El objetivo general descrito en el apartado 2.1 se desglosa en los siguientes objetivos parciales:

- **Op.1.** Estudiar el estado del arte sobre enfoques similares.
- **Op.2.** Analizar, diseñar y desarrollar un analizador sintáctico para la extracción de información de la IU de un sistema heredado.
- **Op.3.** Proveer un mecanismo para la representación de la información independiente de la plataforma e integración con otros artefactos también obtenidos.
- **Op.4.** Definir un medio de extracción de relaciones entre la interfaz de usuario y el resto del sistema.
- **Op.5.** Generar interfaces gráficas de usuario para una nueva aplicación móvil utilizando los modelos obtenidos anteriormente.

Op.6. Analizar, diseñar e implementar un plug-in que soporte la ingeniería inversa de GUIs.

2.3. Objetivos didácticos

Con el desarrollo de este Proyecto de Fin de Carrera, se pretende alcanzar los siguientes objetivos didácticos:

Od.1. Adquirir las destrezas y conocimientos necesarios para el desarrollo de proyectos informáticos.

Od.2. Aprender a realizar un desarrollo software siguiendo la metodología de desarrollo Proceso Unificado de Desarrollo (PUD), que ha sido estudiado en diversas asignaturas durante la carrera.

Od.3. Estudiar la arquitectura de plug-ins de Eclipse y adquirir los conocimientos de desarrollo de los mismos.

Od.4. Instruirse en los estándares internacionales para la representación del conocimiento en el ámbito del software.

Od.5. Crear y utilizar procesadores de lenguajes (*parsers*) para procesar documentos y generar nueva información

2.4. Módulos del proyecto

Partiendo del objetivo principal o global descrito en el apartado 2.1, y teniendo en cuenta todos los sub-objetivos descritos en el apartado 2.2, el proyecto se puede dividir conceptualmente en 3 módulos. Estos módulos vienen a representar una abstracción del objetivo principal para facilitar su comprensión.

Módulo 1: desarrollar las funcionalidades de una herramienta que permita la extracción automática de la información referente a las GUI de un sistema heredado tomado como entrada, y su representación en modelos estandarizados e independientes de la plataforma o la tecnología (metamodelo KDM) (véase la Figura 2.2.). Para la generación de estos modelos, es necesario generar previamente un modelo de árbol sintáctico abstracto (del inglés *abstract syntax* tree o AST), que es dependiente de la plataforma.

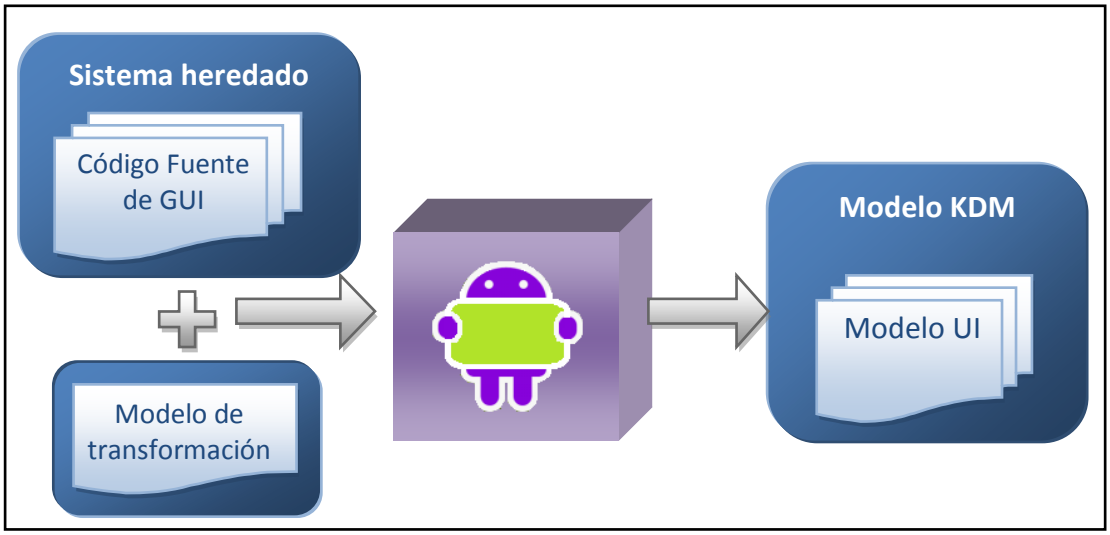


Figura 2.2. Esquema del módulo 1

Módulo 2: desarrollar las funcionalidades de la herramienta que permita la generación automática de interfaces de usuario para una plataforma determinada a partir de la información obtenida en el sub-objetivo 1 (véase la Figura 2.3.).

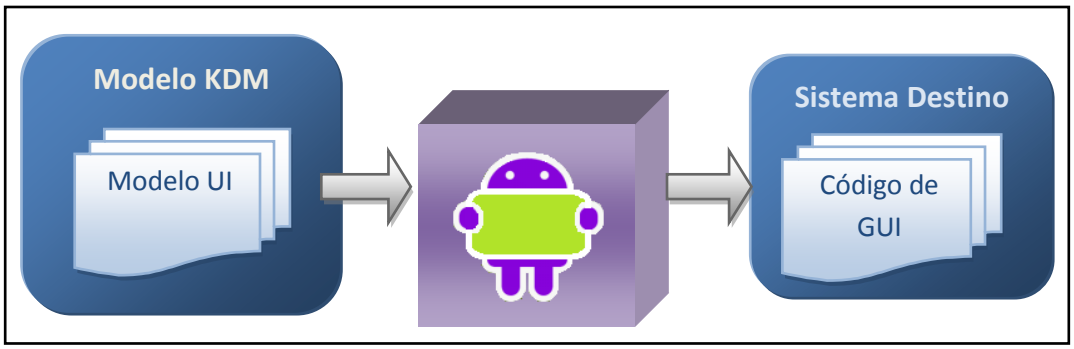


Figura 2.3. Esquema del módulo 2

Módulo 3: desarrollar las funcionalidades de la herramienta que permita la representación gráfica de los modelos de GUI a través de un editor gráfico creado con EMF/GMF. En la Figura 2.4. se puede observar de forma esquematizada este módulo.

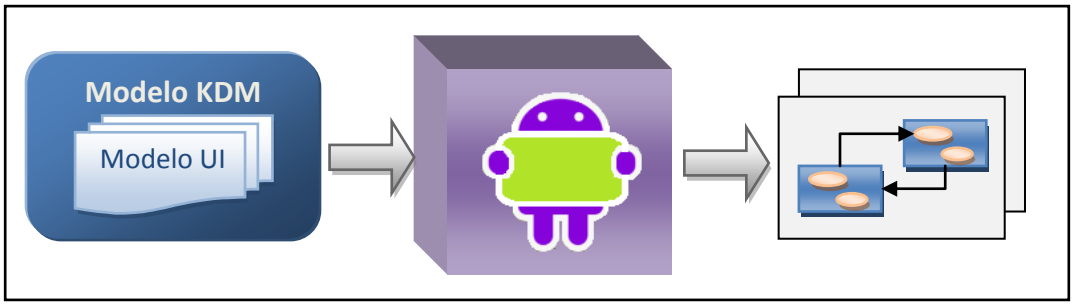


Figura 2.4. Esquema del módulo 3

2.5. Medios empleados

Los medios y recursos, tanto tecnológicos como materiales, utilizados para la realización de los objetivos definidos son los siguientes:

- El proyecto se ha realizado en un ordenador personal con sistema operativo *Microsoft Windows 7 Ultimate Edition*.
- Para las etapas de análisis y diseño, se ha utilizado la herramienta CASE *Visual Paradigm* (v 8.1), con licencia académica proporcionada por la Escuela Superior de Informática. Esta herramienta soporta UML 2.0.
- Para la realización de la planificación del PFC, se ha utilizado la herramienta *Microsoft Project 2003*, con licencia académica proporcionada por la Escuela Superior de Informática.
- Para realizar el diseño de prototipos de las interfaces de usuario de la herramienta AndrIU, se ha utilizado la herramienta *Balsamiq Mockups* (v. 2.1.14), con licencia gratuita de prueba.
- Para la implementación de la herramienta AndrIU se ha utilizado el lenguaje de programación Java 1.6 y el IDE *Eclipse* (Eclipse, 2011a), en su versión Indigo con la distribución *Modeling Tools*[1]. También se han utilizado los paquetes para Eclipse *Eclipse Plug-in Development Environment* y *Android Development Tools.*
- Para la implementación de funcionalidades que requieren de la manipulación de documentos XML se ha utilizado la librería *Jdom 1.1* (Hunter, 2009), así como la librería *KMD.*
- Para la generación de un parser para el lenguaje de programación Java se ha utilizado el generador de analizadores sintácticos *JavaCC 5.0.* (JavaCC, 2009), así como una gramática de Java obtenida de http://javacc.java.net/.

[1] URL de Eclipse Modeling Tools: http://www.xxx.com

- Para la realización de pruebas software se han utilizado los plug-ins de Eclipse *JUnit* 4.8 (Gamma and Beck, 2010) y *Eclemma* (Hoffmann and Janiczak, 2011).

- Para la realización de la memoria y el resto de la documentación se ha utilizado *Microsoft Office Word* 2007, con el plug-in *EndNote X5* para la gestión de notas y referencias bibliográficas.

- Como herramientas de propósito general, se han utilizado *Acrobat Reader 9*, *Gimp 2.6*, *DropBox*, entre otras.

- Como recursos materiales, se han utilizado libros, artículos de investigación, manuales, y otra documentación de interés, tanto en formato físico como en formato digital.

Capítulo 3

Estado de la cuestión

En este capítulo se detalla el estado del arte y la base de conocimiento relacionada con el Proyecto de Fin de Carrera, exponiéndose los conceptos que fundamentan y sirven como base teórica.

3. ESTADO DE LA CUESTIÓN

Primero se introducirá el concepto de sistema de información heredado, así como la problemática que surge al quedar obsoleto para las empresas u organizaciones (ver sección 3.1). Como posible solución tradicional a estos problemas se presenta en segundo lugar el concepto de modernización software, abordando temas como la reingeniería software y las limitaciones que presenta y otros conceptos, metodologías y estándares relacionados con la propia modernización software como evolución de la reingeniería (ver sección 3.2). En tercer lugar, se presenta el metamodelo KDM, un estándar que modela SI, y que ofrece además una completa representación de la parte correspondiente a las GUI de estos sistemas (ver sección 3.3). Tras definir las posibles soluciones a la problemática de los sistemas heredados en cuanto a su modernización, se presentará en cuarto lugar el concepto de interfaz gráfica de usuario de estos sistemas heredados, así como de la importancia que tienen dentro de estos sistemas heredados (ver sección 3.4). A continuación, en quinto lugar, se introduce el concepto de las plataformas móviles, y más concretamente de los sistemas Android como ejemplo de plataformas abiertas que se están imponiendo en la industria actual (ver sección 3.5). Por último, se describe el entorno de desarrollo Eclipse, que ha sido utilizado para la consecución de este proyecto, y de la importancia de las aplicaciones orientadas a plug-ins para el desarrollo de aplicaciones y herramientas software (ver sección 3.6).

3.1. Sistemas de Información Heredados

La evolución que requieren los SI a través de la modificación y agregación de nuevas funcionalidades hace que éstos vayan embebiendo una gran cantidad de lógica y reglas de negocio. Esta información de negocio se va adquiriendo durante el mantenimiento incontrolado que sufren los SI, y en muchas ocasiones la información de negocio podría no estar presente en ningún otro activo de la empresa u organización que no sea código fuente del propio sistema heredado (Sommerville, 2006). Es por ello por lo que los SI resultan imprescindibles en las organizaciones. Pero estos sistemas de información heredados pueden quedar obsoletos tecnológicamente, de manera que se resulte imposible la tarea de realizar nuevas versiones del sistema sobre la tecnología sobre la que se apoya, o que los niveles de calidad y mantenibilidad de dichos sistemas han disminuido por debajo de unos límites aceptables (Polo, Piattini et al., 2003).

En estos casos, se requiere el desarrollo de un nuevo sistema, utilizando tecnologías más actuales y que mantenga toda la lógica y reglas de negocio del sistema LIS. Sin embargo es muy desaconsejable el desarrollo de nuevos sistemas equivalentes desde cero ya que parte de esta información, que únicamente está reflejada en el sistema heredado, podría perderse. Para evitar este problema resulta más apropiado realizar un mantenimiento evolutivo del LIS a fin de preservar la información de negocio embebida. Una de las soluciones más empleadas y con mejores resultados para conseguir el mantenimiento evolutivo de sistemas de información heredados es la reingeniería software.

3.2. Modernización del Software Tradicional

El concepto de modernización del software surge como una evolución de la reingeniería software tradicional, para paliar las carencias de estandarización y formalización, así como de la automatización de estos procesos. De este modo, la modernización software se basa en el desarrollo de procesos de reingeniería siguiendo los principios del Desarrollo Dirigido por Modelos (del inglés *Model Driven Development* y en adelante MDD) (Mellor, 2003).

3.2.1. Reingeniería Software

La reingeniería software es una técnica para realizar el mantenimiento evolutivo de sistemas de información heredados. Se define como el replanteamiento y el rediseño de estos sistemas de información para mejorarlo en aspectos como la calidad, el diseño, la mantenibilidad, el coste, etc. (Hammer and Champy, 1994). De este modo, la reingeniería ofrece la posibilidad de incorporar y desarrollar nuevas reglas de negocio, o de mejorar las existentes, para que las empresas u organizaciones puedan llevar a cabo un proceso de mejora continua que les permita mantener su competencia.

Según (Chikofsky and Cross, 1990), y coincidiendo con (Arnold, 1993), la reingeniería es un proceso mediante el cual, un sistema heredado se ve sometido a las siguientes etapas: (I) ingeniería inversa, (II) restructuración y (III) ingeniería directa.

3.2.1.1. Ingeniería inversa

Es la primera etapa, y en ella se obtiene un conjunto de especificaciones abstractas a partir del sistema heredado, que son usadas a posteriori, para construir una nueva implementación del sistema software. De este modo se obtiene una nueva representación del sistema a un nivel de abstracción mayor, lo que implica en cierto modo la pérdida de información (Chikofsky and Cross, 1990).

3.2.1.2. Restructuración

En esta segunda etapa se toma como partida las representaciones del sistema heredado obtenidas en la etapa anterior, y se realizan transformaciones al mismo nivel de abstracción para obtener nuevas representaciones que mejoran de algún modo las propiedades del sistema. Estas transformaciones deben respetar el comportamiento externo del sistema de origen, para poder preservar la lógica y las reglas de negocio embebidas en el sistema (Chikofsky and Cross, 1990).

3.2.1.3. Ingeniería directa

Una vez realizada la restructuración, en esta etapa se vuelve a bajar en el nivel de abstracción, construyendo una nueva versión operativa del sistema. De este modo se realiza un proceso de ingeniería directa tomando los modelos de representación del sistema fuente obtenidos en la etapa anterior para generar un nuevo sistema con nuevas funcionalidades (Chikofsky and Cross, 1990).

Es muy común representar estas etapas de la reingeniería a través del denominado modelo en herradura (véase Figura 3.1.) (Kazman, Woods et al., 1998). En la etapa de ingeniería inversa aumenta el nivel de abstracción, en la etapa de reestructuración se mantiene el mismo nivel de abstracción, y en la etapa de ingeniería directa vuelve a disminuir el nivel de abstracción.

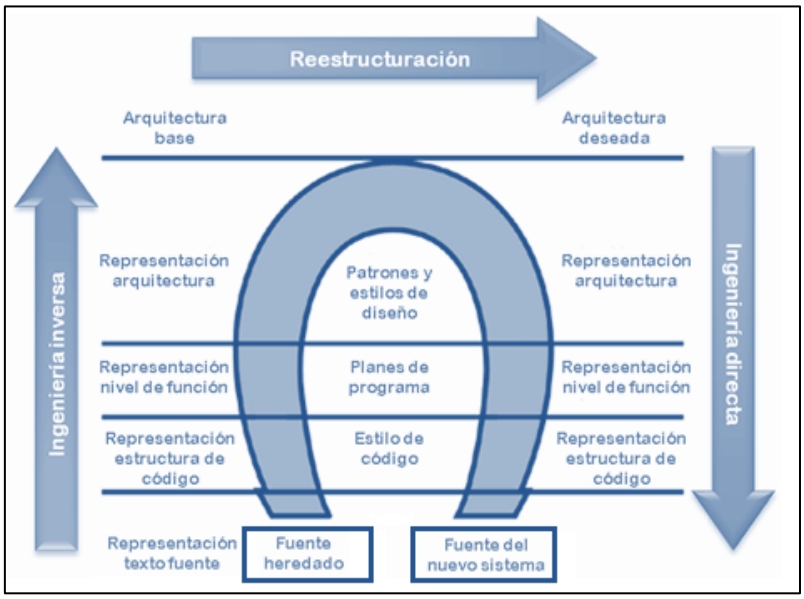


Figura 3.1. Modelo de reingeniería en herradura (Kazman, Woods et al., 1998)

Este proceso de reingeniería software se comenzó a aplicarse en la industria con éxito hace unas décadas. Sin embargo, algunos estudios revelan que más de la mitad de los proyectos actuales de reingeniería suelen fallar (Sneed, 2005). Esto se debe a la falta de formalización y estandarización del propio proceso de reingeniería, así como su falta de automatización (Canfora and Di Penta, 2007). La escasa estandarización y formalización hace que el proceso de reingeniería no pueda ser integrado y reutilizado en otros proyectos, y a su vez esto deriva en el problema de automatización. Al no utilizarse estándares que permitan formalizar el proceso de la reingeniería, resulta casi imposible automatizar los productos obtenidos en cada una de las etapas del proceso entre diferentes proyectos. Los principios del desarrollo dirigido por modelos ayuda a soliviantar los problemas de estandarización y formalización.

3.2.2. Model Driven Development (MDD)

Según (Mellor, 2003), MDD es el concepto de que se puede construir un modelo de un SI para poder transformarlo en otro. Por tanto, en el MDD los artefactos principales del desarrollo son los modelos de los SI (no el código fuente de del software de los SI) y las transformaciones entre dichos modelos. MDD implica una generación semi-automática de la implementación a partir de los modelos.

El concepto de modelo ha sido definido por varios autores de las siguientes formas:

- "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).

- "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, 2003b).

- "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).

3.2.2.1. Model Driven Architecture (MDA)

Para poder aplicar MDD, es necesaria la especificación de algún estándar que lo soporte. Para ello el OMG define MDA (*Model Driven Architecture*, o en español Arquitectura Dirigida por Modelos)(OMG, 2003a; OMG, 2003b). MDA define dos principios:

I. Modelar todos los artefactos como modelos de diferentes niveles de abstracción.

II. Establecer modelos de transformación entre ellos

De este modo, MDA define tres tipos de modelos según su nivel de abstracción (véase Figura 3.2.):

- **Computation Independent Model (CIM):** es una visión del sistema a un alto nivel de abstracción, y es independiente del punto de vista de la computación. Este modelo muestra el sistema en el entorno en el que va a operar y ayuda a

representar la funcionalidad del sistema.

- **Platform Independent Model (PIM):** es una visión del sistema a un nivel de abstracción intermedio, e independiente de la plataforma del mismo. Representa los modelos que describen una solución de software que no contiene detalles de la plataforma concreta en que va a ser implementada.

- **Platform Specific Model (PSM):** es una visión del sistema desde un bajo nivel de abstracción, y está enfocado en una plataforma concreta. Combina las especificaciones del PIM con las particularidades tecnológicas de la plataforma en que se va a implementar el nuevo SI.

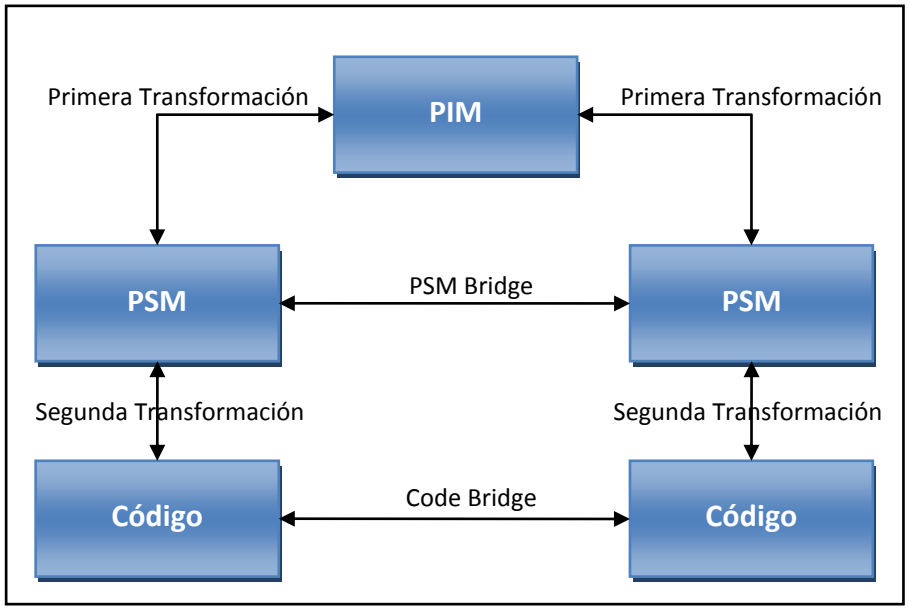


Figura 3.2. Modelos que define MDA

Para pasar de un modelo a otro, deben diseñarse transformaciones automáticas entre dichos modelos, que describen la forma en que un modelo en un determinado nivel de abstracción debe ser transformado en otro modelo en distinto nivel de abstracción. A esto se le conoce como correspondencia entre entidades de ambos modelos (OMG, 2003b), y están especificados en los modelos de transformación. De este modo, todos los modelos representan un mismo sistema, pero a diferentes niveles de abstracción.

3.2.3. Architecture-Driven Modernization (ADM)

La Modernización Dirigida por la Arquitectura (del inglés *Architecture-Driven Modernization* y en adelante ADM) es una iniciativa propuesta por el OMG cuyo objetivo principal es la estandarización del proceso de modernización de los LIS centrándose en los aspectos principales de su arquitectura a través de la definición y uso de metamodelos (OMG, 2007).

De este modo, ADM aplica la modernización del software siguiendo el enfoque de MDA utilizando transformaciones entre modelos para obtener la arquitectura deseada partiendo de la arquitectura del LIS. Por tanto, el modelo en herradura propuesto para el proceso de reingeniería tradicional puede aplicarse del mismo modo a ADM, siguiendo las distintas fases de ingeniería inversa, restructuración e ingeniería directa (véase Figura 3.3.). La modernización software no remplaza el proceso de reingeniería tradicional, sino que lo mejora mediante la aplicación de los principios descritos por MDA.

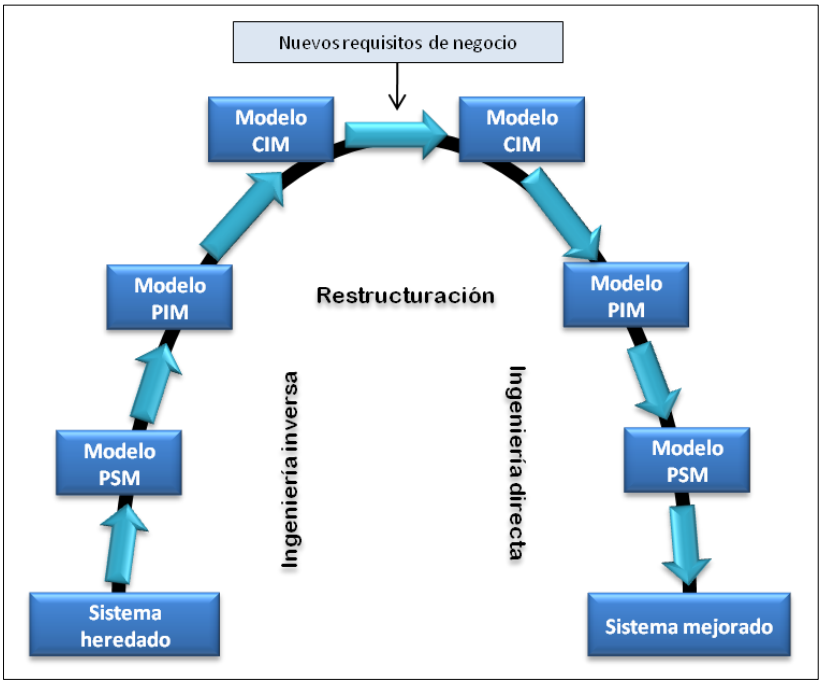


Figura 3.3. Modelo en herradura aplicado a ADM

3.3. Metamodelo KDM (ISO/IEC 19506)

El Metamodelo para el Descubrimiento de Conocimiento (del inglés *Knowledge Discovery Metamodel* y en adelante KDM), estándar internacional ISO/IEC 19506 (ISO/IEC, 2009), especifica un conjunto de conceptos comunes para representar los LIS para la modernización de software, y proporciona infraestructuras para soportar definiciones de conocimiento específicas de dominio, específicas de la aplicación o específicas de la implementación (OMG, 2009).

La estructura de KDM está definida por la combinación diferentes niveles de abstracción que definen capas de abstracción. Cada una de estas capas se encarga de modelar o representar un aspecto determinado de un LIS (OMG, 2009). A su vez, cada una de estas capas contiene un conjunto de paquetes, que se encargan de representar diferentes aspectos dentro de su nivel de abstracción. De este modo, KDM está formado por 12 paquetes, estructurados en 4 capas como muestra la Figura 3.4.

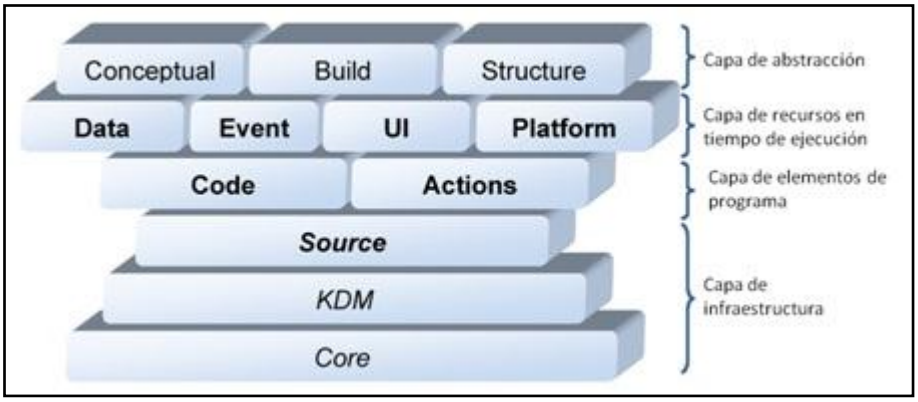


Figura 3.4. Estructura de paquetes KDM (traducción de (OMG, 2009))

A continuación se hará una breve descripción de los diferentes paquetes que conforman KDM según (ISO/IEC, 2009):

- **Core**: define las abstracciones básicas de KDM.
- **Kdm**: proporciona el contexto compartido por todos los modelos KDM.
- **Source**: define el conjunto de artefactos físicos del LIS y permite referenciar partes del código fuente.
- **Code**: define a bajo nivel los procedimientos, tipos de datos, unidades de compilación, etc.

- **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*).
- **Platform**: define los artefactos relacionados con la plataforma de ejecución.
- **UI**: define los aspectos de la interfaz de usuario del LIS.
- **Event**: define conceptos comunes relacionados con la programación dirigida a eventos (lo que comúnmente se llama modelo de eventos).
- **Data**: define los aspectos de los datos persistentes del LIS.
- **Structure**: define los componentes estructurales de los LIS (subsistemas, capas, paquetes, etc.).
- **Conceptual**: define los elementos específicos de dominio del LIS.
- **Build**: define los artefactos finales relativos al LIS.

3.3.1. El paquete UI

Como el objetivo de este proyecto se centra en las IU de los LIS, el paquete de KDM que representará estos elementos, y por tanto en el que se centrará el modelado, el paquete UI, perteneciente a la capa de *recursos en tiempo de ejecución*. El paquete UI de KDM define un conjunto de elementos del metamodelo que permiten representar las los aspectos relacionados a las UI, incluyendo su composición, su secuencia de operaciones y sus relaciones con el LIS (OMG, 2009).

El diagrama de clases del paquete UI de KDM se puede observar en la Figura 3.5. En él se pueden observar los elementos del metamodelo que definen componentes, eventos, acciones, flujos, etc, de los elementos de la GUI, permitiendo modelarlos.

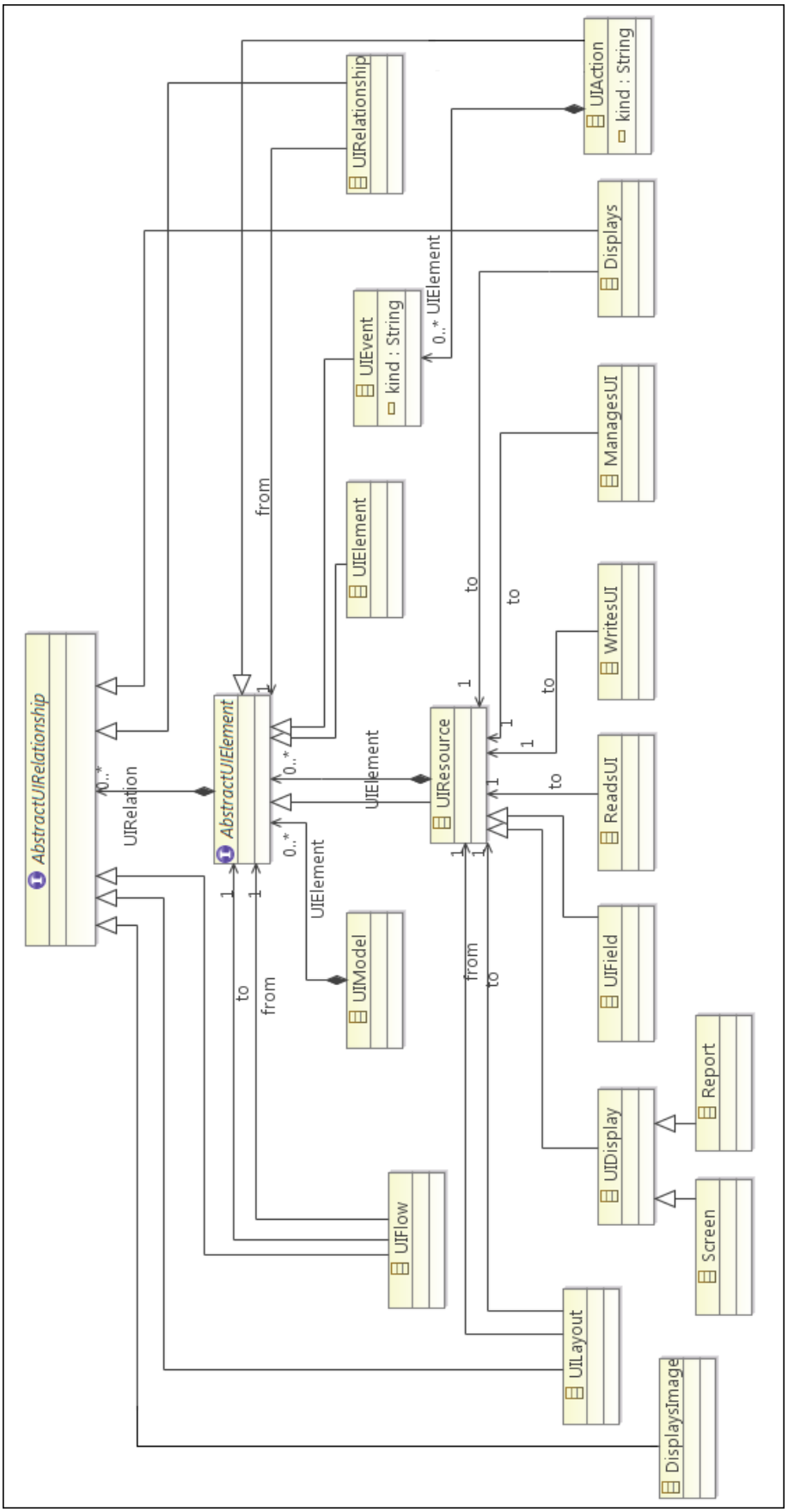


Figura 3.5. Diagrama de Clases del paquete UI de KDM

Este paquete depende además de otros paquetes de KDM, permitiendo relacionar los aspectos de la GUI modelados con otros aspectos del propio LIS. De este modo, los paquetes de los que depende son: Action, Code, kdm, Source y Core (OMG, 2009).

3.4. Interfaces de Usuario

Las interfaces de usuario son los componentes software de un SI que se encargan de realizar la interacción entre el usuario y el propio SI (T. Moran, 1996). Existen muchos tipos de IU, y han ido evolucionando gracias a las TI, desde las primeras interfaces de tipo consola, donde el usuario debía conoces una serie de directivas o instrucciones para comunicarse con el sistema, hasta las interfaces gráficas de usuario, donde el usuario dispone de una gran cantidad de elementos gráficos (ventanas, botones, campos de texto, imágenes, etc) que le permiten realizar esta comunicación de una manera mucho más fácil, rápida y cómoda.

3.4.1. Interfaces Gráficas de Usuario

Las interfaces gráficas de usuario, o GUI (del inglés *Graphical User Interfaces*) son un tipo de IU las cuales se componen de un conjunto de elementos gráficos que facilitan la interacción entre el usuario y el SI. Las GUI están basadas en una metáfora visual, y la más utilizada es la metáfora del escritorio, que proporciona la idea de reproducir una sobremesa de oficina y todos los objetos que se tienen en ella (Julio Abascal, 2001). La base de la metáfora consiste en crear objetos electrónicos que simulan los objetos físicos en una oficina.

De este modo en las GUI clásicas se suelen tener una serie de componentes como elementos de interacción: ventanas, cuadros de diálogo, botones, botones de selección, etiquetas, cuadros y campos de texto, etc. Las interacciones que el usuario puede tener con estos componentes vienen generadas por los llamados eventos. De este modo, pueden ocurrir diferentes eventos sobre un mismo componente, como por ejemplo clic-derecho, clic-izquierdo o dejar pulsado sobre un componente botón. Para cada uno de estos eventos es necesario especificar un controlador de eventos asociado al componente, que se encarga de definir el comportamiento que va a tener el sistema al producirse dicho evento sobre el componente. A este modelo de interacción se le conoce

como “modelo-vista-controlador” (véase Figura 3.6.).

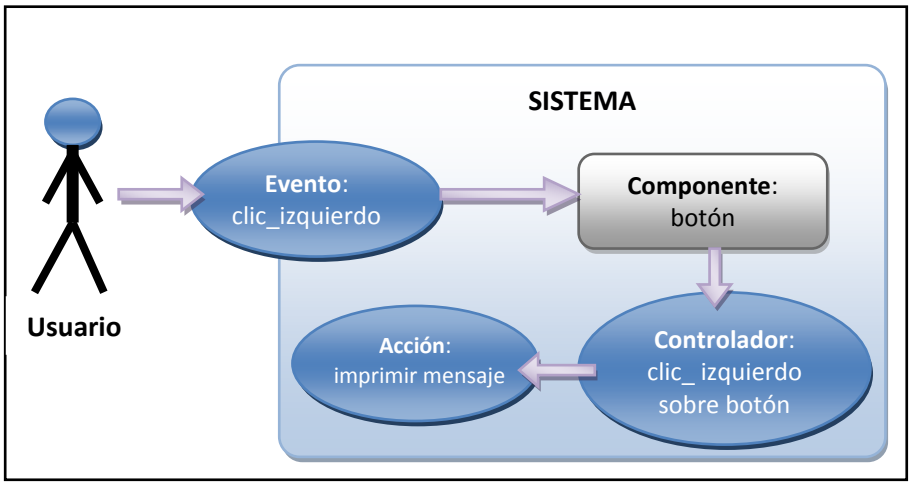


Figura 3.6. Ejemplo de interacción sobre un sistema a través de la GUI

3.4.2. Arquitectura multicapa

En el desarrollo de los SI actuales, uno de los patrones de diseño aplicado con más frecuencia es la “arquitectura multicapa”(Gamma, Helm et al., 1995). Esta arquitectura separa el diseño de la aplicación en varias capas. La más conocida es la arquitectura en tres capas, que define las siguientes capas (véase Figura 3.7.):

- **Capa de presentación**, donde se tienen las funcionalidades de interacción del sistema con el usuario (interfaz de usuario, entrada-salida de datos, etc.). En esta capa se encuentran las UI.
- **Capa de negocio**, donde se encuentran las principales funcionalidades de acuerdo a la lógica de negocio.
- **Capa de persistencia**, que se encarga de las tareas de almacenamiento y recuperación de los datos que manejará el sistema (ficheros, bases de datos, etc.).

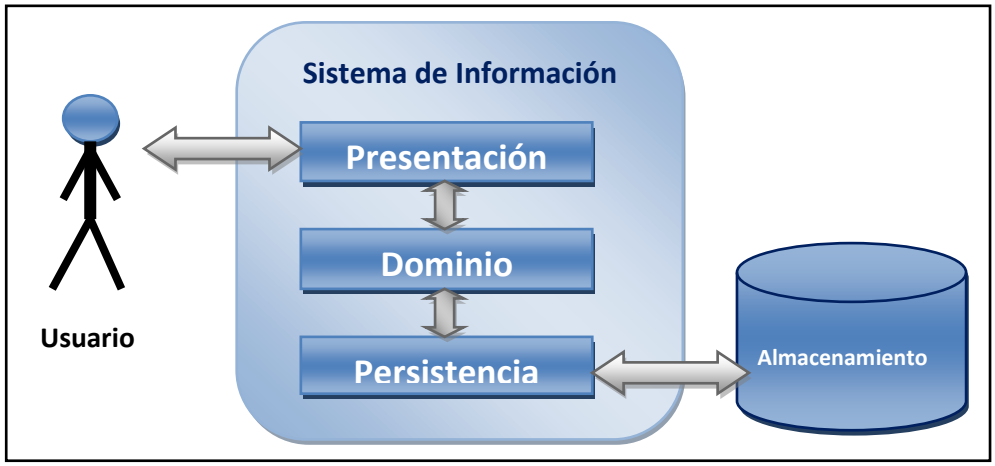


Figura 3.7. Esquema de la arquitectura en 3 capas

3.5. Plataformas móviles

Debido a la revolución del sector de las TIC que se está produciendo en las últimas décadas, el objetivo principal que se busca en este sector es la portabilidad y la capacidad de obtener servicios de forma remota y ubicua. Es por ello por lo que los dispositivos móviles como los *smartphones* o las *tablets* han revolucionando el mercado, gracias además a la relativa potencia que ofrecen en relación al reducido tamaño de los dispositivos y a la cantidad de servicios y aplicaciones que se desarrollan para los mismos. De este modo gran parte de la industria del software se centra en la mejora de las plataformas que dan soporte a estos dispositivos y en el desarrollo de aplicaciones para las mismas. Los sistemas que más éxito han conseguido por el momento son *Windows Mobile*, *iOS*, y especialmente *Android*.

3.5.1. Sistemas Android

Android es una plataforma de software libre para dispositivos móviles que incluye un sistema operativo, middleware y una serie de aplicaciones de propósito general (Android_Developers). Para desarrollar aplicaciones para sistemas basados en Android, es necesario el Android SDK (*Software Development Kit*), que proporciona las herramientas y las APIs (*Application Programming Interfaces*) para utilizar el lenguaje de programación Java. En la Figura 3.8. se muestra a modo de diagrama los principales componentes del sistema operativo Android.

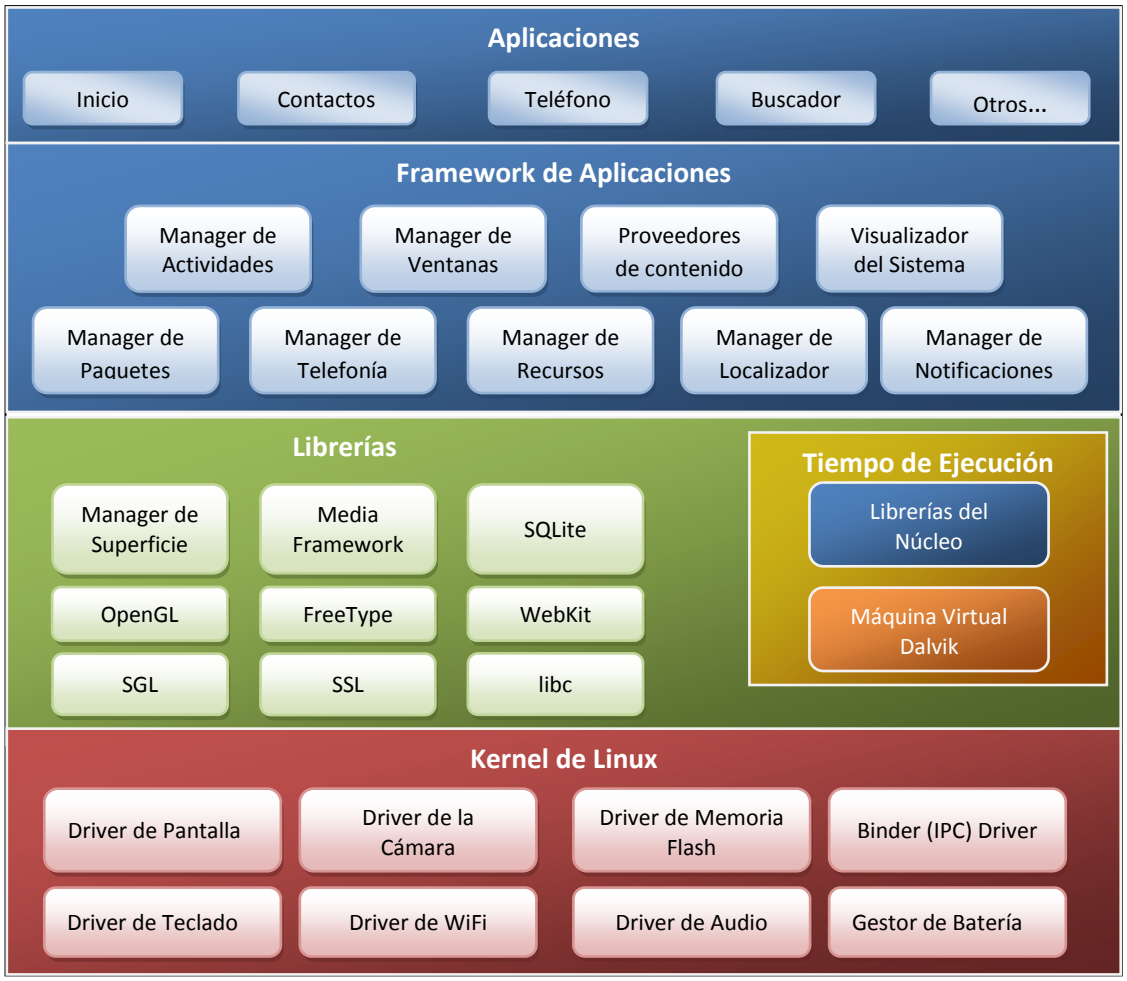


Figura 3.8. Arquitectura de Android (traducción de (Android_Developers)

3.5.2. Interfaces gráficas en Android

En una aplicación Android, la GUI se construye utilizando los objetos *Views* (Vistas) y *ViewGroups* (Grupos de Vistas). Existen muchos tipos de *Views* y *ViewGroups*, cada uno de los cuales hereda de la clase View.

Los objetos View son las unidades básicas de las UI de la plataforma Android, y la clase View sirve como base para las subclases llamadas *Widgets*, que ofrecen una implementación completa de los objetos que conformarán la GUI (como los campos de texto o los botones). La clase *ViewGroup* sirve como base para las subclases llamadas *Layouts*, que ofrecen diferentes tipos de agrupaciones de elementos (en vertical, en horizontal, etc). En la Figura 3.9. se puede observar la jerarquía de estos objetos.

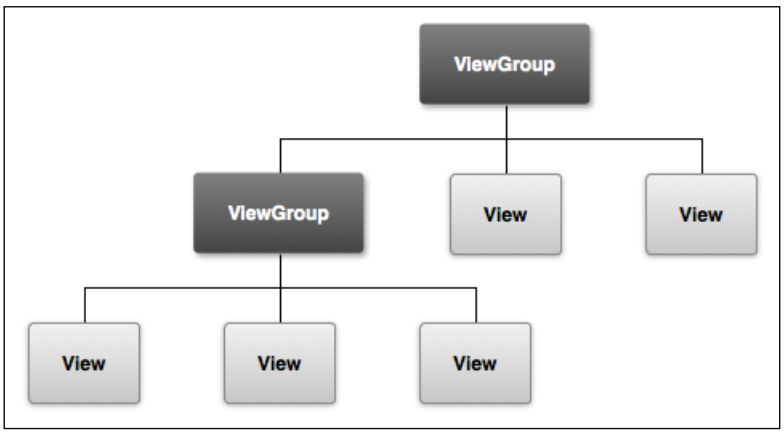


Figura 3.9. Jerarquía de los objetos de GUI de Android

Para definir estos objetos y su jerarquía, se utilizan ficheros de Layout en XML. Cada elemento en el XML es o un objeto *View* o un *ViewGroup*. El nombre del elemento XML hace referencia a la clase Java que lo implementa. En la Figura 3.10. se puede observar un ejemplo de la definición de un Layout que contiene un campo de texto y un botón.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Figura 3.10. Ejemplo de un Layout en XML

3.6. Entorno Integrado de Desarrollo Eclipse

Eclipse es un IDE de código abierto y multiplataforma, que permite la integración de múltiples herramientas que dan soporte al desarrollo software. El proyecto Eclipse fue creado por IBM en 2001 y en 2003 se creó la Fundación Eclipse, una organización sin ánimo de lucro, la cual pasó a hacerse cargo de la administración de Eclipse (Eclipse, 2011a).

De este modo, la Fundación Eclipse se encarga de desarrollar una serie de proyectos para la plataforma Eclipse, que dan lugar a *frameworks* (marcos de trabajo), herramientas y plug-ins, que añaden al IDE Eclipse nuevas funcionalidades. Esta es una

de las grandes ventajas que presenta la plataforma Eclipse, ya que existen numerosas extensiones que ofrecen herramientas para el desarrollo de aplicaciones software. De hecho, este PFC se centra en desarrollar un plug-in que ofrezca un marco de trabajo para labores de reingeniería, tomando como base la plataforma Eclipse.

3.6.1. Plug-ins para Eclipse

Ya que Eclipse es una plataforma que permite la integración de componentes para ampliar el IDE, surgió el concepto de plug-in (del inglés enchufado/conectado) un componente software que aporta nuevas funcionalidades. Un plug-in puede consumir servicios proporcionados por otros plug-ins o puede extender su funcionalidad para ser consumido por otros. Estos plug-ins son cargados dinámicamente por el propio Eclipse en tiempo de ejecución. De hecho, toda la funcionalidad de Eclipse se encuentra en diferentes plug-ins a excepción del núcleo principal (Clayberg and Rubel, 2008).

De este modo, un plug-in para Eclipse se puede ver como una aplicación que es desarrollada por separado y que se empaqueta conteniendo todo el código y los recursos que el plug-in necesita para funcionar correctamente cuando sea agregado al IDE de Eclipse.

El desarrollo de este PFC ha sido llevado a cabo con *Eclipse 3.7. Indigo*, la versión más reciente al comienzo del mismo. Además, de entre los proyectos disponibles para la plataforma *Eclipse Indigo*, para la realización de este PFC se han utilizado *Eclipse Modeling Tools* (Eclipse, 2011b) y *Android Development Tools* (Eclipse, 2011d).

3.6.2. Eclipse Modeling Tools

Este proyecto está formado por un conjunto de componentes de *Eclipse Modeling Proyect*, de entre los cuales se pueden destacar EMF (*Eclipse Modeling Framework*) y GMF (*Graphical Modeling Framework*).

3.6.2.1. Eclipse Modeling Framework (EMF)

EMF (*Eclipse Modeling Framework*) (Steinberg, Budinsky et al., 2008; Eclipse,

2010) es un *framework* de modelado que permite la generación de código para el desarrollo de herramientas a partir de la definición de un modelo de datos estructurado. Esta definición del modelo parte de una especificación en XMI (*XML Metadata Interchange*), creando el modelo Ecore. A partir de este modelo, 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 en forma de árbol (véase Figura 3.11.).

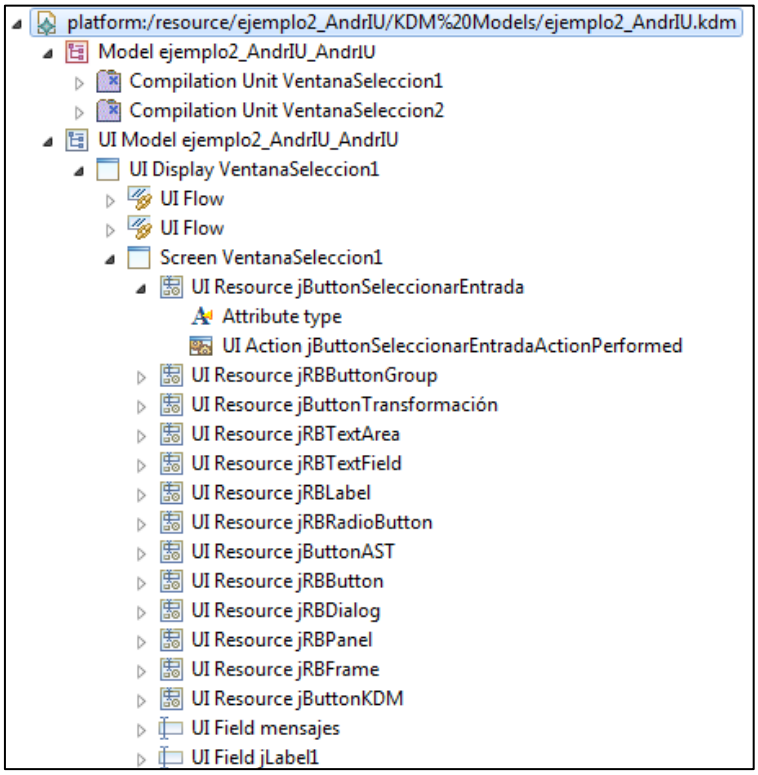


Figura 3.11. Ejemplo del editor de AndrIU con EMF

3.6.2.2. Graphical Modeling Framework (GMF)

GMF es un plug-in Eclipse que permite crear editores gráficos de modelos definidos mediante el metamodelo EMF. Es por ello por lo que GMF depende de EMF, y se debe partir de la definición del modelo Ecore, y los editores creados por EMF para poder generar los editores gráficos que proporciona GMF.

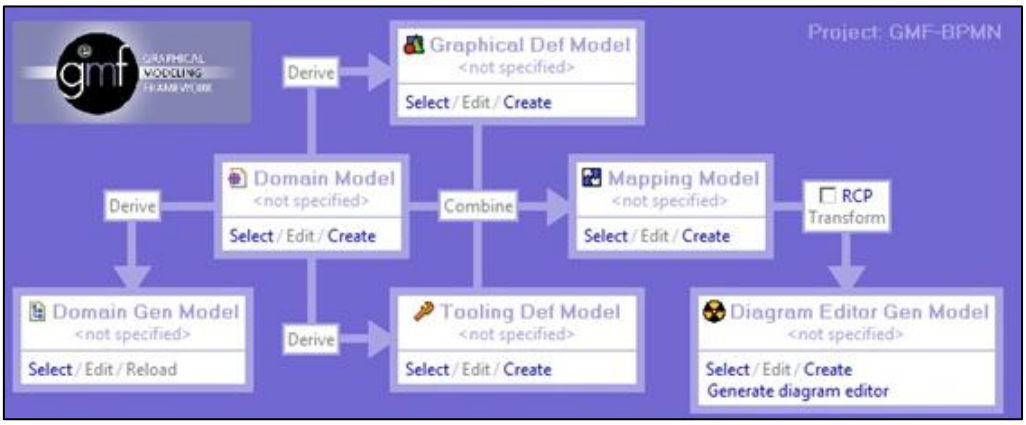


Figura 3.12. Creación de un proyecto GMF

Para crear un editor gráfico con GMF es necesario seguir los pasos que propone el propio GMF (Véase Figura 3.12.). 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.

3.6.3. Android Development Tools (ADT)

Android Development Tools (en adelante ADT) es un plug-in para el IDE Eclipse diseñado para proporcionar un potente entorno para el desarrollo de aplicaciones Android. ADT extiende las funcionalidades de eclipse para permitir la creación de *Android projects* (proyectos Android), el editores para el diseño de las UI de aplicaciones Android, la inclusión de paquetes basados en el API del *Framework* de Android, herramientas de depuración de aplicaciones Android, etc. (Eclipse, 2011d).

Para poder utilizar ATD es imprescindible instalar el Android SDK. Para facilitar la integración de todos estos componentes, ADT proporciona un manager para actualizar a las últimas versiones disponibles (véase Figura 3.13.).

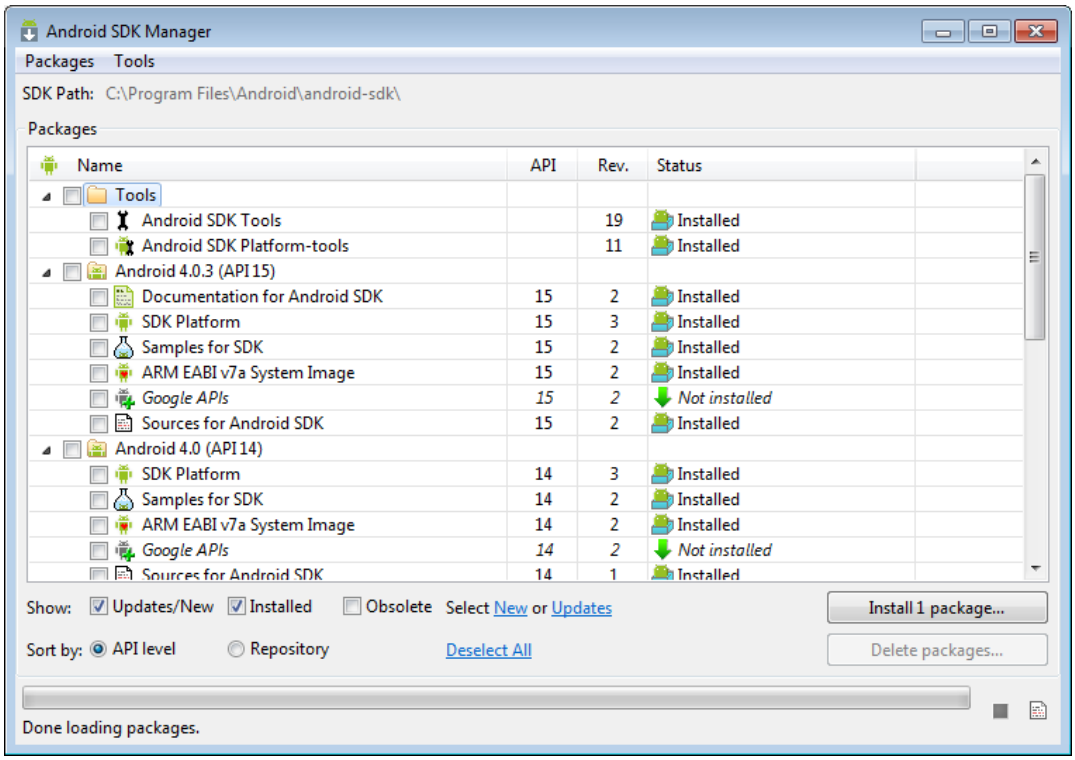


Figura 3.13. Ventana del Android SDK Manager

Capítulo 4

Método de trabajo

En este capítulo se describe el método de trabajo que se ha utilizado para llevar a cabo el desarrollo de la herramienta AndrIU, así como la planificación del proyecto que se ha seguido para su consecución. La sección 4.1 hace mención a la metodología de desarrollo software utilizada, mientras que la sección 4.2 presenta el seguimiento del proyecto, explicando cada una de las iteraciones seguidas.

4. MÉTODO DE TRABAJO

Este capítulo describe el método de trabajo que se ha utilizado para llevar a cabo el desarrollo de la herramienta AndrIU, así como la planificación del proyecto que se ha seguido para su consecución.

4.1. Proceso Unificado de Desarrollo (PUD)

El PUD (Proceso Unificado de Desarrollo) (Jacaboson, 2000) es una metodología de desarrollo software extensible que puede ser adaptada a las necesidades de 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), está basado en componentes software conectados a través de interfaces y usa el lenguaje de modelado UML (*Unified Modelling Language*, Lenguaje Unificado de Modelado) para especificar un conjunto de diagramas y demás artefactos que definen el sistema junto con el código ejecutable.

El principal objetivo del PUD es proporcionar un conjunto de métodos, técnicas y herramientas para facilitar el desarrollo de software de calidad que satisfaga las necesidades de los usuarios.

De este modo el PUD presenta las siguientes características:

- **Dirigido por casos de uso**

  El primer paso a la hora de desarrollar un sistema es identificar las funcionalidades que los usuarios necesitan del sistema. Estas funcionalidades se conocen como requisitos funcionales y de estos requisitos se extraen los denominados casos de uso (CdU) tras realizar un análisis de los requisitos. Cada uno de estos CdU representa un escenario de uso, que define la interacción del usuario con dicha funcionalidad. De este modo se pueden utilizar los CdU para guiar el proceso de desarrollo desde la especificación de requisitos hasta las etapas finales de pruebas.

Al conjunto de todos los CdU que presenta un sistema se le denomina modelo de casos de uso, y se representa a través del diagrama de casos de uso que especifica UML. Es necesario realizar una priorización de estos CdU para planificar el orden en que se van a desarrollar las distintas funcionalidades del sistema, ya que condiciona de manera notable la arquitectura del sistema y la propia evolución del desarrollo.

- **Iterativo e incremental**

  El PUD está compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases está a su vez dividida en una serie de iteraciones, que ofrecen como resultado un *incremento* del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo. Cada iteración se centra en un conjunto de CdU elegidos convenientemente y sigue todos o algunos de los siguientes flujos de trabajo elementales:

  - **Requisitos**: Se recogen todos los requisitos funcionales del sistema.
  - **Análisis**: Se identifican y especifican los CdU.
  - **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**: Los artefactos creados en el diseño se "materializan" a código.
  - **Pruebas**: Se verifica que el sistema cumple los CdU.

- **Centrado en la arquitectura**

  El PUD asume que no existe un modelo único que cubra todos los aspectos del sistema y es por ello por lo que existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema y debe estar relacionada con los CdU para permitir el desarrollo paralelo de los mismos.

- **Enfocado en los riesgos**

  El PUD requiere que el desarrollo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero.

Con todo esto, se puede ver un mapa conceptual del PUD en la Figura 4.1.

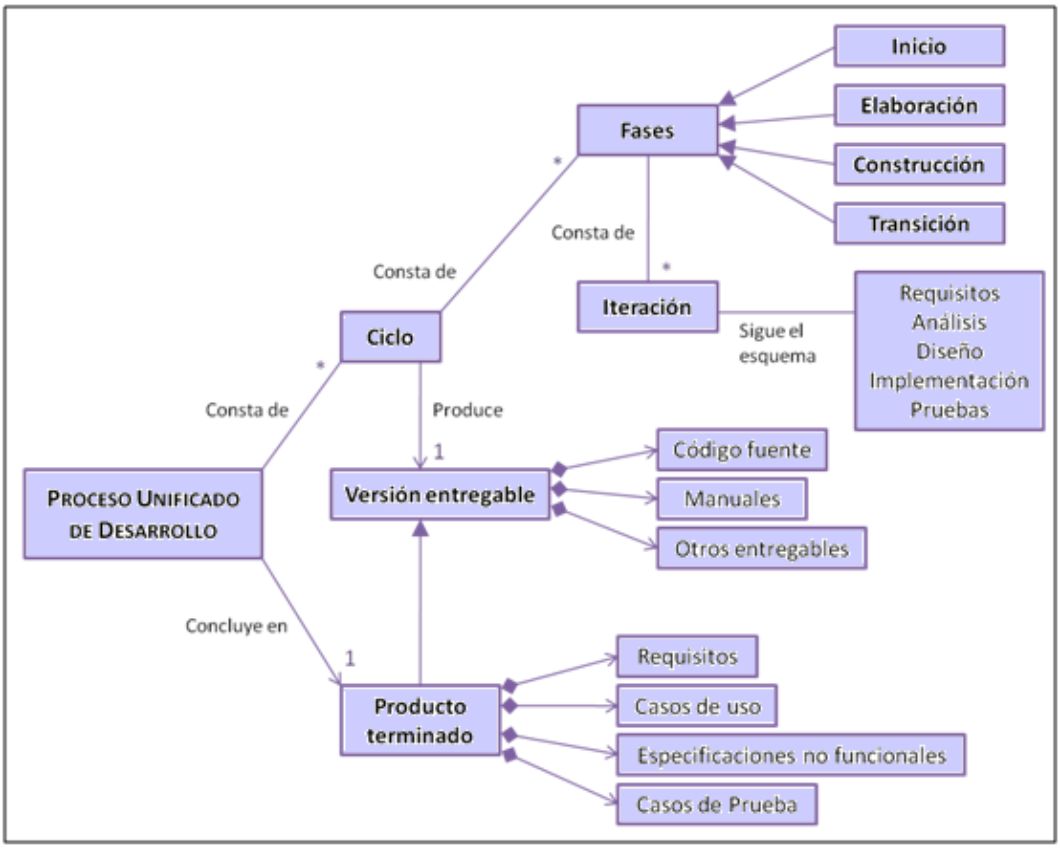


Figura 4.1. Mapa conceptual del PUD

4.1.1. Fases del Proceso Unificado de Desarrollo

El PUD divide el proceso de desarrollo de un proyecto en cuatro fases, como se muestra en la Figura 4.2. Como se ha comentado anteriormente, cada una de estas fases se divide a su vez en iteraciones, contemplando algunos de los flujos de trabajo descritos. Cada una de estas fases termina con un hito, donde se deben cumplir una serie de objetivos. De este modo, las fases del PUD son:

1. **Inicio:** 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.

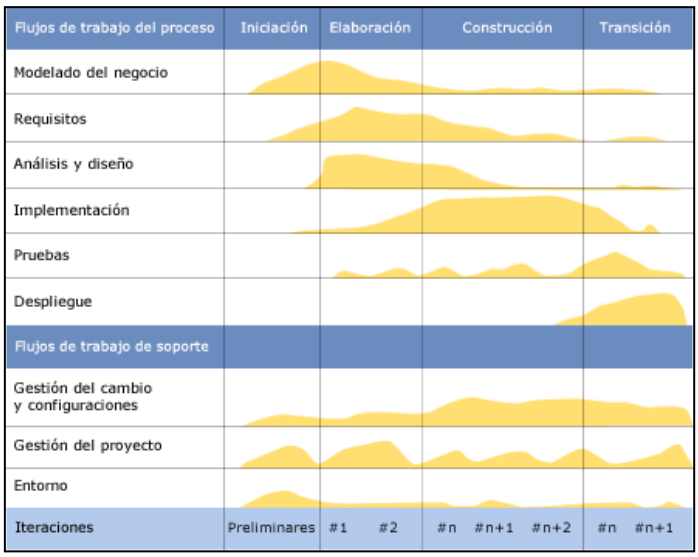


Figura 4.2. Fases del PUD y ejemplo de carga de trabajo por fases y flujos de trabajo

4.1.2. Artefactos

A lo largo del desarrollo software de acuerdo al PUD, se van construyendo diferentes artefactos o modelos software que representan al sistema desde distintos puntos de vista y a diferentes niveles de abstracción, para dar mayor facilidad de comprensión del propio sistema. Estos puntos de vista se definen a través de los siguientes modelos UML:

- **Modelo de Análisis**: define los CdU 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

El desarrollo de la herramienta que se propone en este PFC se divide en una serie de iteraciones. Esta sección describe la planificación de las iteraciones que se han realizado a lo largo de las distintas fases del desarrollo para obtener la herramienta propuesta en este PFC. También se definen los distintos flujos de trabajo que se desarrollan en cada una de las iteraciones así como los productos de salida de cada una de ellas, los cuales se muestran con mayor nivel de detalle en el capítulo 5 de resultados.

4.2.1. Iteración 1

Esta primera iteración es la base para el resto de iteraciones. En ella se define una la lista completa con los requisitos funcionales del sistema y se realiza un análisis de los mismos mediante la identificación preliminar de los casos de uso. Para ello se realiza una primera aproximación del diagrama de casos de uso de la herramienta. La Tabla 4.1. muestra de manera resumida la iteración 1.

Iteración 1	
Fase de PUD:	Inicio
Flujos de trabajo que se realizan:	Requisitos, Análisis
Objetivos	**Productos de Salida**
▪ **OB. 1.1.** Definir los requisitos del sistema. ▪ **OB. 1.2.** Realizar el Análisis de los requisitos mediante la identificación de CdU ▪ **OB. 1.3.** Definir de manera detallada cada uno de los CdU.	▪ **PS. 1.1.** Lista de requisitos del usuario. ▪ **PS. 1.2.** Primera versión del diagrama de casos de uso. ▪ **PS. 1.3.** Especificación de cada uno de los CdU y sus diagramas de flujos de eventos

Tabla 4.1. Resumen de la Iteración 1

Como medida para facilitar la comprensión del sistema, se recomienda ver el Diagrama de Casos de Uso descrito en el capítulo 5 (véase Figura 5.1).

4.2.2. Iteración 2

En esta segunda iteración se realiza la priorización de los CdU obtenidos en la iteración anterior (atendiendo a criterios de complejidad y dependencias entre casos de uso) con el fin de establecer una ordenación a la hora de realizar el análisis, diseño, implementación y pruebas de los CdU. El resultado de la priorización será una lista con los CdU clasificados según su prioridad (véase Tabla 5.16.). Tras esto, se realiza la planificación del PFC, indicando de forma exacta qué CdU se desarrollará en cada iteración. La Tabla 4.2. muestra de manera resumida la iteración 2.

Iteración 2	
Fase de PUD:	Inicio
Flujos de trabajo que se realizan:	Análisis
Objetivos	**Productos de Salida**
▪ **OB.2.1.** Priorizar los CdU ▪ **OB.2.2.** Realizar la planificación del PFC.	▪ **PS.2.1.** Priorización de los CdU. ▪ **PS.2.2.** Planificación del PFC ▪ **PS.2.3.** Esquema de la descripción de la arquitectura.

Tabla 4.2. Resumen Iteración 2

4.2.3. Iteración 3

En esta tercera iteración se comienza la fase de Elaboración siguiendo la planificación. En esta iteración los CdU que se van a abordar son el CdU.4 (Generar Modelos de Transformación) y CdU.5 (Generar Modelos AST).

Esta iteración consistirá en el análisis y diseño de los casos de uso CdU.4 y CdU5, donde se tratará de proporcionar la opción de generar por un lado Modelos de Transformación, donde se relacionarán los elementos de IU del Modelo de Código con elementos de un Modelo PIM, y por otro lado Modelos AST, extraídos directamente del código fuente y conformando el Modelo PSM que define MDA. 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 3, que está alineada con el módulo 1 descrito en el capítulo 2 de objetivos (véase Figura 2.2.).

Iteración 3	
Fase de PUD:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño
Objetivos	**Productos de Salida**
▪ **OB.3.1.** Análisis y Diseño de los CdU: ○ **CdU.4.** Generar Modelos de Transformación ○ **CdU.5.** Generar Modelos AST	▪ **PS.3.1.** Diagramas de secuencia de análisis de los casos de uso CdU4 y CdU5 ▪ **PS.3.2.** Diagramas de clases de análisis de los casos de uso CdU4 y CdU5 ▪ **PS.3.3.** Modelo arquitectónico del proyecto ▪ **PS.3.4.** Diagramas de clases de diseño de los casos de uso CdU4 y CdU5 ▪ **PS.3.5.** Diagramas de secuencia de diseño de los casos de uso CdU4 y CdU5 ▪ **PS.3.6.** Prototipos de las GUIs relativas a los casos de uso CdU4 y CdU5.

Tabla 4.3. Resumen Iteración 3

4.2.4. Iteración 4

En esta iteración se continúa con el análisis y diseño de los siguientes CdU según la priorización realizada (CdU.6 y CdU.7 "Generar Modelos KDM desde fichero Java"). Con estos CdU se pretende proporcionar a la herramienta de la capacidad de generar Modelos de IU siguiendo el estándar KDM, descrito en el Capítulo 1. De este modo, utilizando los modelos que se han generado en la iteración anterior a partir de un fichero de código fuente se generarán como salida modelos KDM.

De manera paralela a este análisis, se comienza la implementación de los CdU diseñados en la iteración anterior, ya que serán necesarios para la implementación de los CdU diseñados en esta iteración. En la Tabla 4.4. se muestra resumida la iteración 4, que abordará el módulo 1 descrito en el capítulo 2 de objetivos (véase Figura 2.2.).

Iteración 4	
Fase de PUD:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación
Objetivos	**Productos de Salida**
▪ **OB.4.1.** Análisis y Diseño de los CdU: ○ **CdU.6.** Generar Modelos KDM ○ **CdU.7.** Generar desde Java File ▪ **OB.4.2.** Implementación de los CdU: ○ **CdU.4.** Generar Modelos de Transformación ○ **CdU.5.** Generar Modelos AST	▪ **PS.4.1.** Diagramas de secuencia de análisis de los casos de uso CdU6 y CdU7 ▪ **PS.4.2.** Diagramas de clases de análisis de los casos de uso CdU6 y CdU7 ▪ **PS.4.3.** Diagrama de clases de CdU6 y CdU7 ▪ **PS.4.4.** Diagrama de secuencia de diseño de los casos de uso CdU6 y CdU7 ▪ **PS.4.5.** Prototipos de las GUIs relativas a los casos de uso CdU6 y CdU7. ▪ **PS.4.6.** Plug-in con la opción de generar Modelos de Transformación y Modelos AST.

Tabla 4.4. Resumen Iteración 4

4.2.5. Iteración 5

La iteración 5 continúa con el análisis y el diseño de los casos de uso CdU.1 (Crear nuevo proyecto AndrIU), CdU.2 (Crear desde cero), CdU.3 (Crear desde proyecto Java existente) y CdU.8 (Generar Modelos KDM desde proyecto AndrIU). De este modo, primero se diseñarán los aspectos relativos a la creación de las estructuras (proyectos para Eclipse) para almacenar los modelos que se generan, y después se continúa con el diseño de los aspectos relativos a la generación de modelos KDM desde estos proyectos.

De manera concurrente, se comenzará la implementación de los casos de uso CdU.6 y CdU.7 diseñados en la anterior iteración, y que servirán de base al CdU.8 que será implementado en la siguiente iteración. En la Tabla 4.5. se muestra de manera resumida la iteración 5, que está alineada con el módulo 1 descrito en el capítulo 2 de objetivos (véase Figura 2.2.).

Iteración 5	
Fase de PUD:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación
Objetivos	**Productos de Salida**
▪ **OB.5.1.** Análisis y Diseño de los CdU: ○ **CdU.1.** Crear nuevo proyecto AndrIU ○ **CdU.2.** Crear desde cero ○ **CdU.3.** Crear desde proyecto Java existente ○ **CdU.8.** Generar modelos KDM desde proyecto AndrIU ▪ **OB.5.2.** Implementación de los CdU: ○ **CdU.6.** Generar Modelos KDM ○ **CdU.7.** Generar desde Java File	▪ **PS.5.1.** Diagramas de secuencia de análisis de los casos de uso CdU1, CdU2, CdU3 y CdU8 ▪ **PS.5.2.** Diagramas de clases de análisis de los casos de uso CdU1, CdU2, CdU3 y CdU8 ▪ **PS.5.3.** Diagrama de clases de los casos de uso CdU1, CdU2, CdU3 y CdU8 ▪ **PS.5.4.** Diagrama de secuencia de diseño de los casos de uso CdU1, CdU2, CdU3 y CdU8 ▪ **PS.5.5.** Prototipos de las GUIs relativas a los casos de uso CdU1, CdU2, CdU3 y CdU8. ▪ **PS.5.6.** Plug-in de Eclipse (AndrIU) con la opción de generar Modelos KDM desde un fichero Java.

Tabla 4.5. Resumen Iteración 5

4.2.6. Iteración 6

En esta sexta iteración se termina de implementar los casos de uso CdU1, CdU2, CdU3 y CdU8, diseñados en la iteración anterior, y que permitirán crear los proyectos para Eclipse que permitirán a la herramienta gestionar los modelos generados. Del mismo modo, se dotará al sistema de la capacidad de generar modelos KDM desde los ficheros de código contenidos en estos proyectos.

Paralelo a esto, se realiza el análisis y el diseño de los casos de uso CdU.9 (Generar Modelos de IU Android), CdU.10 (Generar desde Modelo KDM) y CdU.11 (Generar desde proyecto AndrIU), que permitirán generar los modelos de interfaz de usuario para aplicaciones Android. En la Tabla 4.6. se muestra de manera resumida la iteración 6, que abordará aspectos de los módulos 1 y 2 descritos en el capítulo 2 de objetivos (véase Figura 2.2. y Figura 2.3.).

Iteración 6	
Fase de PUD:	Elaboración
Flujos de trabajo que se realizan:	Análisis, Diseño, Implementación
Objetivos	**Productos de Salida**
▪ **OB.6.1.** Análisis y Diseño de los CdU: ○ **CdU.9.** Generar Modelo IU Android ○ **CdU.10.** Generar desde Modelo KDM ○ **CdU.11.** Generar desde proyecto AndrIU ▪ **OB.6.2.** Implementación de los CdU: ○ **CdU.1.** Crear nuevo proyecto AndrIU ○ **CdU.2.** Crear desde cero ○ **CdU.3.** Crear desde proyecto Java existente ○ **CdU.8.** Generar modelos KDM desde proyecto AndrIU	▪ **PS.6.1.** Diagramas de secuencia de análisis de los casos de uso CdU9, CdU10 y CdU11 ▪ **PS.6.2.** Diagramas de clases de análisis de los casos de uso CdU9, CdU10 y CdU11 ▪ **PS.6.3.** Diagrama de clases de los casos de uso CdU9, CdU10 y CdU11 ▪ **PS.6.4.** Diagrama de secuencia de diseño de los casos de uso CdU9, CdU10 y CdU11 ▪ **PS.6.5.** Prototipos de las GUIs relativas a los casos de uso CdU9, CdU10 y CdU11. ▪ **PS.6.6.** Plug-in de Eclipse (AndrIU) con la opción de crear proyectos AndrIU desde cero o desde un proyecto Java y de generar Modelos KDM desde un proyecto AndrIU.

Tabla 4.6. Resumen Iteración 6

4.2.7. Iteración 7

En esta iteración se realiza la implementación de los casos de uso CdU.9, CdU.10 y CdU.11, que hacen referencia a la generación de los modelos de IU Android (tanto desde un modelo KDM como de un proyecto AndrIU). Adicionalmente, se comienza con la realización de pruebas unitarias de los casos de uso más prioritarios, y que se implementaron en las iteraciones 4 y 5. De este modo, la Tabla 4.7. muestra de manera resumida la iteración 7. Esta iteración está alineada con el módulo 2 descrito en el capítulo 2 (véase Figura 2.3.).

Iteración 7	
Fase de PUD:	Construcción
Flujos de trabajo que se realizan:	Implementación, Pruebas
Objetivos	**Productos de Salida**
▪ **OB.7.1.** Implementación de los CdU: ○ **CdU.9.** Generar Modelo IU Android ○ **CdU.10.** Generar desde Modelo KDM ○ **CdU.11.** Generar desde proyecto AndrIU ▪ **OB.7.2.** Pruebas unitarias de los CdU: ○ **CdU.4.** Generar Modelos de Transformación ○ **CdU.5.** Generar Modelos AST ○ **CdU.6.** Generar Modelos KDM ○ **CdU.7.** Generar desde Java File	▪ **PS.7.1.** Plug-in de Eclipse (AndrIU) con la opción de generar modelos de IU para Android desde un modelo KDM o desde un proyecto AndrIU. ▪ **PS.7.2.** Resultado de las pruebas unitarias

Tabla 4.7. Resumen Iteración 7

4.2.8. Iteración 8

En esta iteración se diseña e implementa el editor gráfico para representar los Modelos generados en KDM. Para ello se sigue la metodología propuesta por EMF/GMF, obteniendo el modelo y el editor, e integrándolos después en la herramienta AndrIU.

De manera concurrente, en la iteración se continúan con las pruebas unitarias relativas a los casos de uso CdU.1, CdU.2, CdU.3 y CdU.8. En la Tabla 4.8. se muestra de manera resumida la iteración 8. Esta iteración aborda el módulo 3 descrito en el capítulo 2 (véase Figura 2.4. Figura 2.3.).

Iteración 8	
Fase de PUD:	Construcción
Flujos de trabajo que se realizan:	Diseño, Implementación, Pruebas
Objetivos	**Productos de Salida**
▪ **OB.8.1.** Diseño e Implementación del CdU.12 ▪ **OB.8.2.** Pruebas unitarias de los CdU: ○ **CdU.1.** Crear nuevo proyecto AndrIU ○ **CdU.2.** Crear desde cero ○ **CdU.3.** Crear desde proyecto Java existente ○ **CdU.8.** Generar modelos KDM desde proyecto AndrIU	▪ **PS.8.1.** Generación de código del modelo EMF (.genmodel) ▪ **PS.8.2.** Especificación de la correspondencia (.gmfmap) ▪ **PS.8.3.** Generación del modelo generador (.gmfgen) ▪ **PS.8.4.** Plug-in con el editor EMF/GMF (AndrIUGraphicalEditor) para visualizar y editar diagramas de IU, que será integrado con el anterior plug-in (AndrIU) ▪ **PS. 8.5.** Resultados de las pruebas unitarias.

Tabla 4.8. Resumen Iteración 8

4.2.9. Iteración 9

En esta iteración se terminarán de implementar los últimos casos de uso, de manera que quede terminada por completo la implementación de la herramienta. Se dota a la herramienta de una página de configuración y administración de la herramienta.

Una vez terminada la implementación completa del plug-in de Eclipse correspondiente a la herramienta AndrIU se procede a terminar de realizar las pruebas unitarias respectivas y a crear las distintas pruebas de integración y aceptación. La Tabla 4.9. muestra de manera resumida la iteración 9.

Iteración 9	
Fase de PUD:	Construcción
Flujos de trabajo que se realizan:	Implementación, Pruebas
Objetivos	**Productos de Salida**
▪ **OB.9.1.** Implementación del CdU.13 ▪ **OB.9.2.** Pruebas unitarias de los CdU: ○ **CdU.9.** Generar Modelo IU Android ○ **CdU.10.** Generar desde Modelo KDM ○ **CdU.11.** Generar desde proyecto AndrIU ○ **CdU.12.** Configurar el entorno AndrIU	▪ **PS.9.1.** Incorporación de la opción en el plug-in AndrIU para "configurar parámetros del entorno AndrIU" ▪ **PS. 9.2.** Resultados de las pruebas unitarias. ▪ **PS. 9.3.** Resultados de las pruebas de integración. ▪ **PS. 9.4.** Resultados de las pruebas de aceptación.

Tabla 4.9. Resumen Iteración 9

4.2.10. Iteración 10

Esta iteración pone fin al proceso de desarrollo de la herramienta propuesta. De este modo se prepara el producto final para su distribución bajo licencia EPL (*Eclipse Public License*). También se confecciona la documentación de la herramienta para desarrolladores y un manual de usuario. En paralelo se llevan a cabo varios casos de estudio reales para facilitar la trasferencia a la industria. La Tabla 4.10. muestra de manera resumida la iteración 10.

Iteración 10	
Fase de PUD:	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 4.10. Resumen Iteración 10

4.2.11. Resumen de las iteraciones

En este apartado se hace un resumen del desarrollo de cada uno de los casos de uso en cada una de las iteraciones que han sido expuestas a lo largo de este capítulo. En la Tabla 4.11. se muestra dicho resumen.

Caso de Uso	Análisis	Diseño	Implementación	Pruebas
CdU.1	it5	it5	it6	it6
CdU.2	it5	it5	it6	it8
CdU.3	it5	it5	it6	it8
CdU.4	it3	it3	it4	it7
CdU.5	it3	it3	it4	it7
CdU.6	it4	it4	it5	it7
CdU.7	it4	it4	it5	it7
CdU.8	it5	it5	it6	it8
CdU.9	it6	it6	it7	it9
CdU.10	it6	it6	it7	it9
CdU.11	it6	it6	it7	it9
CdU.12	it8	it8	it8	it9
CdU.13	it5	it5	it6	it8

Tabla 4.11. Resumen de las Iteraciones

A continuación se presentan dos gráficos (véase Figura 4.3) donde se muestran la distribución de esfuerzos según los flujos de trabajo en cada una de las iteraciones del desarrollo. De este modo se puede observar de un solo vistazo la cantidad de análisis, diseño, implementación y pruebas que se ha realizado en cada una de las iteraciones.

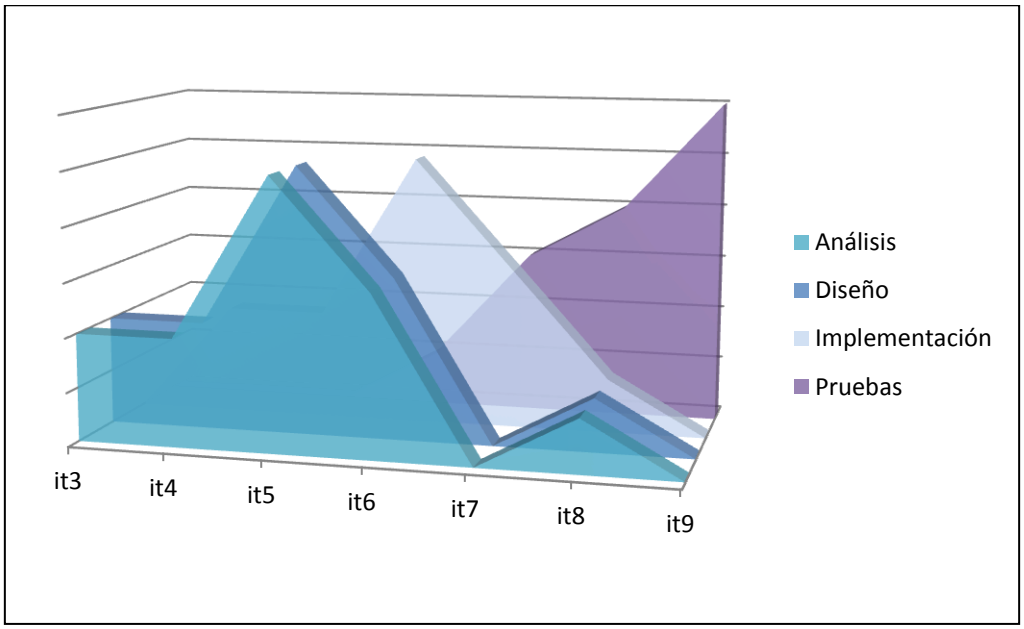


Figura 4.3. Gráfico con la carga de flujos de trabajo en cada iteración (II)

Capítulo 5

Resultados

En este capítulo se describirán de manera detallada los resultados obtenidos en cada una de las iteraciones al realizar el desarrollo del presente PFC.

5. RESULTADOS

Este capítulo describe de manera detallada los entregables y resultados obtenidos en cada una de las iteraciones que han llevado a cabo para la consecución del presente PFC, de acuerdo al método propuesto en el capítulo anterior.

5.1. Iteración 1

En la iteración 1 se han generado los productos de salida que se detallan a continuación (véase el resumen en la Tabla 4.1.), y se presentan agrupados según el flujo de trabajo al que pertenecen (requisitos y análisis).

5.1.1. Especificación de Requisitos

De acuerdo a la metodología PUD, una de las primeras tareas consiste en especificar los requisitos que presenta el producto software. Es una tarea importante a la hora de lleva 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 en dos listas numeradas, que corresponden al producto de salida PS.1.1 (véase Tabla 4.1.).

5.1.1.1. Requisitos Funcionales

Los requisitos funcionales definen el comportamiento interno del sistema, y son los que guiarán el desarrollo del proyecto. A continuación se muestra en la Tabla 5.1. los requisitos funcionales, incluyendo un identificador y una breve descripción.

Identificador	Descripción
RF.01	Permitir la generación de modelos de IU representados en KDM siguiendo el estándar ISO/IEC 19506.
RF.02	Permitir la generación de modelos de transformación para la definición y correspondencia de los elementos de IU que se van a tratar.

Identificador	Descripción
RF.03	Permitir el análisis de código fuente de un sistema de información existente a fin de reconocer ciertos elementos de IU para ser representados en un modelo específico de plataforma (PSM).
RF.04	Permitir la creación de una nueva estructura (proyecto) que almacene y organice los modelos que se generarán.
RF.05	Permitir la creación de una estructura que almacene y organice los modelos que se generarán a partir de un sistema de información heredado.
RF.06	Permitir que la información generada sea persistente, es decir, que se aloje físicamente para su posterior recuperación.
RF.07	Permitir la representación gráfica de los modelos de IU generados a fin de que estos puedan ser utilizados para facilitar las tareas en fases posteriores de reingeniería y migración.
RF.08	Permitir la generación de GUI para Android a partir de los modelos de IU generados, y su inclusión en proyectos Android.
RF.09	Permitir la edición de los modelos generados a fin de realizar correcciones y/o modificaciones.
RF.10	Permitir la configuración de ciertos parámetros para realizar las distintas tareas que permita la herramienta. Los cambios en estos parámetros serán persistentes, de manera que permanezcan en posteriores sesiones.

Tabla 5.1. Requisitos Funcionales

5.1.1.2. Requisitos No Funcionales

Los requisitos no funcionales presentes en el Proyecto de Fin de Carrera se definen a continuación en la Tabla 5.2. .

Identificador	Descripción
Tecnologías empleadas	
RnF.1	La herramienta será implementada como un plug-in para el IDE Eclipse.
RnF.2	Se generará un parser del lenguaje 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.3	Se empleará EMF-GMF para el desarrollo de los editores gráficos.
RnF.4	Se utilizará la librería JDom para el manejo de ficheros XML donde se almacenará la información de los modelos generados.

Identificador	Descripción
RnF.5	Se utilizará el framework "Android Development Tool" (ADT) de Eclipse para la generación de proyectos Android donde se almacenarán los ficheros de GUI generados. Además permitirá utilizar sus editores gráficos para manipular estos ficheros generados.
Estándares aplicados	
RnF.6	Estándar KDM versión 1.1
RnF.7	Estándar para la representación de GUI Android 1.6
Requisitos de Desarrollo	
RnF.8	Plataforma de desarrollo: "Eclipse Indigo"
RnF.9	Lenguaje de programación: "Java 1.6"

Tabla 5.2. Requisitos no Funcionales

5.1.2. Análisis

Una vez identificados los requisitos (tanto funcionales como no funcionales) de AndrIU, se realiza un análisis de los mismos. De este análisis se obtendrán los casos de uso (de ahora en adelante CdU), que guiarán el desarrollo del presente proyecto, como especifica PUD. Con los CdU extraídos del análisis, se obtiene una primera versión del diagrama de casos de uso, y la descripción detallada de todos sus elementos.

5.1.2.1. Diagrama de Casos de Uso

Los CdU identificados representa la vista funcional del sistema, mostrando las relaciones existentes entre los actores y cada uno de los casos de uso que representan las funcionalidades del sistema a desarrollar. Es por ello que los CdU tienen la finalidad de especificar el comportamiento y la comunicación del sistema a través de su interacción con los usuarios y otros sistemas externos. En la Figura 5.1 se muestra el diagrama de casos de uso que se ha extraído del análisis de los requisitos especificados anteriormente, y en los subapartados posteriores se describen de manera detallada cada uno de estos casos de uso. El presente diagrama de casos de uso corresponde al producto de salida PS.1.2 y las tablas donde se describe cada uno de los CdU corresponden al producto de salida PS.1.3 (véase la Tabla 4.1.).

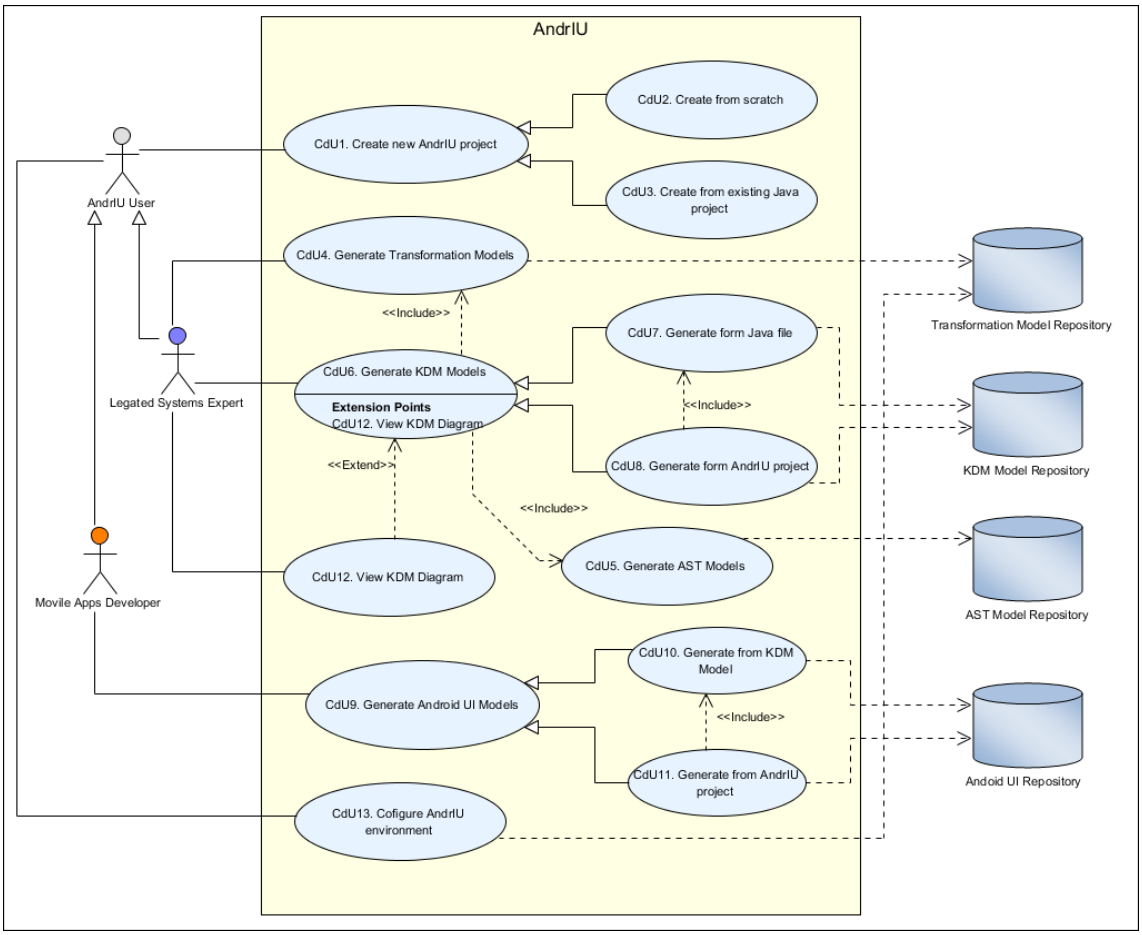


Figura 5.1. Diagrama de Casos de Uso

5.1.2.2. Roles

En el diagrama de casos de uso se pueden apreciar dos roles diferentes, que heredan de un tercero más general. Estos roles pueden observarse en el margen izquierdo de la Figura 5.1, y se describen a continuación.

- **AndrIU User (Usuario AndrIU)**

  Es el usuario que accederá a la herramienta. De manera general, este usuario tiene conocimientos en el desarrollo de aplicaciones software, y de manara más concreta tiene conocimientos en aspectos de reingeniería y sistemas heredados.

- **Legacy Systems Expert (Experto en Sistemas Heredados)**

  Es un usuario AndrIU que además es experto en LIS. Este usuario será conocedor del sistema heredado de entrada y del estándar KDM, con el que se representarán los modelos que dicho usuario generará a través del sistema.

- **Movile Applications Developer (Desarrollador de Aplicaciones Móviles)**
  Es un usuario AndrIU que además es experto el desarrollo de aplicaciones móviles, más concretamente aplicaciones Android. Este usuario deberá tener conocimientos sobre ingeniería inversa y sobre el estándar KDM.

5.1.2.3. Descripción de los Casos de Uso

En la siguiente sección se realizará una descripción detallada de cada uno de los CdU, incluyendo los requisitos funcionales que aborda, los actores que involucra y una lista de condiciones a priori ya posteriori. En las tablas Tabla 5.3. a la Tabla 5.15. se pueden observar detalladamente la descripción de los CdU, y se corresponden con el producto de salida PS.1.3 (véase la Tabla 4.1.).

CdU 1. Crear Nuevo Proyecto	
Descripción	Se encarga de crear un nuevo tipo de proyecto para Eclipse asociado a la herramienta AndrIU, con una estructura específica. La estructura de estos proyectos constará de cuatro carpetas o directorios para almacenar los distintos modelos (Transformación, AST y KDM), además de los ficheros de código fuente (Java).
Requisitos funcionales	RF.4
Actores involucrados	"Experto en Sistemas Heredados" y "Experto en Dispositivos Móviles".
Precondiciones	No existen precondiciones.
Postcondiciones	El nuevo proyecto creado se almacenará automáticamente en el Workspace activo de Eclipse del equipo.

Tabla 5.3. Detalles del CdU.1

CdU 2. Crear desde Cero	
Descripción	Se encarga de crear un nuevo tipo de proyecto para Eclipse desde cero (a través de un "Wizard") asociado a la herramienta AndrIU, con una estructura específica. La estructura de estos proyectos constará de cuatro carpetas o directorios (ver descripción de CdU1). Inicialmente estas carpetas estarán vacías, salvo la de "Transformation Models", que incluirá el modelo de transformación configurado por defecto.
Requisitos funcionales	RF.5, RF.6
Actores involucrados	"Experto en Sistemas Heredados" y "Experto en Dispositivos Móviles".
Precondiciones	No existen precondiciones.
Postcondiciones	El nuevo proyecto creado se almacenará automáticamente en el Workspace activo de Eclipse del equipo.

Tabla 5.4. Detalles del CdU.2

CdU 3. Crear desde proyecto existente	
Descripción	Se encarga de crear un nuevo tipo de proyecto a partir de un proyecto Java de Eclipse, con una estructura específica. La estructura de estos proyectos constará de cuatro carpetas o directorios (ver descripción de CdU1), y la carpeta "source" contendrá los ficheros de código fuente del proyecto de origen. El resto de carpetas estarán vacías.
Requisitos funcionales	RF.5, RF.6
Actores involucrados	"Experto en Sistemas Heredados" y "Experto en Dispositivos Móviles".
Precondiciones	Debe existir al menos un proyecto Java en el Workspace para que lo tome como fuente para generar el nuevo proyecto.
Postcondiciones	El nuevo proyecto creado se almacenará automáticamente en el Workspace activo de Eclipse del equipo.

Tabla 5.5. Detalles del CdU.3

CdU 4. Generar Modelo de Transformación	
Descripción	Se encarga de generar un nuevo modelo de transformación. Para ello se deberán seleccionar de una lista los elementos de IU que se desean incluir en el nuevo modelo, que se almacenará en un fichero XML.
Requisitos funcionales	RF.2
Actores involucrados	"Experto en Sistemas Heredados".
Precondiciones	No existen precondiciones.
Postcondiciones	El nuevo modelo creado se almacenará automáticamente en el equipo.

Tabla 5.6. Detalles del CdU.4

CdU 5. Generar Modelos AST	
Descripción	Se encarga de generar un nuevo modelo de transformación. Para ello se deberán seleccionar de una lista los elementos de IU que se desean incluir en el nuevo modelo, que se almacenará en un fichero XML.
R Requisitos funcionales	RF.3
Actores involucrados	Este CdU depende de otros, pero no lo utiliza ningún actor directamente. Se ha separado por complejidad del CdU.6
Precondiciones	Debe existir un fichero de código fuente al menos para poder generar su correspondiente modelo AST.
Postcondiciones	El nuevo modelo o modelos creados se almacenarán automáticamente en la carpeta "AST models" del proyecto de Eclipse.

Tabla 5.7. Detalles del CdU.5

CdU 6. Generar Modelos KDM	
Descripción	Se encarga de generar los modelos KDM. Para ello, toma como entrada uno o más ficheros de código fuente y un modelo de transformación, y genera los modelos KDM, que se almacenarán en el proyecto.
Requisitos funcionales	RF.01
Actores involucrados	"Experto en Sistemas Heredados".
Precondiciones	Debe existir al menos un fichero de código fuente Java y un modelo de transformación.
Postcondiciones	El nuevo modelo o modelos creados se almacenarán automáticamente en la carpeta "KDM models" del proyecto de Eclipse.

Tabla 5.8. Detalles del CdU.6

CdU 7. Generar desde fichero Java	
Descripción	Se encarga de generar el modelo KDM para un fichero de código fuente Java. Para ello toma como entrada el propio fichero Java y un modelo de transformación, y genera el modelos KDM, que se almacenará en el proyecto.
Requisitos funcionales	RF.01, RF.06
Actores involucrados	"Experto en Sistemas Heredados".
Precondiciones	Debe existir el fichero de código fuente Java y un modelo de transformación.
Postcondiciones	El nuevo modelo creado se almacenará automáticamente en la carpeta "KDM models" del proyecto de Eclipse.

Tabla 5.9. Detalles del CdU.7

CdU 8. Generar desde proyecto	
Descripción	Se encarga de generar los modelos KDM para cada uno de los ficheros de código fuente que se encuentren en la carpeta "source" del proyecto AndrIU. Para ello utilizará además un fichero de transformación
Requisitos funcionales	RF.01, RF.0 6
Actores involucrados	"Experto en Sistemas Heredados".
Precondiciones	Debe existir un proyecto AndrIU con al menos un fichero Java en la carpeta "source", y un modelo de transformación.
Postcondiciones	El nuevo modelo creado se almacenará automáticamente en la carpeta "KDM models" del proyecto de Eclipse.

Tabla 5.10. Detalles del CdU.8

CdU 9. Generar modelos de IU Android	
Descripción	Se encarga de generar modelos de IU para los sistemas Android. Para ello tomará un modelo KDM y un proyecto Android de Eclipse (con ADT), y generará los correspondientes ficheros en el proyecto Android, además de en el propio proyecto AndrIU.
Requisitos funcionales	RF.08
Actores involucrados	"Experto en Dispositivos Móviles".
Precondiciones	Debe existir un proyecto AndrIU con al menos un modelo KDM en la carpeta "KDM models", y un proyecto Android de ADT en el workspace de Eclipse.
Postcondiciones	Los modelos generados se almacenarán en la carpeta "Android" del proyecto AndrIU y en el directorio donde se alojan los modelos de IU para el proyecto Android.

Tabla 5.11. Detalles del CdU.9

CdU 10. Generar desde modelo KDM	
Descripción	Se encarga de generar modelos de IU para sistemas Android a partir de un modelo KDM. Para ello tomará el modelo KDM y generará los correspondientes modelos en la carpeta "Android" del proyecto AndrIU.
Requisitos funcionales	RF.06, RF.08
Actores involucrados	"Experto en Dispositivos Móviles".
Precondiciones	Debe existir un proyecto AndrIU con al menos un modelo KDM en la carpeta "kdm models".
Postcondiciones	Los modelos generados se almacenarán en la carpeta "android" del proyecto AndrIU.

Tabla 5.12. Detalles del CdU.10

CdU 11. Generar desde proyecto	
Descripción	Se encarga de generar modelos de IU para sistemas Android a partir de un proyecto AndrIU. Para ello tomará los modelos KDM de la carpeta "kdm models" del proyecto y un proyecto Android del workspace, y generará los correspondientes modelos en la carpeta "Android" del proyecto AndrIU y en el directorio donde se alojan los modelos de IU para el proyecto Android.
Requisitos funcionales	RF.06, RF.08
Actores involucrados	"Experto en Dispositivos Móviles".
Precondiciones	Debe existir un proyecto AndrIU con al menos un modelo KDM en la carpeta "kdm models", y un proyecto Android de ADT en el workspace de Eclipse.
Postcondiciones	Los modelos generados se almacenarán en la carpeta "android" del proyecto AndrIU y en el directorio donde se alojan los modelos de IU para el proyecto Android.

Tabla 5.13. Detalles del CdU.11

CdU 12. Ver diagramas en Editor	
Descripción	Se encarga de mostrar los modelos creados con editores, tanto en forma de árbol como en forma de diagramas.
Requisitos funcionales	RF.07, RF.09
Actores involucrados	"Experto en Sistemas Heredados".
Precondiciones	Debe existir algún modelo KDM.
Postcondiciones	No existen postcondiciones.

Tabla 5.14. Detalles del CdU.12

CdU13. Configurar el entorno AndrIU	
Descripción	Se encarga de almacenar los parámetros de configuración de la herramienta. Estos parámetros se configurarán desde una ventana de preferencias de eclipse asociada a la herramienta AndrIU, y quedarán guardados para las posteriores sesiones.
Requisitos funcionales	RF.10
Actores involucrados	"Experto en Sistemas Heredados" y "Experto en Dispositivos Móviles".
Precondiciones	No existen precondiciones.
Postcondiciones	Los cambios en los parámetros de configuración se guardarán para la siguiente sesión.

Tabla 5.15. Detalles del CdU.13

5.2. Iteración 2

En esta segunda iteración se describe la planificación del PFC. Para ello primero se realiza una priorización de los casos de uso descritos en el apartado anterior. A continuación se realiza la planificación en sí, ayudados de *Microsoft Project*, y finalmente se describe la arquitectura que tendrá AndrIU.

5.2.1. Priorización de los Casos de Uso

La priorización de los casos de uso viene dada por la complejidad y la dependencia con otros casos de uso, ya que el desarrollo de algún CdU puede requerir el desarrollo de otro CdU previamente. La complejidad se establece en un rango de 1 y 5 (indicando el 5 mayor prioridad) y se puede observar en la Tabla 5.16. De este modo dicha tabla muestra los CdU con su correspondiente valor de prioridad, así como la iteración en la que comienza su desarrollo. Esta priorización se ha realizado teniendo en cuenta que se deben desarrollar antes los CdU que pueden condicionar el desarrollo de

alguna de las partes o del todo de AndrIU. Dicha priorización de los CdU es el producto de salida PS.2.1.

Caso de Uso	Prioridad	Iteración
CdU1	****	5
CdU2	****	5
CdU3	***	5
CdU4	*****	3
CdU5	*****	3
CdU6	****	4
CdU7	****	4
CdU8	***	5
CdU9	**	6
CdU10	**	6
CdU11	**	6
CdU12	*	8
CdU13	*	9

Tabla 5.16. Priorización de los casos de uso

5.2.2. Planificación

De acuerdo a la priorización de los CdU se realiza la planificación de la ejecución del PFC. Para llevar a cabo esta planificación se ha utilizado la herramienta *Microsoft Project 2003*. En las siguientes figuras (Figura 5.1 – Figura 5.10) puede observarse el diagrama de Gantt por partes, de manera que en cada una se describe la planificación de cada iteración. Estos diagramas representan una estimación aproximada del esfuerzo necesario para llevar a cabo cada una de las iteraciones y se corresponden con el producto de salida PS.2.2 de la segunda iteración.

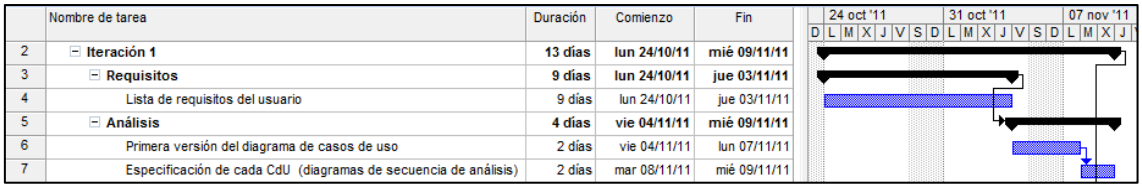


Figura 5.2. Planificación de la Iteración 1

	Nombre de tarea	Duración	Comienzo	Fin
8	**Iteración 2**	**10 días**	**jue 10/11/11**	**mié 23/11/11**
9	**Análisis**	**10 días**	**jue 10/11/11**	**mié 23/11/11**
10	Priorización de los CdU	2 días	jue 10/11/11	vie 11/11/11
11	Esquema de la descripción de la arquitectura	4 días	lun 14/11/11	jue 17/11/11
12	Planificación (???)	4 días	vie 18/11/11	mié 23/11/11

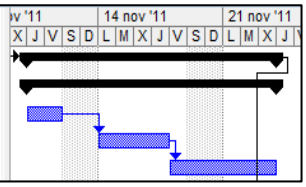


Figura 5.3. Planificación de la Iteración 2

	Nombre de tarea	Duración	Comienzo	Fin
13	**Iteración 3**	**12 días**	**jue 24/11/11**	**vie 09/12/11**
14	**Análisis**	**3 días**	**jue 24/11/11**	**lun 28/11/11**
15	Diagramas de secuencia de análisis de los casos de uso CdU4 y C	1 día	jue 24/11/11	jue 24/11/11
16	Diagramas de clases de análisis de los casos de uso CdU4 y CdU5	2 días	vie 25/11/11	lun 28/11/11
17	**Diseño**	**9 días**	**mar 29/11/11**	**vie 09/12/11**
18	Modelo arquitectónico del proyecto	2 días	mar 29/11/11	mié 30/11/11
19	Diagramas de clases de diseño de los casos de uso CdU4 y CdU5	3 días	jue 01/12/11	lun 05/12/11
20	Diagramas de secuencia de diseño de los casos de uso CdU4 y Cd	3 días	mar 06/12/11	jue 08/12/11
21	Prototipos de las GUIs relativas a los casos de uso CdU4 y CdU5	1 día	vie 09/12/11	vie 09/12/11

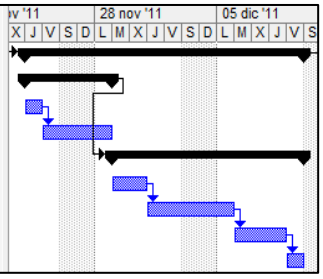


Figura 5.4. Planificación de la Iteración 3

	Nombre de tarea	Duración	Comienzo	Fin
22	**Iteración 4**	**15 días**	**lun 12/12/11**	**vie 30/12/11**
23	**Análisis**	**3 días**	**lun 12/12/11**	**mié 14/12/11**
24	Diagramas de secuencia de análisis de los casos de uso CdU6 y C	1 día	lun 12/12/11	lun 12/12/11
25	Diagramas de clases de análisis de los casos de uso CdU6 y CdU7	2 días	mar 13/12/11	mié 14/12/11
26	**Diseño**	**7 días**	**jue 15/12/11**	**vie 23/12/11**
27	Diagrama de clases de los casos de uso CdU6 y CdU7	3 días	jue 15/12/11	lun 19/12/11
28	Diagrama de secuencia de diseño de los casos de uso CdU6 y CdU	3 días	mar 20/12/11	jue 22/12/11
29	Prototipos de las GUIs relativas a los casos de uso CdU6 y CdU7	1 día	vie 23/12/11	vie 23/12/11
30	**Implementación**	**15 días**	**lun 12/12/11**	**vie 30/12/11**
31	Generar Modelos de Transformación	3 días	lun 12/12/11	mié 14/12/11
32	Generar Modelos AST desde Java file	12 días	jue 15/12/11	vie 30/12/11

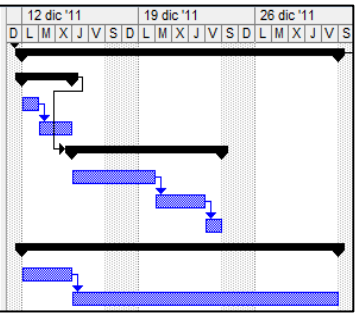


Figura 5.5. Planificación de la Iteración 4

	Nombre de tarea	Duración	Comienzo	Fin
33	**Iteración 5**	**14 días**	**lun 02/01/12**	**jue 19/01/12**
34	**Análisis**	**3 días**	**lun 02/01/12**	**mié 04/01/12**
35	Diagramas de secuencia de análisis de los casos de uso CdU1, Cd	1 día	lun 02/01/12	lun 02/01/12
36	Diagramas de clases de análisis de los casos de uso CdU1, CdU2,	2 días	mar 03/01/12	mié 04/01/12
37	**Diseño**	**5 días**	**jue 05/01/12**	**mié 11/01/12**
38	Diagrama de clases de los casos de uso CdU1, CdU2, CdU3 y CdU	2 días	jue 05/01/12	vie 06/01/12
39	Diagrama de secuencia de diseño de los casos de uso CdU1, CdU2	2 días	lun 09/01/12	mar 10/01/12
40	Prototipos de las GUIs relativas a los casos de uso CdU1, CdU2, C	1 día	mié 11/01/12	mié 11/01/12
41	**Implementación**	**14 días**	**lun 02/01/12**	**jue 19/01/12**
42	Generar Modelos KDM desde Java file	14 días	lun 02/01/12	jue 19/01/12

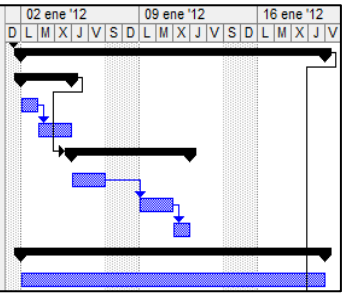


Figura 5.6. Planificación de la Iteración 5

	Nombre de tarea	Duración	Comienzo	Fin
43	**Iteración 6**	**20 días**	**vie 20/01/12**	**jue 16/02/12**
44	**Análisis**	**2 días**	**vie 20/01/12**	**lun 23/01/12**
45	Diagramas de secuencia de análisis de los casos de uso CdU9, Cd	1 día	vie 20/01/12	vie 20/01/12
46	Diagramas de clases de análisis de los casos de uso CdU9, CdU10	1 día	lun 23/01/12	lun 23/01/12
47	**Diseño**	**5 días**	**mar 24/01/12**	**lun 30/01/12**
48	Diagrama de clases de los casos de uso CdU9, CdU10 y CdU11	2 días	mar 24/01/12	mié 25/01/12
49	Diagrama de secuencia de diseño de los casos de uso CdU9, CdU	2 días	jue 26/01/12	vie 27/01/12
50	Prototipos de las GUIs relativas a los casos de uso CdU9, CdU10 y	1 día	lun 30/01/12	lun 30/01/12
51	**Implementación**	**20 días**	**vie 20/01/12**	**jue 16/02/12**
52	Crear nuevo proyecto AndrIU desde cero	4 días	vie 20/01/12	mié 25/01/12
53	Crear nuevo proyecto AndrIU desde proyecto Java existente	3 días	jue 26/01/12	lun 30/01/12
54	Generar modelos KDM desde proyecto AndrIU	13 días	mar 31/01/12	jue 16/02/12

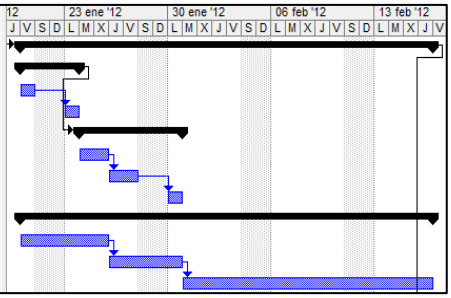


Figura 5.7. Planificación de la Iteración 6

	Nombre de tarea	Duración	Comienzo	Fin
55	**Iteración 7**	**19 días**	**vie 17/02/12**	**mié 14/03/12**
56	**Implementación**	**19 días**	**vie 17/02/12**	**mié 14/03/12**
57	Generar Modelo IU Android desde Modelo KDM	14 días	vie 17/02/12	mié 07/03/12
58	Generar Modelo IU Android desde proyecto AndrIU	5 días	jue 08/03/12	mié 14/03/12
59	**Pruebas**	**8 días**	**vie 17/02/12**	**mar 28/02/12**
60	Pruebas unitarias de los casos de uso CdU4, CdU5, CdU6 y CdU7	8 días	vie 17/02/12	mar 28/02/12

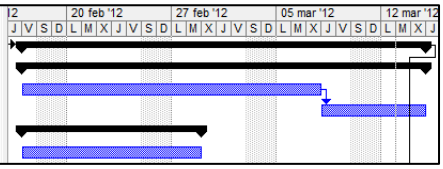


Figura 5.8. Planificación de la Iteración 7

	Nombre de tarea	Duración	Comienzo	Fin
61	**Iteración 8**	**9 días**	**jue 15/03/12**	**mar 27/03/12**
62	**Diseño**	**1 día**	**jue 15/03/12**	**jue 15/03/12**
63	Diseño del editor gráfico AndrIUGraphicalEditor	1 día	jue 15/03/12	jue 15/03/12
64	**Implementación**	**3 días**	**vie 16/03/12**	**mar 20/03/12**
65	**Editor gráfico AndrIUGraphicalEditor**	**3 días**	**vie 16/03/12**	**mar 20/03/12**
66	Generación de código del modelo EMF (.genmodel)	1 día	vie 16/03/12	vie 16/03/12
67	Especificación de la correspondencia (.gmfmap)	1 día	lun 19/03/12	lun 19/03/12
68	Generación del modelo generador (.gmfgen)	1 día	mar 20/03/12	mar 20/03/12
69	**Pruebas**	**9 días**	**jue 15/03/12**	**mar 27/03/12**
70	Pruebas unitarias de los casos de uso CdU1, CdU2, CdU3 y CdU8	9 días	jue 15/03/12	mar 27/03/12

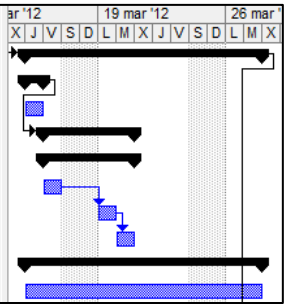


Figura 5.9. Planificación de la Iteración 8

	Nombre de tarea	Duración	Comienzo	Fin
71	**Iteración 9**	**14 días**	**mié 28/03/12**	**lun 16/04/12**
72	**Implementación**	**3 días**	**mié 28/03/12**	**vie 30/03/12**
73	Configurar parámetros del entorno AndrIU	3 días	mié 28/03/12	vie 30/03/12
74	**Pruebas**	**14 días**	**mié 28/03/12**	**lun 16/04/12**
75	Pruebas unitarias de los casos de uso CdU9, CdU10, CdU11 y CdU'	8 días	mié 28/03/12	vie 06/04/12
76	Pruebas de integración	3 días	lun 09/04/12	mié 11/04/12
77	Pruebas de aceptación	3 días	jue 12/04/12	lun 16/04/12

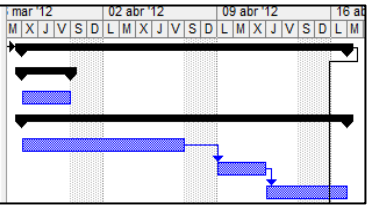


Figura 5.10. Planificación de la Iteración 9

	Nombre de tarea	Duración	Comienzo	Fin
78	**Iteración 10**	**27 días**	**mar 17/04/12**	**mié 23/05/12**
79	**Pruebas**	**2 días**	**mar 17/04/12**	**mié 18/04/12**
80	Últimas pruebas de la herramienta	2 días	mar 17/04/12	mié 18/04/12
81	**Documentación**	**25 días**	**jue 19/04/12**	**mié 23/05/12**
82	Memoria del Proyecto Fin de Carrera	20 días	jue 19/04/12	mié 16/05/12
83	Manual de usuario de la herramienta	4 días	jue 17/05/12	mar 22/05/12
84	Informe sobre la distribución de la herramienta	1 día	mié 23/05/12	mié 23/05/12
85	**Entregable final**	**1 día**	**jue 19/04/12**	**jue 19/04/12**
86	Plug-in AndrIU	1 día	jue 19/04/12	jue 19/04/12

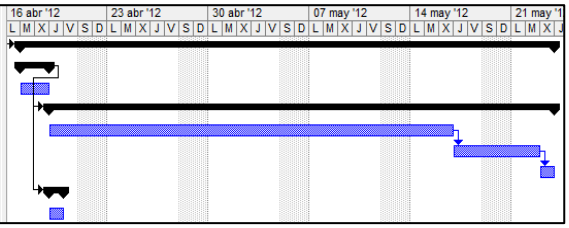


Figura 5.11. Planificación de la Iteración 10

5.2.2.1. Evolución del proyecto

La Tabla 5.17. resume la evolución del proyecto, mostrando las iteraciones realizadas, así como los diferentes flujos de trabajo que son desarrollados en cada una de ellas. Estos productos de salida son explicados detalladamente en el capítulo 5. Esta evolución es definida tomando los productos de salida de estas dos primeras iteraciones (PS.1.1.”Requisitos Funcionales”, PS.1.2.”Diagrama de Casos de Uso”, PS1.3.”Descripción de los casos de Uso”, PS2.1.”Priorización de los Casos de Uso” y PS.2.2.”Planificación del PFC”).

	Iteración	Casos de Uso	Análisis	Diseño	Implem.	Pruebas
INICIO	1	Diagrama de CdU	✓			
	2	Priorización de los CdU	✓			
ELABORACIÓN	3	CdU4	✓	✓		
		CdU5	✓	✓		
	4	CdU4			✓	
		CdU5			✓	
		CdU6	✓	✓		
		CdU7	✓	✓		
	5	CdU1	✓	✓		
		CdU2	✓	✓		
		CdU3	✓	✓		
		CdU6			✓	
		CdU7			✓	
		CdU8	✓	✓		
		CdU13	✓	✓		
	6	CdU1			✓	
		CdU2			✓	
		CdU3			✓	
		CdU8			✓	
		CdU9	✓	✓		
		CdU10	✓	✓		
		CdU11	✓	✓		
CONSTRUCCIÓN	7	CdU4				✓
		CdU5				✓
		CdU6				✓
		CdU7				✓
		CdU9			✓	
		CdU10			✓	
		CdU11			✓	
	8	CdU1				✓
		CdU2				✓
		CdU3				✓
		CdU8				✓
		CdU12	✓	✓	✓	
	9	CdU9				✓
		CdU10				✓
		CdU11				✓
		CdU12				✓
		CdU13	✓	✓	✓	✓
TRANSICIÓN	10	Documentación				

Tabla 5.17. Resumen de la evolución del PFC

5.2.3. Arquitectura

La arquitectura de AndrIU que se desarrolla en el proyecto está orientada a componentes ya que es un herramienta que se compone de una serie de plug-ins para Eclipse. Una de las ventajas de este tipo de arquitectura es que resulta sencillo crear en un futuro nuevos plug-ins que se integren con AndrIU mediante interfaces bien definidas con el fin de añadir nuevas funcionalidades.

El conjunto de plug-ins consisten en la propia herramienta AndrIU, donde se encuentran las funcionalidades principales, y los editores. Entre los editores se encuentran el editor gráfico KDM creado para la representación gráfica (en forma de diagramas) de los modelos generados según el estándar KDM y el editor en forma de árbol para la visualización y modificación de los modelos KDM.

Las librerías en las que se apoya la herramienta son: la librería *JDom* para la creación, edición y modificación de archivos XML, las librerías de *KDM* y las librerías pertenecientes al parser de Java generado. Todo lo anterior puede verse reflejado en la Figura 5.12.

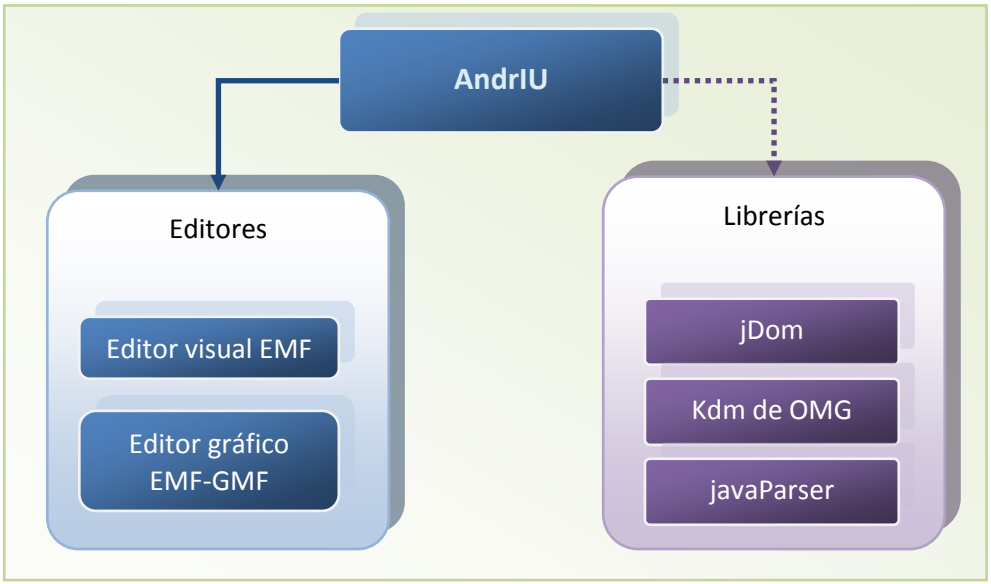


Figura 5.12. Estructura de los componentes de la herramienta

5.3. Iteración 3

En esta tercera iteración se realiza el análisis y diseño del parser que generará los modelos AST de código fuente correspondiente al CdU.5. De manera paralela se aborda el análisis y diseño relativo a la generación de los modelos de transformación necesarios en etapas posteriores, y que se corresponde con el CdU.4.

5.3.1. Análisis

En este flujo de trabajo se obtendrán los productos de salida siguientes, los cuales se explican en detalle en las siguiente subsecciones:

PS.3.1. Diagramas de secuencia de análisis de los casos de uso CdU.4 y CdU.5

PS.3.2. Diagramas de clases de análisis de los casos de uso CdU.4 y CdU.5

5.3.1.1. Diagramas de Secuencia de Análisis

Los diagramas de secuencia de análisis muestra desde un punto de vista de muy alto nivel la interacción entre el usuario y el sistema en cada uno de los casos de uso. Se utilizan para comprender el sistema y se corresponden con los flujos de eventos de los CdU. Estos diagramas corresponden con el producto de salida PS.3.1.

5.3.1.1.1. CdU.4. Generate Transformation Models

a) Escenario principal

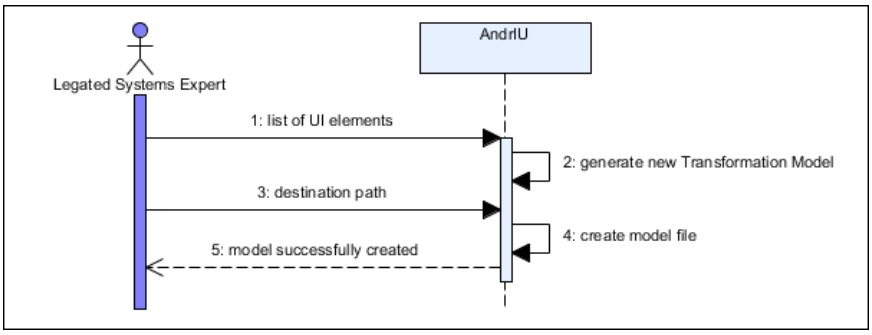


Figura 5.13. Diagrama de Secuencia de Análisis CdU.4 (Escenario Normal)

b) Escenario alternativo 1 (error en la generación del modelo de transformación)

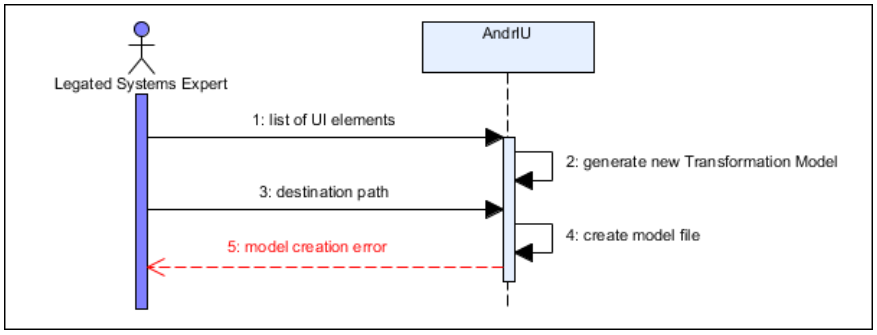


Figura 5.14. Diagrama de Secuencia del Flujo de Eventos CdU.4 (Escenario Alternativo 1)

5.3.1.1.2. CdU5. Generate AST Models

a) Escenario principal

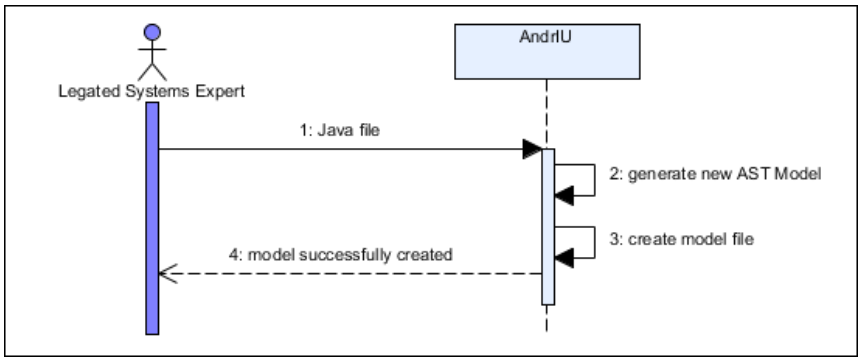


Figura 5.15. Diagrama de Secuencia de Análisis CdU.5 (Escenario Normal)

b) Escenario alternativo 1 (error en la generación de modelos AST)

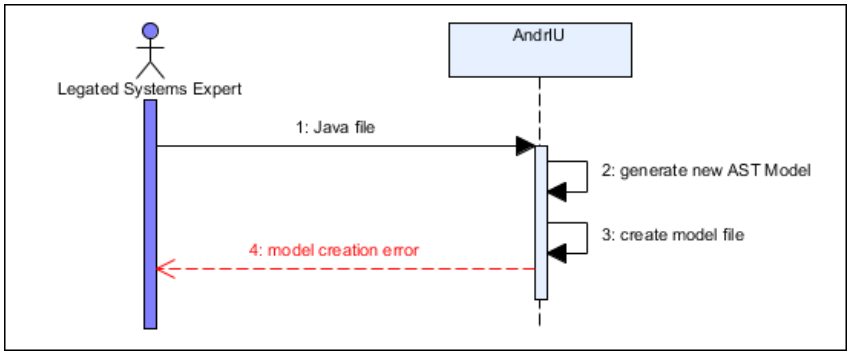


Figura 5.16. Diagrama de Secuencia de Análisis CdU.5 (Escenario Alternativo 1)

5.3.1.2. Diagramas de Clases de Análisis

La Figura 5.17 y la Figura 5.18 muestran los diagramas de clases de análisis realizados para los caso de uso CdU.4 y CdU.5. Este tipo de diagramas facilita la comprensión del sistema que finalmente será implementado, mediante la representación de las clases de análisis estereotipadas como *Boundaries* (clases relativas a la interfaz), *Controls* (clases relativas al control) y *Entities* (clases relativas a entidades). Esta parte corresponde al producto de salida PS. 3.2 (véase Tabla 4.3.).

5.3.1.2.1. CdU.4. Generate Transformation Models

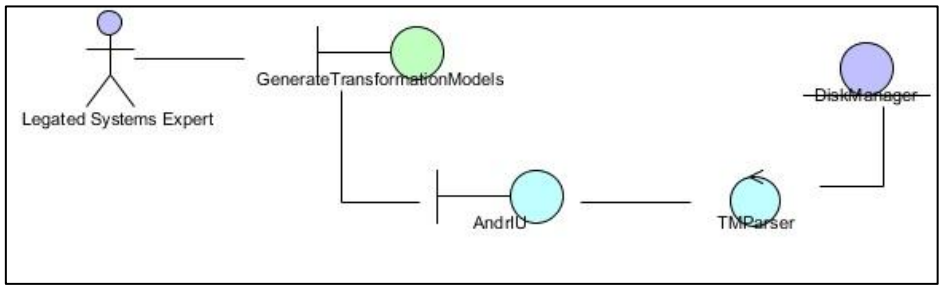


Figura 5.17. Diagrama de Clases de Análisis CdU.4

5.3.1.2.2. CdU5. Generate AST Models

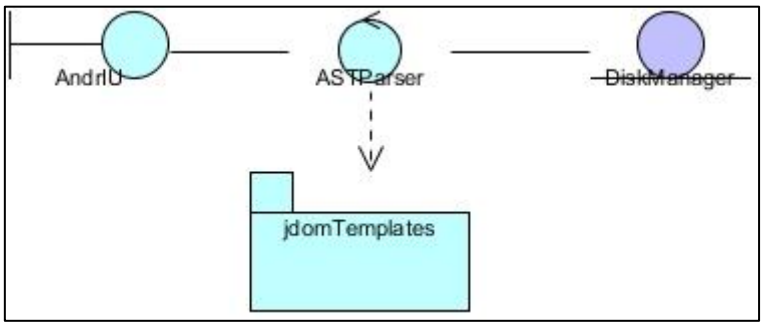


Figura 5.18. Diagrama de Clases de Análisis CdU.5

5.3.2. Diseño

En este apartado se describe el diseño realizado en la iteración 3. En este flujo de trabajo se obtendrán los productos de salida siguientes:

PS.3.3. Modelo arquitectónico

PS.3.4. Diagramas de clases de diseño de los casos de uso CdU.4 y CdU.5

PS.3.5. Diagramas de secuencia de diseño de los casos de uso CdU.4 y CdU.5

PS.3.6. Prototipos de la GUI de los casos de uso CdU.4 y CdU.5

5.3.2.1. Modelo multicapa

A la hora de realizar el diseño de la herramienta AndrIU se ha optado por la aplicación de un modelo multicapa, más concretamente tres capas, que organizan la arquitectura en paquetes, permitiendo separar el comportamiento entre las capas. Esto es muy importante, ya que facilita el mantenimiento de la herramienta y permite una mejor comprensión del sistema. De este modo 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*).

En la Figura 5.19 se muestra de manera resumida la estructura de la arquitectura en tres capas de la herramienta, con las dependencias existentes entre las capas, y corresponde al producto de salida PS.3.3.

> **Nota:** Se ha seguido un patrón de colores para diferenciar las clases que pertenecen a una capa u otra. De este modo las clases pertenecientes a la capa de presentación tienen color verde, las que pertenecen a la capa de dominio tienen color azul y las de la capa de persistencia tienen color morado.

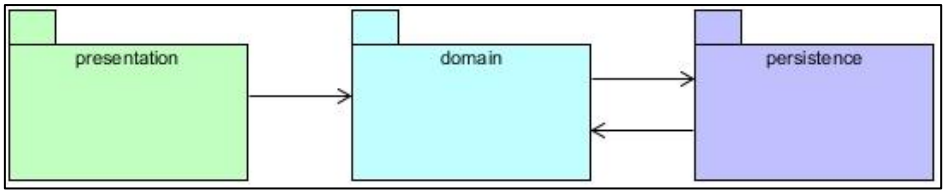


Figura 5.19. Diagrama de Paquetes en una arquitectura en tres capas.

Capa de presentación: Esta capa contiene 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. Del mismo modo muestra al usuario la información obtenida por la capa de dominio.

Capa de lógica de dominio: Esta capa contiene todas las clases específicas del dominio de negocio que se pretende modelar. Esto es, 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 contiene todas aquellas clases que permiten la persistencia de los datos a lo largo de las diferentes ejecuciones de la herramienta. En el caso de esta herramienta, la persistencia hace referencia al almacenamiento de ficheros en los proyectos. De este modo se maneja la entidad 'Proyecto' como la unidad que almacena y gestiona toda la información relativa a los modelos de IU generados a partir de un sistema heredado.

5.3.2.2. Patrones de diseño

Los patrones son buenas soluciones a problemas bien conocidos y frecuentes. Dichos patrones responden a soluciones que responden bien ante determinadas situaciones. Proponen soluciones genéricas muy sencillas de adaptar a problemas concretos (Gamma et al. 1995; Larman).

Se denominan patrones de diseño debido a que son utilizados principalmente en la etapa de diseño dentro del proceso de desarrollo software. Al realizar un desarrollo aplicando patrones de diseño, hay que seleccionar con cuidado los patrones que se van a aplicar, ya que el resultado puede variar en gran medida y el objetivo es aportar la mejor solución posible.

De este modo, los patrones de diseño utilizados en este Proyecto de Fin de Carrera son los siguientes:

5.3.2.2.1. Patrón *Fachada*

También conocido como *Facade, e*l patrón Fachada es un patrón de propósito estructural que permite centralizar el punto de acceso a las clases de un subsistema, simplificando el acceso al mismo. Este patrón permite reducir considerablemente el acoplamiento entre las clases de los distintos subsistemas.

En este proyecto se utiliza este patrón para desarrollar una fachada para toda la

capa de dominio, de manera que la capa de presentación accede a todas las funcionalidades de la capa de dominio a través de esta fachada. Esta fachada es implementada por la clase AndrIU del paquete *domain* (véase Figura 5.20), de modo que todas las operaciones que ofrece la capa de dominio a la capa de presentación están definidas en dicha clase.

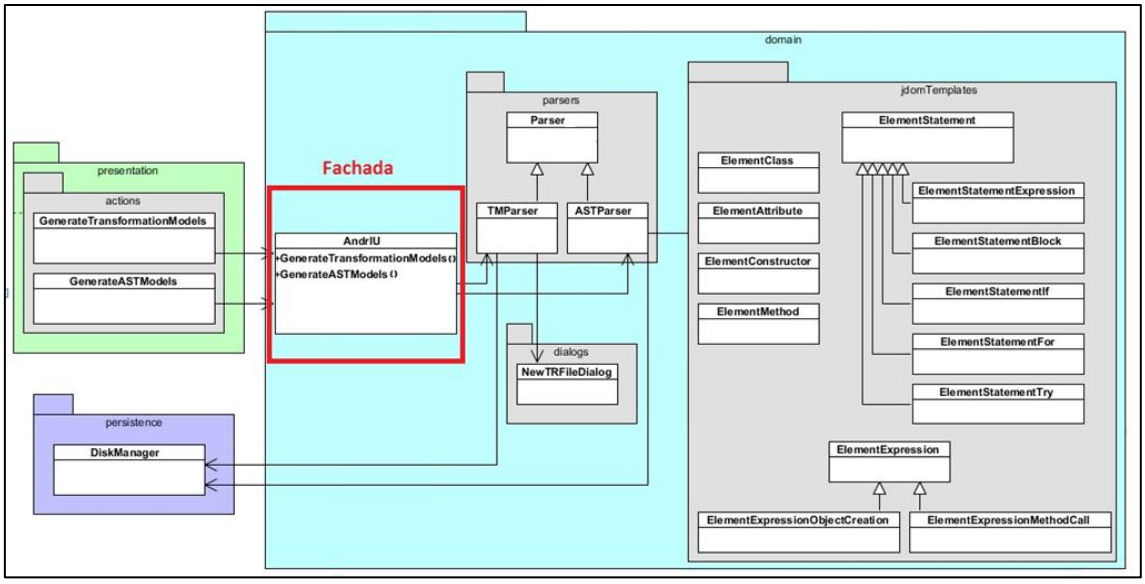


Figura 5.20. Ejemplo del patrón Fachada en el PFC

5.3.2.2.2. Patrón *Singleton*.

A veces conviene soportar la visibilidad global o un solo punto de acceso a una instancia individual de una clase y no a alguna otra forma de visibilidad (Gamma et al. 1995). Esto se consigue obligando a que tan sólo se pueda crear una instancia de la clase *Singleton*. En el caso de este PFC, este patrón de creación se ha utilizado en la clase *DiskManager*, que se encarga de la persistencia del sistema. De este modo, si existe una instancia de *DiskManager*, al ser requerido se devolverá la propia instancia y en caso contrario, se creará la instancia. Este patrón se puede ver reflejado en la Figura 5.21, y el fragmento de código correspondiente al mismo en la Figura 5.22.

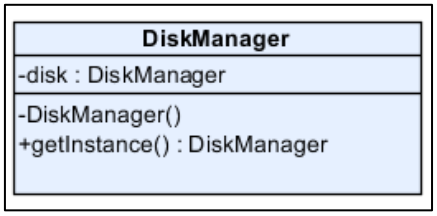


Figura 5.21. Ejemplo de Patrón Singleton

```
public class DiskManager {

    private static DiskManager disk=null;

    private DiskManager(){}

    public static DiskManager getInstance(){
        if(disk==null)
            disk=new DiskManager();
        return disk;
    }
```

Figura 5.22. Código fuente para la implementación del patrón Singleton

5.3.2.3. Diagramas de Clases de Diseño

Una vez definida la arquitectura de la herramienta, se procede a realizar el diagrama de clases correspondiente al diseño de los primeros casos de uso. En la figura Figura 5.23 se muestra el diagrama de Clases de Diseño correspondiente al caso de uso CdU.4 y en la Figura 5.24 se muestra el diagrama de Clases de Diseño correspondiente al caso de uso CdU.5.

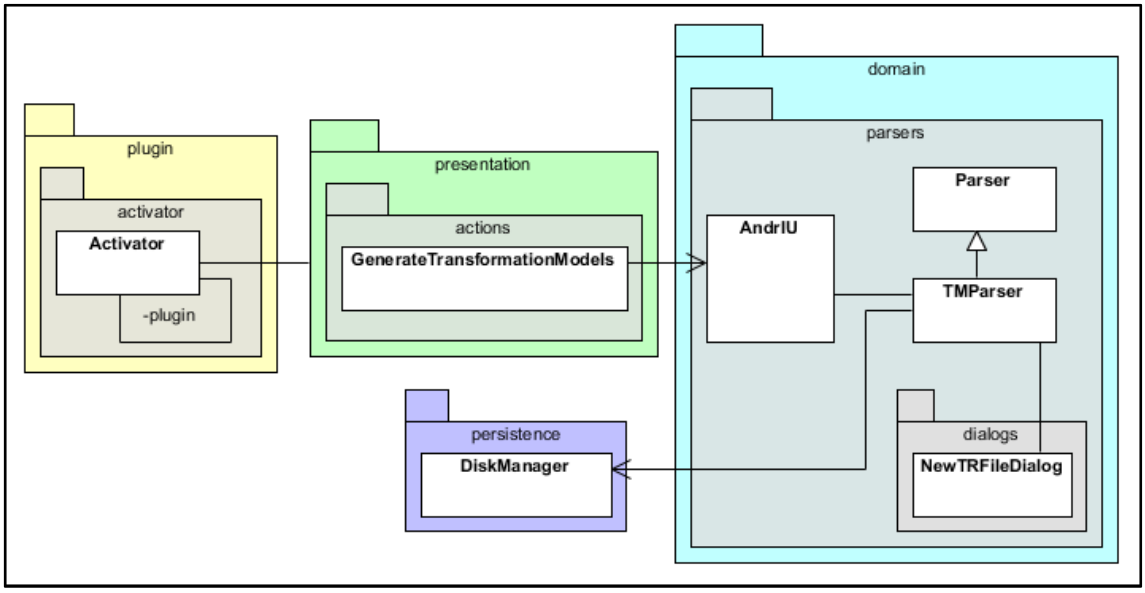


Figura 5.23. Diagrama de Clases de Diseño del CdU.4 (Generar Modelos de Transformación)

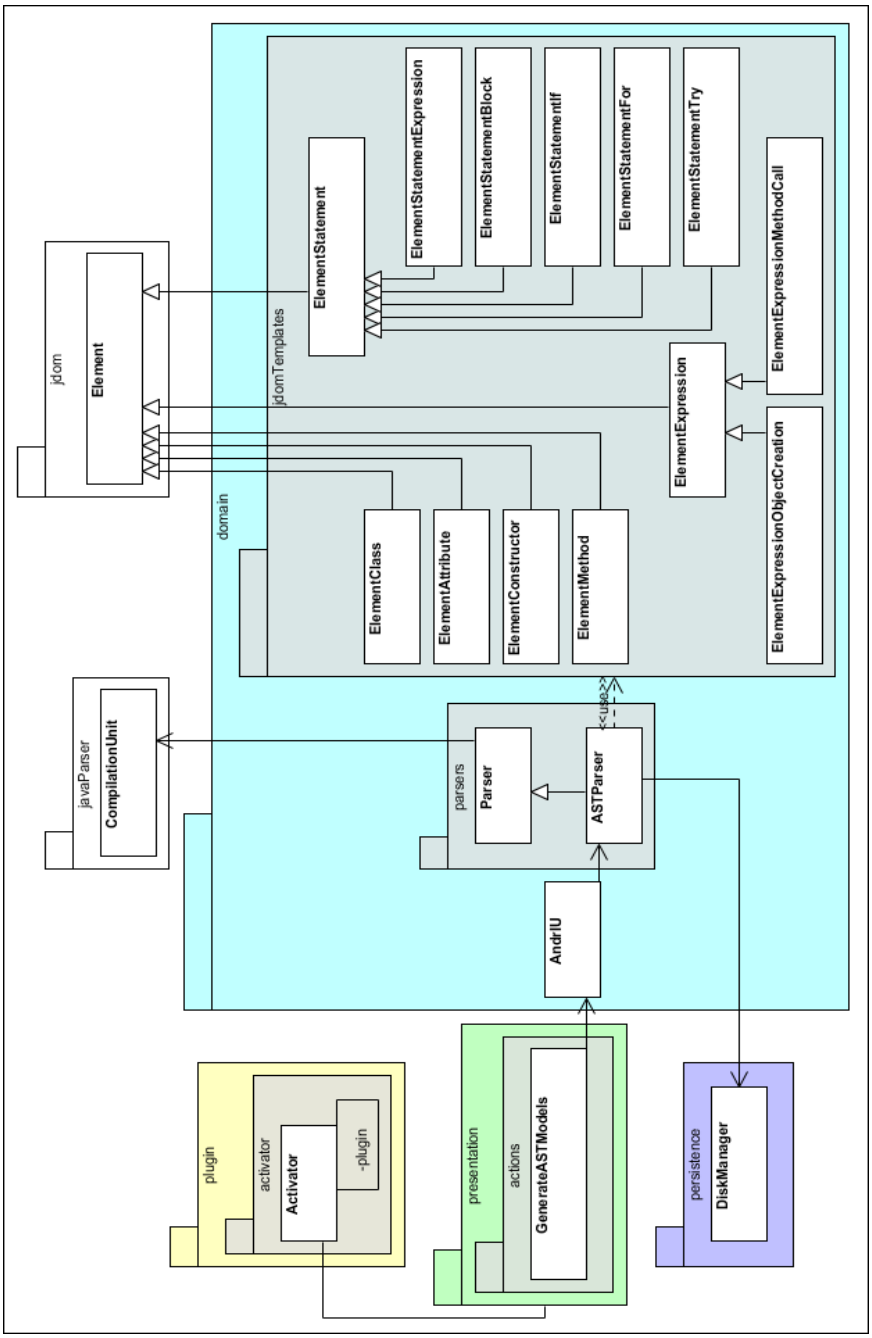


Figura 5.24. Diagrama de Clases de Diseño del CdU.5 (Generar Modelos AST)

5.3.2.4. Diagramas de Secuencia de Diseño

Con las clases diseñadas, se procede a la realización de los diagramas de secuencia, correspondientes al producto de salida PS.3.5. Estos diagramas de manera mucho más detallada el funcionamiento del sistema, permitiendo una comprensión más profunda del mismo. En las figuras Figura 5.25 y Figura 5.26 se pueden observar los diagramas de secuencia para los casos de uso CdU.4 y CdU.5 respectivamente.

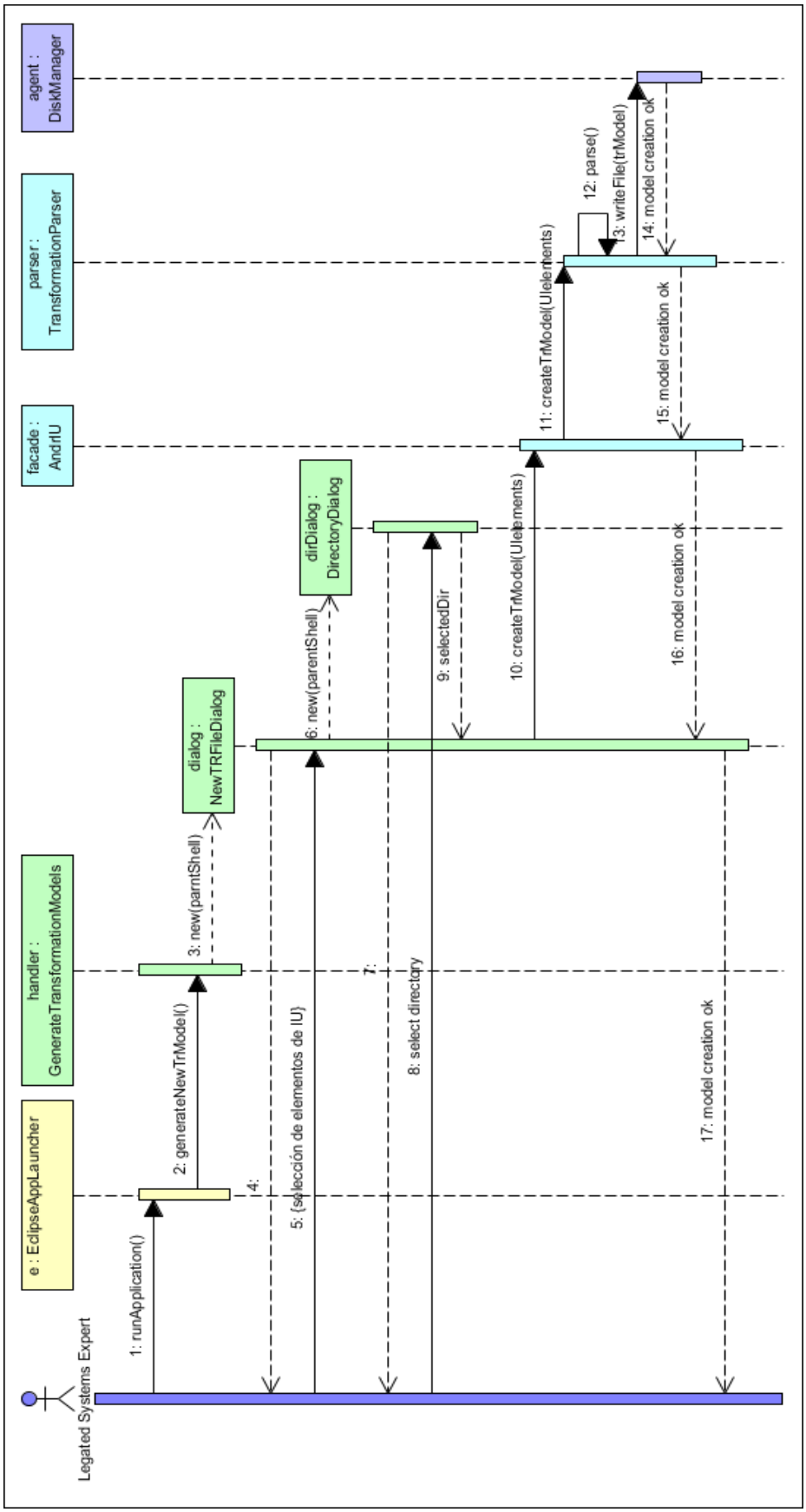


Figura 5.25. Diagrama de Secuencia de Diseño del CdU.4 (Generar Modelos de Transformación)

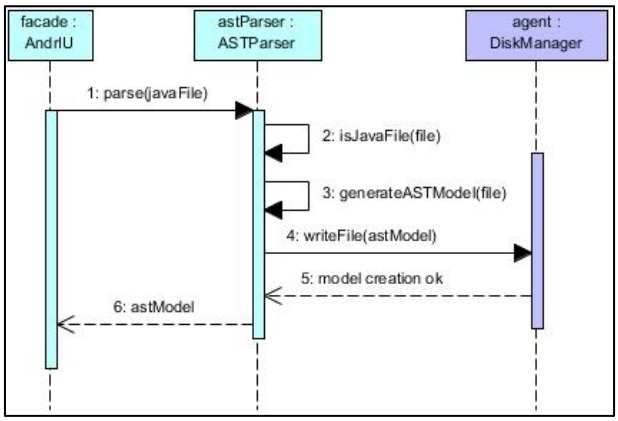


Figura 5.26. Diagrama de Secuencia de Diseño del CdU.5 (Generar Modelos AST)

Como se puede observar, en el diagrama de secuencia del CdU.5 no aparece ningún actor. Como se ha comentado anteriormente, esta funcionalidad será utilizada por otras y no directamente, pero por complejidad se ha decidido separarla. Por ello es la propia clase fachada la que ofrece esta funcionalidad.

5.3.2.5. Prototipos de la GUI

A continuación se realiza un diseño de los elementos de la interfaz que presentará la herramienta AndrIU de acuerdo a los casos de uso que se desarrollan en esta iteración. Esto es muy importante, ya que es imprescindible presentar al usuario la herramienta de manera sencilla e intuitiva. Al ser un plug-in para Eclipse, los elementos de la GUI son los que ofrece el propio Eclipse, y se basan en menús, barras de herramientas y ventanas de diálogo. Para realizar el diseño de los prototipos iniciales de la GUI de AndrIU se ha utilizado la herramienta *Balsamiq Mockups*.

De este modo, como primer prototipo se ha realizado un diseño de la perspectiva AndrIU (véase Figura 5.27), que tiene como elementos principales un menú en la barra de menús (arriba), otro menú contextual desplegable, el explorador (a la izquierda) y la consola (abajo).

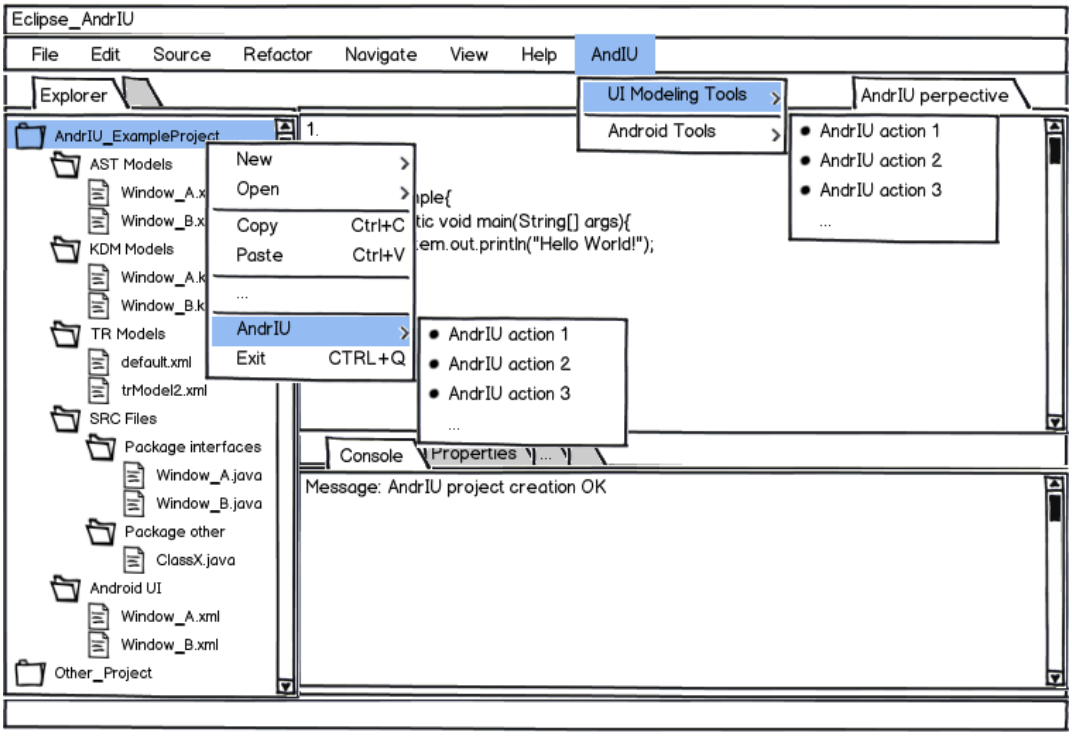


Figura 5.27. Prototipo de la perspectiva AndrIU

En la Figura 5.28 se muestra el prototipo del menú AndrIU de la barra de menús, y en la Figura 5.29 el menú contextual. Con estos menús el usuario podrá utilizar la funcionalidad de generar modelos de transformación que ofrece AndrIU.

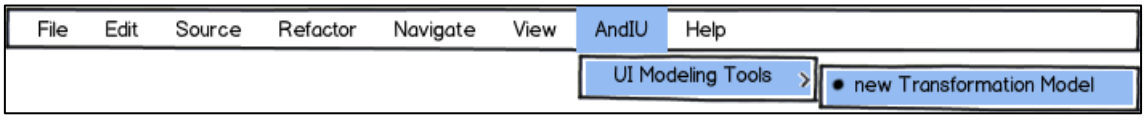


Figura 5.28. Prototipo del menú para el CdU.4

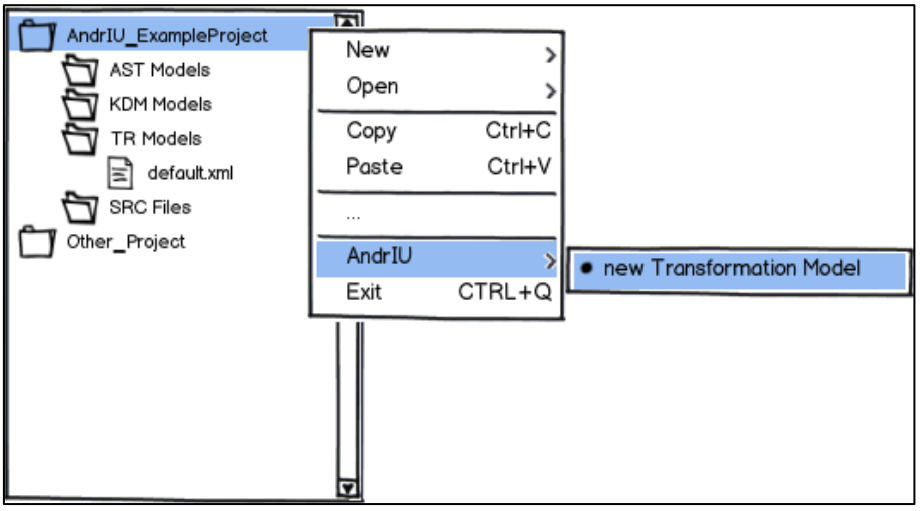


Figura 5.29. Prototipo del menú contextual para el CdU.4

Por último, en la Figura 5.30 se muestra el prototipo de la ventana de diálogo con las opciones para la creación del modelo de transformación. En esta ventana, el usuario selecciona los elementos que se van a generar en el nuevo modelo de transformación. Estos prototipos se corresponden con el producto de salida PS.3.6.

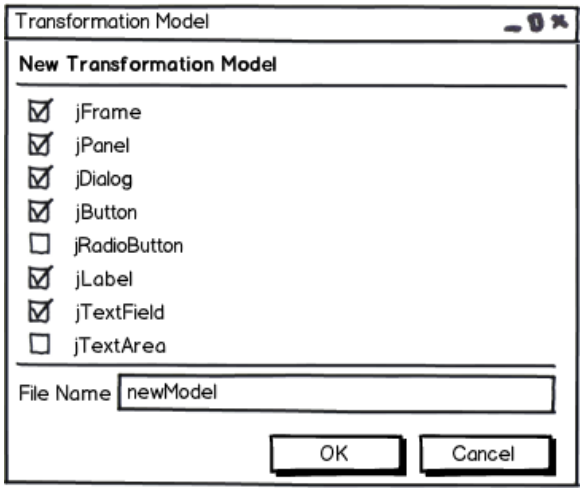


Figura 5.30. Prototipo de la ventana de creación de modelo de transformación

5.4. Iteración 4

En esta cuarta iteración se comenzará con el desarrollo de CdU.6 y CdU.7, abordando las fases de análisis y diseño de los mismos. De forma paralela se realizará la implementación de los casos de uso CdU.4 y CdU.5, diseñados en la iteración anterior.

5.4.1. Análisis

En este flujo de trabajo se obtendrán los productos de salida siguientes:

PS.4.1. Diagramas de secuencia de análisis de los casos de uso CdU.6 y CdU.7

PS.4.2. Diagramas de clases de análisis de los casos de uso CdU.6 y CdU.7

5.4.1.1. Diagramas de Secuencia de Análisis

Se han utilizado diagramas de secuencia de análisis para facilitar la comprensión del sistema. En las figuras siguientes se presentan los diagramas de secuencia de análisis correspondientes a los casos de uso CdU.6 y CdU.7, que corresponden con el producto de salida PS.4.1.

5.4.1.1.1. CdU.6-CdU.7. Generate KDM Models from Java file

a) Escenario principal

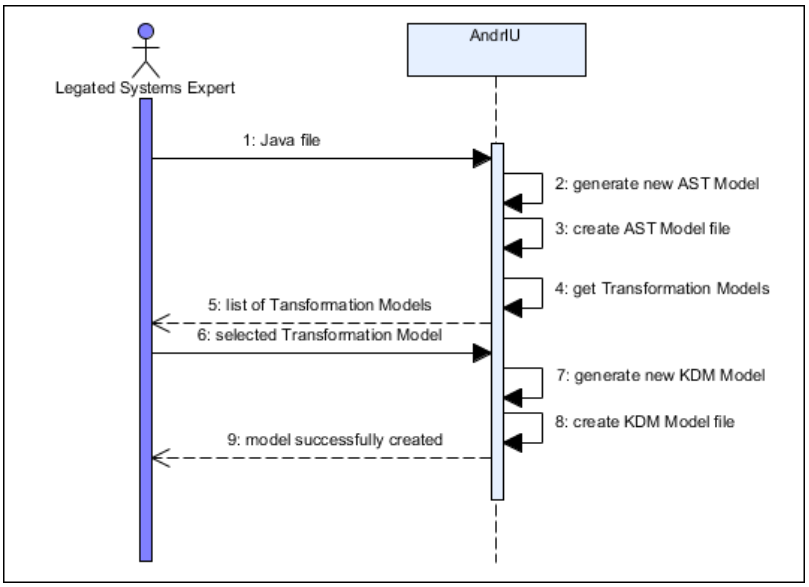


Figura 5.31. Diagrama de Secuencia de Análisis de los CdU.6-CdU.7 (Escenario normal)

b) Escenario alternativo 1 (error en la generación del modelo AST)

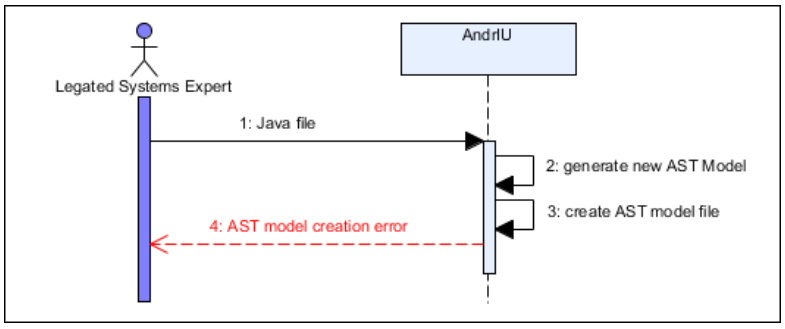


Figura 5.32. Diagrama de Secuencia de Análisis de los CdU.6-CdU.7 (Escenario alternativo 1)

c) Escenario alternativo 2 (error en el listado de modelos de transformación)

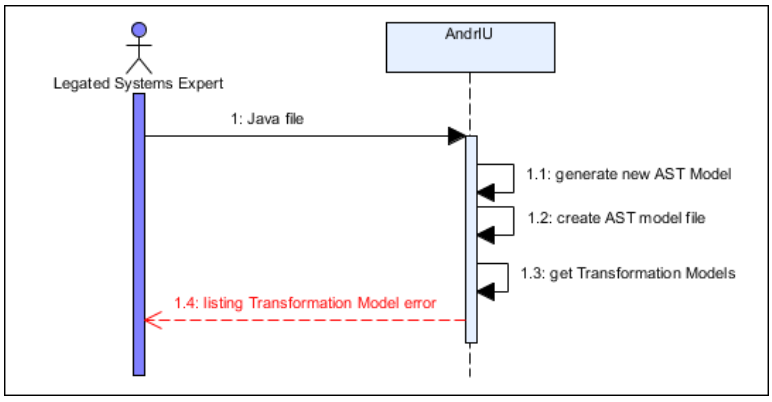


Figura 5.33. Diagrama de Secuencia de Análisis de los CdU.6-CdU.7 (Escenario alternativo 2)

d) Escenario alternativo 3 (error en la generación de modelos KDM)

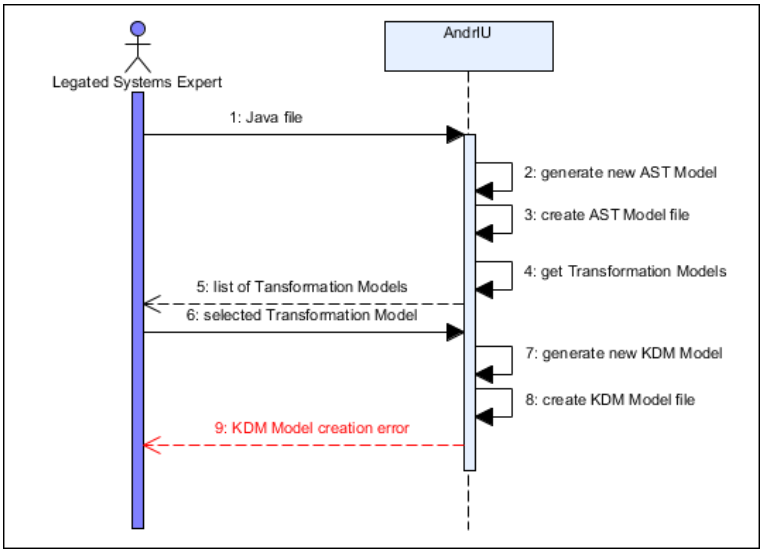


Figura 5.34. Diagrama de Secuencia de Análisis de los CdU.6-CdU.7 (Escenario alternativo 3)

5.4.1.2. Diagramas de Clases de Análisis

Tras definir los diagramas de secuencia de análisis se procede a realizar los diagramas de clases de análisis al igual que en la iteración anterior. En la Figura 5.35 se puede observar dicho diagrama, que corresponde con el producto de salida PS.4.2.

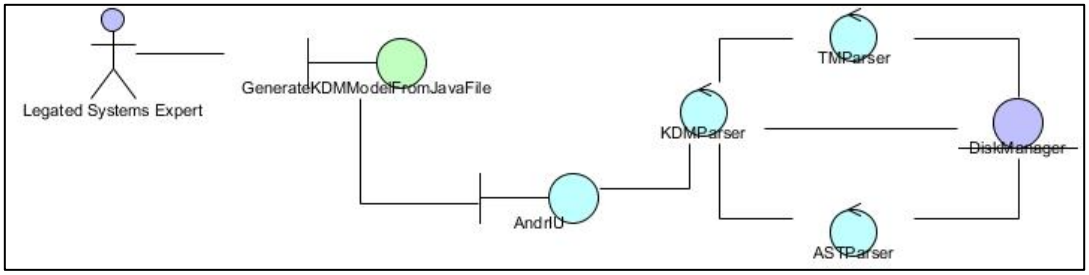


Figura 5.35. Diagrama de Clases de análisis de los CdU.6-CdU.7

5.4.2. Diseño

En este flujo de trabajo se obtendrán los productos de salida siguientes:

PS.4.3. Diagramas de clases de diseño de los casos de uso CdU.6 y CdU.7

PS.4.3. Diagramas de secuencia de diseño de los casos de uso CdU.6 y CdU.7

PS.4.4. Prototipos de la GUI de los casos de uso CdU.6 y CdU.7

5.4.2.1. Diagramas de Clases de Diseño

Una vez terminado el análisis de los casos de uso relativos a la generación de modelos KDM a partir de un fichero de código Java, se diseñan las clases que implementarán esta funcionalidad. En la Figura 5.36 se muestra el esquema de clases y en las posteriores se detallan las clases más importantes. Esto se corresponde con el producto de salida PS.4.3.

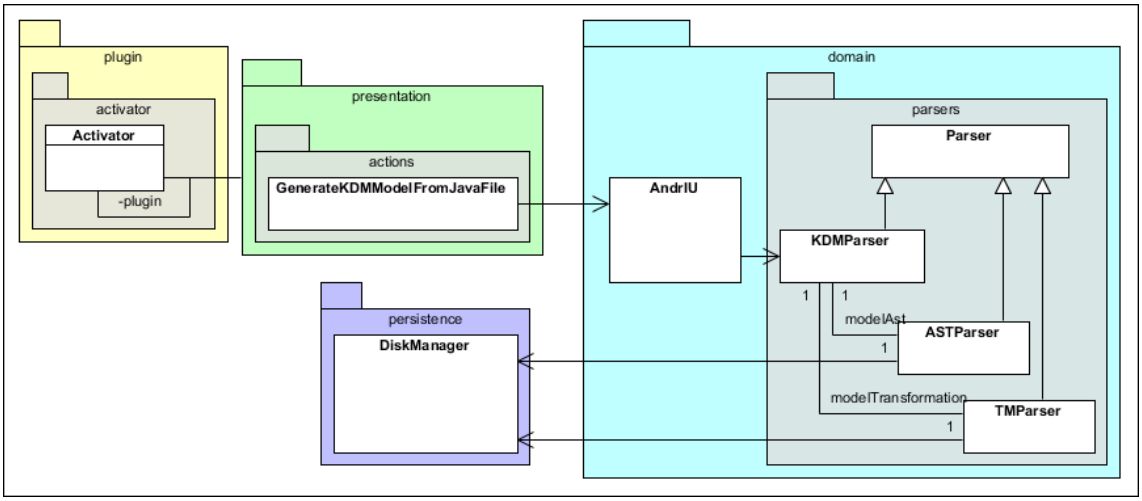


Figura 5.36. Diagrama de Clases de Diseño de la iteración 4

5.4.2.2. Diagramas de Secuencia de Diseño

Tras diseñar las clases, se realizan los diagramas de secuencia de diseño que proporcionan una visión más detallada del funcionamiento. En la Figura 5.37 se puede observar el diagrama de secuencia correspondiente a los casos de uso CdU.6 y CdU.7, que se corresponde con el producto de salida PS.4.4.

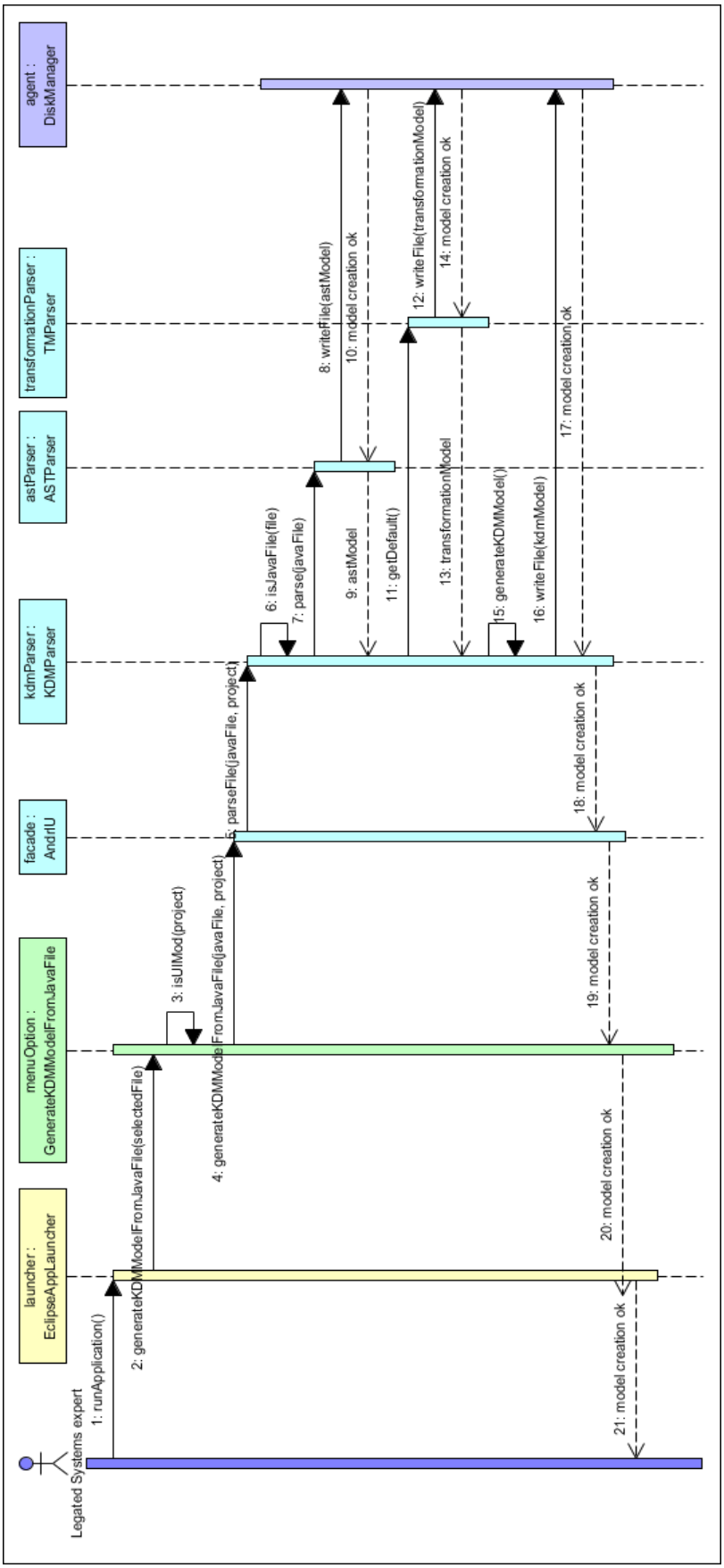


Figura 5.37. Diagrama de Secuencia de la iteración 4

5.4.2.3. Prototipos de la GUI

Del mismo modo que se ha hecho en al iteración anterior, en este apartado se diseñan los prototipos de la GUI que presentará la herramienta que se está desarrollando.

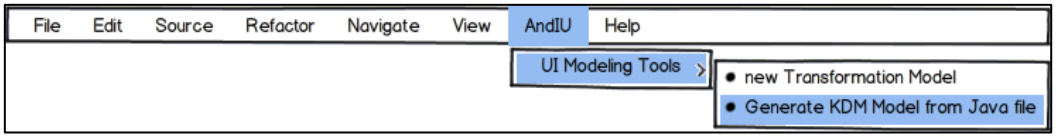


Figura 5.38. Prototipo del menú para la iteración 4

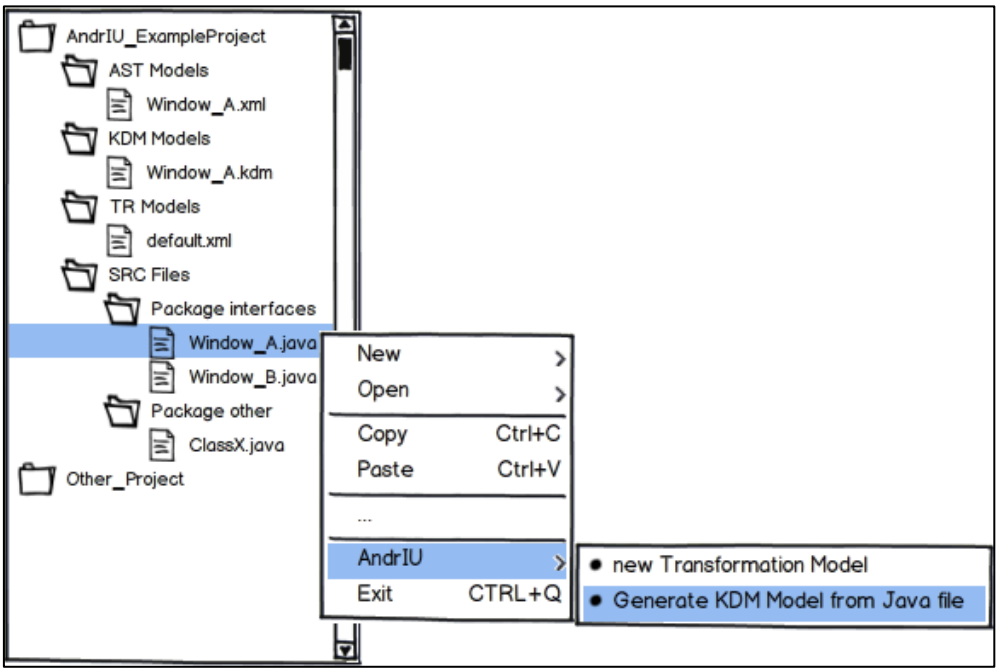


Figura 5.39. Prototipo del menú contextual para la iteración 4

5.4.3. Implementación

En esta iteración se comienza la implementación de los primeros casos de uso de la herramienta (CdU.4 y CdU.5), que serán necesarios para implementar otros casos de uso posteriores. 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.

5.4.3.1. Implementación del CdU.4

En primer lugar se implementa el CdU.4, que permite la creación de modelos de transformación. La herramienta utilizará estos modelos de transformación para definir qué elementos del modelo de código (AST) se transforman en elementos KDM. De este modo, los elementos que se han tomado inicialmente se muestran en la Tabla 5.18. .

Elementos Java del modelo de código	Elementos KDM
JFrame	Screen
JPanel	Screen
JDialog	Report
JButton	UIResource
JRadioButton	UIResource
JButtonGroup	UILayout
JLabel	UIField
JTextField	UIField
JTextArea	UIField

Tabla 5.18. Elementos del modelo de transformación

Estos modelos de transformación se almacenan en ficheros XML, ya que de este modo se facilita su acceso, modificación, etc., cuando se utilicen los parsers. Para la definición de cada elemento de GUI, se observa un elemento "declaration" con dos campos: input, que hace referencia al elemento del modelo de código, y output, que hace referencia al elemento KDM. En la Figura 5.40 se muestra un ejemplo de modelo de transformación en XML. Para facilitar las tareas de lectura y escritura en ficheros XML se ha utilizado la librería JDOM (Hunter, 2009).

```
<?xml version="1.0" encoding="UTF-8"?>
<TansformationModel name=example>
  <declaration input="JFrame" output="Screen" />
  <declaration input="JPanel" output="Screen" />
  <declaration input="JDialog" output="Report" />
  <declaration input="JButton" output="UIResource" />
  <declaration input="JRadioButton" output="UIResource" />
  <declaration input="JButtonGroup" output="UILayout" />
  <declaration input="JLabel" output="UIField" />
  <declaration input="JTextField" output="UIField" />
  <declaration input="JTextArea" output="UIField" />
</TansformationModel>
```

Figura 5.40. Ejemplo de un modelo de código en XML

A continuación se muestra la clase *TMParser* (véase Figura 5.41), que se encarga de la generación de los modelos de transformación. Esta clase hereda de Parser, que contiene algunas de funciones básicas de lectura y escritura de ficheros (y que serán utilizadas por futuros parsers). Estas operaciones las tratará el *DiskManager* (véase Figura 5.42), la clase que se encarga de gestionar la persistencia de la herramienta.

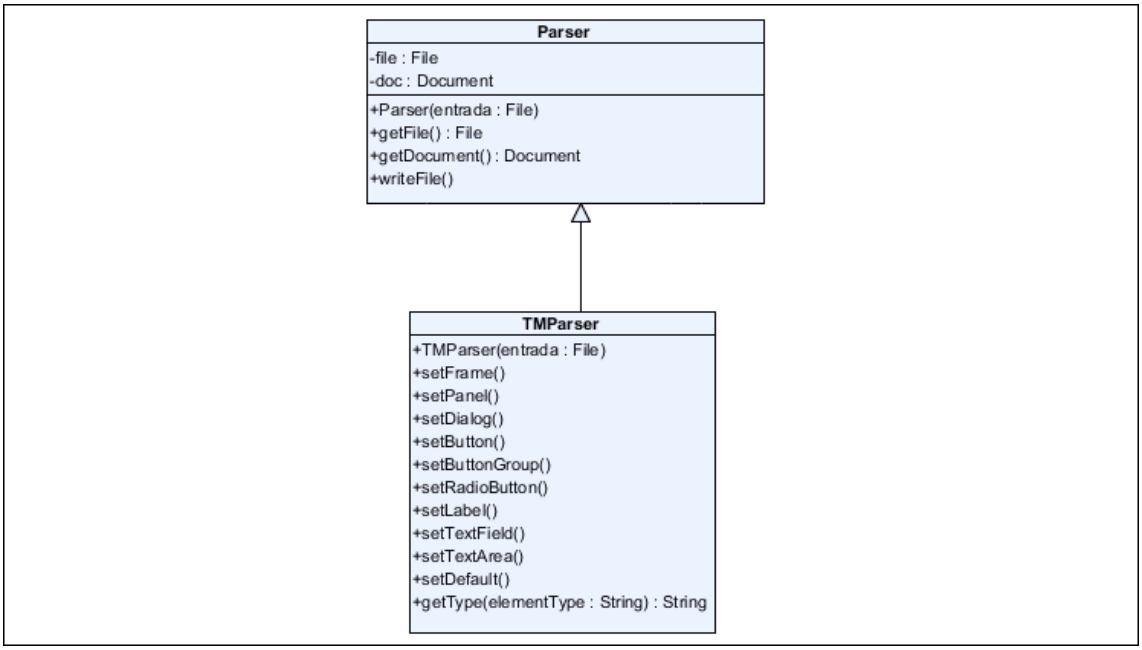


Figura 5.41. Clases Parser y TMParser

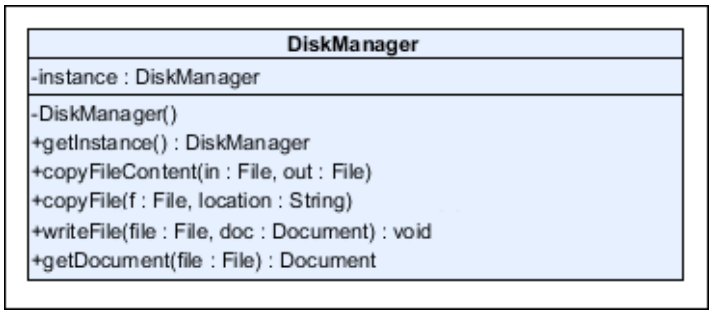


Figura 5.42. Clase DiskManager (I)

5.4.3.2. Implementación del CdU.5

La implementación del CdU.5 consiste en la generación de los modelos de código o modelos AST, para lo cual se ha creado un parser de Java, que extrae el modelo de código en forma de un árbol sintáctico abstracto. Este parser de java se ha generado con la herramienta *JavaCC* a partir de la gramática de Java 1.6. Como resultado se obtiene un proyecto Java con las funcionalidades necesarias para obtener el modelo de código AST de un fichero Java de entrada. En la Figura 5.43 se puede observar la estructura del parser de Java obtenido.

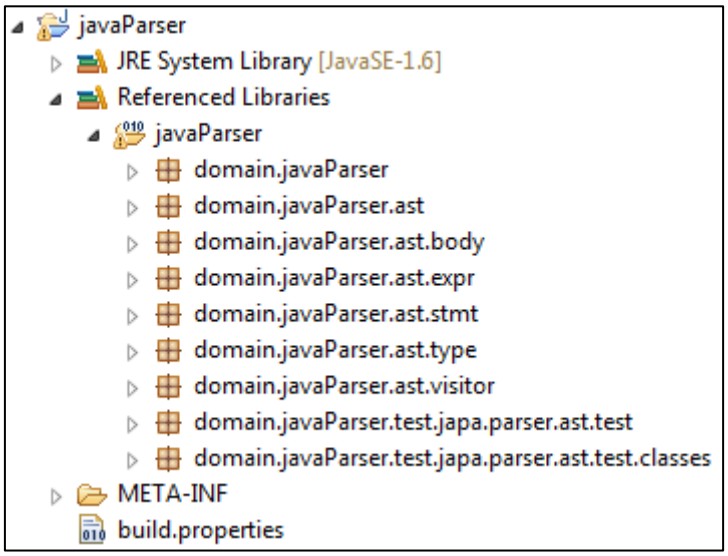


Figura 5.43. Estructura del parser de Java obtenido

Una vez obtenido el parser para extraer el modelo de código, se implementa un parser (*ASTParser*) para almacenar en forma de ficheros XML este modelo de código. Para ello, se recorre el árbol sintáctico abstracto obtenido con el parser de Java (que se almacena en memoria) a través de las funciones que dicho parser ofrece, y se van

escribiendo los elementos en un fichero XML gracias a las funciones que ofrece la librería JDOM. En la Figura 5.44 se puede observar un fragmento de un fichero XML con un modelo AST de AndrIU.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmlns:xmi="http://www.omg.org/XMI" xmi:version="2.0">
  <Class xmi:id="VentanaSeleccion1" name="VentanaSeleccion1" type="javax.swing.JFrame">
    <Attribute xmi:id="serialVersionUID" name="serialVersionUID" type="long" />
    <Attribute xmi:id="jButtonSeleccionarEntrada" name="jButtonSeleccionarEntrada" type="JButton" />
    <Attribute xmi:id="jRBButtonGroup" name="jRBButtonGroup" type="JRadioButton" />
    <Attribute xmi:id="jButtonTransformación" name="jButtonTransformación" type="JButton" />
    <Attribute xmi:id="jRBTextArea" name="jRBTextArea" type="JRadioButton" />
    <Attribute xmi:id="jRBTextField" name="jRBTextField" type="JRadioButton" />
    <Attribute xmi:id="jRBLabel" name="jRBLabel" type="JRadioButton" />
    <Attribute xmi:id="jRBRadioButton" name="jRBRadioButton" type="JRadioButton" />
    <Attribute xmi:id="mensajes" name="mensajes" type="JTextArea" />
    <Attribute xmi:id="jLabel1" name="jLabel1" type="JLabel" />
    <Attribute xmi:id="jTextFieldFicheroSeleccionado" name="jTextFieldFicheroSeleccionado" type="JTextField" />
    <Method xmi:id="main" name="main" returnType="void">
      <Parameters>
        <Attribute xmi:id="args" name="args" type="String[]" />
      </Parameters>
      <Body>
        <Statement TIPO="ExpressionStatement">
          <Expression TIPO="MethodCallExpression" name="invokeLater">
            <Scope content="SwingUtilities" />
            <Arguments>
              <Expression TIPO="ObjectCreationExpression" type="Runnable">
                <BODY_DECLARATION>
                  <Method xmi:id="run" name="run" returnType="void">
                    <Parameters />
                    <Body>
                      <Statement TIPO="ExpressionStatement">
                        <Expression TIPO="VariableDeclarationExpression" Type="VentanaSeleccion1">
                          <Variable name="inst" />
```

Figura 5.44. Fragmento de un AST model en XML

La principal clase que implementa esta funcionalidad es *ASTParser* (véase Figura 5.45), que también hereda de *Parser*. Ésta clase es la que utiliza las funcionalidades que ofrece el parser de Java generado. Además *ASTParser* utiliza una serie de clases definidas en el paquete *jdomTemplates* (véase Figura 5.46), que hacen de plantillas para crear los elementos del fichero XML que almacenará el modelo AST. Estas clases heredan de la clase Element de la librería JDOM.

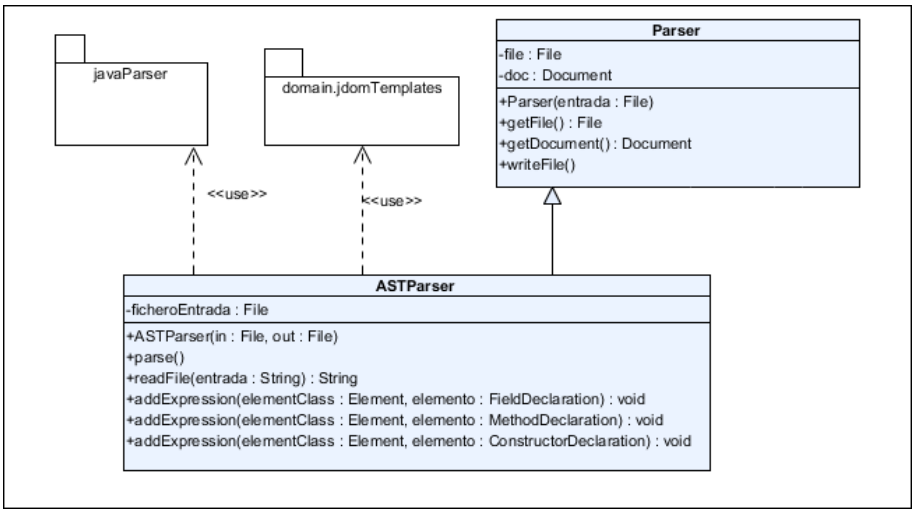


Figura 5.45. Clase ASTParser

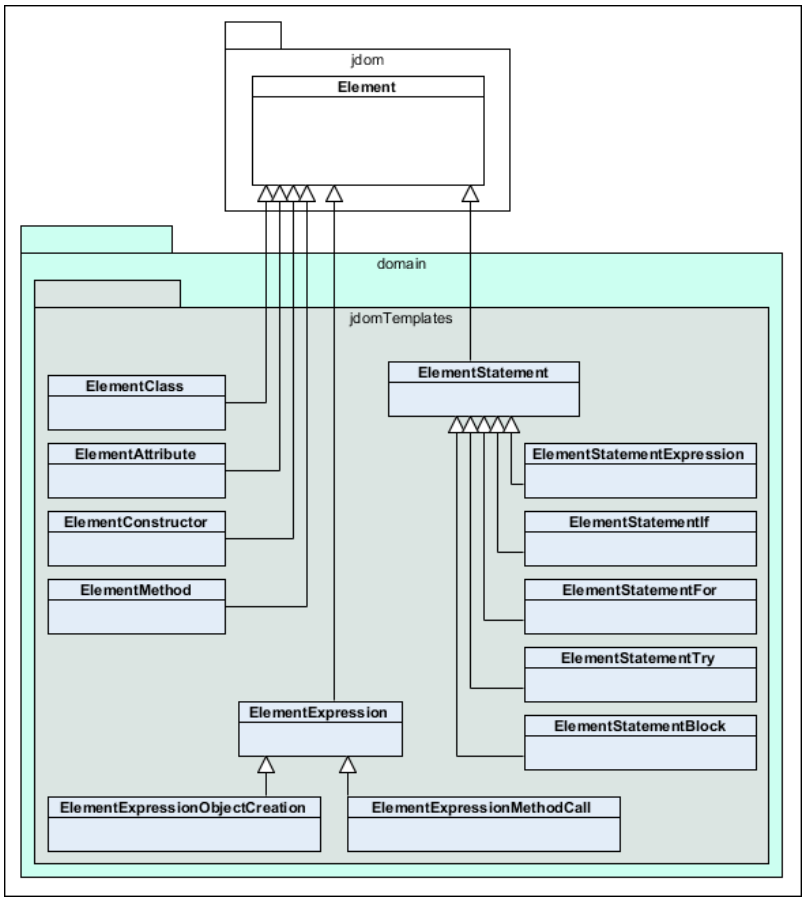


Figura 5.46. Paquete jdomTemplates

La clase *GenerateTR*, perteneciente a la capa de presentación, que implementa el manejador para los menús del plug-in se puede observar en la Figura 5.47. Además, utiliza la ventana de diálogo *NewTRFileDialog*, que se puede observar en la Figura 5.48. La generación de modelos AST no tiene un manejador de eventos, ya que es una

acción que no se ejecutará directamente desde una opción de ningún menú. Esta funcionalidad irá incluida al generar los modelos KDM en las siguientes iteraciones.

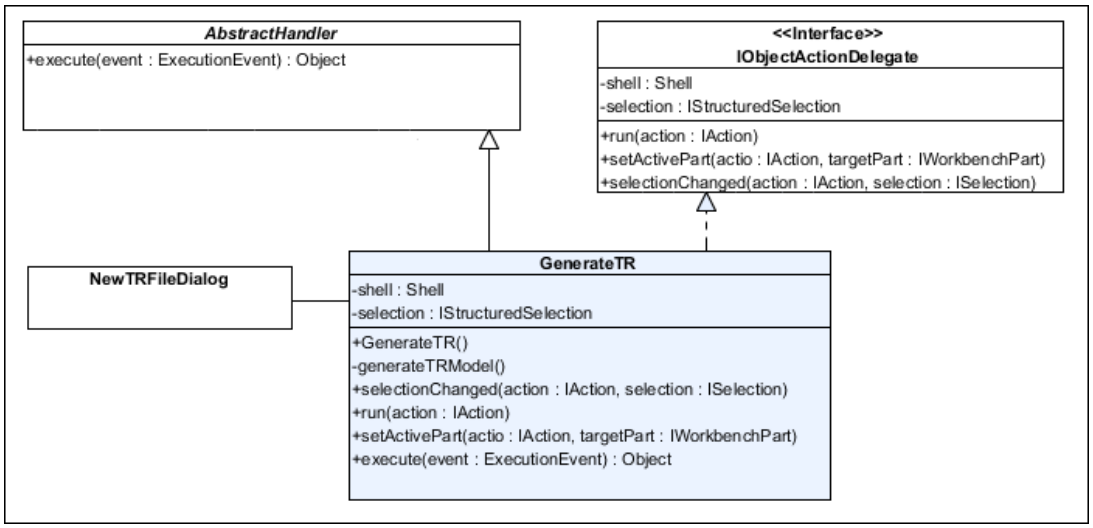


Figura 5.47. Clase GenerateTR

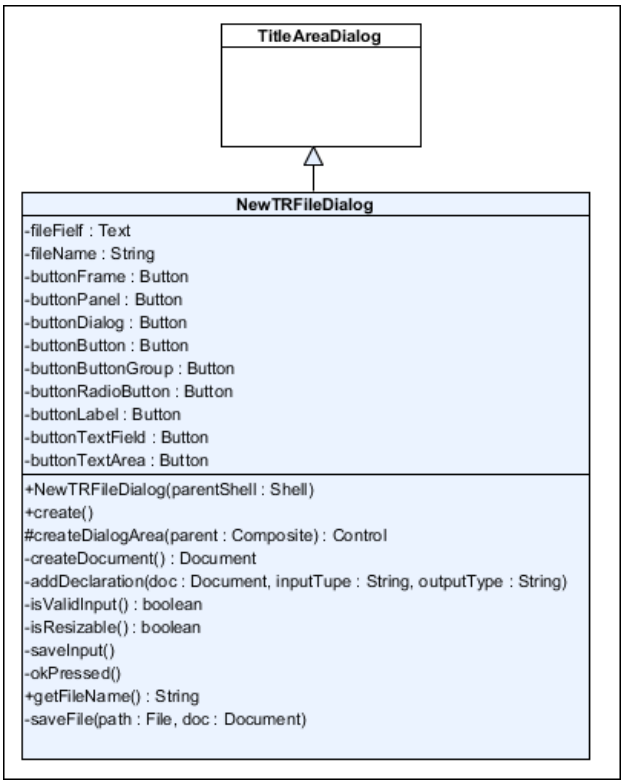


Figura 5.48. Clase NewTRFileDialog

5.5. Iteración 5

En esta quinta iteración se desarrollarán los casos de uso CdU.1, CdU.2, CdU.3 y CdU.8. Se comienza con el análisis y el diseño de los tres primeros casos de uso, referentes a la creación de proyectos que contengan los modelos que se van a generar. Del mismo modo se realiza el análisis y diseño del CdU.8, que genera modelos KDM desde estos proyectos.

De manera paralela se comienza con la implementación de los casos de uso CdU.6 y CdU.7 diseñados en la iteración anterior.

5.5.1. Análisis

En este flujo de trabajo se obtendrán los productos de salida siguientes:

PS.5.1. Diagramas de secuencia de análisis de CdU.1, CdU.2, CdU.3 y CdU.8.

PS.5.2. Diagramas de clases de análisis de CdU.1, CdU.2, CdU.3 y CdU.8.

5.5.1.1. Diagramas de Secuencia de Análisis

Al igual que en la iteración anterior, se ha optado por utilizar diagramas de secuencia de eventos para comprender en mayor medida el sistema. A continuación se presentan los diagramas de secuencia de análisis de los casos de uso CdU.1, CdU.2, CdU.3 y CdU.8 que se corresponden con los flujos de eventos de los CdU. Estos diagramas corresponden con el producto de salida PS.5.1 (véase Tabla 4.5. .

5.5.1.1.1. CdU.1. Create new AndrIU project

a) Escenario principal

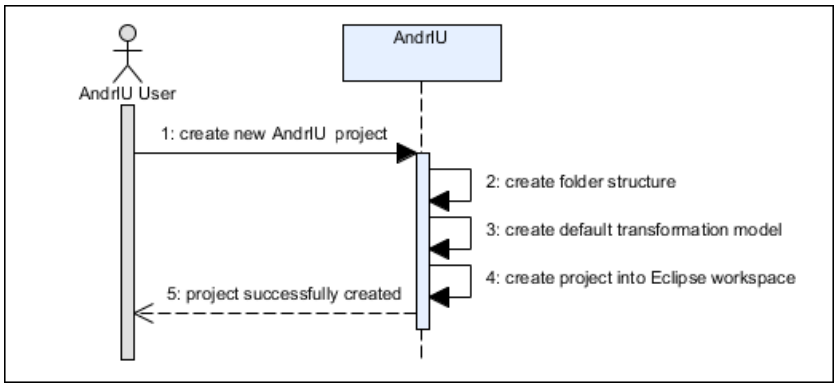


Figura 5.49. Diagrama de Secuencia de Análisis del CdU.1 (Escenario normal)

b) Escenario alternativo (error en la creación del proyecto)

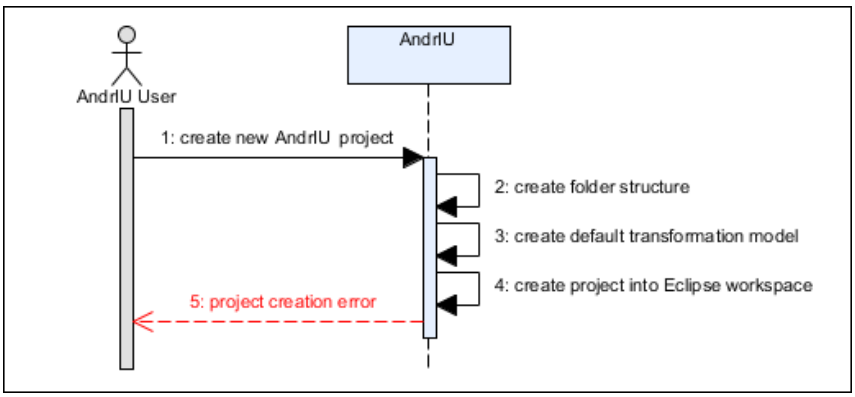


Figura 5.50. Diagrama de Secuencia de Análisis del CdU.1 (Escenario alternativo)

5.5.1.1.2. CdU.2. Create new AndrIU project from scratch

a) Escenario principal

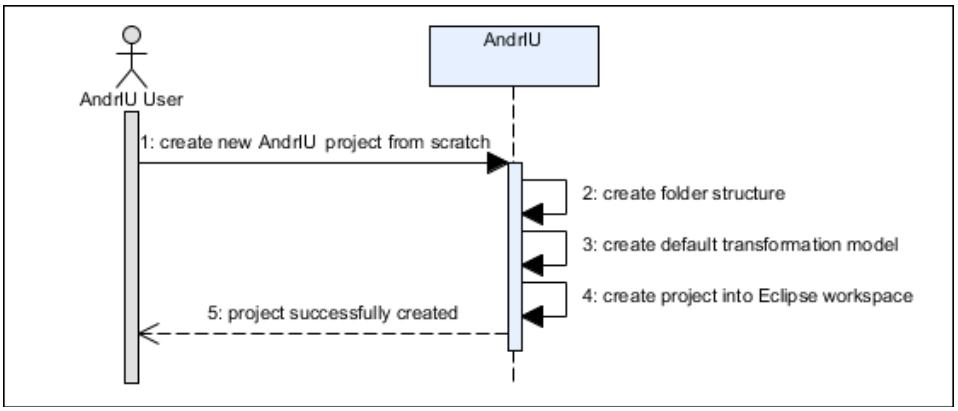


Figura 5.51. Diagrama de Secuencia de Análisis del CdU.2 (Escenario normal)

b) Escenario alternativo (error en la creación del nuevo proyecto)

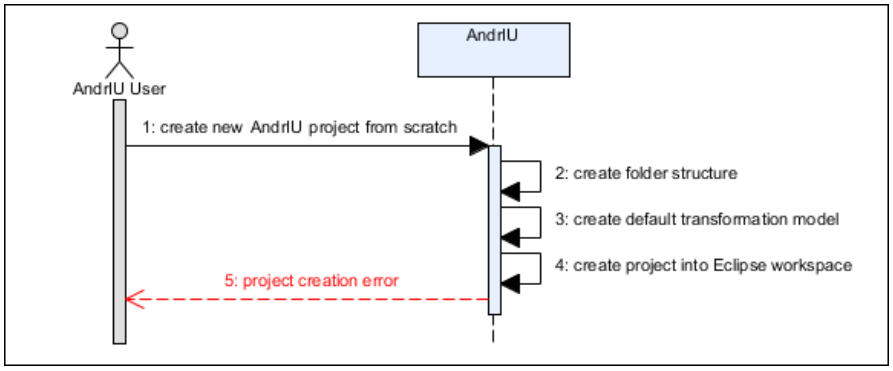


Figura 5.52. Diagrama de Secuencia de Análisis del CdU.2 (Escenario alternativo)

5.5.1.1.3. CdU.3. Create new AndrIU project from existing Java project

a) Escenario principal

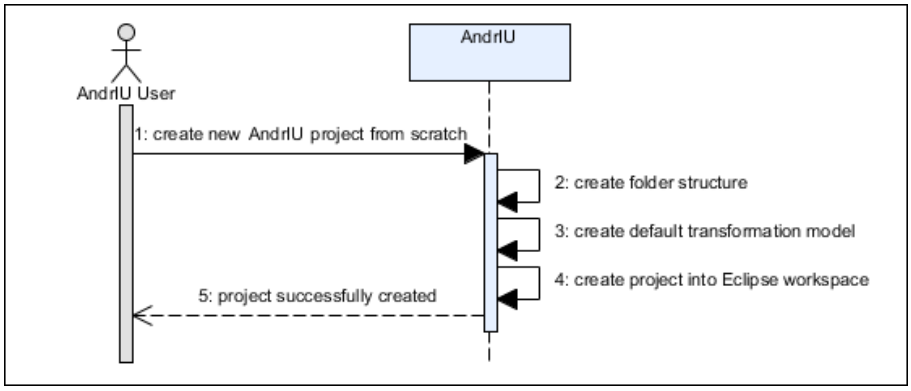


Figura 5.53. Diagrama de Secuencia de Análisis del CdU.3 (Escenario normal)

b) Escenario alternativo (error en la creación del proyecto)

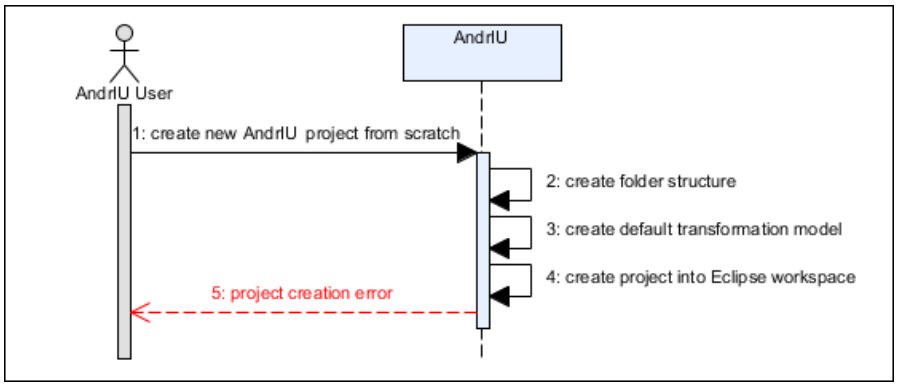


Figura 5.54. Diagrama de Secuencia de Análisis del CdU.3 (Escenario alternativo)

5.5.1.1.4. CdU.8. Generate KDM Models form AndrIU project

a) Escenario principal

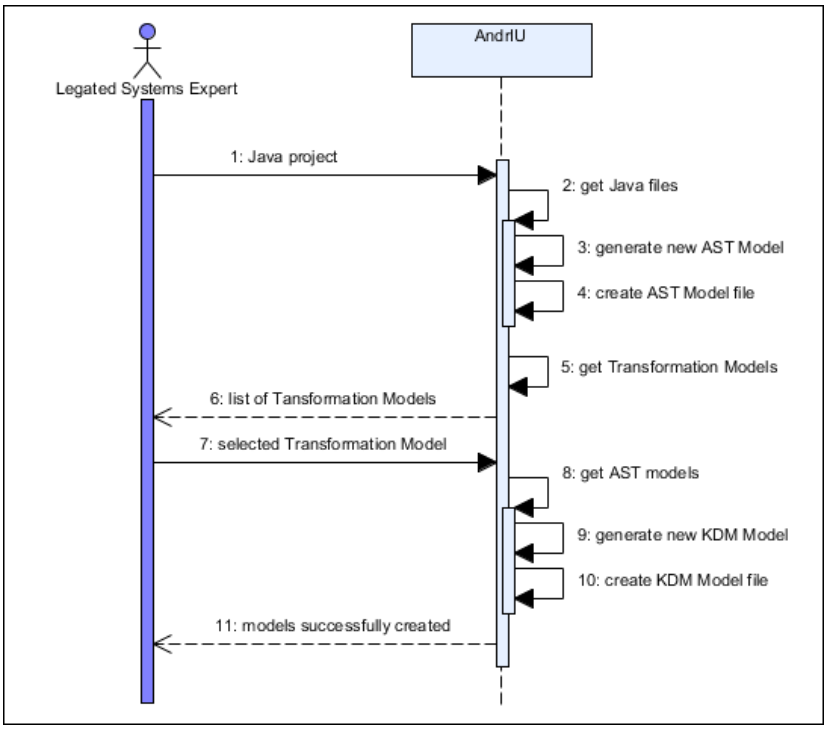


Figura 5.55. Diagrama de Secuencia de Análisis del CdU.8 (Escenario normal)

b) Escenario alternativo 1 (error en la generación del modelo AST)

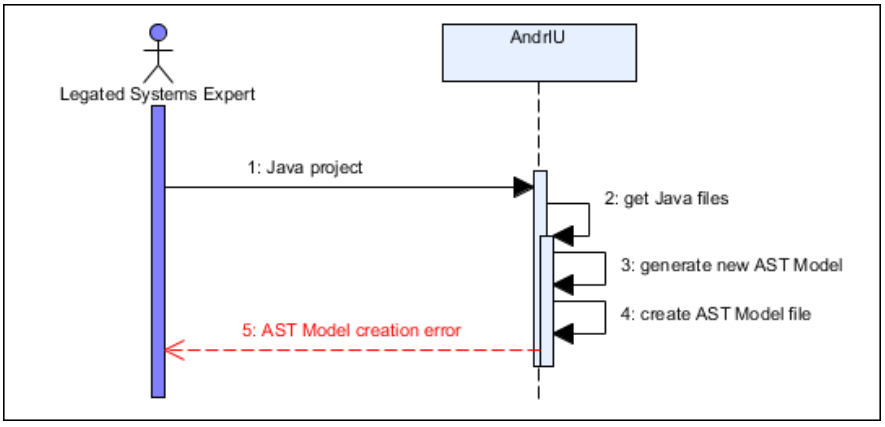


Figura 5.56. Diagrama de Secuencia de Análisis del CdU.8 (Escenario alternativo 1)

c) Escenario alternativo 2 (error en el listado de modelos de transformación)

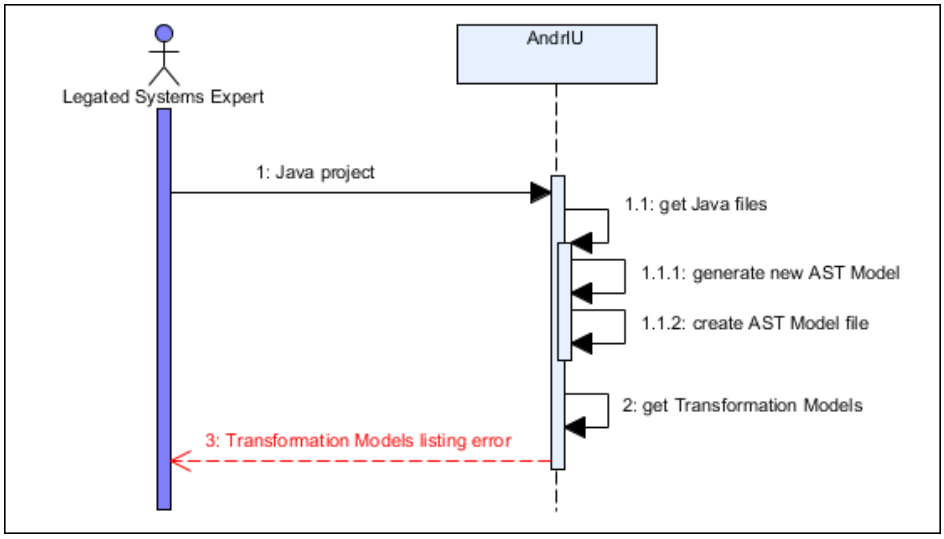


Figura 5.57. Diagrama de Secuencia de Análisis del CdU.8 (Escenario alternativo 2)

d) Escenario alternativo 3 (error en la generación de modelos KDM)

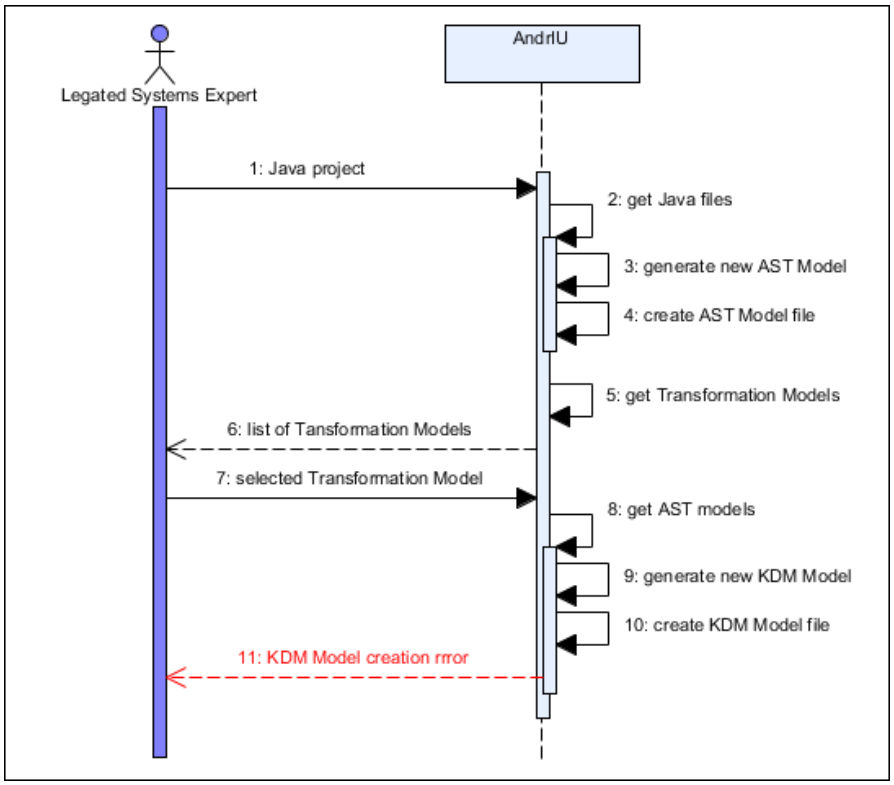


Figura 5.58. Diagrama de Secuencia de Análisis del CdU.8 (Escenario alternativo 3)

5.5.1.2. Diagramas de Clases de Análisis

Tras definir los diagramas de secuencia de análisis se procede a realizar los diagramas de clases de análisis al igual que en la iteración anterior. En las siguientes figuras se pueden observar dichos diagramas, que corresponden con el producto de salida PS.5.2.

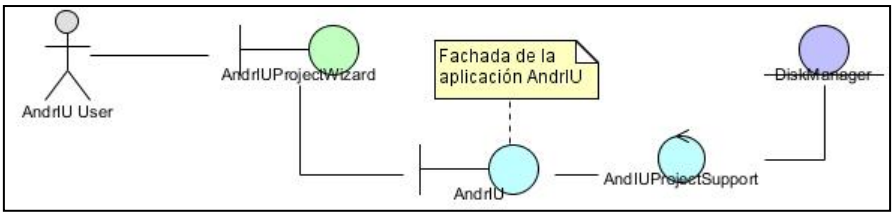


Figura 5.59. Diagrama de Clases de análisis de los CdU.1-CdU.2

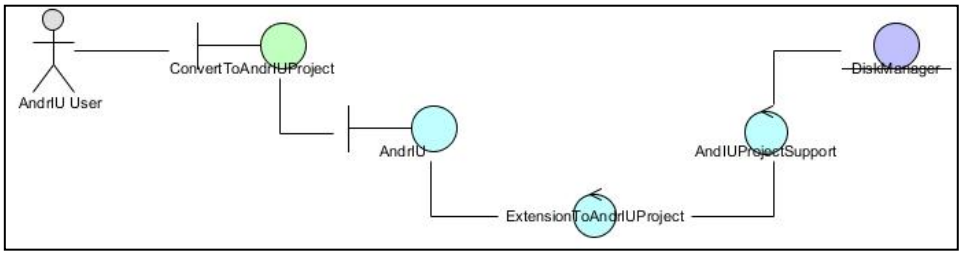


Figura 5.60. Diagrama de Clases de análisis del CdU.3

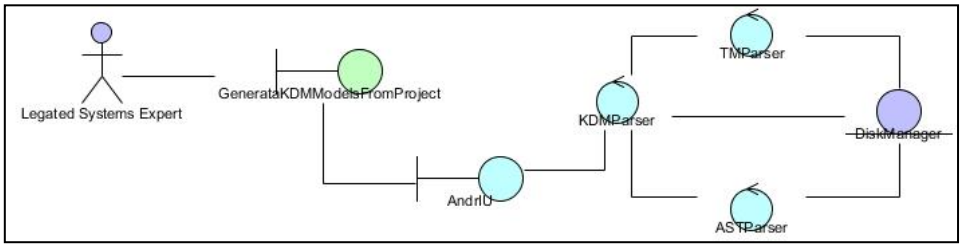


Figura 5.61. Diagrama de Clases de análisis del CdU.8

5.5.2. Diseño

En el flujo de trabajo de diseño de la iteración 5 se obtendrán los productos de salida siguientes:

PS.5.3. Diagramas de clases de diseño de CdU.1, CdU.2, CdU.3 y CdU.8.

PS.5.3. Diagramas de secuencia de diseño de CdU.1, CdU.2, CdU.3 y CdU.8.

PS.5.4. Prototipos de la GUI de CdU.1, CdU.2, CdU.3 y CdU.8.

5.5.2.1. Diagramas de Clases de Diseño

Una vez terminado el análisis de los casos de uso anteriormente mencionados, se procede con en diseño de las clases que implementarán estas funcionalidades. En la Figura 5.62 se muestra el diagrama de clases de la iteración completa, y en la Figura 5.63, Figura 5.64 y Figura 5.65 se presentan los diagramas de clases separandos según cada uno de los CdU de la iteración para facilitar su comprensión. Esto se corresponde con el producto de salida PS.5.3.

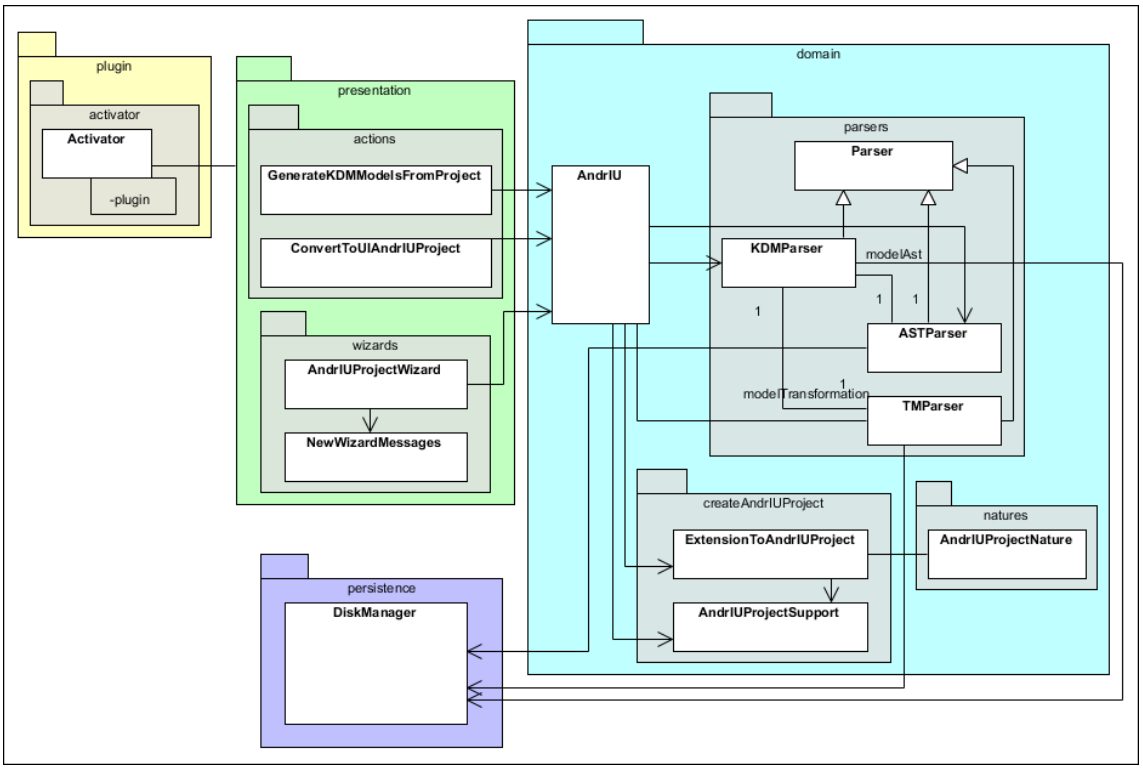


Figura 5.62. Diagrama de clases de la iteración 5

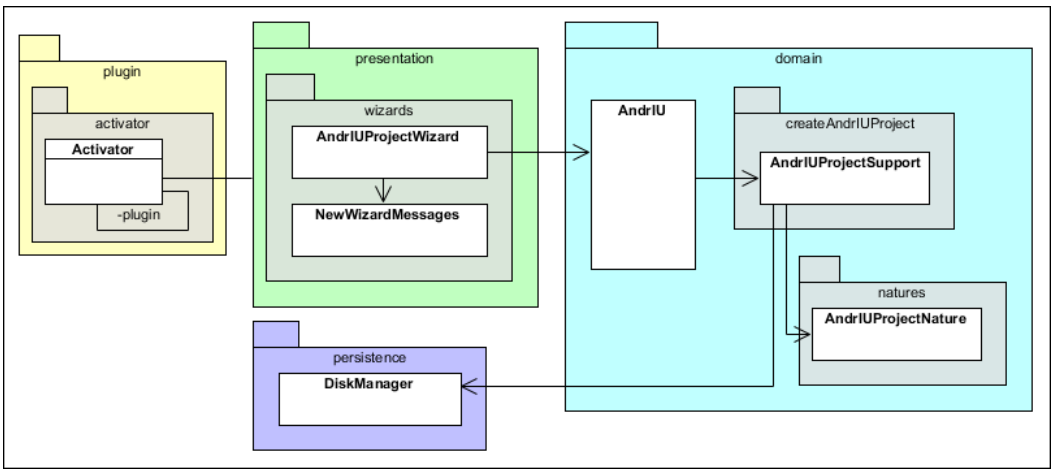


Figura 5.63. Diagrama de clases de CdU.1-CdU.2 (Crear nuevo proyecto AndrIU)

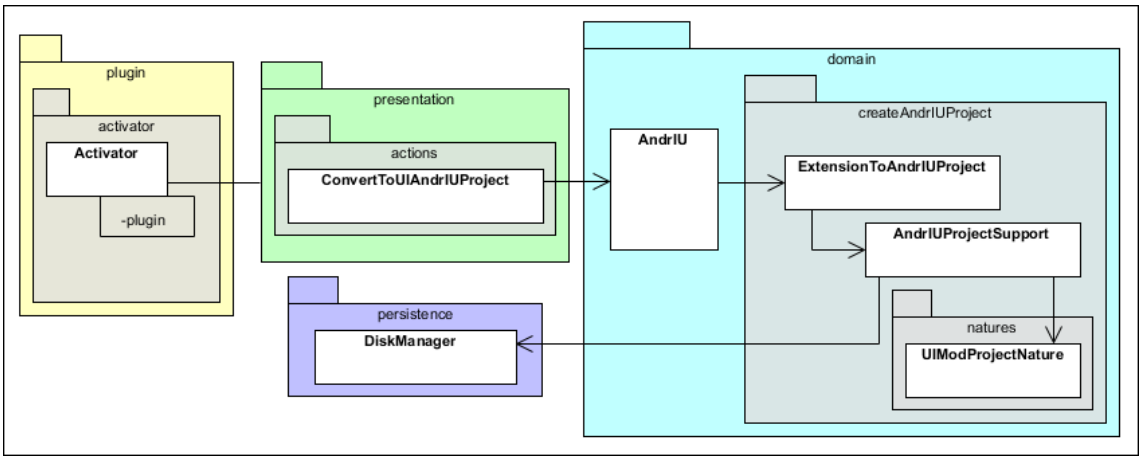


Figura 5.64. Diagrama de clases de CdU.1-CdU.3 (Crear proyecto AndrIU desde proyecto existente)

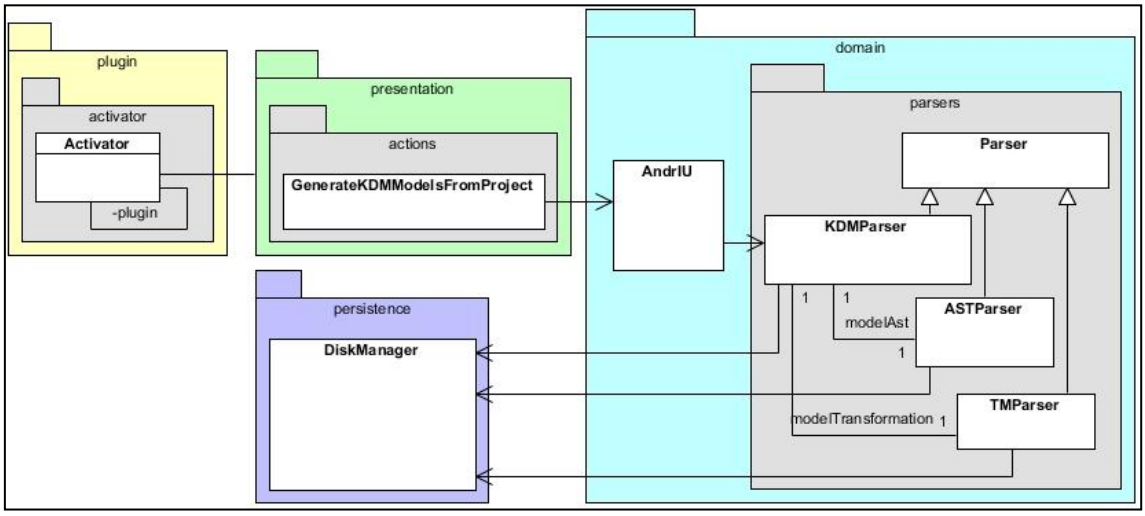


Figura 5.65. Diagrama de clases del caso de uso CdU.8 (Generar modelos KDM desde proyecto)

5.5.2.2. Diagramas de Secuencia de Diseño

Tras diseñar las clases, se realizan los diagramas de secuencia de diseño que proporcionan una visión más detallada del funcionamiento. En las siguientes se muestran los diagramas de secuencia correspondientes a los casos de uso CdU.1, CdU.2, CdU.3 y CdU.8, que se corresponde con el producto de salida PS.5.4.

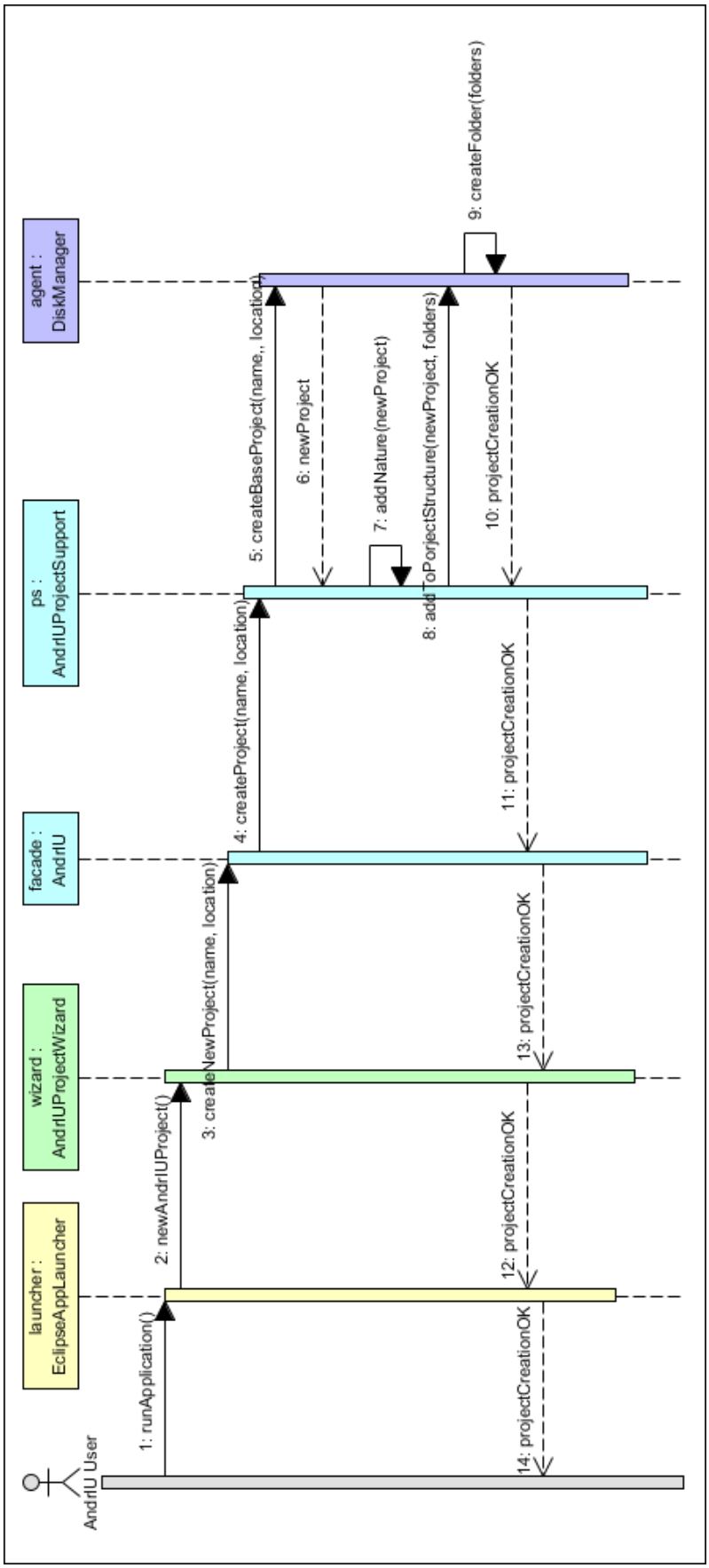


Figura 5.66. Diagrama de secuencia de CdU.1-CdU.2 (Crear nuevo proyecto AndrIU)

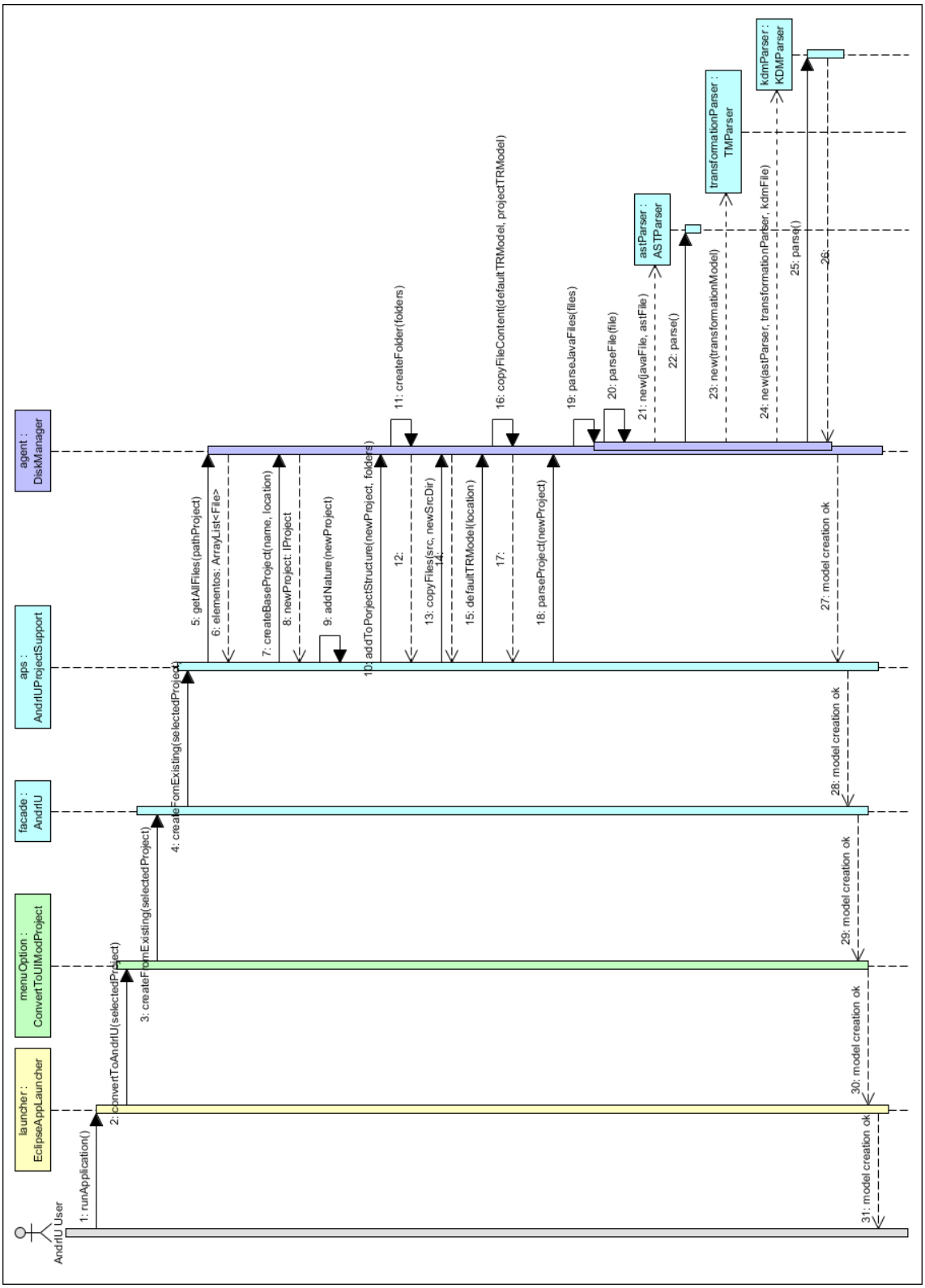


Figura 5.67. Diagrama de secuencia de CdU.3 (Crear proyecto AndrIU desde proyecto existente)

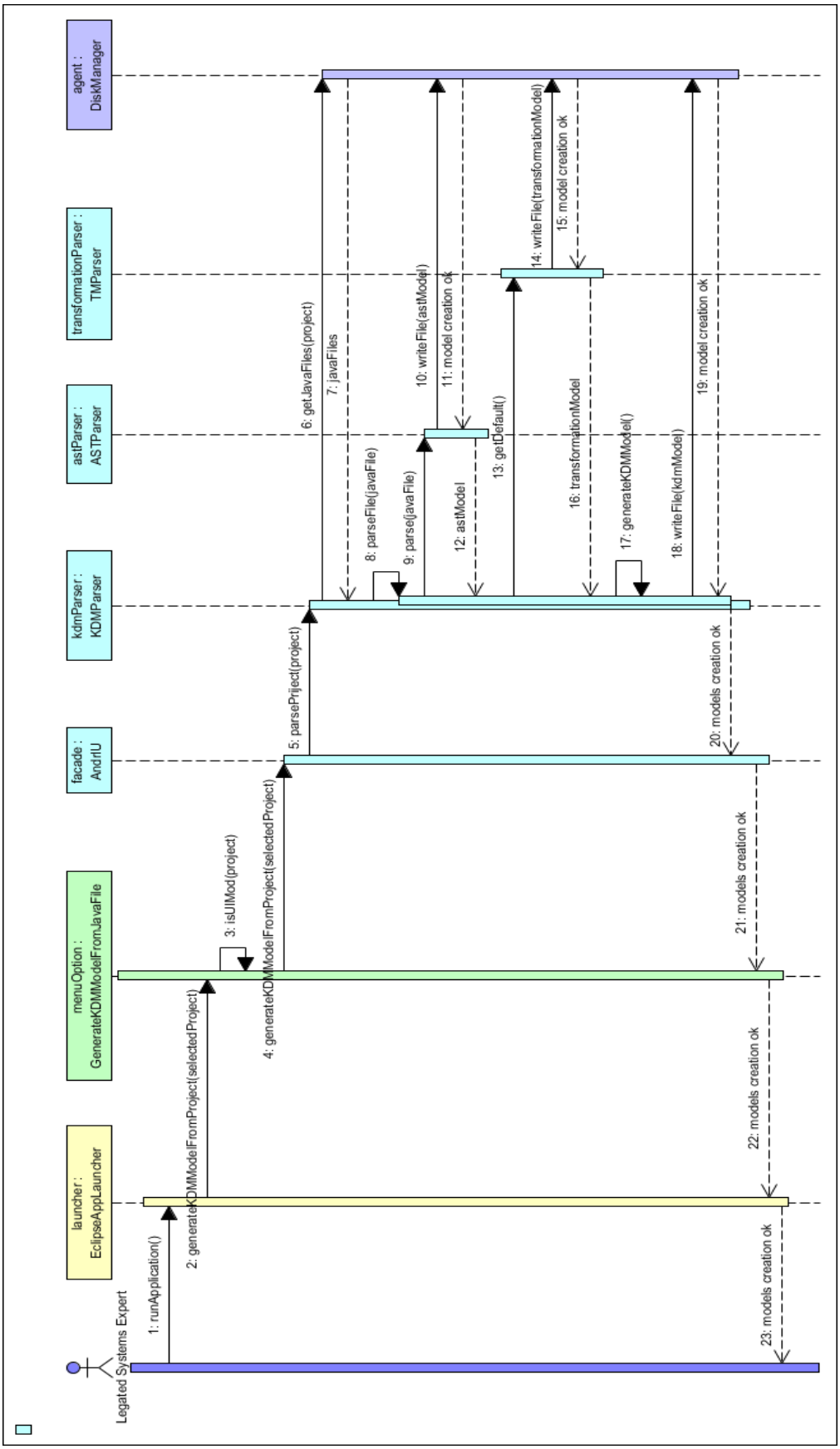


Figura 5.68. Diagrama de secuencia de CdU.8 (Generar modelos KDM desde proyecto)

5.5.2.3. Prototipos de la GUI

En esta iteración también se diseñan los prototipos de la GUI que presentará la herramienta que se está desarrollando. En primer lugar, se diseña el wizard de Eclipse para la creación de nuevos proyectos AndrIU (véase Figura 5.69).

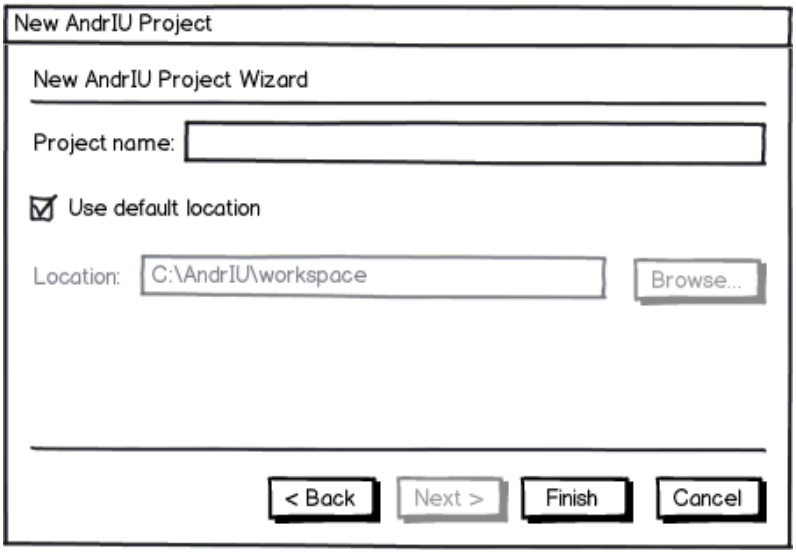


Figura 5.69. Prototipo del Wizard para la creación de nuevos proyectos AndrIU

A continuación se pueden observar los prototipos de los menús con las dos nuevas opciones: generación de modelos KDM desde un proyecto AndrIU y convertir un proyecto Java en un proyecto AndrIU (véase Figura 5.70 y Figura 5.71).

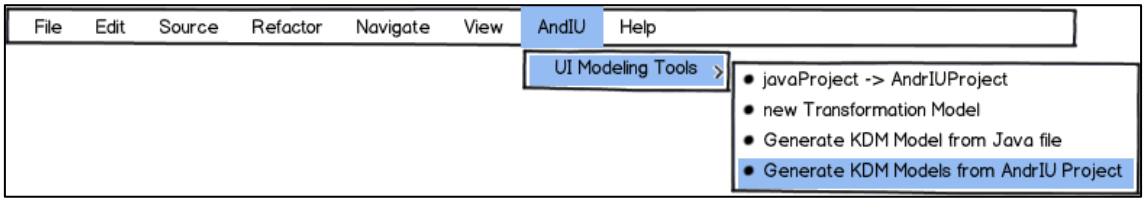


Figura 5.70. Prototipo del menú para la iteración 5

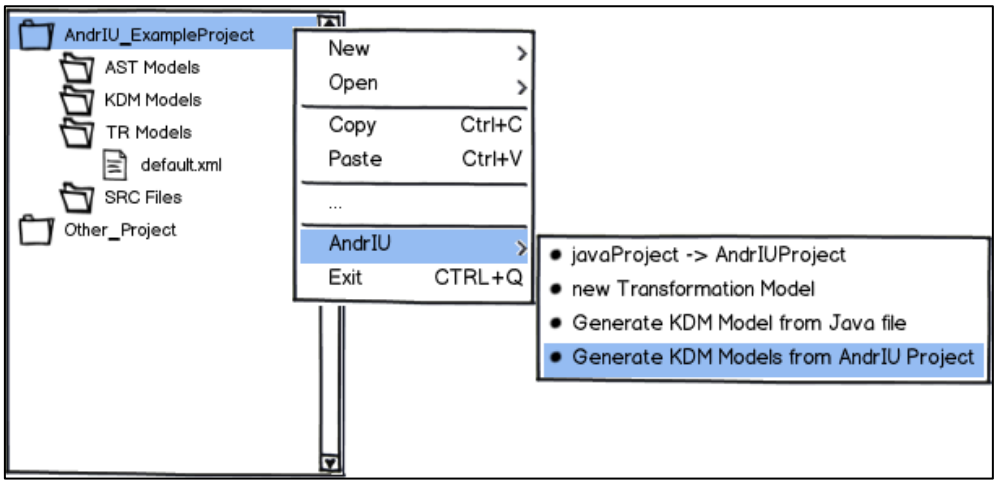


Figura 5.71. Prototipo del menú contextual para la iteración 5

5.5.3. Implementación

En esta iteración se comienzan a implementar los casos de uso CdU.6 y CdU.7, correspondiente a la generación de modelos KDM desde un fichero de código Java. Esta implementación se apoya fuertemente en la realizada en la iteración anterior, ya que para generar los modelos KDM desde un fichero Java es necesario generar primero su modelo AST y disponer de un modelo de transformación.

De este modo primero se toma el modelo AST y se genera el modelo de código en KDM. Después se recorren las clases y se comprueba su tipo. Si el tipo se corresponde con alguno de los elementos definidos en el *modelo de Transformación* se añade al modelo UI de KDM. Después, dentro de cada clase se recorren los atributos de la clase, los métodos y los constructores. De los atributos, se compara el tipo con los elementos definidos en el modelo de transformación, y del mismo modo se añaden al modelo KDM. Después se comprueban los métodos para buscar los manejadores de los eventos de los elementos ya definidos, generando así los elementos "*UIAction*" en el modelo KDM. A su vez se buscan acciones de navegabilidad entre ventanas, y añadir de este modo los "*UIFlow*" al modelo KDM, que definen la navegabilidad entre dos ventanas. En la Figura 5.72 se puede observar un fragmento de un modelo KDM en el editor de AndrIU.

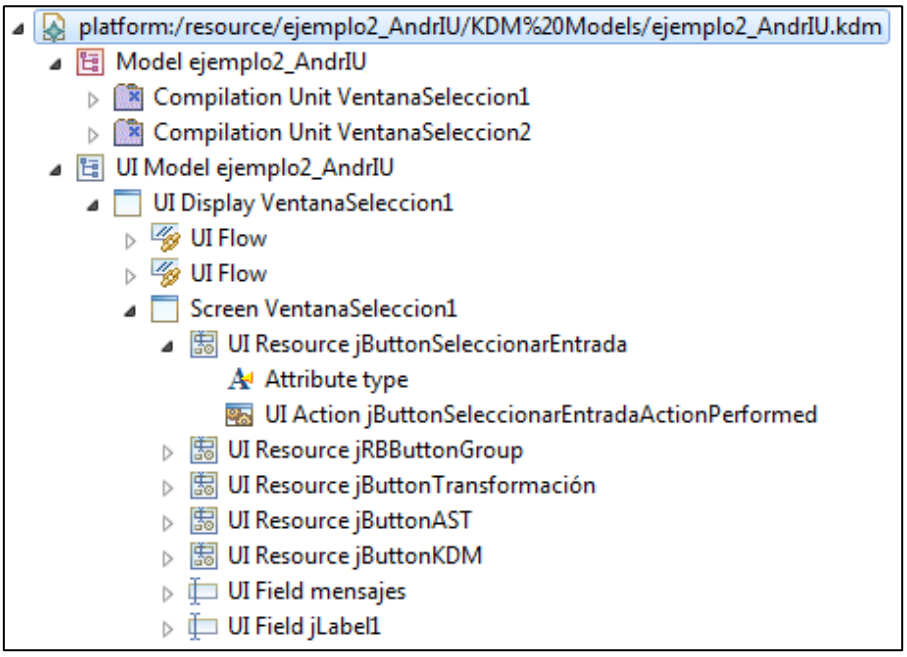


Figura 5.72. Fragmento de un modelo KDM

Para la generación de estos modelos KDM, se utiliza la clase *KDMParser* (véase Figura 5.74), que utiliza a su vez la clases *ASTParser* y *TMParser*, implementadas en la iteración anterior. A la clase *DiskManager* también se la han añadido nuevos métodos para gestionar la persistencia de esta nueva funcionalidad (véase Figura 5.73). Del mismo modo que antes, la clase *GenerateKDMModelFromJavaFile* funciona como manejador de eventos de la capa de presentación para las opciones de los menús de la herramienta (véase Figura 5.75).

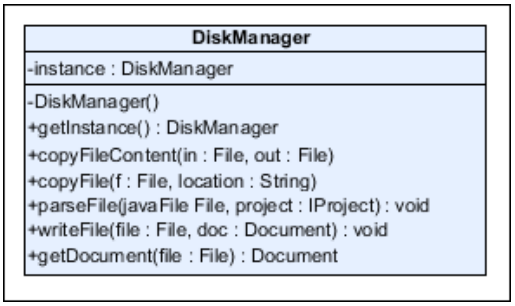


Figura 5.73. Clase DiskManager (II)

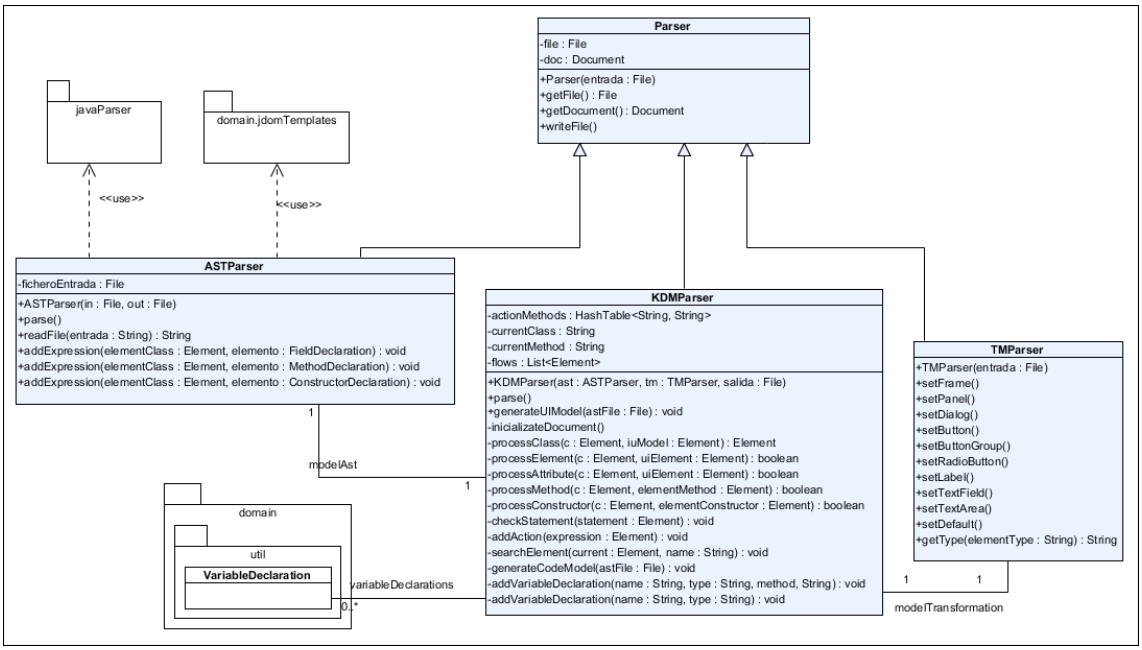


Figura 5.74. Clase KDMParser (I)

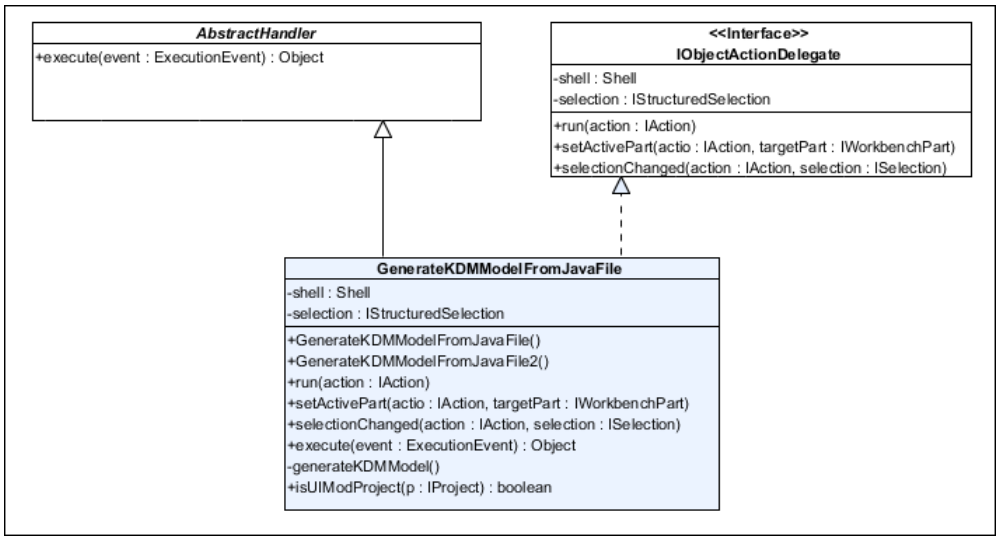


Figura 5.75. Clase GenerateKDMModelFromJavaFile

5.6. Iteración 6

En esta sexta iteración se comienza el análisis y diseño de los casos de uso CdU.9, CdU.10 y CdU.11, referentes a la generación de los modelos de interfaz de usuario Android. De forma paralela se comienza con la implementación de los casos de uso diseñados en la iteración anterior, y que permitirán la creación de proyectos AndrIU y la generación de modelos KDM a partir de estos.

5.6.1. Análisis

En este flujo de trabajo se obtendrán los productos de salida siguientes:

PS.5.1. Diagramas de secuencia de análisis de CdU.9, CdU.10 y CdU.11

PS.5.2. Diagramas de clases de análisis de CdU.9, CdU.10 y CdU.11

5.6.1.1. Diagramas de Secuencia de Análisis

Se van a emplear diagramas de secuencia de análisis para comprender en mayor medida el sistema. En las figuras siguientes se presentan los diagramas de secuencia de análisis correspondientes a los casos de uso CdU.9, CdU.10 y CdU.11 que corresponden con el producto de salida PS.6.1.

5.6.1.1.1. CdU.9-CdU.10. Generate Android UI Model from KDM Model

a) Escenario principal

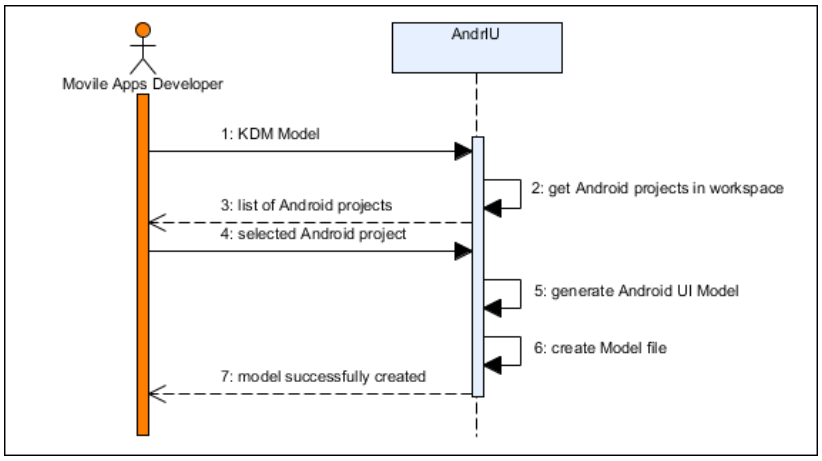


Figura 5.76. Diagrama de Secuencia de Análisis de los CdU.9-CdU.10 (Escenario normal)

b) Escenario alternativo 1 (error en el listado de proyectos Android)

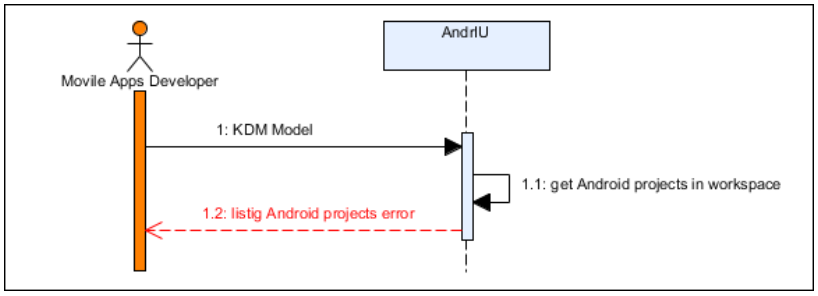


Figura 5.77. Diagrama de Secuencia de Análisis de los CdU.9-CdU.10 (Escenario alternativo 1)

c) Escenario alternativo 2 (error en la generación de GUI para Android)

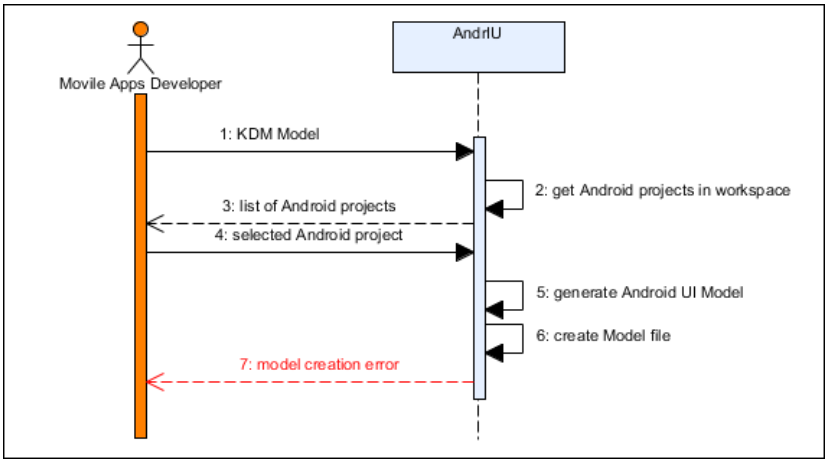


Figura 5.78. Diagrama de Secuencia de Análisis de los CdU.9-CdU.10 (Escenario alternativo 2)

5.6.1.1.2. CdU.11. Generate Android UI Model from AndrIU project

a) Escenario principal

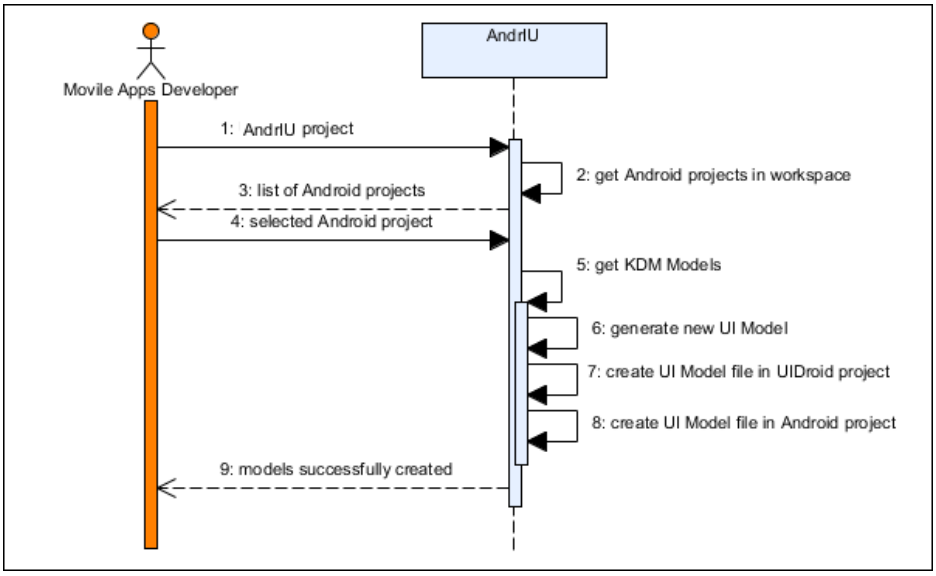


Figura 5.79. Diagrama de Secuencia de Análisis del CdU.11 (Escenario normal)

b) Escenario alternativo 1 (error en el listado de proyectos Android)

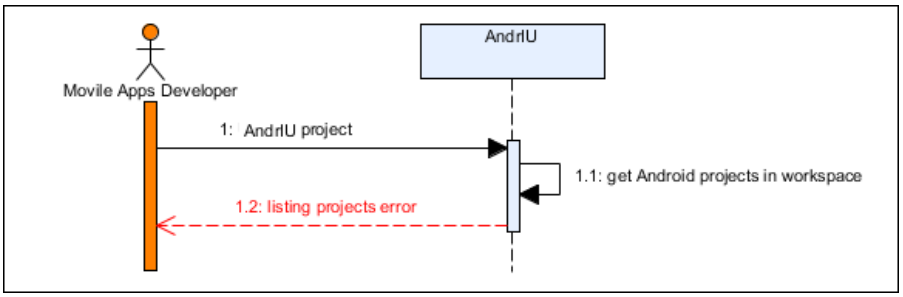


Figura 5.80. Diagrama de Secuencia de Análisis del CdU.11 (Escenario alternativo 1)

c) Escenario alternativo 2 (error en la generación de GUI para Android)

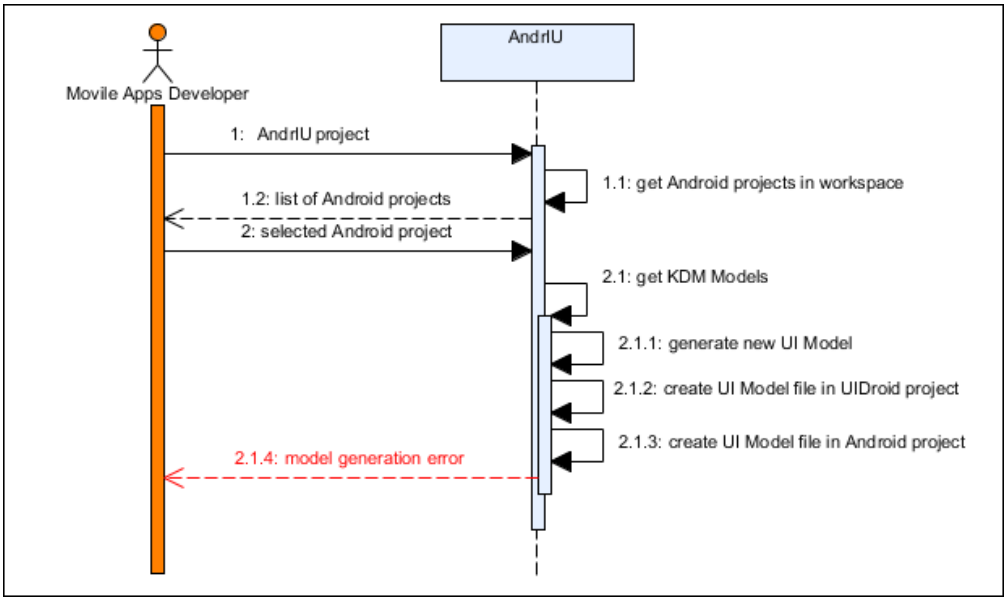


Figura 5.81. Diagrama de Secuencia de Análisis del CdU.11 (Escenario alternativo 2)

5.6.1.2. Diagramas de Clases de Análisis

Tras definir los diagramas de secuencia de análisis se procede a realizar los diagramas de clases de análisis al igual que en la iteración anterior. En la Figura 5.82 y Figura 5.83 se pueden observar dichos diagramas, que corresponden con el producto de salida PS.62.

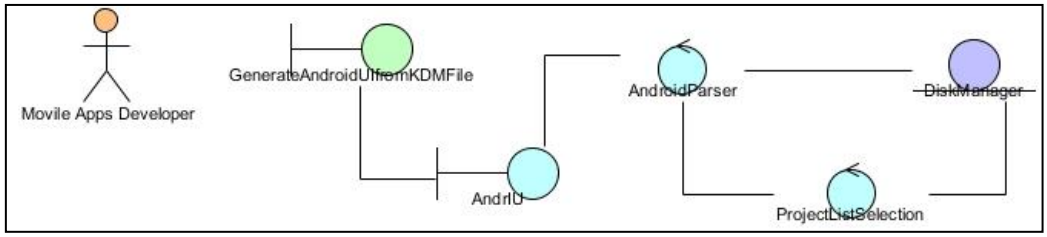


Figura 5.82. Diagrama de Clases de análisis de los CdU.9-CdU.10

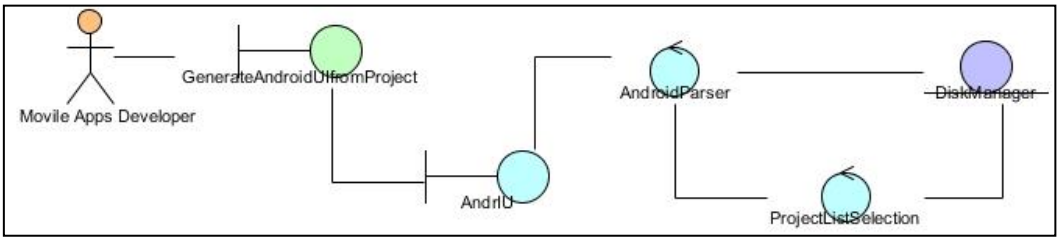


Figura 5.83. Diagrama de Clases de análisis del CdU.11

5.6.2. Diseño

En este flujo de trabajo se obtendrán los productos de salida siguientes:

PS.5.3. Diagramas de clases de diseño de los casos de uso CdU.9, CdU.10 y CdU.11
PS.5.3. Diagramas de secuencia de diseño de los casos de uso CdU.9, CdU.10 y CdU.11
PS.5.4. Prototipos de la GUI de los casos de uso CdU.9, CdU.10 y CdU.11

5.6.2.1. Diagramas de Clases de Diseño

Una vez terminado el análisis de los casos de CdU.9, CdU.10 y CdU.11, se realiza el diseño de las clases que implementarán estos casos de uso. En la Figura 5.84 y en la Figura 5.85 se muestran los diagramas de clases y en las posteriores se detallan las clases más importantes. Esto se corresponde con el producto de salida PS.5.3.

A continuación se presentan los diagramas de clases separandos según cada uno de los casos de uso de la iteración para facilitar su comprensión.

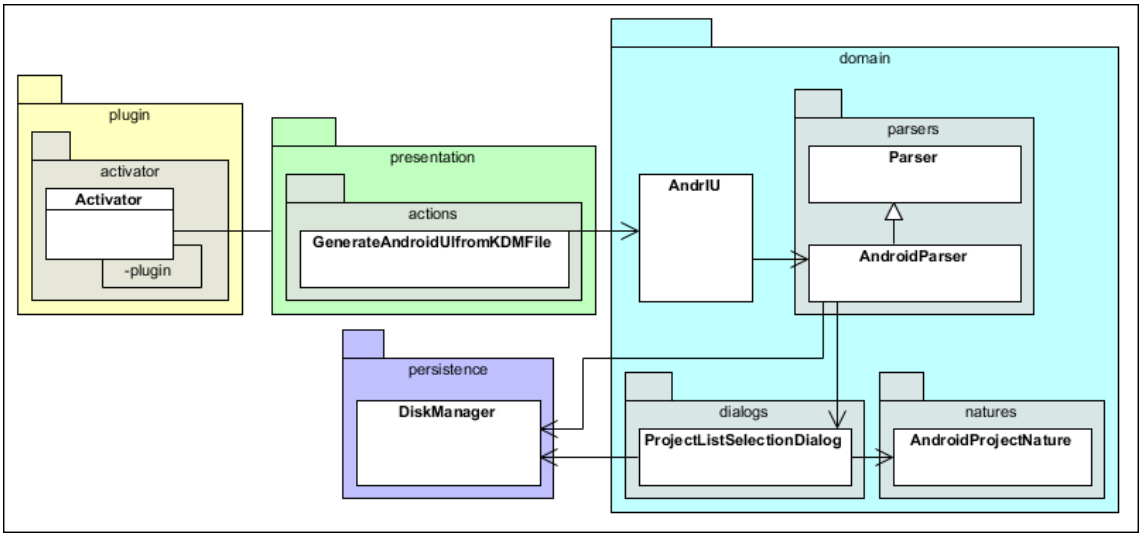


Figura 5.84. Diagrama de clases de los casos de uso CdU.9-CdU.10

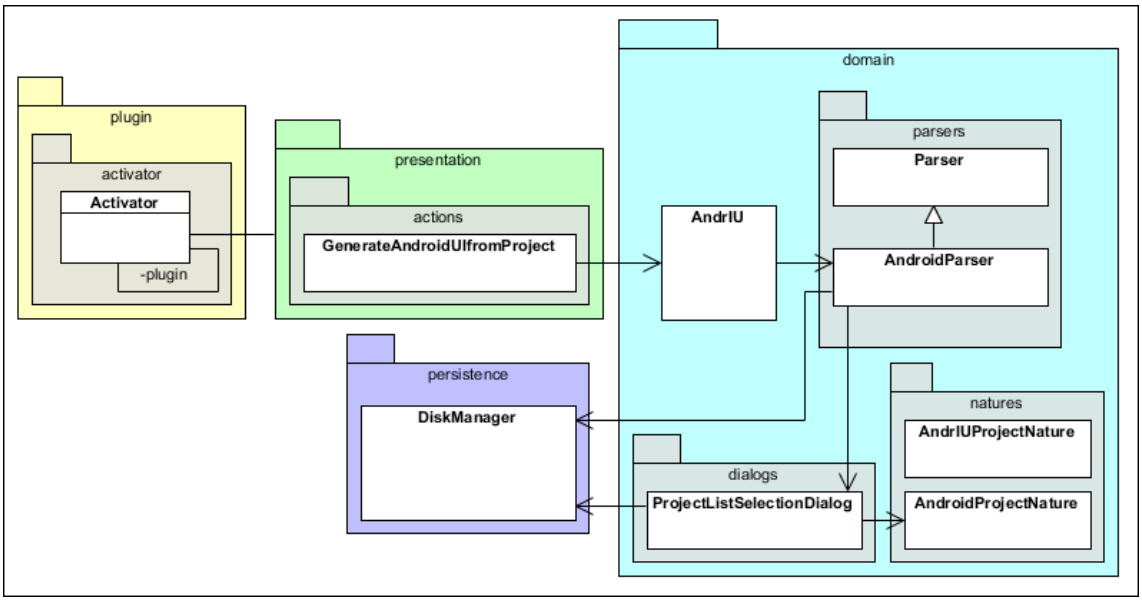


Figura 5.85. Diagrama de clases del caso de uso CdU.11

5.6.2.2. Diagramas de Secuencia de Diseño

Tras realizar el diseño de las clases, se realizan los diagramas de secuencia de diseño para proporcionar una visión más detallada del funcionamiento. En las siguientes se muestran los diagramas de secuencia correspondientes a los casos de uso CdU.9, CdU.10 y CdU.11, que se corresponde con el producto de salida PS.6.4.

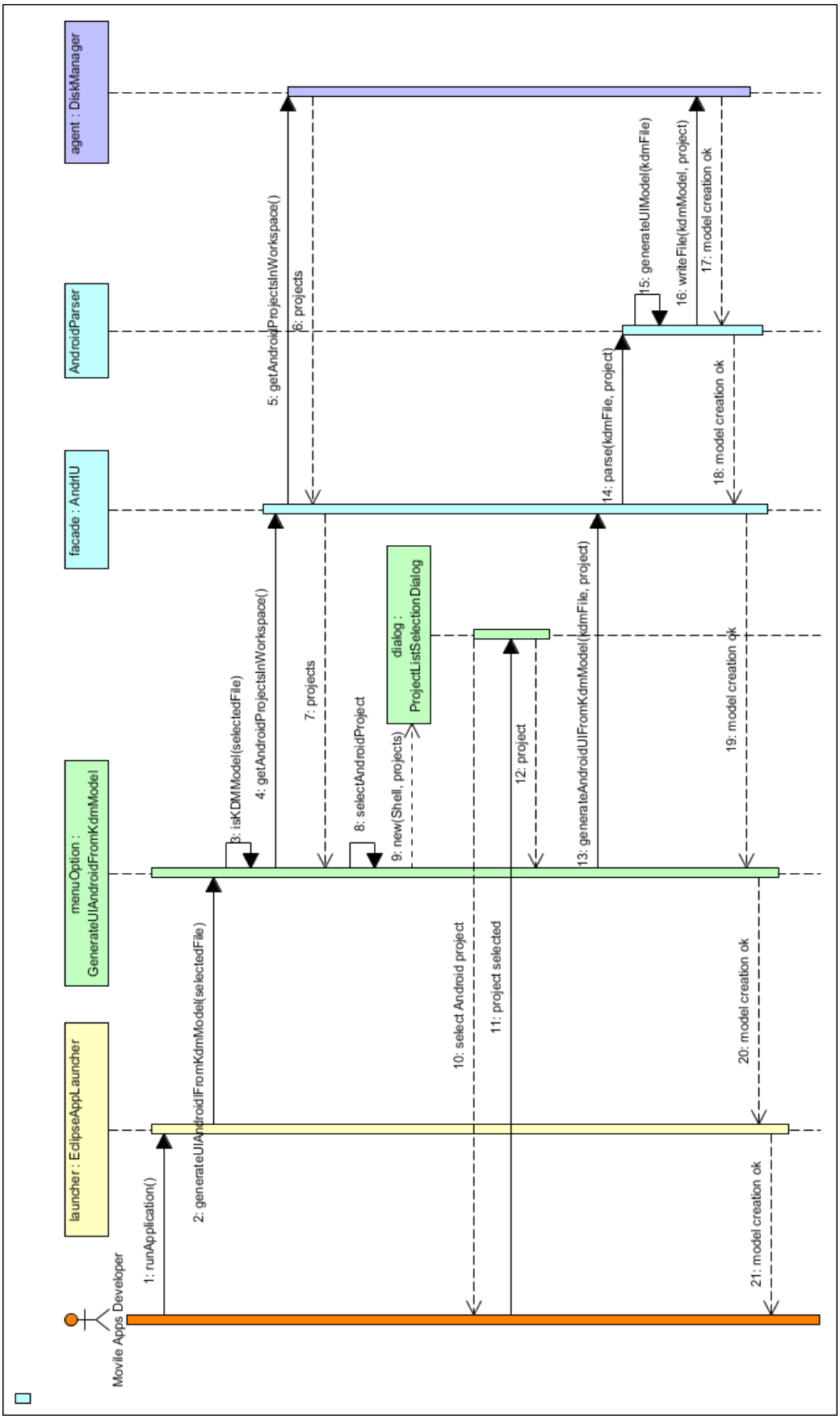


Figura 5.86. Diagrama de secuencia de los casos de uso CdU9-CdU10

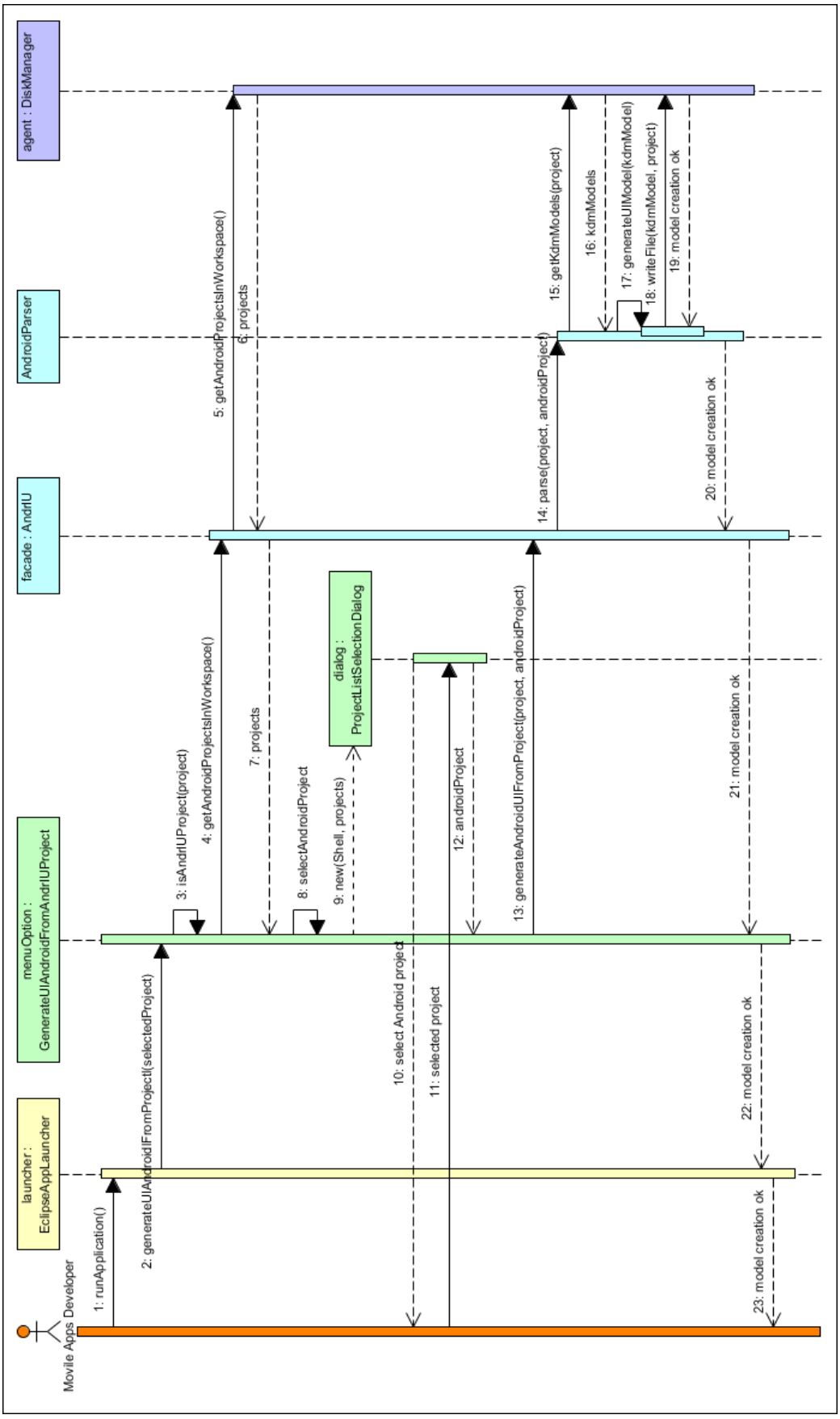


Figura 5.87. Diagrama de secuencia des caso de uso CdU11

5.6.2.3. Prototipos de la GUI

Del mismo modo que se ha hecho en al iteración anterior, en este apartado se diseñan los prototipos de la GUI que presentará la herramienta que se está desarrollando. En primer lugar, en la Figura 5.88 y en la Figura 5.89, se pueden observar los prototipos de los menús, con las nuevas opciones, y en la Figura 5.90 la ventana de selección de proyecto, necesaria para seleccionar el proyecto donde se almacenarán los modelos que se generan.

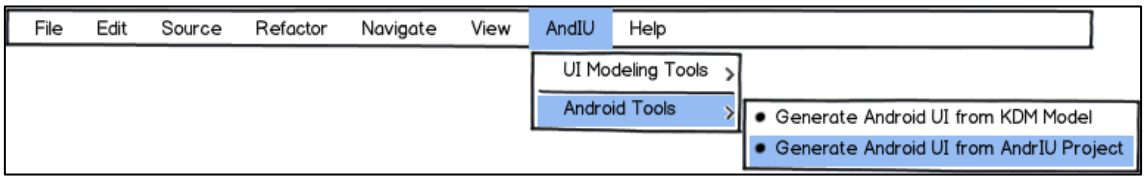


Figura 5.88. Prototipo del menú para la iteración 6

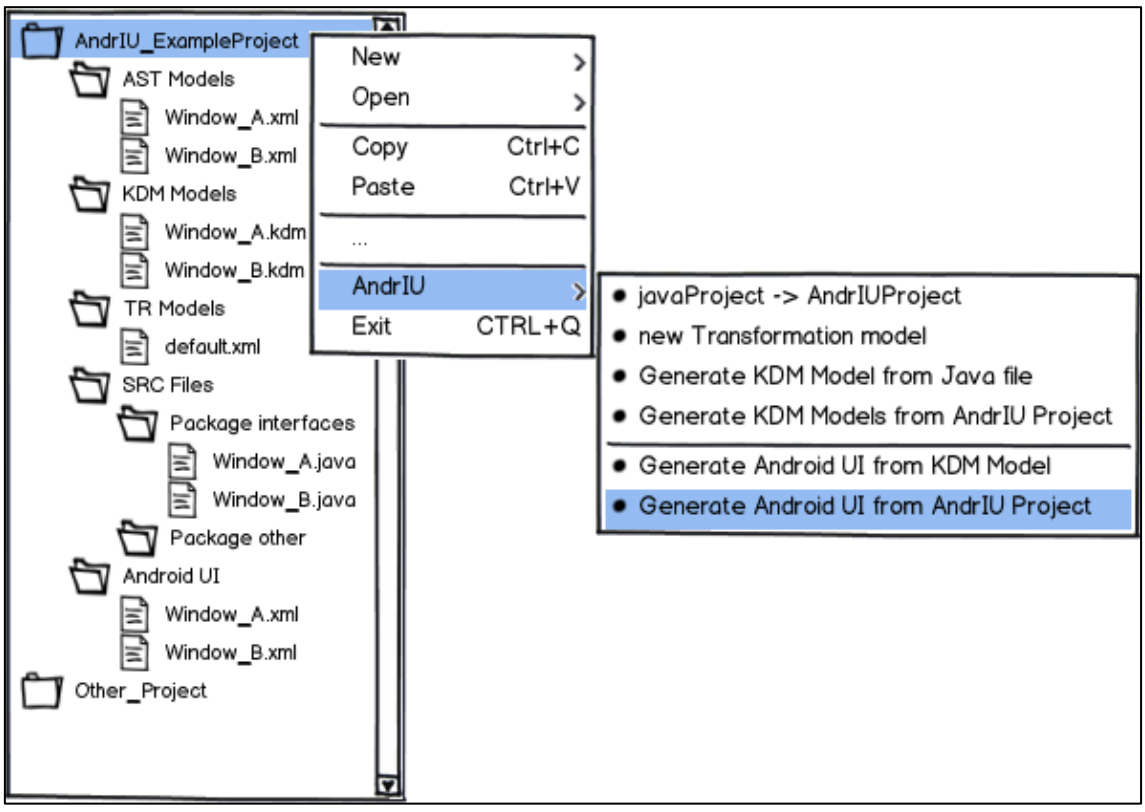


Figura 5.89. Prototipo del menú contextual para la iteración 6

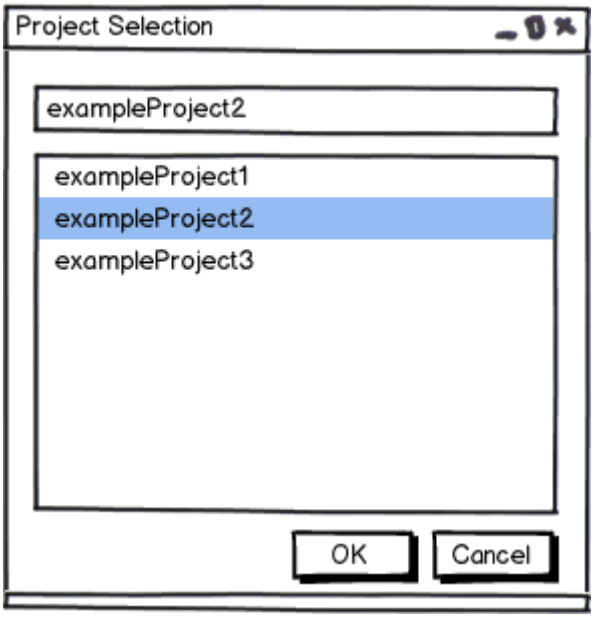


Figura 5.90. Prototipo de la ventana de selección de proyecto

5.6.3. Implementación

En esta iteración se comienzan a implementar los CdU diseñados en la iteración anterior, correspondientes a las funcionalidades de la creación de proyectos AndrIU (tanto proyectos nuevos desde un Wizard como desde un proyecto Java existente) y de la generación de modelos KDM desde un proyecto AndrIU.

5.6.3.1. Implementación de los CdU.1-CdU.2

Para la implementación de estos casos de uso, se ha utilizado la clase *AndrIUProjectSupport* (véase Figura 5.91), que se encarga de generar los proyectos AndrIU nuevos, es decir, desde cero. Para ello, se ha creado una extensión del plug-in denominada *Wizard* (véase Figura 5.92), que añade un nuevo tipo de proyecto al menú de Eclipse. El *Wizard* está implementado por la clase *AndrIUProjectWizard*, que utiliza la clase *NewWizardMessages* para definir las constantes necesarias (véase Figura 5.93).

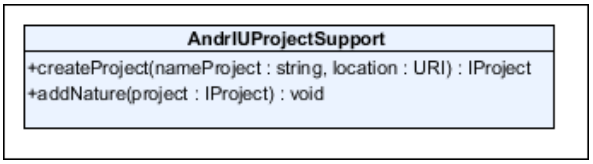


Figura 5.91. Clase AndrIUProjectSupport

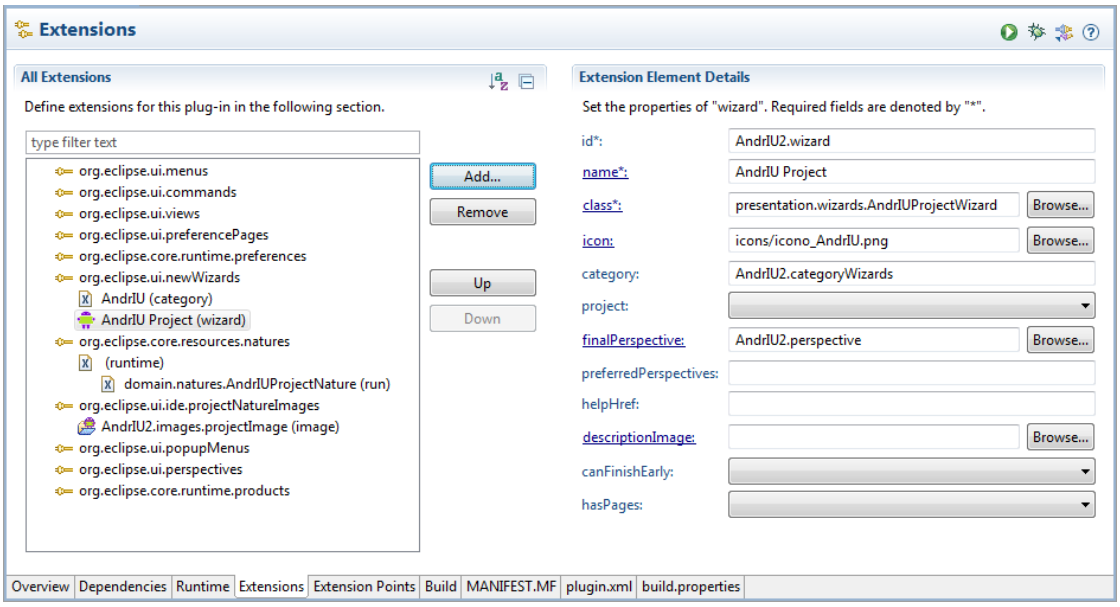


Figura 5.92. Extensiones del plug-in para definir el Wizard y la nature

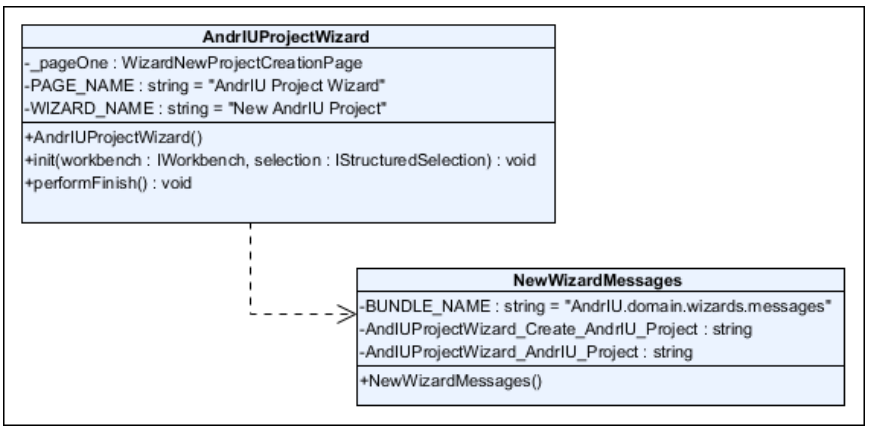


Figura 5.93. Clases AndrIUProjectWizard y NewWizardMessages

Para crear un nuevo tipo de proyecto en Eclipse, también es necesario definir en el plug-in una *nature*[2], otra extensión de Eclipse, que sirve para identificar la naturaleza del proyecto (véase Figura 5.92). Esto se implementa a través de la clase *AndrIUProjectNature* (véase Figura 5.94).

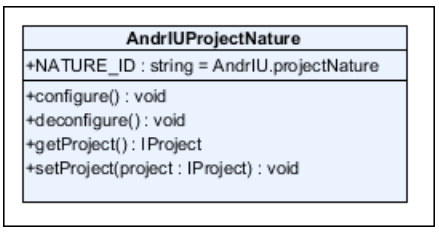


Figura 5.94. Clase AndrIUProjectNature

[2] Las *natures* son unas propiedades que tiene los proyectos en Eclipse que indican la naturaleza de los mismos para atribuirles propiedades o utilidades características.

En esta iteración también se añadieron métodos a la clase *DiskManager* (véase Figura 5.95), para crear el sistema de directorios que componen el proyecto AndrIU.

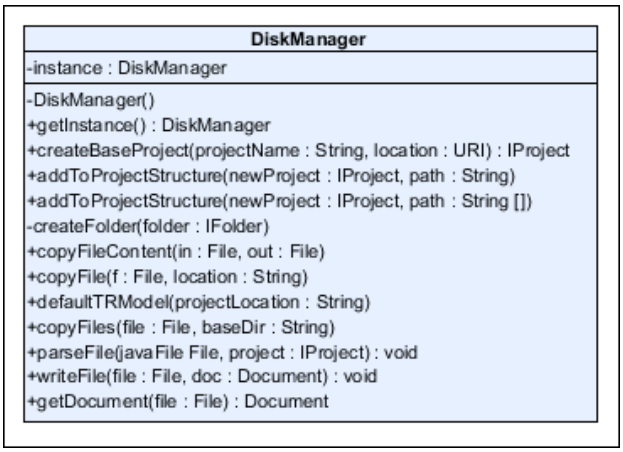


Figura 5.95. Clase DiskManager (III)

5.6.3.2. Implementación del CdU.3

Para la implementación de este caso de uso, consistente en la conversión de un proyecto Java existente en un proyecto AndrIU, se han utilizado las clases *AndrIUProjectSupport* y *AndrIUProjectNature* implementadas anteriormente, así como la nueva clase *ExtensionToAndrIUProject* (véase Figura 5.96).

La conversión consiste en la creación de un nuevo AndrIU Project, con el nombre del proyecto Java que lo genera seguido del sufijo "_AndrIU". De este modo, a partir de un proyecto Java llamado "example" se generará un nuevo proyecto AndrIU llamado "example_AndrIU".

Una vez creado el nuevo proyecto, con la *nature* que se ha definido anteriormente, se crean la estructura de directorios de los proyectos AndrIU. Estos directorios son: "AST Models", "KDM Models", "TR Models" y "SRC Files". Inicialmente los dos primeros directorios están vacíos (sirven para contener los modelos AST y KDM que se creen en el futuro). En el directorio "TR Models" se creará un modelo de transformación por defecto. Y por último, en el directorio "SRC Files" se realizará una copia de todos los paquetes y ficheros de código fuente del proyecto Java de origen. Estos ficheros servirán para generar los distintos modelos AST, KDM, etc.

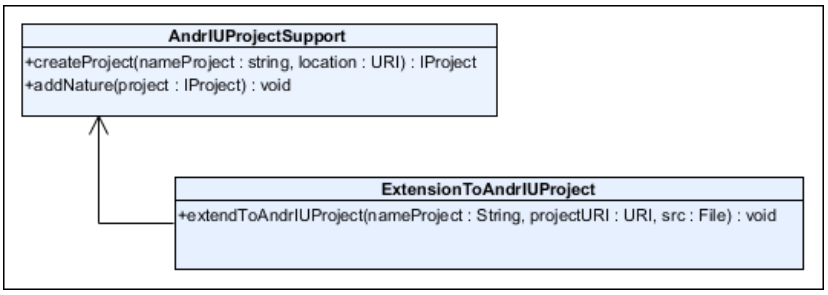


Figura 5.96. Clase ExtensionToAndrIUProject

Ya que la opción mencionada se realiza desde los menús de la herramienta, fue necesario incluir las nuevas extensiones en los menús correspondientes, así como la implementación del manejador de eventos *ConvertToAndrIUProject* (véase Figura 5.97).

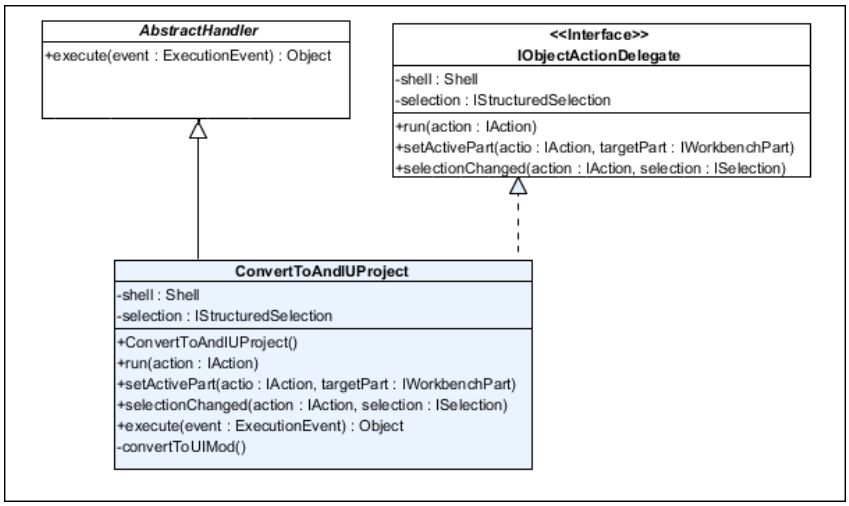


Figura 5.97. Clase ConvertToAndrIUProject

5.6.3.3. Implementación del CdU.8

Una vez se tienen los proyectos AndrIU, se implementa la función de generación de los modelos KDM desde dicho proyecto. Para ello, se toman como fuente los ficheros java que se encuentran en el directorio "SRC Files" del proyecto AndrIU, se toma el fichero de transformación generado por defecto del directorio "TR Models" y se generan en primer lugar los modelos AST en el directorio "AST Models" y a continuación, tomando éstos se generan los modelos KDM en el directorio "KDM Models". La generación de estos modelos se realiza de la misma manera que en el CdU.7, salvo que ahora se tiene un conjunto de ficheros Java.

Para la implementación de esta funcionalidad se han añadido algunas funciones nuevas a la clase *KDMParser* (véase Figura 5.98), así como a *DiskManager* (véase Figura 5.100). El manejador de eventos de esta nueva opción de los menús de la herramienta es la clase *GenerateKDMModelsFromProject* (véase Figura 5.99).

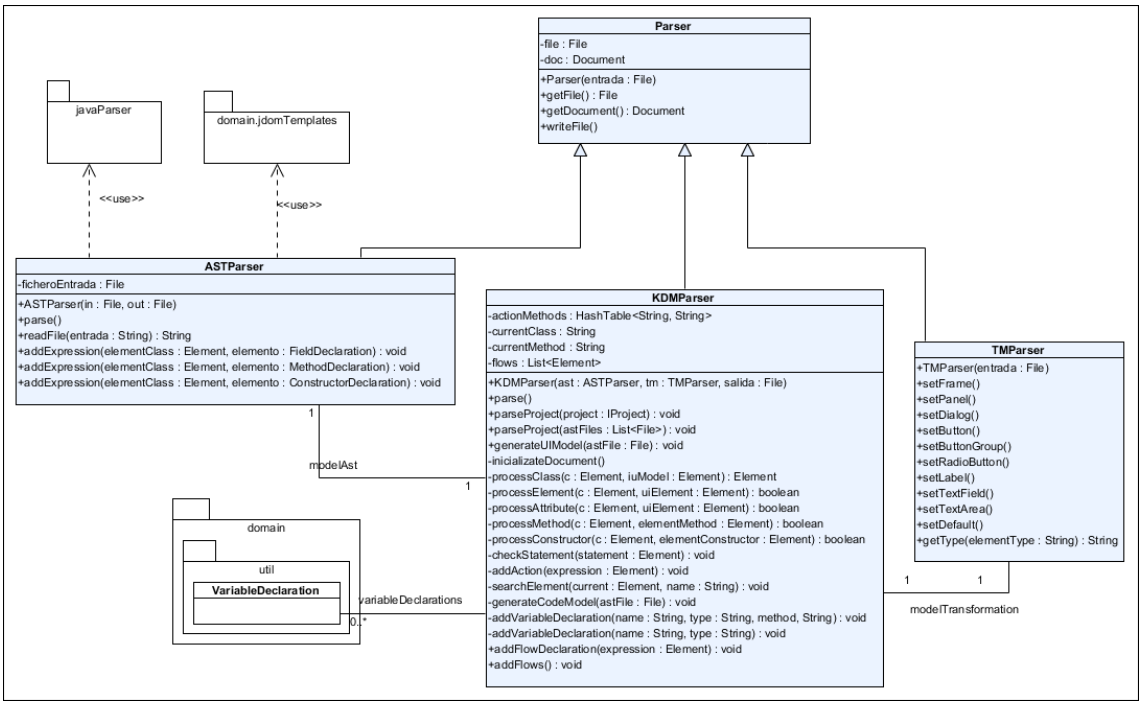


Figura 5.98. Clase KDMParser (II)

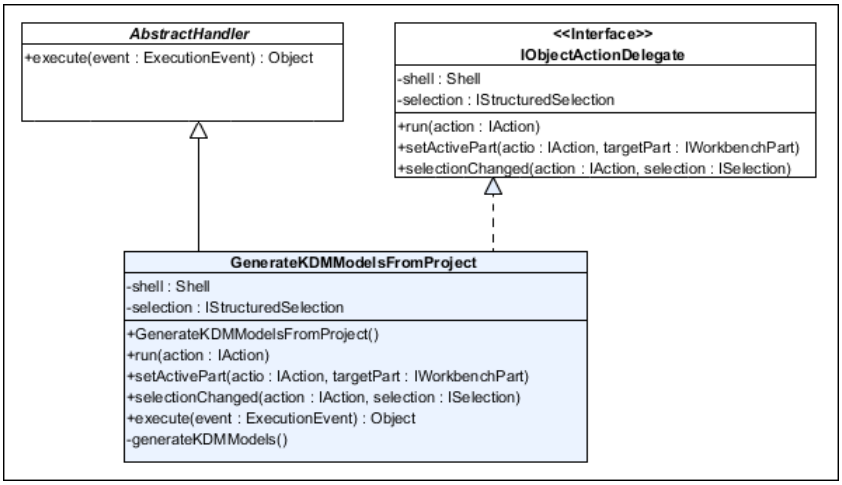


Figura 5.99. Clase GenerateKDMModelsFromProject

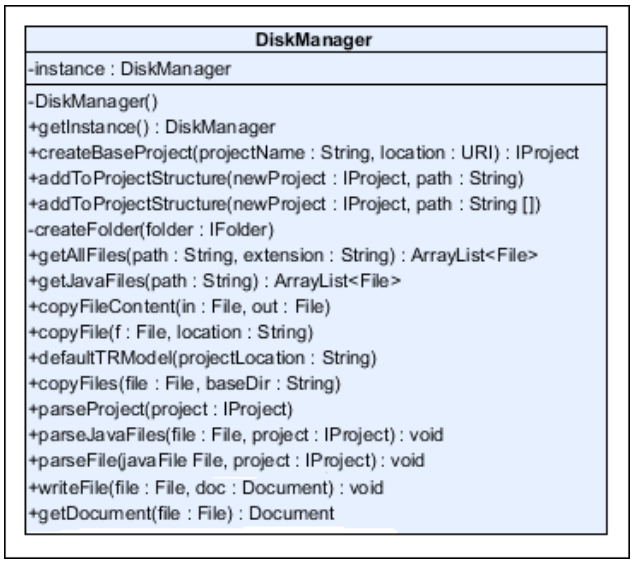


Figura 5.100. Clase DiskManager (IV)

5.7. Iteración 7

En la séptima iteración se realizó la implementación de los casos de uso CdU.9, CdU.10 y CdU.11 (diseñados en la iteración anterior) referentes a la generación de GUI para sistemas Android a partir de los modelos KDM que se han generado con la herramienta AndrIU. Además se comienza a realizar las pruebas software con los casos de uso CdU.4, CdU.5, CdU.6 y CdU.7. En este apartado se realizarán pruebas sobre las clases principales de dominio así como de la capa de persistencia.

5.7.1. Implementación

5.7.1.1. Implementación de los CdU.9-CdU.10

Para la implementación de los casos de uso referentes a la generación de GUI para Android a partir de un modelo KDM se ha seguido el mismo principio que en anteriores iteraciones: para la capa de presentación se ha desarrollado la clase *GenerateAndroidUIFromKDMFile* que hace de manejador de eventos para las opciones de los menús de la herramienta (véase Figura 5.101), para la capa de dominio se tiene la clase *AndroidParser* (véase Figura 5.102) que como su propio nombre indica consiste en un parser, y para la capa de persistencia se han añadido nuevos métodos a la clase *DiskManager*.

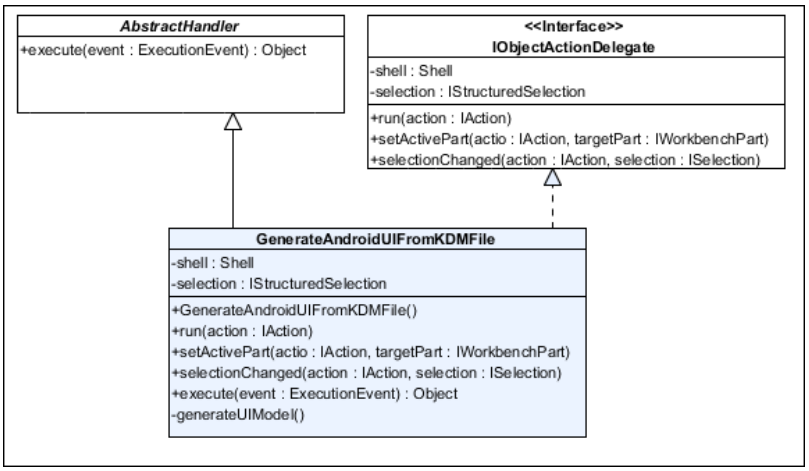


Figura 5.101. Clase GenerateAndroidUIFromKDMFile

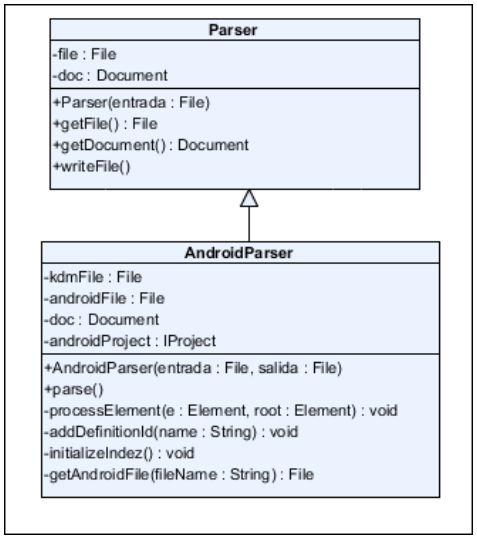


Figura 5.102. Clase AndroidParser

La pieza más importante de esta implementación es el parser que genera las GUI a partir de un modelo KDM. Para ello, el parser parte del "*UIModel*" del modelo KDM y toma todos los elementos "*UIDisplay*" que éste contenga. Por cada "*UIDisplay*" se genera un fichero XML con el mismo nombre que representará a un *Layout* de Android (elemento de visualización). Los elementos contenidos en el "*UIDisplay*" (*UIResources*, *UIFields*, etc) se transforman en elementos como botones, cuadros de texto, etc, en los *Layouts*.

Estos *Layouts* se almacenan en un nuevo directorio llamado "Android UI" en el propio proyecto AndrIU donde está el modelo KDM que se toma para realizar la generación.

5.7.1.2. Implementación del CdU.11

La implementación de este CdU se apoya en la clase *AndroidParser* para la generación las GUI, y únicamente es necesario implementar otro manejador para esta nueva opción. Esto se hace mediante la clase *GenerateAndroidUI* (véase Figura 5.103).

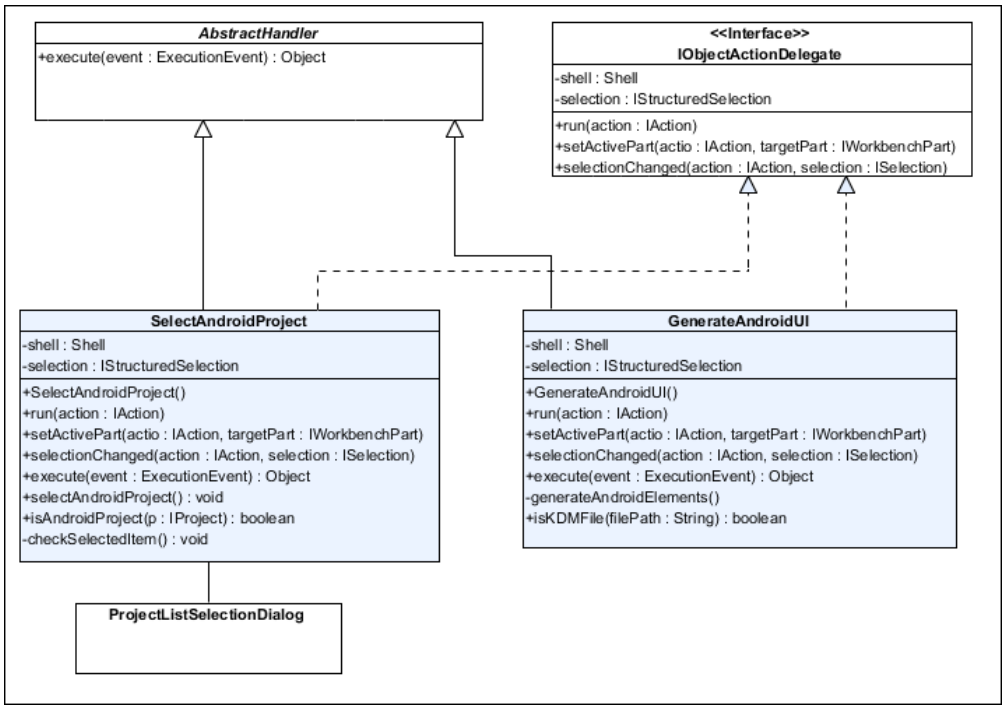


Figura 5.103. Clases GenerateAndroidUI y SelectAndoidProject

Esta nueva funcionalidad solicita al usuario que se elija un proyecto Android (generado con las herramientas que proporciona ADT) para que genere los *Layouts* dentro de este proyecto. Esto se hace con la ventana de diálogo que implementa la clase *ProjectListSelectionDialog* (véase Figura 5.103 y Figura 5.104). Además de generar los propio ficheros XML en el directorio "gen/layout" del proyecto Android, añade la declaración de los elementos del *Layout* en la clase Java que se encarga de ello (generalmente se llama "*R.java*", pero la herramienta solicita al usuario que seleccione dicha clase Java).

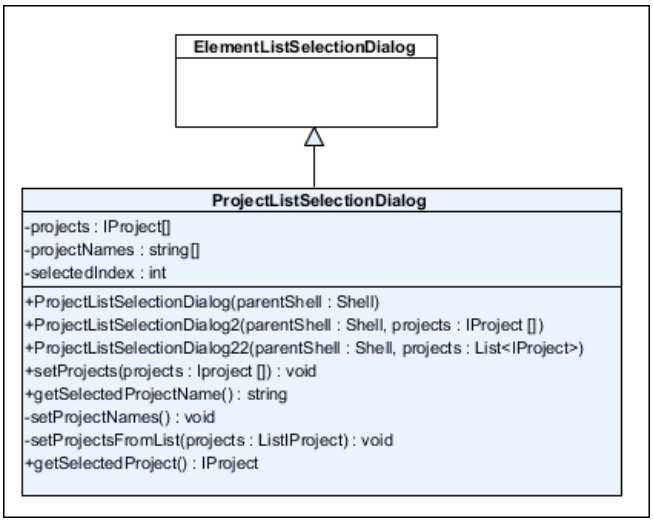


Figura 5.104. Clase ProjectListSelectionDialog

5.7.2. Pruebas

Es en esta iteración se comienza con las pruebas de las distintas piezas software diseñadas e implementadas en iteraciones anteriores para verificar y validar su correcto funcionamiento. En concreto, se comienza con pruebas unitarias para los primeros casos de uso implementados: CdU.4, CdU.5, CdU.6 y CdU.7. Estas pruebas unitarias han consistido en la implementación de un conjunto de casos de prueba siguiendo un enfoque de "caja negra". Esto significa que el código probado ha sido considerado como un bloque cerrado, sin importar lo que hay dentro, y sólo se han tenido en cuenta los datos de entrada y los datos de salida esperados.

Es importante comentar que estas pruebas únicamente se han realizado para las clases que forman la capa de dominio y la capa de persistencia, ya que son las más susceptibles a errores por su complejidad y la cantidad de lógica de dominio que encierran. La capa de presentación, al estar basada en las extensiones que proporciona Eclipse, se ha probado manualmente (mediante las herramientas de ejecución y depuración que ofrece Eclipse) junto al resto de la funcionalidad con resultados satisfactorios.

Para la ejecución de las pruebas unitarias se ha utilizado el conjunto de plug-ins de Eclipse llamado *JUnit*, que es específico para el lenguaje de programación Java. Además, para evaluar la cobertura sobre el código fuente de estas pruebas se ha utilizado otro plug-in de Eclipse llamado *Eclemma*, que soporta las ejecuciones de los

casos de prueba de *JUnit*. Los resultados de estas pruebas se muestran a continuación y se corresponden con el producto de salida PS.7.2.

5.7.2.1. Pruebas unitarias del CdU.4

El primer caso de uso que ha sido sometido a pruebas es el CdU.4 "Generar Modelo de Transformación". Para probar la funcionalidad se ha creado un *test suite* compuesto por varios *test cases*, que se pueden observar en la Figura 5.105. Estas pruebas han consistido principalmente en la creación del parser para los modelos de transformación y la correcta generación de dichos modelos en ficheros XML.

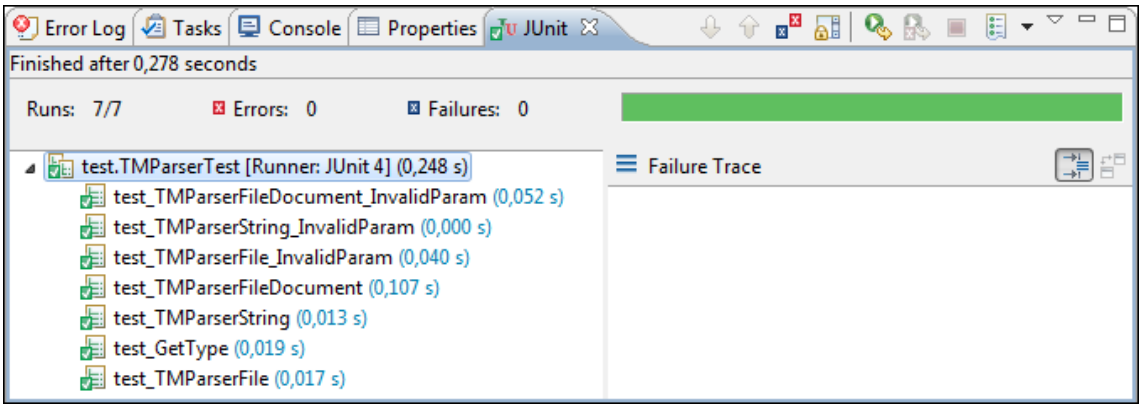


Figura 5.105. Casos de prueba para el CdU.4

Como se puede observar en la Figura 5.106 el código probado cubre las clases *TMParser* (94%) y *Parser* (55%) de la capa de dominio, y una parte de la clase *DiskManager* (7'3%) de la capa de persistencia.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	4,4 %	524	11255	11779
src	4,4 %	524	11255	11779
test	16,3 %	201	1032	1233
TMParserTest.java	63,0 %	201	118	319
AllTests.java	0,0 %	0	35	35
AndrIUProjectSupportTest.java	0,0 %	0	68	68
AndroidParserTest.java	0,0 %	0	47	47
ASTParserTest.java	0,0 %	0	185	185
ExtensionToAndrIUProjectTest.java	0,0 %	0	95	95
KDMParserTest.java	0,0 %	0	219	219
KDMParserTest2.java	0,0 %	0	265	265
persistence	7,3 %	60	767	827
DiskManager.java	7,3 %	60	767	827
domain.parsers	4,9 %	259	5023	5282
TMParser.java	94,0 %	237	15	252
Parser.java	55,0 %	22	18	40
AndroidParser.java	0,0 %	0	454	454
ASTParser.java	0,0 %	0	464	464
KDMParser.java	0,0 %	0	2504	2504
domain	0,0 %	0	123	123
domain.createAndrIUProject	0,0 %	0	159	159
domain.idomtemplates	0,0 %	0	1549	1549

Figura 5.106. Cobertura de las pruebas unitarias para el CdU.4

5.7.2.2. Prueba del CdU.5

Del mismo modo que para el caso de uso anterior, las pruebas unitarias para el CdU.5 Generar Modelos AST, se basan en la creación del parser correspondiente y la generación de modelos AST a partir de unos ficheros de código fuente. Los casos de prueba se pueden observar en la Figura 5.107.

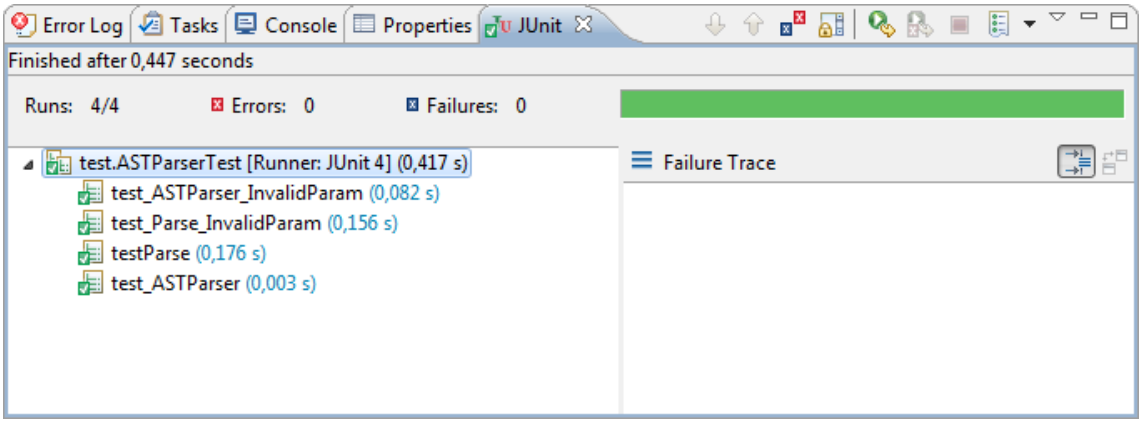


Figura 5.107. Casos de prueba para el CdU.5

Los resultados de la cobertura de estas pruebas obtenidos con *Eclemma* se pueden observar en la Figura 5.108. Las clases implicadas son Parser (47'5%) y *ASTParser* (88'5%) de la capa de dominio y *DiskManager* (3'1%) de la capa de persistencia.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	1,9 %	224	11555	11779
src	1,9 %	224	11555	11779
exceptions	16,7 %	4	20	24
IncorrectPathException.java	100,0 %	4	0	4
IncorrectFolderException.java	0,0 %	0	4	4
IncorrectListException.java	0,0 %	0	4	4
IncorrectNameProjectException.java	0,0 %	0	4	4
IncorrectProjectException.java	0,0 %	0	4	4
InvalidNameProjectException.java	0,0 %	0	4	4
test	8,2 %	101	1132	1233
ASTParserTest.java	54,6 %	101	84	185
AllTests.java	0,0 %	0	35	35
AndrIUProjectSupportTest.java	0,0 %	0	68	68
AndroidParserTest.java	0,0 %	0	47	47
ExtensionToAndrIUProjectTest.java	0,0 %	0	95	95
KDMParserTest.java	0,0 %	0	219	219
KDMParserTest2.java	0,0 %	0	265	265
TMParserTest.java	0,0 %	0	319	319
persistence	3,1 %	26	801	827
DiskManager.java	3,1 %	26	801	827
domain.parsers	2,3 %	93	4035	4128
Parser.java	47,5 %	19	21	40
ASTParser.java	88,5 %	411	53	464
AndroidParser.java	0,0 %	0	822	822
KDMParser.java	0,0 %	0	2550	2550
TMParser.java	0,0 %	0	252	252
domain	0,0 %	0	123	123
domain.createAndrIUProject	0,0 %	0	159	159

Figura 5.108. Cobertura de las pruebas unitarias del CdU.5

5.7.2.3. Prueba de los CdU.6-CdU.7

Para realizar las pruebas unitarias de los casos de uso CdU.6 y CdU.7, consistentes en la generación de modelos KDM a partir de un fichero de código fuente Java, se ha utilizado el *test suite* que se muestra en la Figura 5.109.

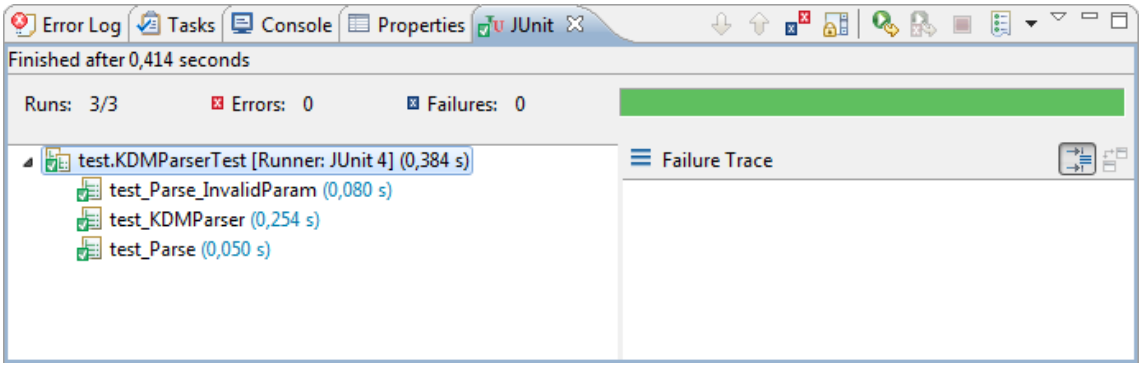


Figura 5.109. Casos de prueba para el CdU.6 y CdU.7

Los resultados de la cobertura de estas pruebas pueden observar en la Figura 5.110. Las clases sometidas a las pruebas han sido *KDMParser* (35'5%), Parser (77'5%), *ASTParser* (15'9%) y *TMParser* (76'2%) de la capa de dominio y *DiskManager* (11'5%) de la capa de persistencia.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	5,6 %	654	11125	11779
src	5,6 %	654	11125	11779
domain.parsers	28,7 %	1202	2926	4128
Parser.java	77,5 %	31	9	40
TMParser.java	76,2 %	192	60	252
KDMParser.java	35,5 %	905	1645	2550
ASTParser.java	15,9 %	74	390	464
AndroidParser.java	0,0 %	0	822	822
test	12,0 %	148	1085	1233
KDMParserTest.java	67,6 %	148	71	219
AllTests.java	0,0 %	0	35	35
AndrIUProjectSupportTest.java	0,0 %	0	68	68
AndroidParserTest.java	0,0 %	0	47	47
ASTParserTest.java	0,0 %	0	185	185
ExtensionToAndrIUProjectTest.java	0,0 %	0	95	95
KDMParserTest2.java	0,0 %	0	265	265
TMParserTest.java	0,0 %	0	319	319
persistence	11,5 %	95	732	827
DiskManager.java	11,5 %	95	732	827

Figura 5.110. Cobertura de las pruebas unitarias de CdU.6 y CdU.7

5.8. Iteración 8

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.

5.8.1. Análisis

Para realizar el análisis de los casos de uso de la iteración 8 se ha optado por la utilización de diagramas de secuencia de análisis.

a) Escenario Normal

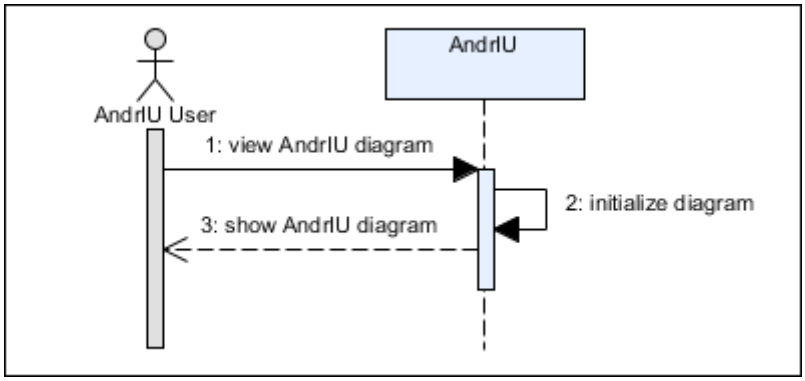


Figura 5.111. Diagrama de Secuencia de Análisis del CdU.12 (Escenario normal)

b) Escenario Alternativo (error al inicializar el diagrama)

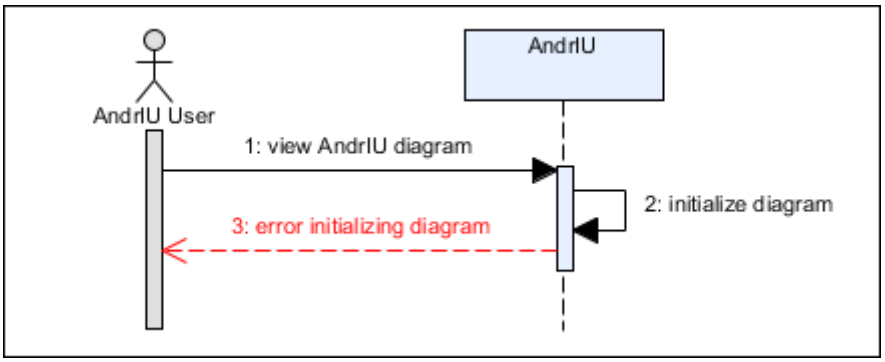


Figura 5.112. Diagrama de Secuencia de Análisis del CdU.12 (Escenario alternativo)

5.8.2. Diseño

Para realizar el diseño del caso de uso referente al editor no se han seguido los pasos de PUD sino que se ha seguido los pasos propuestos por EMF/GMF (véase Figura 3.12.) para el diseño de un editor gráfico. En esta iteración se obtienen los siguientes productos de salida:

En la figura se muestran los componentes de un proyecto GMF que se describen a continuación:

- **Modelo de dominio (.ecore)**

El modelo de dominio es el metamodelo base del Editor Gráfico, y está expresado en el lenguaje ECORE. A partir de este modelo, se irán generando el resto de modelos que necesita el editor GMF. En la Figura 5.113 se muestra un fragmento de

este modelo.

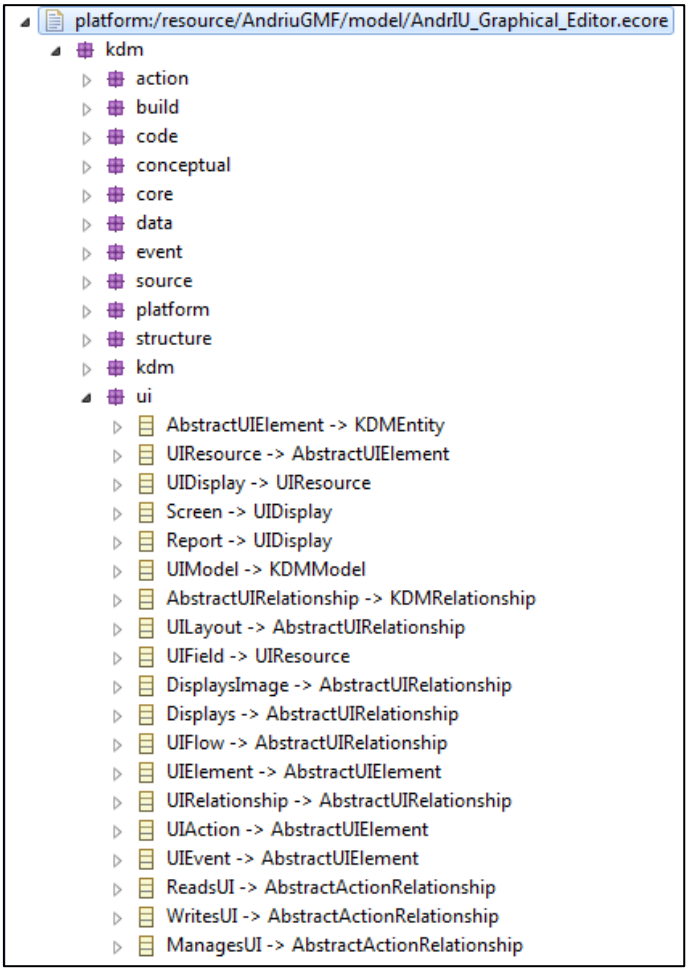


Figura 5.113. Fragmento del modelo de dominio kdm

- **Modelo de definición gráfica (.gmfgraph)**

El modelo de definición gráfica se utiliza para definir las figuras, nodos, conexiones, etc. es decir, todo aquello que se muestra en el diagrama. En este modelo se establece el aspecto que tendrán los componentes del editor: forma, color, etiquetas, etc. En la Figura 5.114 se muestra un fragmento del modelo de definición gráfica del editor gráfico de AndrIU.

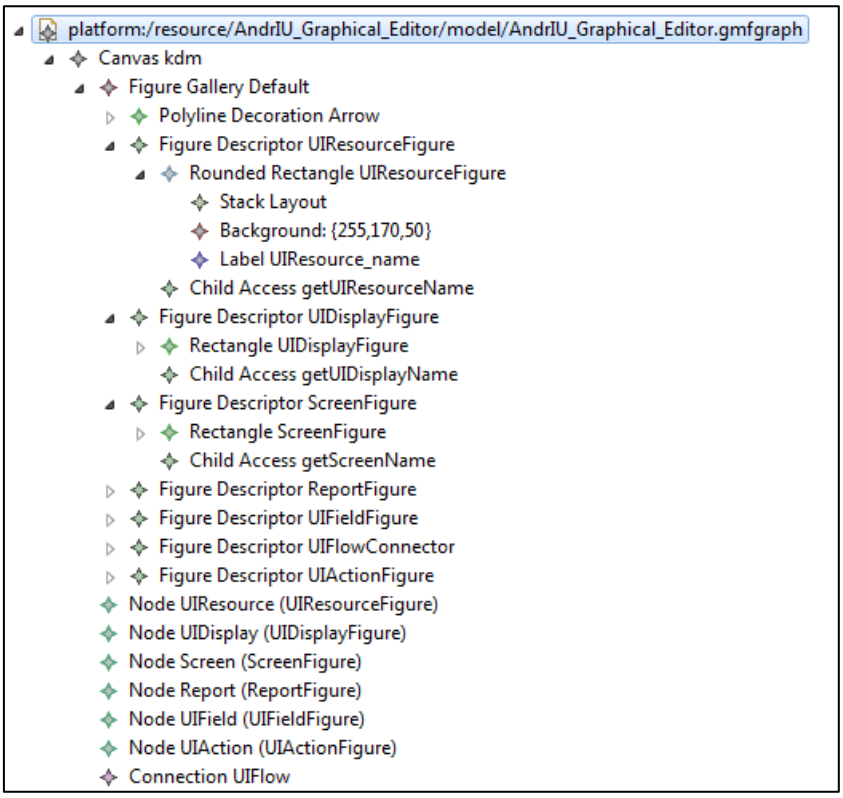


Figura 5.114. Fragmento del modelo de definición gráfica

- **Modelo de definición de herramientas (.gmftool)**

En este modelo se establecen las herramientas que aparecerán en la paleta de herramientas del editor gráfico. En este caso, estas herramientas sirven para añadir y modificar los elementos del modelo que se muestran en el editor. En la Figura 5.115 se muestra un fragmento del modelo de definición de herramientas.

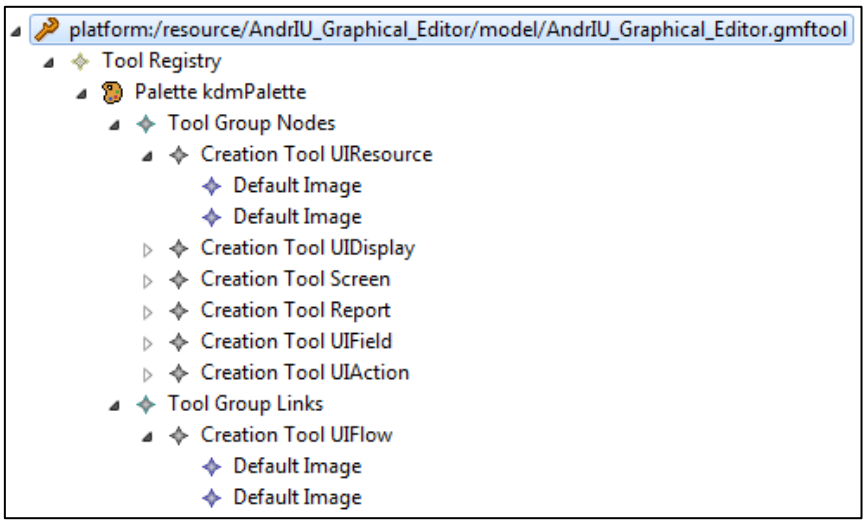


Figura 5.115. Fragmento del modelo de definición de herramientas

5.8.3. Implementación

Del mismo modo que en el diseño, para la implementación del editor se han seguido los pasos propuestos por EMF/GMF (véase Figura 3.12.). Así, los productos de salida son los siguientes:

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

Este modelo se obtiene del modelo base ECORE y permite generar automáticamente el código asociado al meta-modelo y a los editores no gráficos. En la Figura 5.116 se muestra un fragmento del modelo generador obtenido.

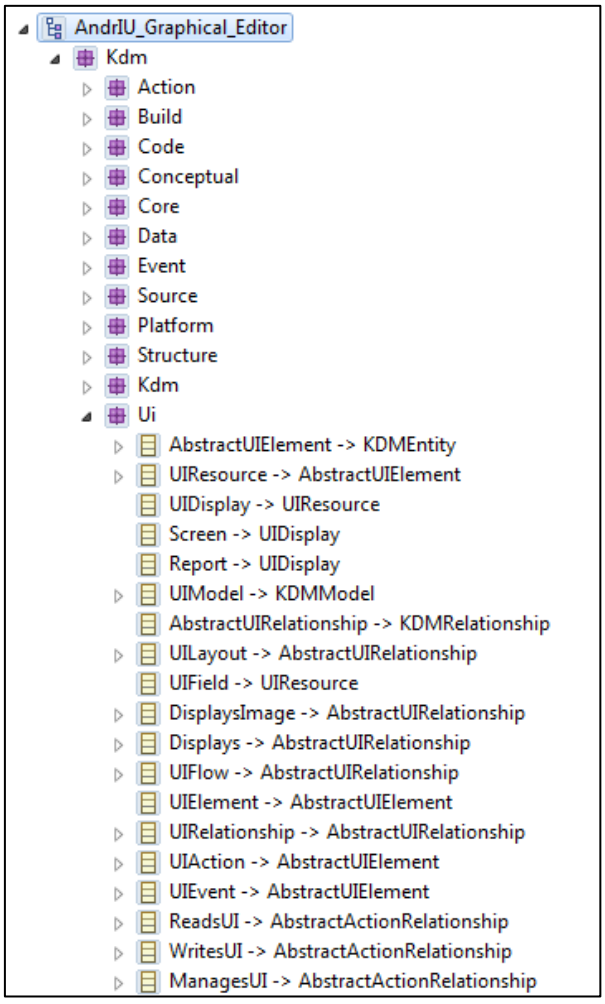


Figura 5.116. Fragmento del generador de código del modelo KDM

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

El modelo de definición de correspondencia o mapeado es el que relaciona los tres modelos creados anteriormente: el de dominio (ecore), el gráfico (gmfgraph) y el de herramienta (gmftool). Con este modelo se relaciona cada elemento del modelo de dominio con el elemento gráfico que lo representará y con la herramienta del editor que permitirá dibujarlo.

De este modo, se definen los elementos que representarán los nodos y los enlaces del diagrama. A su vez, si algún nodo puede contener otros nodos (lo cual es bastante habitual), se definen como nodos hijos del primero. Una vez definidos, se establecen las correspondencias con los modelos de dominio, gráfico y de herramienta como se ha comentado anteriormente. En la Figura 5.117 se muestra un fragmento del modelo de correspondencia del editor gráfico de AndrIU.

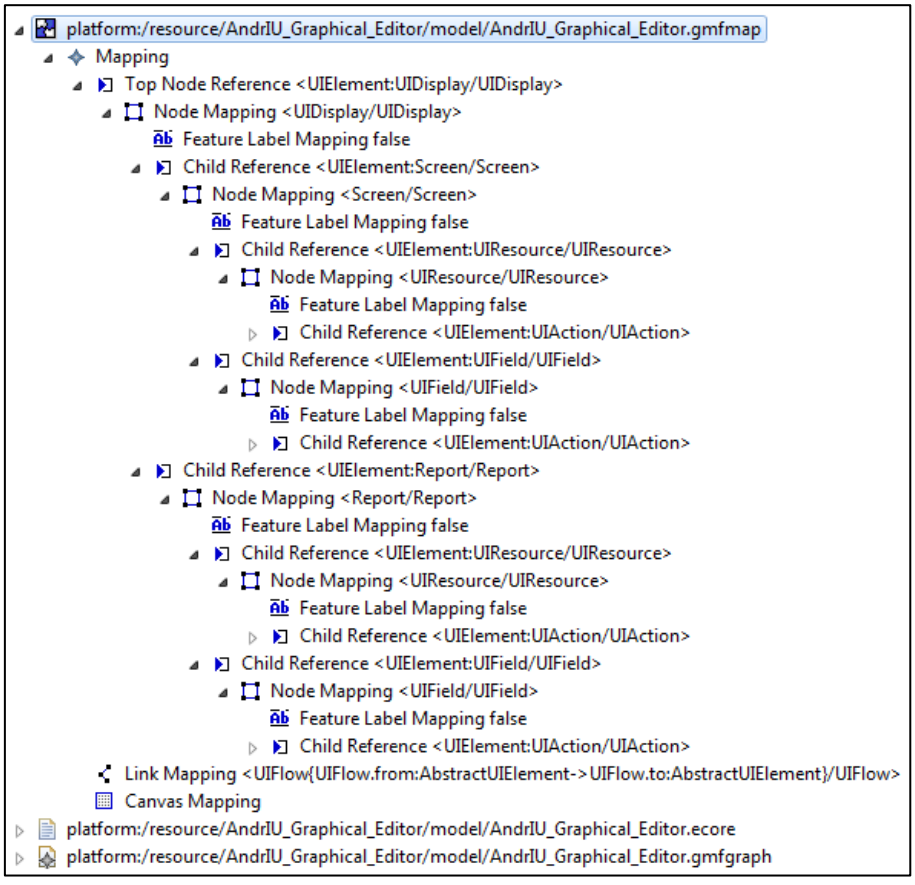


Figura 5.117. Fragmento de la especificación de correspondencia

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

El modelo generador se obtiene a partir del modelo de especificación de correspondencia (gmfmap), y es el encargado de generar todo el código del editor de diagramas. En la Figura 5.118 se muestra un fragmento del modelo generador del editor gráfico de AndrIU.

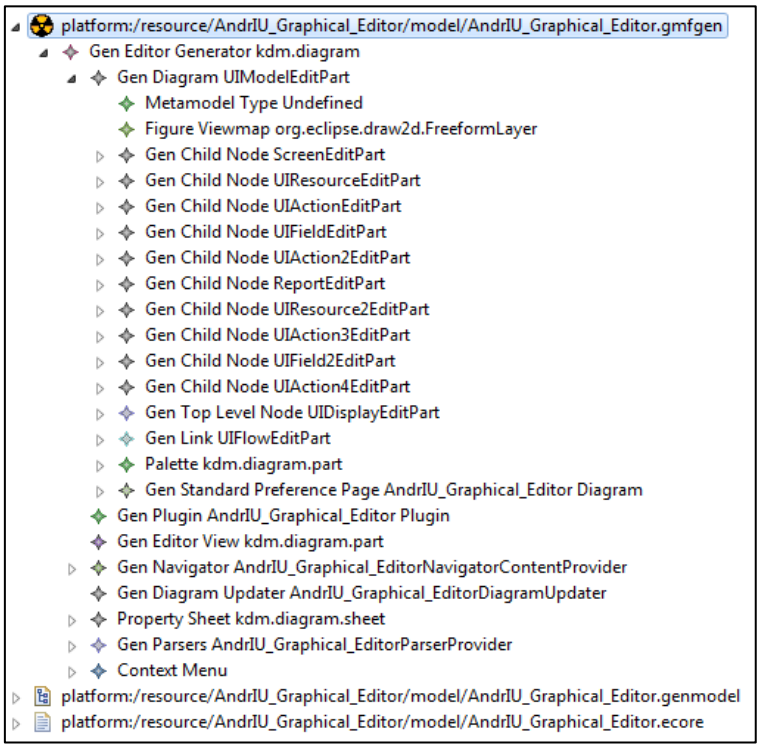


Figura 5.118. Fragmento del modelo generador

Una vez que creado el modelo generador, se pasa al siguiente paso de desarrollo del editor utilizando GMF que es la generación del código del editor gráfico. Esta generación de código se realiza de manera automática a partir de los modelos descritos y se crea la estructura de directorios mostrada en la Figura 5.119.

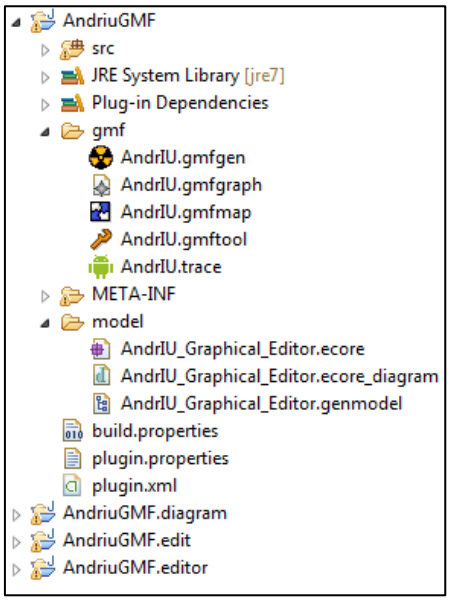


Figura 5.119. Estructura de directorios del editor EMF/GMF

El resultado es un conjunto de plug-ins que contienen el editor gráfico de AndrIU. Una vez integrados estos componentes en el plug-in principal de AndrIU, el resultado del editor se muestra en la Figura 5.120.

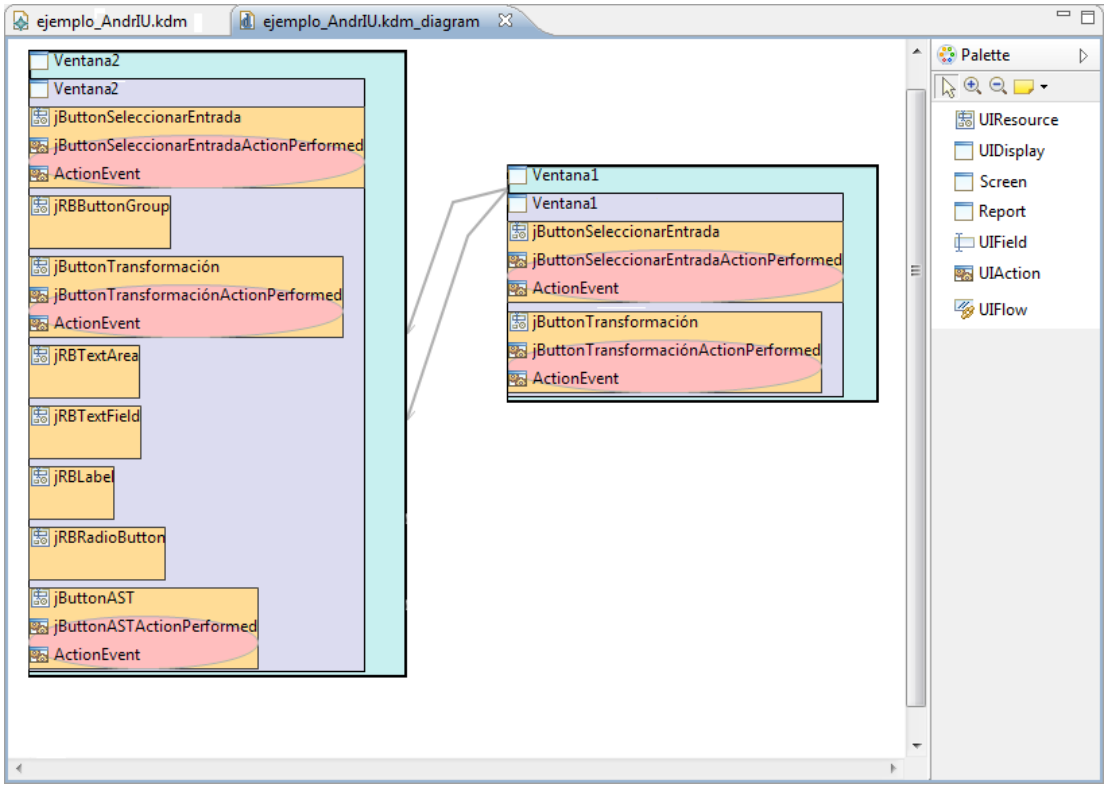


Figura 5.120. Resultado del editor gráfico de AndrIU

5.8.4. Pruebas

De la misma forma que en la iteración anterior, el flujo de trabajo de pruebas de la iteración 8 ha consistido en pruebas unitarias de los casos de uso ya implementados en otras iteraciones. Se han utilizado los plug-ins *JUnit* y *Eclemma* de Eclipse para ejecutar los casos de prueba y medir su cobertura. Los resultados de estas pruebas se muestran a continuación y corresponden con el producto de salida PS.8.5.

5.8.4.1. Pruebas unitarias de los CdU.1-CdU.2

Las pruebas unitarias para estos dos casos de uso han consistido en la creación de proyectos AndrIU nuevos. Para ello se ha utilizado los casos de prueba que se muestran en la Figura 5.121

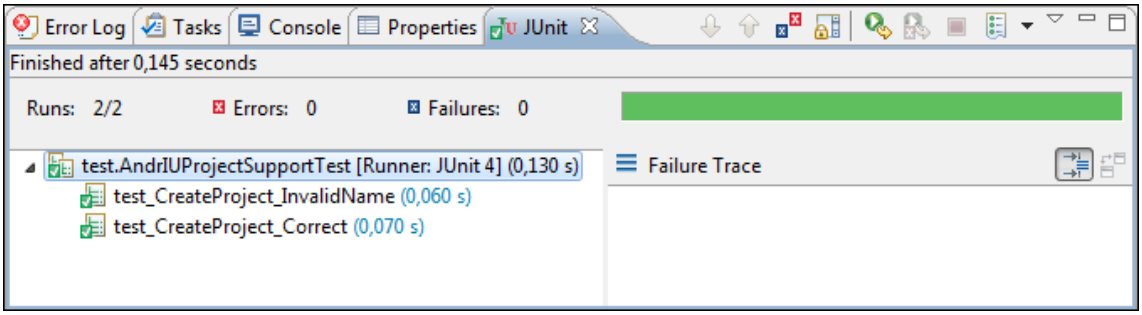


Figura 5.121. Casos de prueba para el CdU.1 y CdU.2

La cobertura de estas pruebas se puede ver en la Figura 5.122. Las clases involucradas en estas pruebas han sido *AndrIUProjectSupport* (84'4%), *AndrIUProjectNature*(62'5%) y *DiskManager* (11'5%). La cobertura se mide en el tanto por ciento de líneas de código que se han ejecutado en las pruebas realizadas. De este modo, en las pruebas realizadas sobre la clase *AndrIUProjectSupport* se ha ejecutado el 84'4% del código de la clase.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	0,4 %	52	11727	11779
src	0,4 %	52	11727	11779
domain.createAndrIUProject	52,8 %	84	75	159
AndrIUProjectSupport.java	84,8 %	84	15	99
ExtensionToAndrIUProject.java	0,0 %	0	60	60
domain.natures	31,2 %	5	11	16
AndroidProjectNature.java	0,0 %	0	8	8
AndrIUProjectNature.java	62,5 %	5	3	8
persistence	11,5 %	95	732	827
DiskManager.java	11,5 %	95	732	827
test	2,3 %	28	1205	1233
AndrIUProjectSupportTest.java	41,2 %	28	40	68
AllTests.java	0,0 %	0	35	35
AndroidParserTest.java	0,0 %	0	47	47
ASTParserTest.java	0,0 %	0	185	185
ExtensionToAndrIUProjectTest.java	0,0 %	0	95	95
KDMParserTest.java	0,0 %	0	219	219
KDMParserTest2.java	0,0 %	0	265	265
TMParserTest.java	0,0 %	0	319	319
domain	0,0 %	0	123	123
domain.jdomtemplates	0,0 %	0	1549	1549
domain.parsers	0,0 %	0	5282	5282
domain.util	0,0 %	0	149	149

Figura 5.122. Cobertura de las pruebas unitarias del CdU.1 y CdU.2

5.8.4.2. Pruebas unitarias del CdU.3

Los casos de prueba para la creación de proyectos AndrIU a partir de un proyecto Java existente se pueden observar en la Figura 5.123. Los resultados de la cobertura de estas pruebas aparecen en la Figura 5.124. Las clases involucradas han sido *AndrIUProjectSupport* (84´8%), *ExtensionToAndrIUProject* (83'3%), *AndrIUProjectNature* (32'6%) y *Diskmanager* (62'5%).

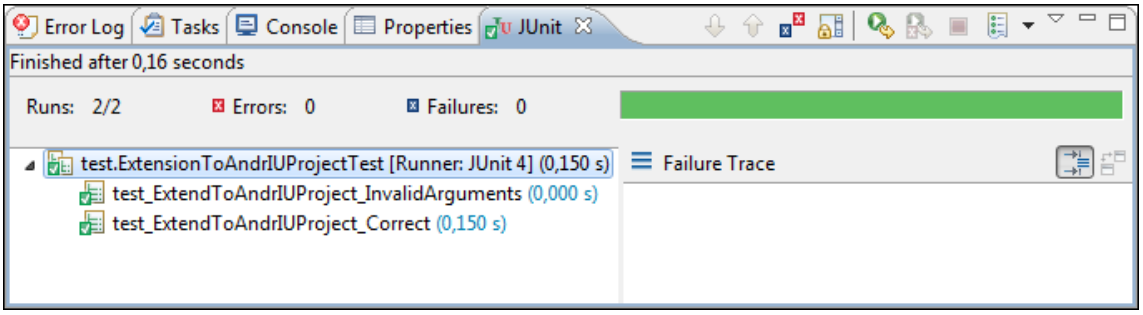


Figura 5.123. Casos de prueba para el CdU.3

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	0,7 %	79	11700	11779
src	0,7 %	79	11700	11779
domain.createAndrIUProject	84,3 %	134	25	159
AndrIUProjectSupport.java	84,8 %	84	15	99
ExtensionToAndrIUProject.java	83,3 %	50	10	60
test	3,4 %	53	1191	1233
ExtensionToAndrIUProjectTest.java	62,5 %	53	42	95
AllTests.java	0,0 %	0	35	35
AndrIUProjectSupportTest.java	0,0 %	0	68	68
AndroidParserTest.java	0,0 %	0	47	47
ASTParserTest.java	0,0 %	0	185	185
KDMParserTest.java	0,0 %	0	219	219
KDMParserTest2.java	0,0 %	0	265	265
TMParserTest.java	0,0 %	0	319	319
persistence	1,7 %	14	813	827
DiskManager.java	32,6 %	270	557	827
domain	0,0 %	0	123	123
domain.jdomtemplates	0,0 %	0	1549	1549
domain.natures	31,2 %	5	11	16
AndrIUProjectNature.java	62,5 %	5	3	8
AndroidProjectNature.java	0,0 %	0	8	8
domain.parsers	0,0 %	0	5282	5282
domain.util	0,0 %	0	149	149

Figura 5.124. Cobertura de las pruebas unitarias del CdU.3

5.8.4.3. Pruebas unitarias del CdU.8

Las pruebas unitarias referentes a la creación de modelos KDM desde un proyecto AndrIU son parecidas a las del CdU.7, con la excepción de que se han utilizado varios ficheros Java para la generación del modelo KDM. Los casos de prueba se pueden observar en la Figura 5.125, y los resultados de la cobertura en la Figura 5.126. Las clases sometidas a las pruebas han sido *KDMParser* (37'4%), *Parser* (77'5%), *TMParser* (76'2%), *ASTParser* (15'9%) y *DiskManager* (11'5%).

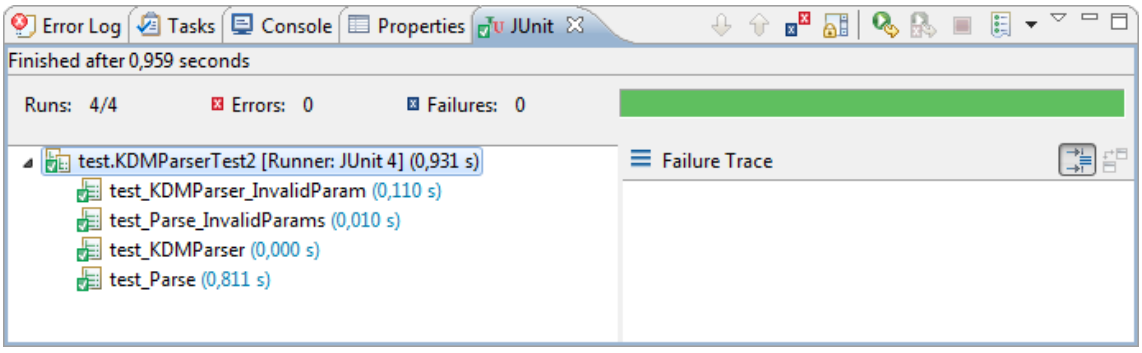


Figura 5.125. Casos de prueba para el CdU.8

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	6,6 %	776	11003	11779
src	6,6 %	776	11003	11779
test	15,7 %	193	1040	1233
domain.parsers	10,4 %	551	4731	5282
Parser.java	77,5 %	31	9	40
TMParser.java	76,2 %	192	60	252
ASTParser.java	15,9 %	74	390	464
KDMParser.java	37,4 %	931	1573	2504
AndroidParser.java	0,0 %	0	454	454
persistence	11,5 %	95	732	827
DiskManager.java	11,5 %	95	732	827
domain	0,0 %	0	123	123
domain.createAndrIUProject	0,0 %	0	159	159
domain.jdomtemplates	0,0 %	0	1549	1549
domain.natures	0,0 %	0	16	16

Figura 5.126. Cobertura de las pruebas unitarias del CdU.8

5.9. Iteración 9

Como se indica en la Tabla 4.8. 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.

5.9.1. Análisis

Para realizar el análisis de los casos de uso de la iteración 9 se han utilizado diagramas de secuencia de análisis.

a) Escenario Normal

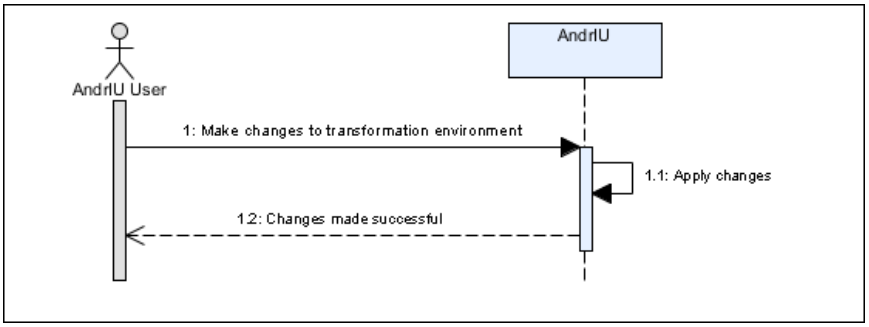


Figura 5.127. Diagrama de Secuencia de Análisis del CdU.13 (Escenario normal)

b) Escenario Alternativo (error)

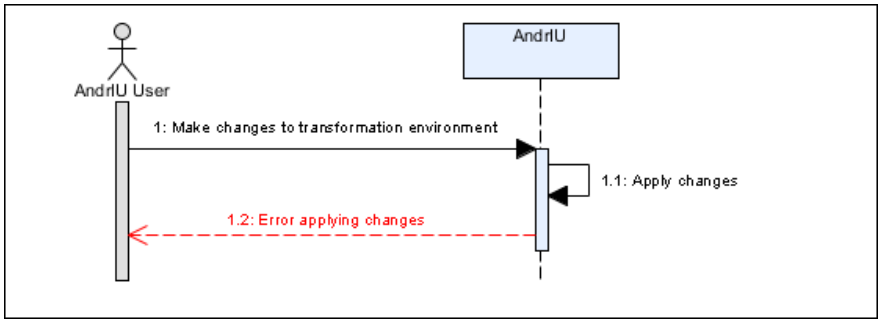


Figura 5.128. Diagrama de Secuencia de Análisis del CdU.13 (Escenario alternativo)

5.9.2. Diseño

La funcionalidad del caso de uso CdU13 ha sido 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 (véase Figura 5.129) con las clases añadidas. Además, en la Figura 5.130 se muestra el diseño del prototipo de la ventana de preferencias.

5.9.2.1. Diagrama de clases de diseño

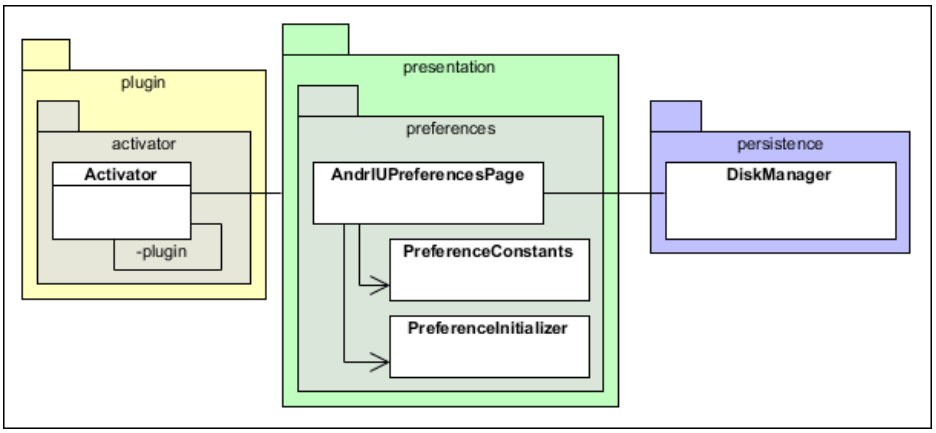


Figura 5.129. Diagrama de clases de la iteración 9

Prototipo de la GUI

En la ventana de preferencias diseñada (véase Figura 5.130), se tienen dos opciones de configuración: una selección booleana (verdadero/falso) y un campo de selección de fichero. La primera opción es si se desea tomar un modelo de transformación por defecto en las funcionalidades de generación de modelos KDM. La segunda opción es la selección de dicho modelo de transformación en el equipo donde se ejecute AndrIU.

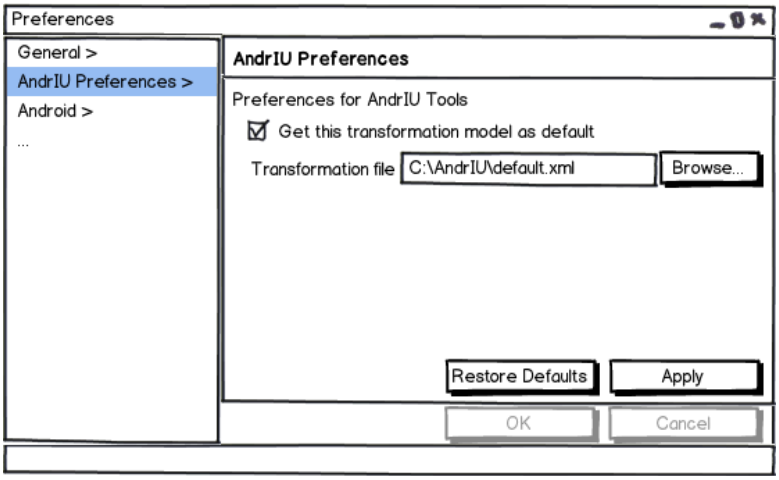


Figura 5.130. Prototipo de la ventana de preferencias de AndrIU

5.9.3. Implementación

La página de preferencias de la herramienta AndrIU contiene un campo booleano que da la opción de seleccionar el modelo de transformación por defectos al generar los modelos KDM, o que sea el usuario el que elija el modelo deseado. Además contiene otro campo textual donde se ha de poner el fichero con el modelo de transformación por defecto.

La página de preferencias está implementada por la clase *AndrIUPreferencesPage*. La clase *PreferenceConstants* proporciona las constantes relativas a las preferencias y la clase *PreferenceInitialicer* tiene la responsabilidad de inicializar los valores de estas preferencias. Todas estas clases se pueden ver en detalle en la Figura 5.131.

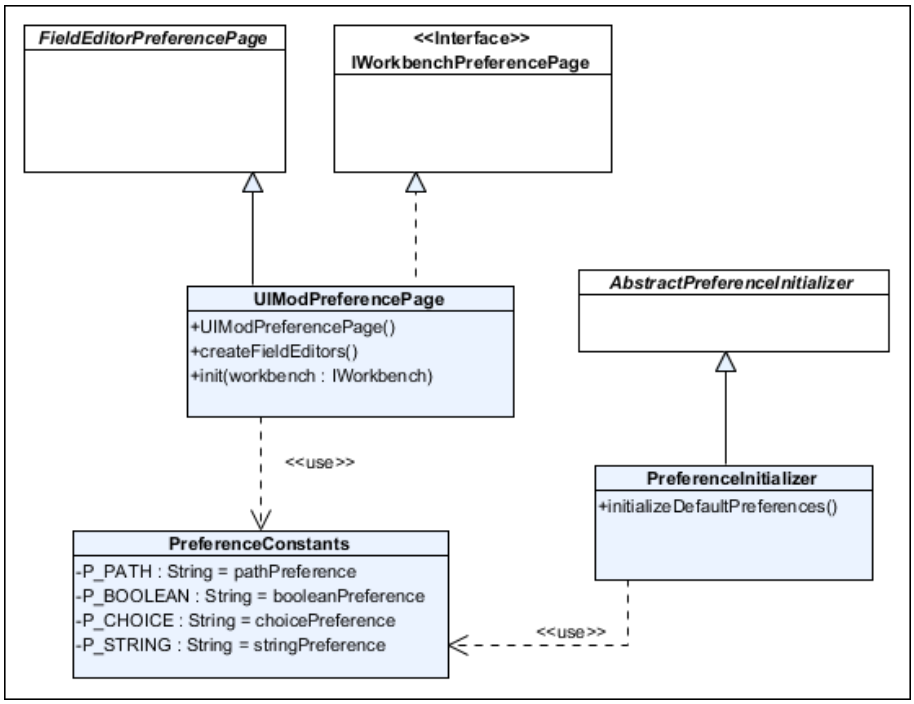


Figura 5.131. Diagrama de clases de la página de preferencias de AndrIU

5.9.4. Pruebas

En esta iteración se concluyen las pruebas unitarias y realizan las pruebas de integración y pruebas de aceptación del sistema implementado. Para el CdU.12 no se han realizado pruebas unitarias, ya que hace referencia al editor gráfico y las pruebas se han realizado manualmente. Para el CdU.13 tampoco se han realizado pruebas con *JUnit* y *Eclemma* ya que su funcionalidad se refiere a la página de preferencias de la herramienta AndrIU en el entorno de Eclipse, y pertenece a la capa de presentación (ya que la persistencia la gestiona Eclipse).

5.9.4.1. Pruebas unitarias de los CdU.9, CdU.10 y CdU.11

La construcción de las pruebas unitarias ha seguido la misma estructura que en iteraciones anteriores de modo que sólo se han realizado pruebas para las clases de la capa de dominio y persistencia. Estos casos de uso hacen referencia a la generación de GUI para sistemas Android a partir de modelos KDM, ya sea directamente desde el modelo o desde un proyecto AndrIU. Por este motivo, las pruebas referidas a estos tres casos de uso se han unificado en el mismo *test suite* que se muestra en la Figura 5.132. Estas pruebas corresponden al producto de salida PS.9.2.

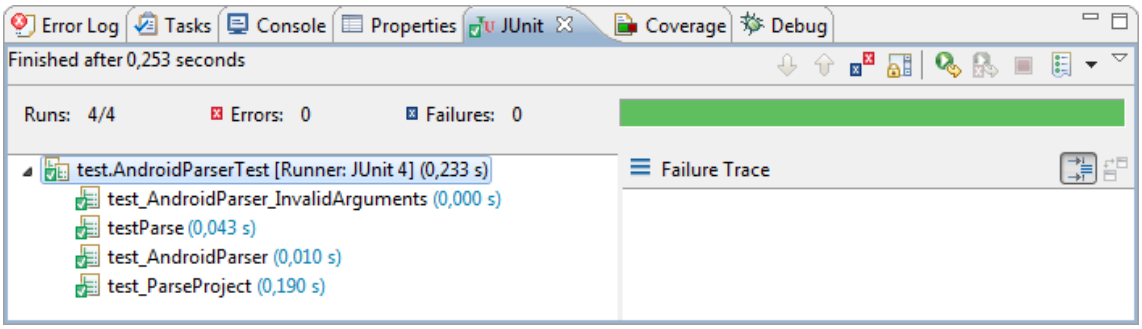


Figura 5.132. Casos de prueba para el CdU.9, CdU.10 y CdU.11

La cobertura de estas pruebas se muestra en la Figura 5.133. Las clases sometidas a prueba han sido *AndroidParser* (92'5%), *Parser* (35%) y *DiskManager* (4'4%).

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	4,3 %	506	11273	11779
src	4,3 %	506	11273	11779
domain.parsers	8,2 %	434	4848	5282
AndroidParser.java	92,5 %	420	34	454
Parser.java	35,0 %	14	26	40
ASTParser.java	0,0 %	0	464	464
KDMParser.java	0,0 %	0	2504	2504
TMParser.java	0,0 %	0	252	252
persistence	4,4 %	36	791	827
DiskManager.java	4,4 %	36	791	827
test	2,9 %	36	1197	1233
AndroidParserTest.java	76,6 %	36	11	47
AllTests.java	0,0 %	0	35	35
AndrIUProjectSupportTest.java	0,0 %	0	68	68
ASTParserTest.java	0,0 %	0	185	185
ExtensionToAndrIUProjectTest.java	0,0 %	0	95	95
KDMParserTest.java	0,0 %	0	219	219
KDMParserTest2.java	0,0 %	0	265	265
TMParserTest.java	0,0 %	0	319	319
domain	0,0 %	0	123	123
domain.createAndrIUProject	0,0 %	0	159	159
domain.jdomtemplates	0,0 %	0	1549	1549
domain.natures	0,0 %	0	16	16
domain.util	0,0 %	0	149	149

Figura 5.133. Cobertura de las pruebas unitarias del CdU.9, CdU.10 y CdU.11

5.9.4.2. Pruebas de integración

Las pruebas de integración se realizan para verificar que todos los componentes software construidos interaccionan entre sí correctamente. Estas pruebas se han realizado con *JUnit* y *Eclemma* (véase Figura 5.134 y Figura 5.135), y corresponden al producto de salida PS. 9.2. Las clases pertenecientes a la capa de presentación muestran una cobertura del 0%, lo cual es totalmente comprensible ya que las pruebas se han realizado sólo sobre las capas de dominio y persistencia.

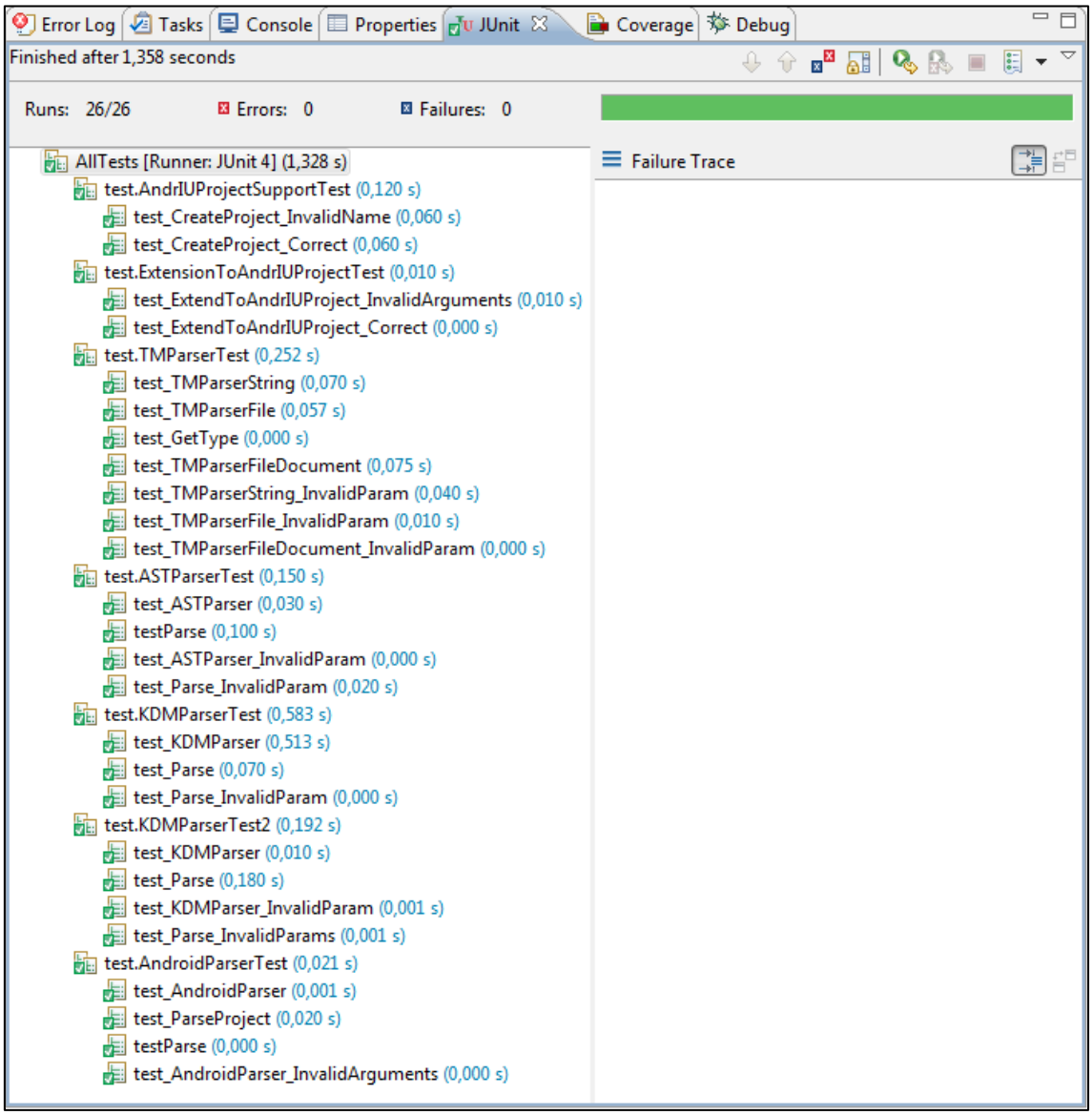


Figura 5.134. Test suite de las pruebas de integración

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
AndrIU2	43,4 %	5107	6672	11779
src	43,4 %	5107	6672	11779
test	65,0 %	801	432	1233
AllTests.java	80,0 %	28	7	35
KDMParserTest2.java	77,4 %	205	60	265
AndroidParserTest.java	76,6 %	36	11	47
KDMParserTest.java	71,2 %	156	63	219
TMParserTest.java	63,0 %	201	118	319
ASTParserTest.java	56,8 %	105	80	185
ExtensionToAndrIUProjectTest.java	44,2 %	42	53	95
AndrIUProjectSupportTest.java	41,2 %	28	40	68
domain.parsers	63,5 %	3352	1930	5282
TMParser.java	94,8 %	239	13	252
AndroidParser.java	92,5 %	420	734	793
KDMParser.java	91,6 %	2293	211	2504
Parser.java	90,0 %	36	4	40
ASTParser.java	78,4 %	364	100	464
domain.natures	62,5 %	10	16	32
AndroidProjectNature.java	62,5 %	5	3	8
AndrIUProjectNature.java	62,5 %	5	3	8
domain.jdomtemplates	52,6 %	815	734	1549
ElementExpressionMethodCall.java	100,0 %	77	0	77
ElementStatementBlock.java	100,0 %	48	0	48
ElementStatementIf.java	100,0 %	57	0	57
ElementExpressionVariableDeclaration	89,4 %	59	7	66
ElementStatement.java	88,9 %	104	13	117
ElementStatementTry.java	85,5 %	59	10	69
ElementStatementExpression.java	82,5 %	66	14	80
ElementExpression.java	82,0 %	82	18	100
ElementExpressionObjectCreation.java	71,8 %	94	37	131
ElementAttribute.java	38,8 %	33	52	85
ElementMethod.java	38,0 %	84	137	221
ElementConstructor.java	23,9 %	52	166	218
ElementClass.java	0,0 %	0	128	128
ElementStatementFor.java	0,0 %	0	152	152
exceptions	50,0 %	12	12	24
IncorrectNameProjectException.java	100,0 %	4	0	4
IncorrectPathException.java	100,0 %	4	0	4
InvalidNameProjectException.java	100,0 %	4	0	4
IncorrectFolderException.java	0,0 %	0	4	4
IncorrectListException.java	0,0 %	0	4	4
IncorrectProjectException.java	0,0 %	0	4	4
persistence	33,2 %	275	552	827
DiskManager.java	33,2 %	275	552	827
domain.util	28,2 %	42	107	149
VariableDeclaration.java	47,7 %	42	46	88
AndrIUConsole.java	0,0 %	0	57	57
Util.java	0,0 %	0	4	4
domain.createAndrIUProject	15,7 %	25	134	159
ExtensionToAndrIUProject.java	31,7 %	19	41	60
AndrIUProjectSupport.java	6,1 %	6	93	99
domain	0,0 %	0	123	123
plugin.activator	0,0 %	0	21	21
presentation.actions	0,0 %	0	1463	1463
presentation.dialogs	0,0 %	0	739	739
presentation.preferences	0,0 %	0	60	60
presentation.wizards	0,0 %	0	69	69
presetation.perspectives	0,0 %	0	65	65

Figura 5.135. Cobertura de las pruebas de integración

5.9.4.3. Pruebas de aceptación

El objetivo de las pruebas de aceptación es validar que el sistema desarrollado cumple con el funcionamiento esperado, además de ofrecer al usuario de dicho sistema un garantía para su aceptación, desde el punto de vista de su funcionalidad y rendimiento. Esta parte corresponde al producto de salida PS. 9.4.

Para desempeñar estas pruebas de aceptación se han llevado a cabo dos casos de estudio (aplicaciones de escritorio implementadas en Java) como ejemplos de aplicación real de la herramienta AndrIU. Estos casos de estudio se pueden consultar en detalle en el Anexo II.

5.10. Iteración 10

Como se indica en la Tabla 4.10. , en esta iteración se ha completado la memoria del Proyecto Fin de Carrera (correspondiente al producto de salida PS. 10.1), incluyendo el manual de usuario (se puede consultar en el Anexo I y se corresponde al producto de salida PS. 10.2). También se ha producido el producto software del 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. Adicionalmente, la herramienta puede descargarse del repositorio de plug-in *Elipse Marketplace* en http://marketplace.eclipse.org/content/andriu.

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 I.

5.10.2. Distribución de la herramienta

Como último paso en la planificación realizada en el apartado 5.2.2 se muestra la distribución que se ha realizado de la herramienta, correspondiente al producto de salida

PS. 10.4:

- La herramienta *AndrIU* se encuentra disponible para su descarga en la página web http://www.businessprocessarcheology.org/. En este sitio web se puede encontrar la herramienta de dos formas:
    - Una distribución de Eclipse basada en *Eclipse Modeling Tools*, con ADT y AndrIU instalados.
    - Los componentes correspondientes a AndrIU y las librerías necesarias (como jdom, el parser de java, los editores GMF, etc) para que sean instaladas en una distribución cualquiera de Eclipse Indigo.
- Adicionalmente, la herramienta *AndrIU* se encuentra disponible en el sitio oficial Eclipse Market Place (Eclipse, 2011a) en http://marketplace.eclipse.org/content/andriu.

Capítulo 6

Conclusiones y propuestas

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

6. CONCLUSIONES Y PROPUESTAS

6.1. Conclusiones

El presente proyecto de fin de carrera definió como objetivo principal desarrollar una herramienta software que permitiera la migración de GUIs de sistemas de información existentes a través de la extracción y representación de estas GUIs mediante modelos estandarizados. Se han seguido técnicas de ingeniería inversa para la extracción de los modelos a partir del sistema de información heredado, y de ingeniería directa para transformar estos modelos en GUIs para otra plataforma diferente a la del sistema de origen abarcando de este modo todo el proceso de reingeniería.

El resultado de este proyecto ha sido la herramienta software *AndrIU*, que contribuye a la modernización del software al extraer y representar las GUIs de sistemas existentes desarrollados en Java a través de modelos estandarizados KDM (ISO/IEC 19506), y dando soporte a la migración de estas GUIs a plataformas móviles como Android.

Este proyecto plantea además una forma de representación de GUIs a través del paquete UI del estándar KDM, y su integración con otros elementos del modelo de código. La notación de esta propuesta se detalla en el Anexo III.

AndrIU ha sido desarrollada mediante un conjunto de plug-ins para la plataforma Eclipse, lo que facilita su aplicación en la industria de desarrollo software. Además permite y facilita una futura extensión de esta herramienta con mejoras, nuevas funcionalidades, integración con otras herramientas de modernización software, etc.

Tras la finalización del presente Proyecto de Fin de carrera, y contrastando los resultados obtenidos con la lista de objetivos marcados al inicio del mismo, se considera que se han alcanzado los objetivos marcados. Por tanto, el resultado final de este PFC ha sido satisfactorio, tanto personal como profesionalmente.

En la Tabla 6.1 se muestra una lista con los requisitos funcionales que se extrajeron de los objetivos al inicio del proyecto. Además se muestra si los requisitos han sido satisfechos y su justificación.

Id	Requisito Funcional	OK	Justificación
RF.01	Permitir la generación de modelos de IU representados en KDM siguiendo el estándar ISO/IEC 19506.	✓	Se generan modelos de IU en KDM a partir de ficheros de código fuente en Java.
RF.02	Permitir la generación de modelos de transformación para la definición y correspondencia de los elementos de IU que se van a tratar.	✓	Se generan modelos de transformación a través de una opción de la herramienta que defina la correspondencia entre elementos de GUI Java y elementos de UI de KDM.
RF.03	Permitir el análisis de código fuente de un sistema de información existente a fin de reconocer ciertos elementos de IU para ser representados en un modelo específico de plataforma (PSM).	✓	Se ha generado un parser de Java que permite analizar el código fuente y extraer de éste modelos específicos de la plataforma (los modelos AST).
RF.04	Permitir la creación de una nueva estructura (proyecto) que almacene y organice los modelos que se generarán.	✓	Se ha creado una estructura "proyecto AndrIU" vacía que permite albergar los modelos que se generarán.
RF.05	Permitir la creación de una estructura que almacene y organice los modelos que se generarán a partir de un sistema de información heredado.	✓	Se ha creado una opción que permite crear un "proyecto AndrIU" a partir de un "proyecto Java", que permite almacenar los modelos que se van a generar.
RF.06	Permitir que la información generada sea persistente, es decir, que se aloje físicamente para su posterior recuperación.	✓	Los modelos que se generan son almacenados en ficheros (.xml, .kdm, etc), que se estructuran en el sistema de directorios de los "proyectos AndrIU".
RF.07	Permitir la representación gráfica de los modelos de IU generados a fin de que éstos puedan ser utilizados para facilitar las tareas en fases posteriores de reingeniería y migración.	✓	Se ha facilitado un editor gráfico para la representación de los modelos de IU de KDM.
RF.08	Permitir la generación de GUI para Android a partir de los modelos de IU generados, y su inclusión en proyectos Android.	✓	Se generan ficheros XML con los Layout que contienen las GUI para Android a partir de los modelos KDM generados.
RF.09	Permitir la edición de los modelos generados a fin de realizar correcciones y/o modificaciones.	✓	Se ha facilitado otro editor que permite la modificación de todos los modelos KDM generados.
RF.10	Permitir la configuración de ciertos parámetros para realizar las distintas tareas que permita la herramienta. Los cambios en estos parámetros serán persistentes, de manera que permanezcan en posteriores sesiones.	✓	La herramienta proporciona una página de preferencias que permite configurar de manera persistente ciertos parámetros necesarios para el resto de utilidades.

Tabla 6.1.Requisitos funcionales cumplidos tras el PFC

6.2. Trabajo futuro

En este apartado se comentan algunas líneas de trabajo futuro que nacen directamente de la finalización del Proyecto de Fin de Carrera. Como ya se ha señalado anteriormente, la herramienta *AndrIU* ha sido implementada como un plug-in, lo que facilita su ampliación a través de nuevas extensiones que añadan nueva funcionalidad. De este modo, tras el desarrollo de la herramienta *AndrIU* se proponen las siguientes líneas de trabajo futuro:

- Soportar el análisis de sistemas de información heredados basados en otras tecnologías o plataformas diferentes a Java, para extraer los modelos de UI en KDM, que es independiente de la plataforma.
- Del mismo modo, proporcionar la generación de GUIs a partir de los modelos KDM generados para otras plataformas diferentes a *Android* como *iOS* o *Windows Mobile*.
- Mejorar la técnica de extracción de información de las GUI para limitar aún más la pérdida de semántica característica de cualquier técnica de ingeniería inversa.
- Mantener la herramienta de manera que se mejoren o añadan nuevas funcionalidades que puedan resultar interesantes para los procesos de reingeniería y no se han contemplado en esta primera versión.
- Mejorar la distribución de los elementos de las GUI de acuerdo a diferentes *Layouts* automáticos en las GUI de Android.

7. REFERENCIAS

[1] Android_Developers. "Android Developers." from http://developer.android.com/ index.html.

[2] Arnold, R. S. (1993). "Software Reengineering." IEEE Computer Society Press.

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

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

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

[6] E. Stroulia, M. E.-R., L. Kong, P. Sorenson (1999). "Reverse Engineerin Legacy Interfaces: An interaction-Driven Approach."

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

[8] Eclipse. (2011a). "Eclipse." from http://www.eclipse.org/.

[9] Eclipse. (2011b). "Eclipse Modeling Tools." from http://www.eclipse.org/downloads/packages/eclipse-modeling-tools-includes-incubating-components/heliossr2.

[10] Eclipse. (2011d). "Android Development Tools." from http://developer.android.com/sdk/eclipse-adt.html.

[11] Gamma, E. and K. Beck (2010). JUnit.

[12] Gamma, E., R. Helm, et al. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Inc. Boston, MA, USA, Addison Wesley.

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

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

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

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

[17] Jacaboson, I., Booch, G., Rumbaugh J. (2000). El Proceso Unificado de Desarrollo de Software.

[18] JavaCC. (2009). "Java Compiler Compiler." from http://javacc.java.net/.

[19] Julio Abascal, I. A., José J. Cañas, Miguel Gea, Ana Belén Gil, Jesús Lorés, Ana Belén Martínez, Manuel Ortega, Pedro Valero, Manuel Vélez (2001). La interacción persona-ordenador.

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

[21] Kleppe, A., J. Warmer, et al. (2003). MDA explained: the model driven architecture: practice and promise, Addison-Wesley.

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

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

[24] OMG (2003a). Model Driven Architecture.

[25] OMG (2003b). MDA Guide Version 1.0.1. OMG**:** 62.

[26] OMG (2006). "ADM Glossary of Definitions and Terms."

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

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

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

[30] Pérez-Castillo, R., I. García-Rodríguez de Guzmán, et al. (2011). "Business

Process Archeology using MARBLE." Information and Software Technology **In Press**.

[31] Polo, M., M. Piattini, et al. (2003). Advances in software maintenance management: technologies and solutions, Idea Group Publishing.

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

[33] Schmidt, A., B. Pfleging, F. Alt, A.S. Shirazi, and G. Fitzpatrick (2012). "Interacting with 21st-Century Computers." IEEE Pervasive Computing Magazine.

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

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

[36] Steinberg, D., F. Budinsky, et al. (2008). EMF: Eclipse Modeling Framework, Addison-Wesley Professional,.

[37] T. Moran, L. C. (1996). Design rationale: concepts, techniques and use.

[38] Van Agten, T. (2012). "Google Android Market Tops 400,000 Applications."

Anexos

En la siguiente sección se presentan una serie de anexos que tienen como finalidad completar o aclarar la información presentada en los capítulos anteriores.

8. ANEXOS

8.1. Anexo I. Manual de Usuario

Este anexo corresponde al manual de usuario de la herramienta desarrollada a lo largo del presente Proyecto de Fin de Carrera. En este manual únicamente se explicarán los detalles de las funcionalidades que ofrece AndrIU, sin entrar en detalles del propio entorno de desarrollo, dando por supuesto que el usuario conoce en mayor o menor medida Eclipse.

La herramienta AndrIU se ofrece de dos maneras: como una aplicación Eclipse consistente en una distribución de Eclipse que contiene la herramienta; o como un conjunto de plug-ins que deben ser instalados en una distribución de Eclipse de la manera habitual. En este manual se toma la aplicación Eclipse que contiene las herramientas de AndrIU integradas. Para ello, hay que copiar el directorio AndrIU en el lugar del equipo que se desee, ya que no se requiere de una instalación previa.

Para ejecutar la aplicación, hacer doble clic sobre el ejecutable "*eclipse.exe*" (véase Figura 8.1) y aparecerá la imagen de la Figura 8.2 mientras carga el entorno.

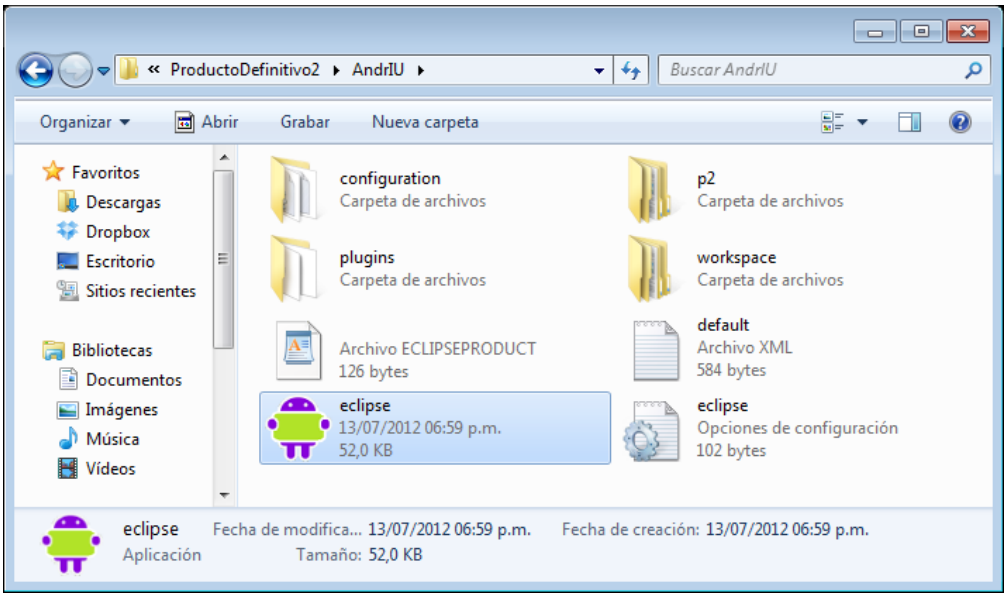


Figura 8.1. Directorio de la aplicación AndrIU

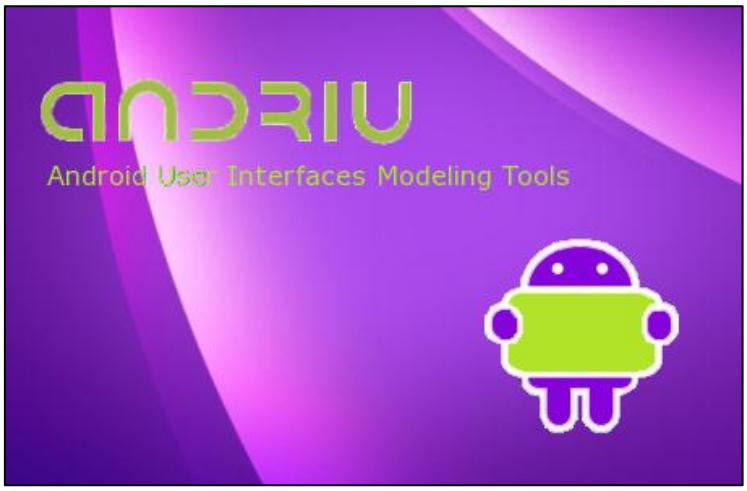


Figura 8.2. Imagen de carga de la aplicación AndrIU

Tras esto, se pedirá la selección del *Workspace* o directorio de trabajo (véase Figura 8.3) y una vez seleccionado se cargará el entorno de Eclipse como se muestra en la Figura 8.4. A continuación, para poder utilizar todas las herramientas que ofrece AndrIU hay que seleccionar la "perspectiva AndrIU" como se muestra en la Figura 8.5 de la manera habitual (véase Figura 8.6).

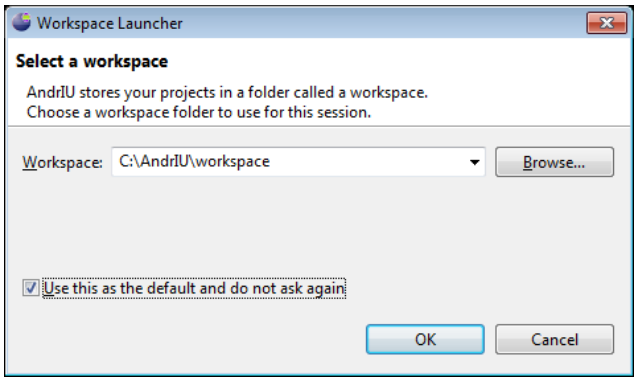


Figura 8.3. Ventana de selección del *Workspace* inicial

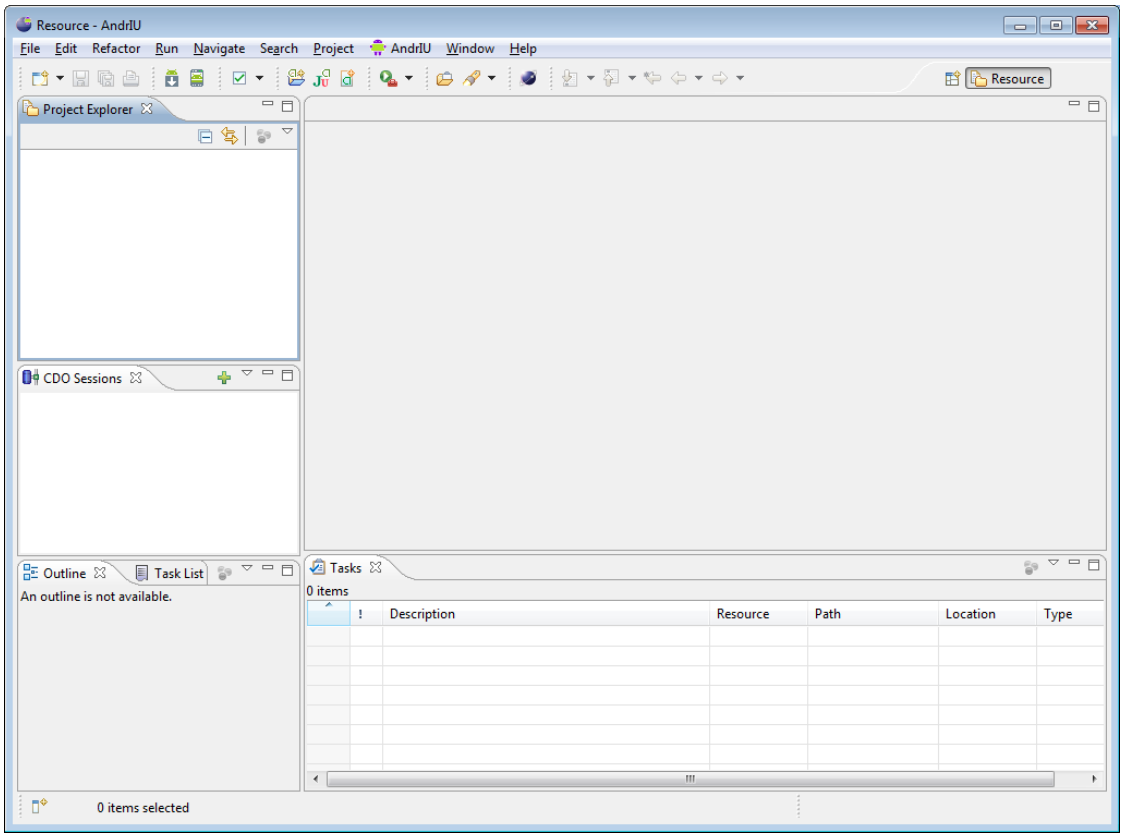


Figura 8.4. Vista inicial de Eclipse

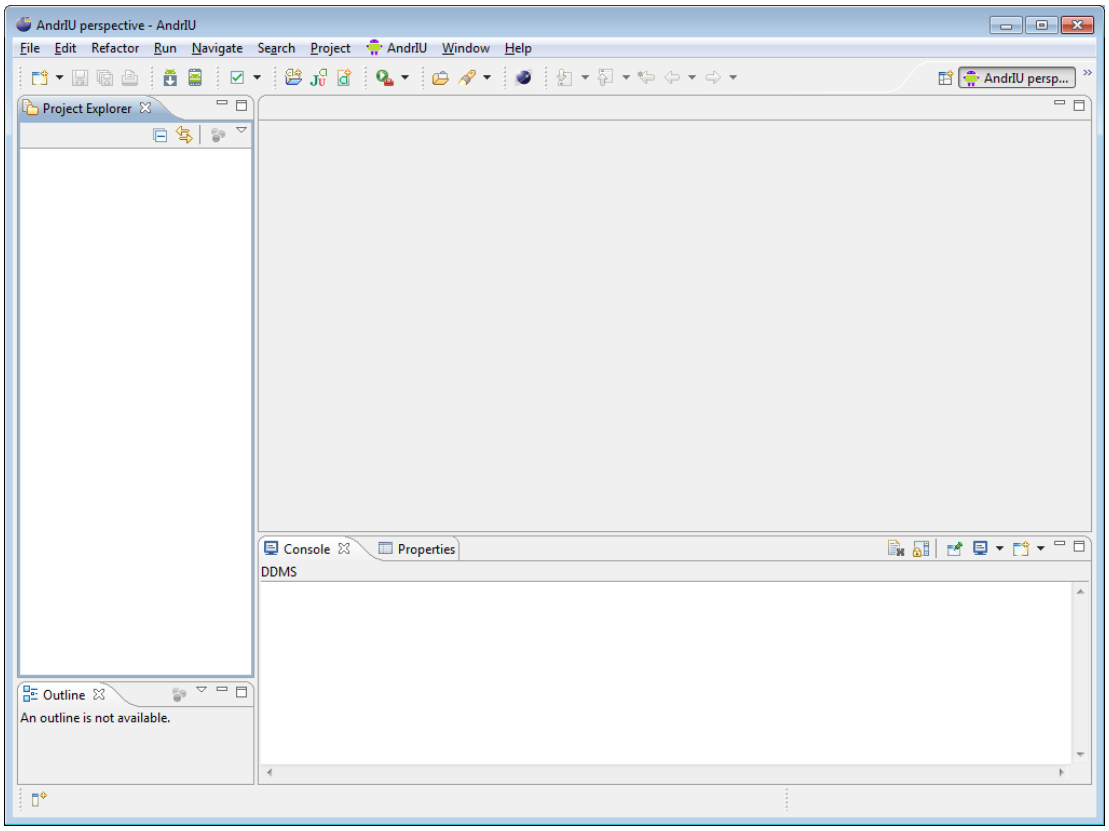


Figura 8.5. Vista de Eclipse con la perspectiva AndrIU

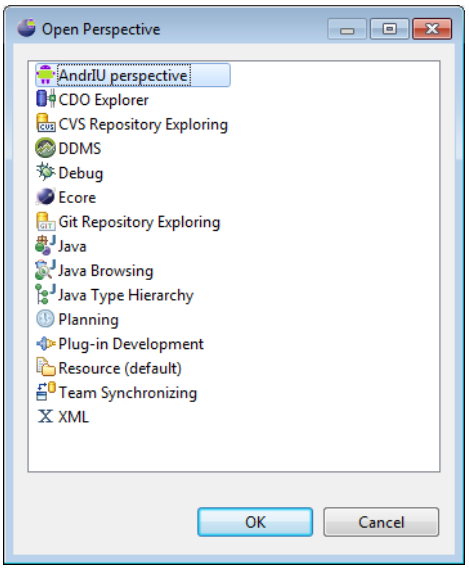


Figura 8.6. Ventana de selección de perspectivas

8.1.1. Configurando el entorno AndrIU

Una vez se tenga el entorno con la perspectiva AndrIU, lo primero que se debe hacer es configurar las opciones para generar los modelos. Para ello, hay que ir a la ventana de preferencias de AndrIU de la manera habitual (en *Window>preferences*), y seleccionar dicha página de preferencias (véase Figura 8.7). En esta ventana hay que seleccionar un modelo de transformación por defecto, que la herramienta tomará posteriormente a la hora de generar proyectos AndrIU. Si no se selecciona ninguno, el modelo de transformación *default.xml* que se genere en los proyectos estará vacío. Para que se pueda seleccionar un modelo de transformación inicial, se incluye el fichero *default.xml* en el directorio de la aplicación. También se puede generar uno como se explica más adelante y seleccionarlo desde la página de preferencias.

Además, se puede configurar una opción para la generación de los modelos KDM de manera que se tome el modelo de transformación *default.xml* o que se pueda seleccionar otro modelo dentro del proyecto. Para ello, hay que deseleccionar la opción "*Get this transformation model as default"*.

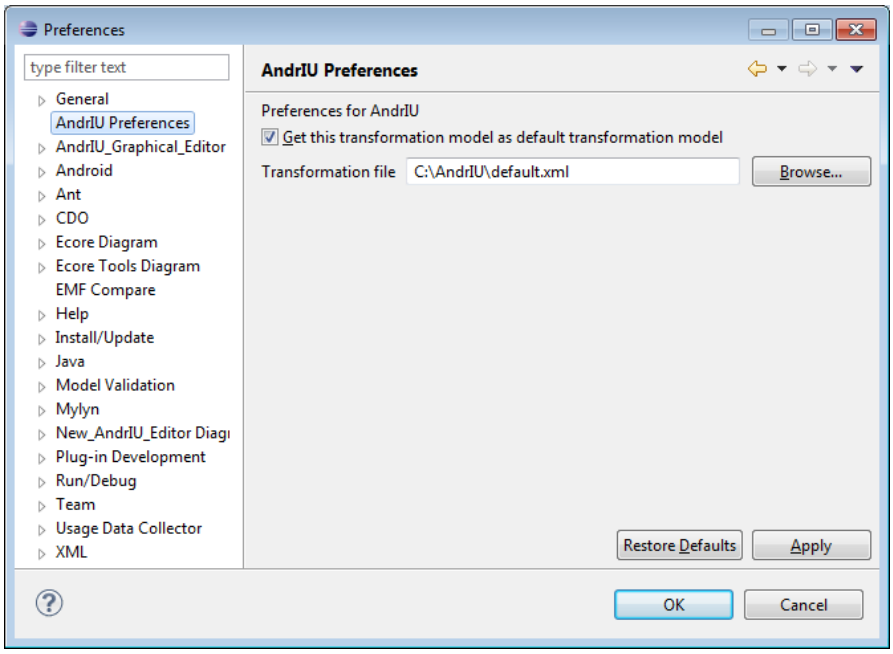


Figura 8.7. Página de preferencias de AndrIU

8.1.2. Creando proyectos AndrIU

La creación de proyectos AndrIU se pueden hacer de dos maneras: creando un nuevo proyecto en blanco desde el *Wizard* o creando un proyecto AndrIU desde un proyecto Java existente. Se recomienda la segunda opción si se dispone del proyecto Java original del que se desean extraer los modelos de IU, ya que la herramienta copia todos los ficheros fuente que se encuentren en el directorio "src" del proyecto Java. En caso contrario, hay que copiar manualmente estos ficheros de código fuente en el directorio "SRC Files" del proyecto AndrIU.

8.1.2.1. Creando proyectos AndrIU desde el Wizard

De esta manera se crea un proyecto AndrIU en blanco (únicamente contiene los directorios necesarios). Para ello, hay que seleccionar "*File>New>Other…"* o pulsar "Ctrl+N" y aparecerá la ventana de selección de Wizard (véase Figura 8.8). Seleccionar la opción de "*AndrIU project" y aparecerá el Wizard de AndrIU* (véase Figura 8.9).

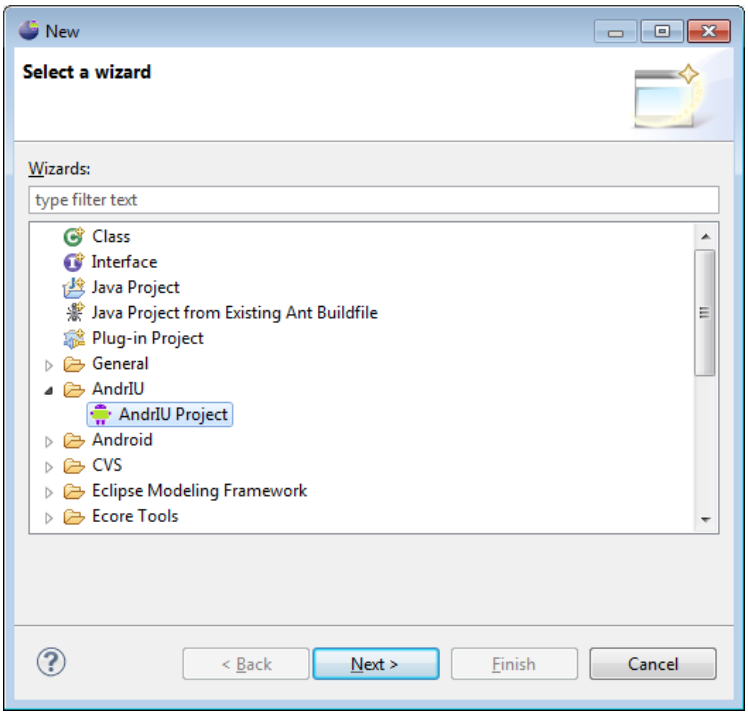


Figura 8.8. Ventana de selección de Wizard

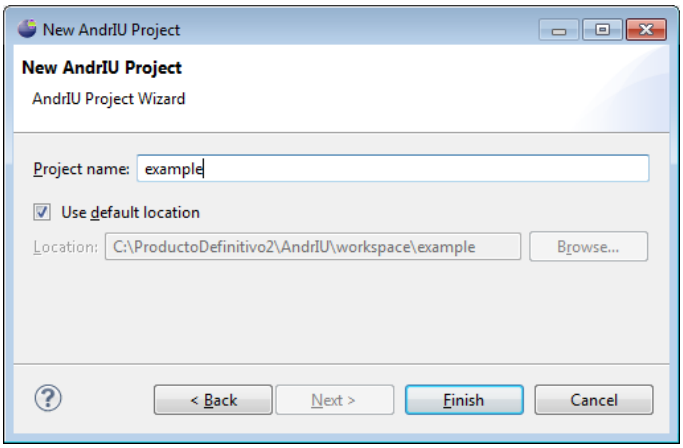


Figura 8.9. Ventana del Wizard de AndrIU

Escribir un nombre para el proyecto AndrIU que se va a crear y pulsar en "Finish". Tras esto, se puede observar en la Figura 8.10 como aparece el nuevo proyecto en blanco creado.

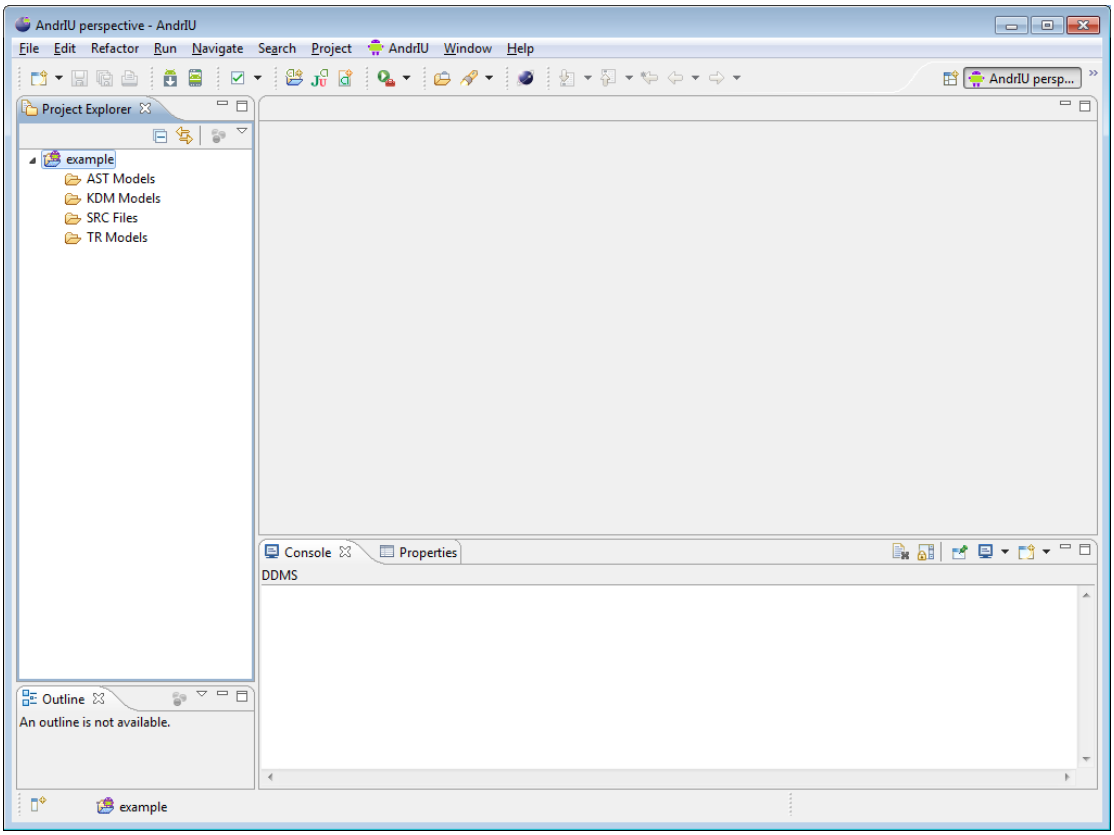


Figura 8.10. Perspectiva AndrIU con un nuevo proyecto en blanco

8.1.2.2. Creando proyectos AndrIU desde un proyecto Java existente

En el caso de tener un proyecto Java, se puede utilizar la opción que ofrece AndrIU para crear un proyecto AndrIU a partir de dicho proyecto Java. Esto se puede hacer desde el menú de AndrIU que aparece en la barra de menús o desde el menú contextual que aparece al pulsar el botón derecho del ratón sobre el proyecto. De ahora en adelante, el resto de funciones que ofrece AndrIU se pueden hacer de estas dos maneras, y el usuario puede utilizar la que prefiera. Tener en cuenta que para utilizar las opciones del menú de la barra de menús hay que tener seleccionado igualmente el proyecto o fichero sobre el que se va a realizar la operación.

Para crear un proyecto AndrIU desde un proyecto Java existente, seleccionar la opción "*Convert to AndrIU*" (véase Figura 8.11). El nuevo proyecto AndrIU creado tendrá el mismo nombre que el proyecto Java seguido de "*_AndrIU*", y contendrá los directorios necesarios (véase Figura 8.12). En el directorio "SRC Files" se almacenará una copia de todos los ficheros .java que el proyecto Java original tenía en el directorio "src". Además, en el directorio "TR Models" se creará el modelo de transformación

default.xml que se habrá configurado previamente (véase sección 8.1.1).

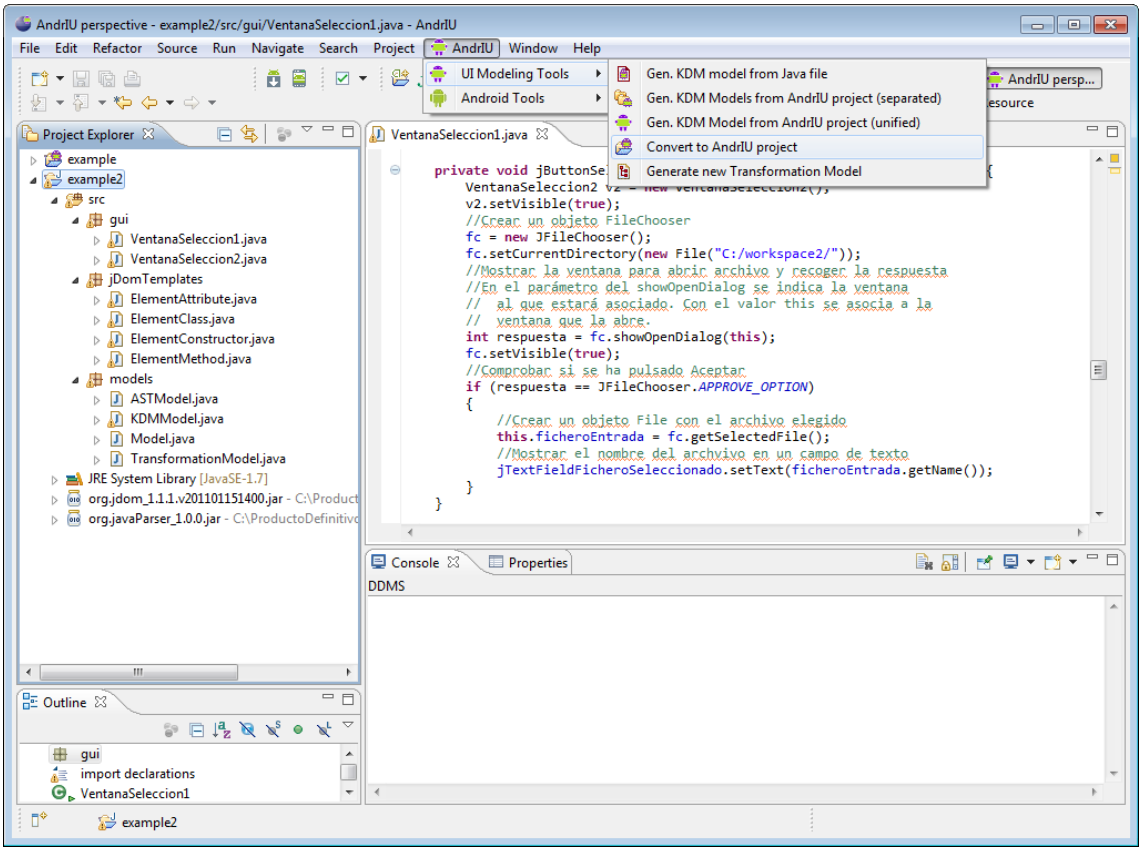


Figura 8.11. Opción de crear un proyecto AndrIU desde un proyecto Java

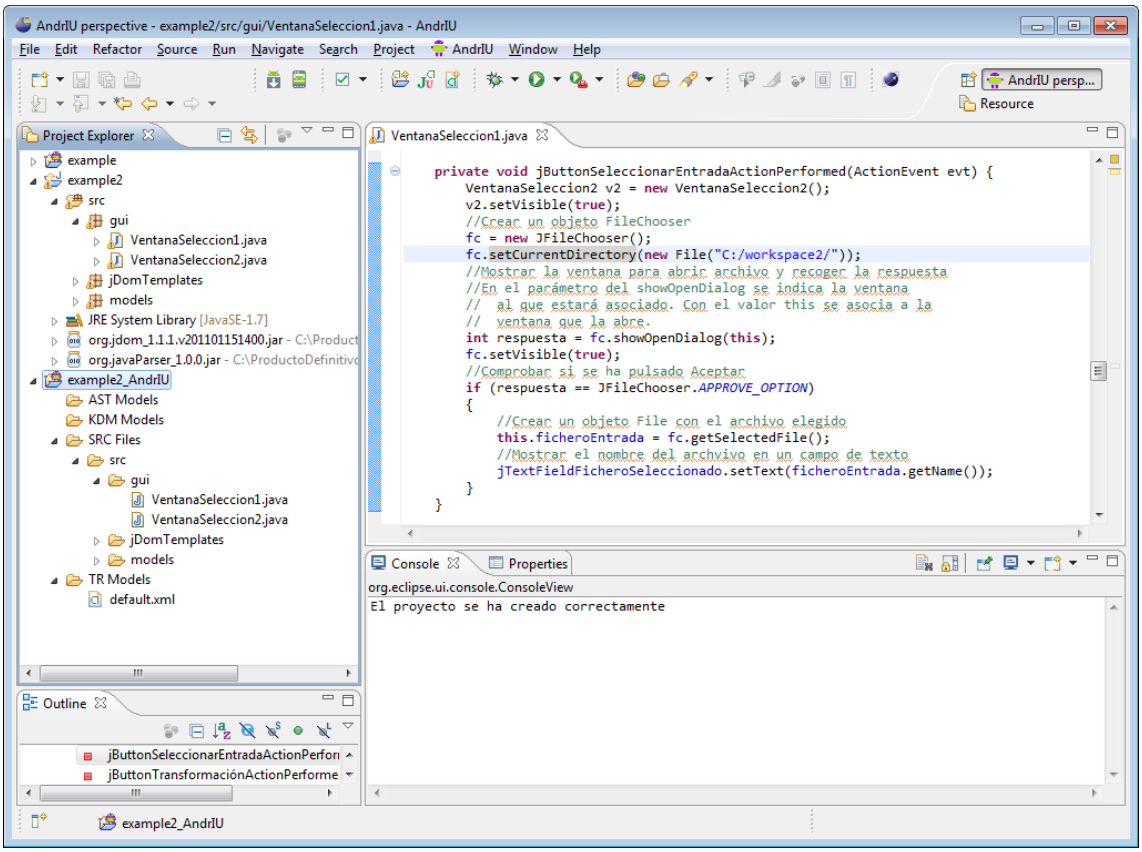


Figura 8.12. Proyecto AndrIU creado a partir de un proyecto Java

8.1.3. Generando modelos de transformación

Los modelos de transformación se almacenan en el directorio *TR Models* de un proyecto AndrIU, por tanto es necesario tener algún proyecto AndrIU para poder generar un modelo de transformación. Para generar un nuevo modelo de transformación, hay que seleccionar el proyecto AndrIU donde se almacenará el nuevo modelo y elegir la opción del menú o del menú contextual "*Generate new Transformation Model"* (véase Figura 8.13).

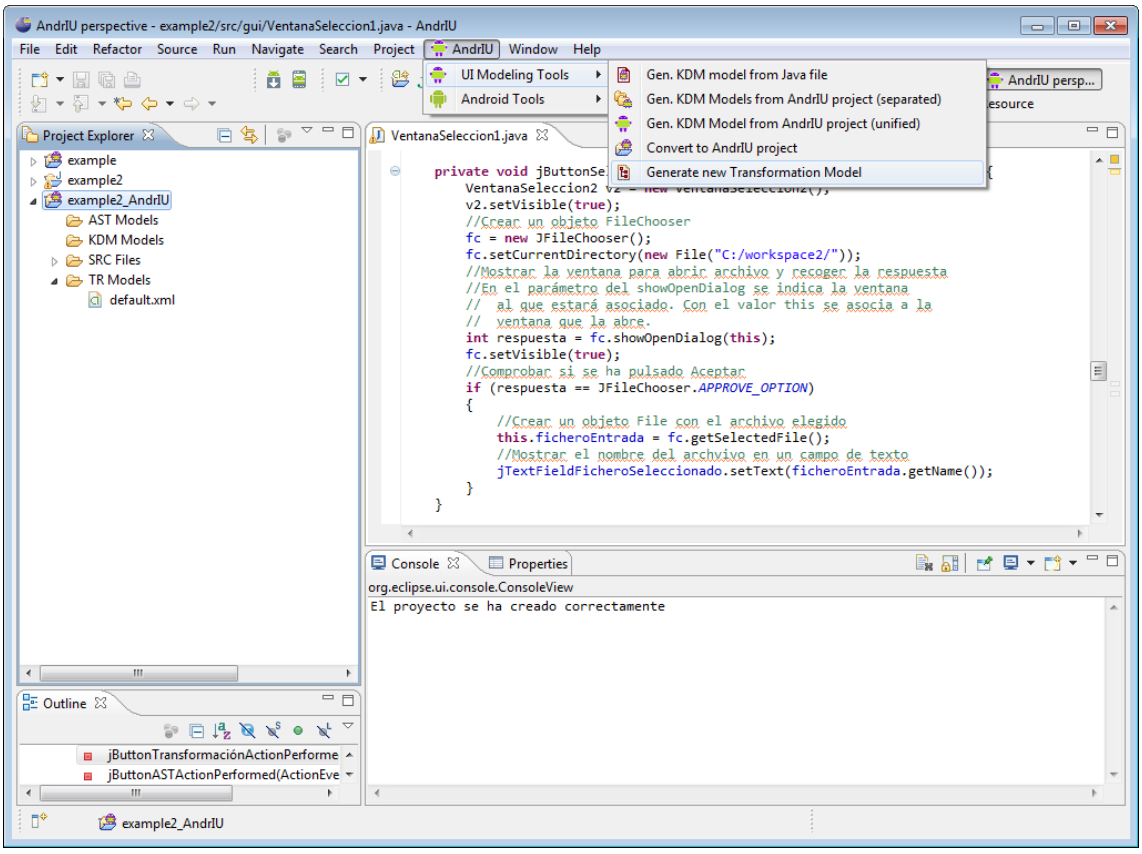


Figura 8.13. Opción de generar nuevo modelo de transformación

Tras esto, aparecerá la ventana de la Figura 8.14, donde hay que seleccionar los elementos de GUI que se desean incluir en el nuevo modelo de transformación, escribir un nombre para dicho fichero y pulsar "OK". Tras esto, el fichero que contiene el nuevo modelo de transformación se habrá creado en el directorio correspondiente (véase Figura 8.15).

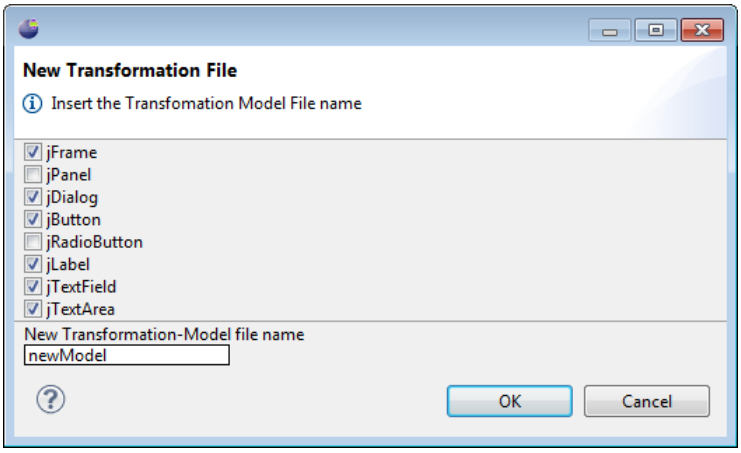


Figura 8.14. Ventana "Nuevo modelo de transformación"

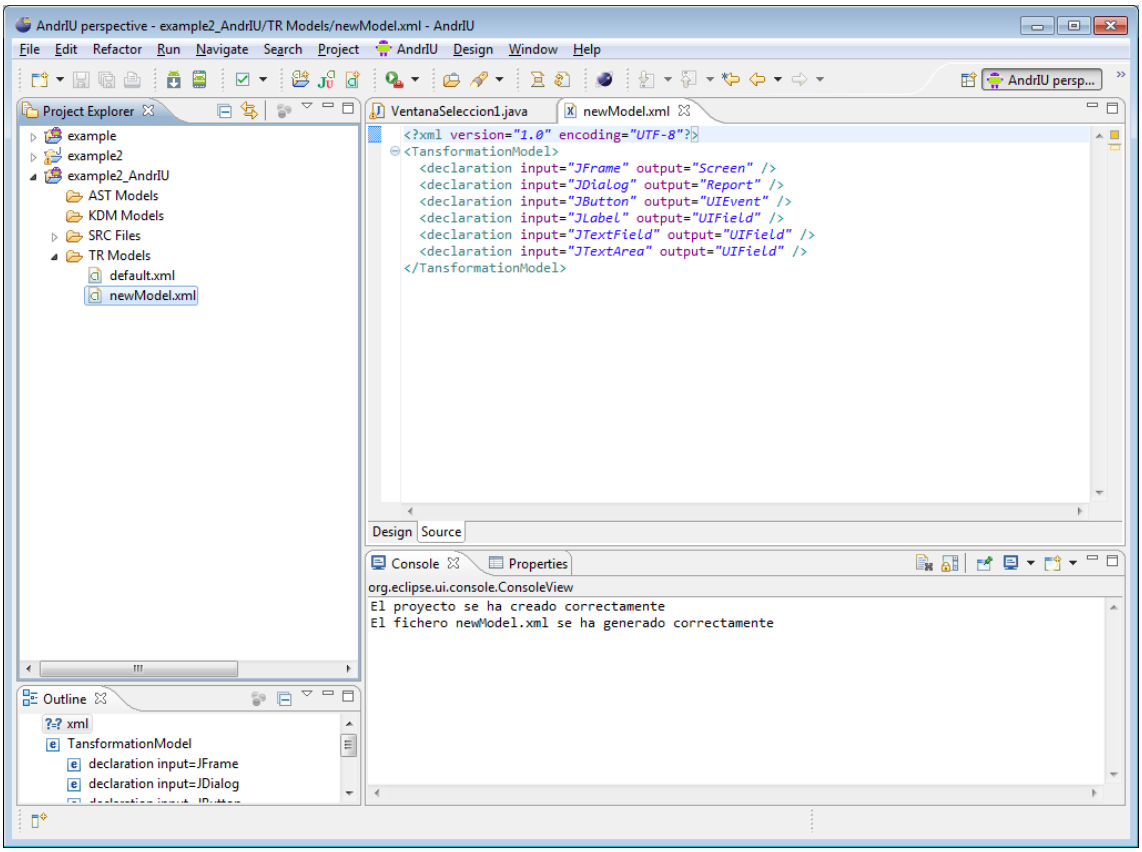


Figura 8.15. Nuevo modelo de transformación creado

8.1.4. Generando modelos KDM

Al igual que con los modelos de transformación, para generar modelos KDM hay que tener un proyecto AndrIU desde el que se van a generar dichos modelos. Al generar los modelos KDM también se generan los modelos AST, ya que son un paso intermedio. La generación de modelos KDM se pueden hacer de dos formas: de un fichero *.java* que se encuentre en el proyecto AndrIU, o de todos los ficheros *.java* que se encuentren en el directorio *SRC Files* del proyecto AndrIU.

8.1.4.1. Generando el modelo KDM de un fichero Java

Para generar el modelo KDM de un fichero *.java*, hay que seleccionarlo y elegir la opción del menú o del menú contextual "*Generate KDM model from Java file*" (véase Figura 8.16). Si la opción de configuración no es la de seleccionar el modelo de transformación por defecto (véase apartado 8.1.1), la ventana de la Figura 8.17 aparecerá para seleccionar alguno de los modelos de transformación del proyecto

AndrIU. Si no se dispone de ninguno, hay que generarlo primero (véase apartado 8.1.3). Tras esto, y como se puede observar en la Figura 8.18, los modelos KDM y AST se han generado en sus directorios correspondientes.

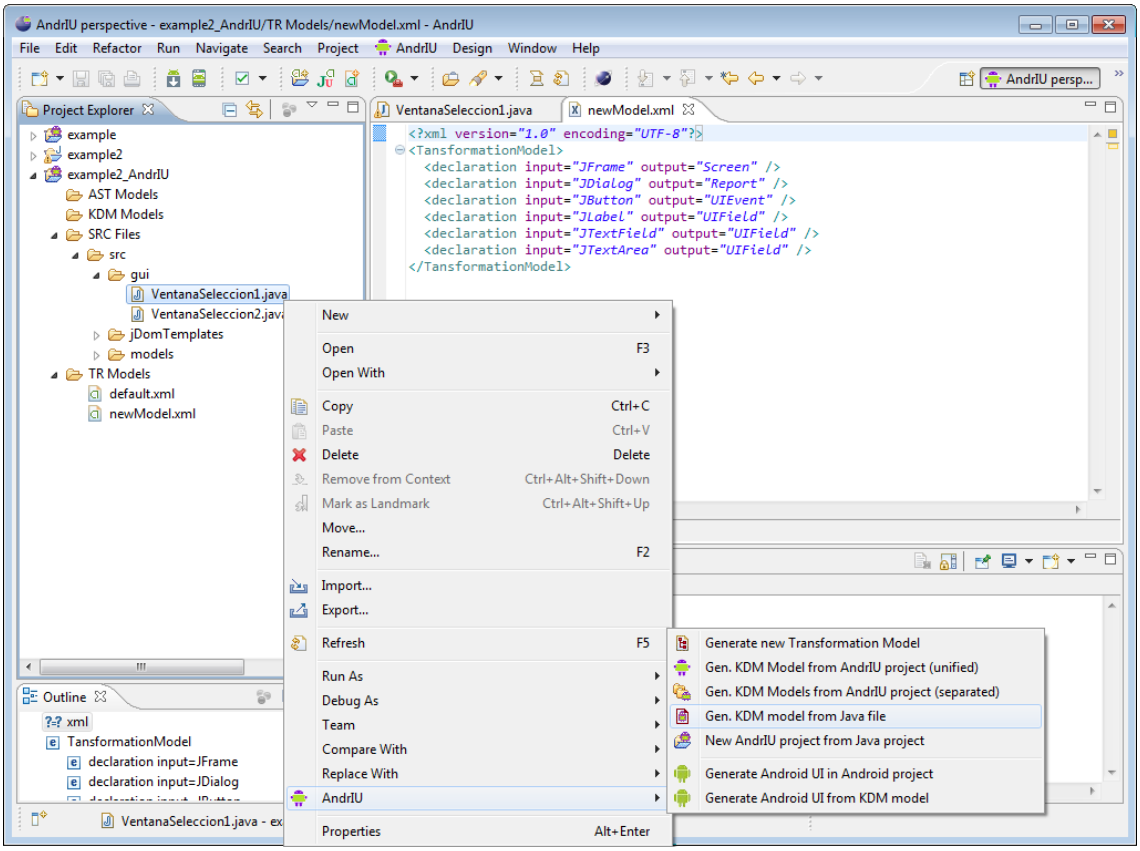


Figura 8.16. Opción del menú contextual para generar modelos KDM desde un fichero Java

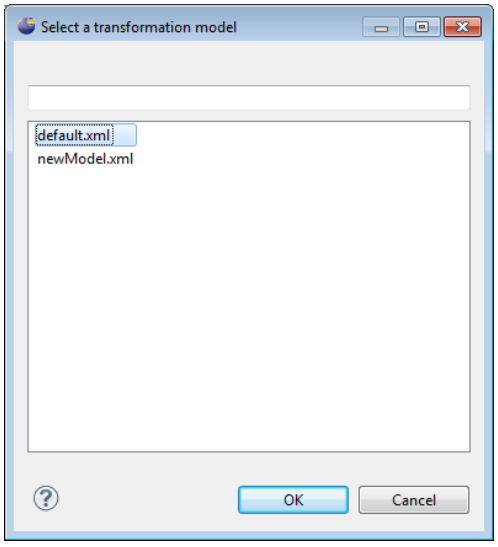


Figura 8.17. Ventana de selección del modelo de transformación

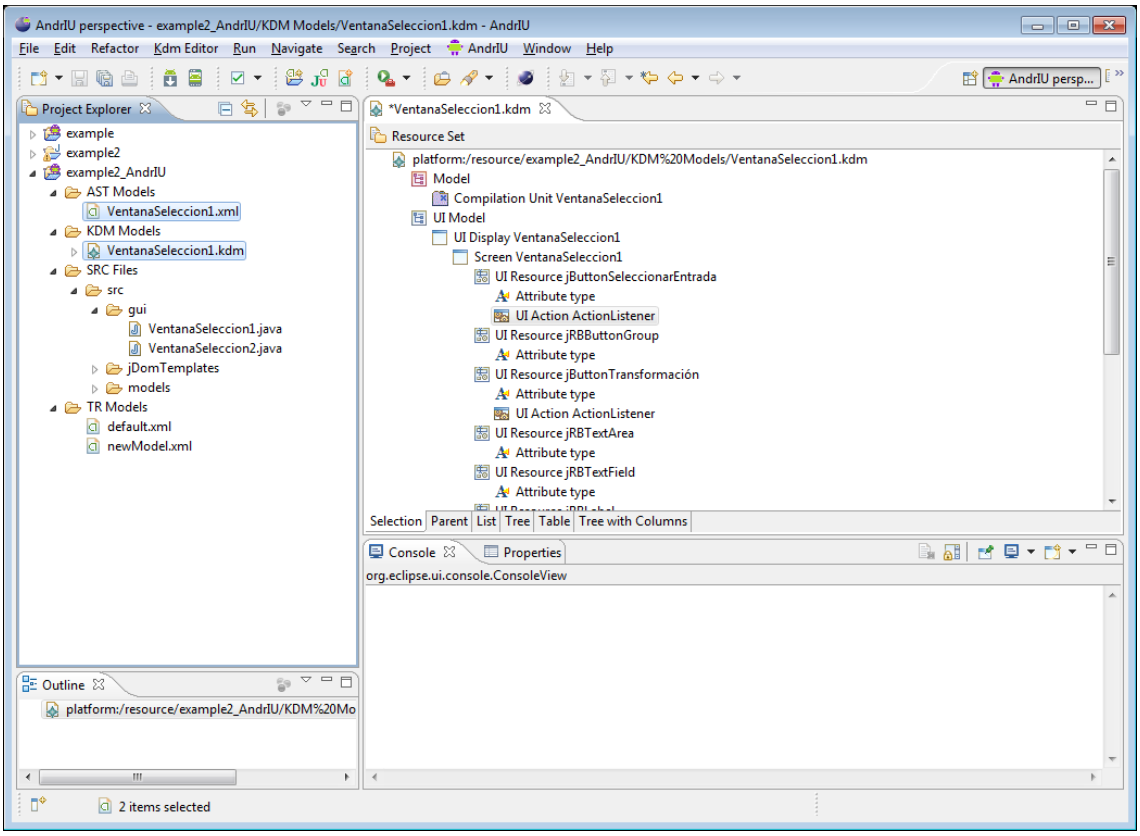


Figura 8.18. Modelos AST y KDM generados del fichero Java

8.1.4.2. Generando modelos KDM de un proyecto AndrIU

Para generar los modelos KDM de los ficheros Java de un proyecto AndrIU, se tienen dos opciones: generar un modelo KDM independiente para cada fichero *.java*, o generar un único modelo KDM que contenga los elementos de todos los ficheros *.java*. Ambas operaciones se realizan seleccionando el proyecto AndrIU y eligiendo la opción respectiva del menú o del menú contextual que ofrece la herramienta *AndrIU* (véase Figura 8.19 y Figura 8.20). En cualquier caso, habrá que elegir el modelo de transformación a utilizar si se ha configurado de este modo y los modelos AST y KDM se generarán en los directorios correspondientes.

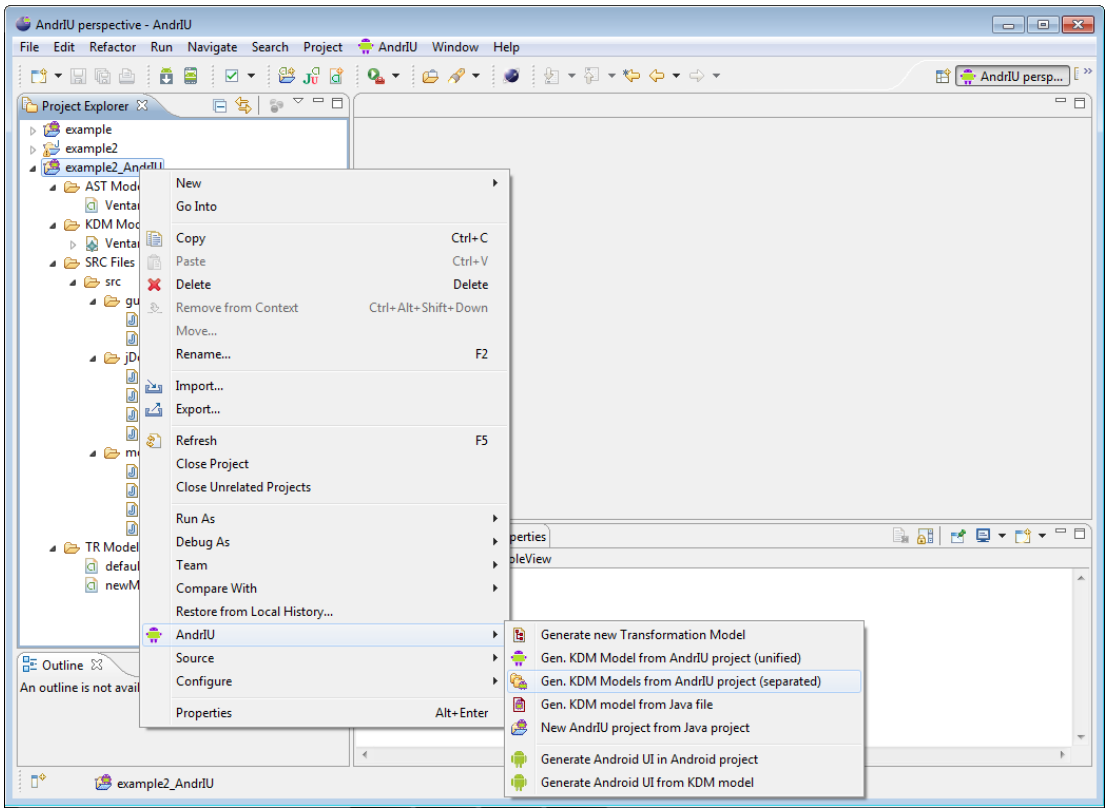


Figura 8.19. Opción del menú contextual para la generación de modelos KDM separados

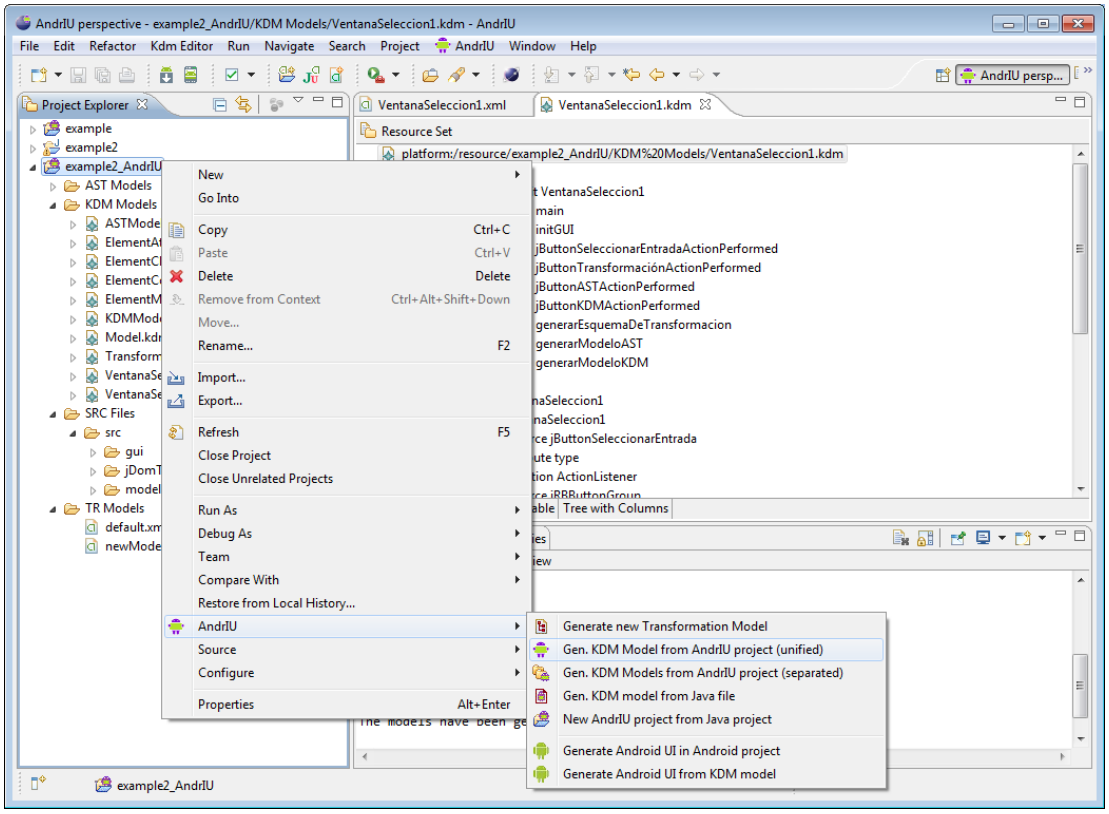


Figura 8.20. Opción del menú contextual para la generación del modelo KDM unificado

Para la generación de modelos separados, cada modelo KDM tendrá el mismo nombre que el fichero de código fuente, mientras que para la generación del modelo unificado, el modelo KDM tendrá el nombre del propio proyecto (véase Figura 8.21). Los modelos AST también se generan en el directorio *"AST Models"*.

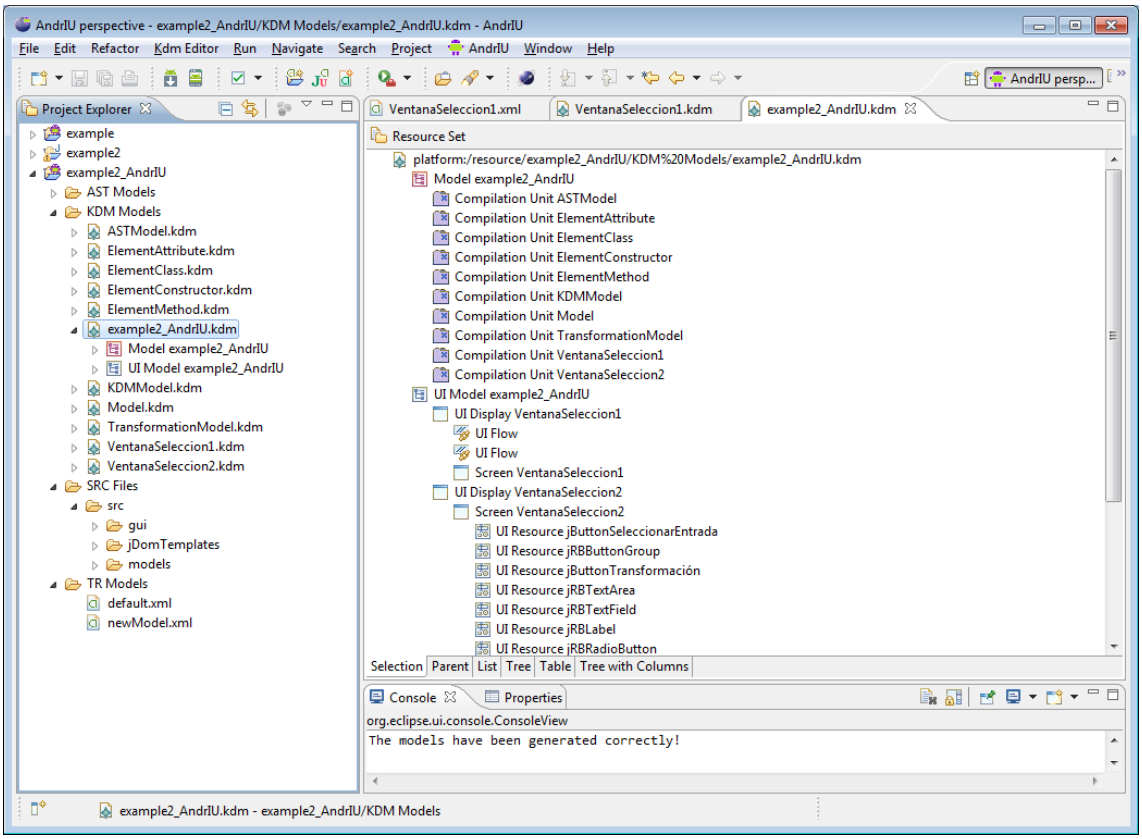


Figura 8.21. Modelos KDM generados en el proyecto AndrIU

8.1.5. Generando interfaces de usuario para Android

Para generar los ficheros que contienen las GUI para Android, existen dos posibilidades: generarlos en un nuevo directorio del propio proyecto AndrIU, o generarlos en un proyecto Android existente.

8.1.5.1. Generando UI Android en un proyecto AndrIU

Para generar un fichero de GUI para Android en el propio proyecto AndrIU, hay que seleccionar el modelo KDM del que se quiere generar y a continuación seleccionar la opción del menú o del menú contextual *"Generate Android UI in AndrIU Project"* (véase Figura 8.22). Tras esto, se creará un nuevo directorio *"Android UI"* en el

proyecto AndrIU, que contendrá los ficheros de GUI que se han generado (véase Figura 8.23).

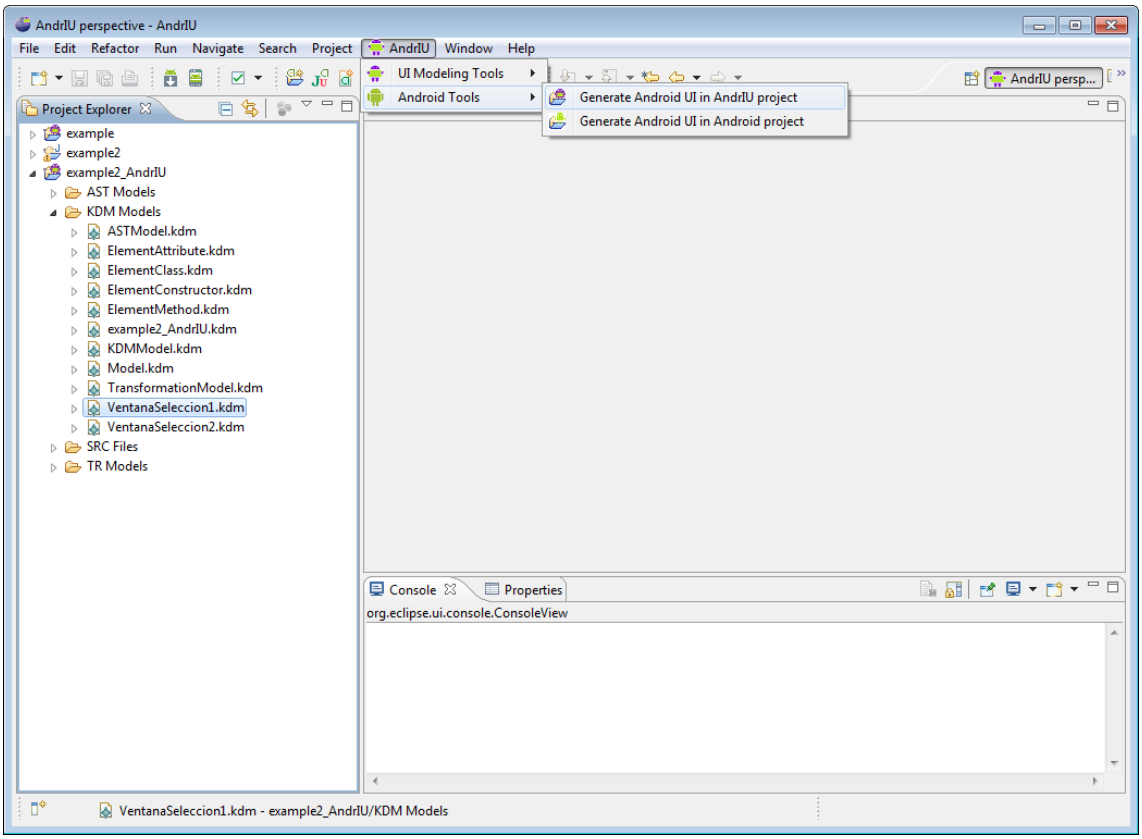


Figura 8.22. Opción del menú para generar GUI para Android en el proyecto AndrIU

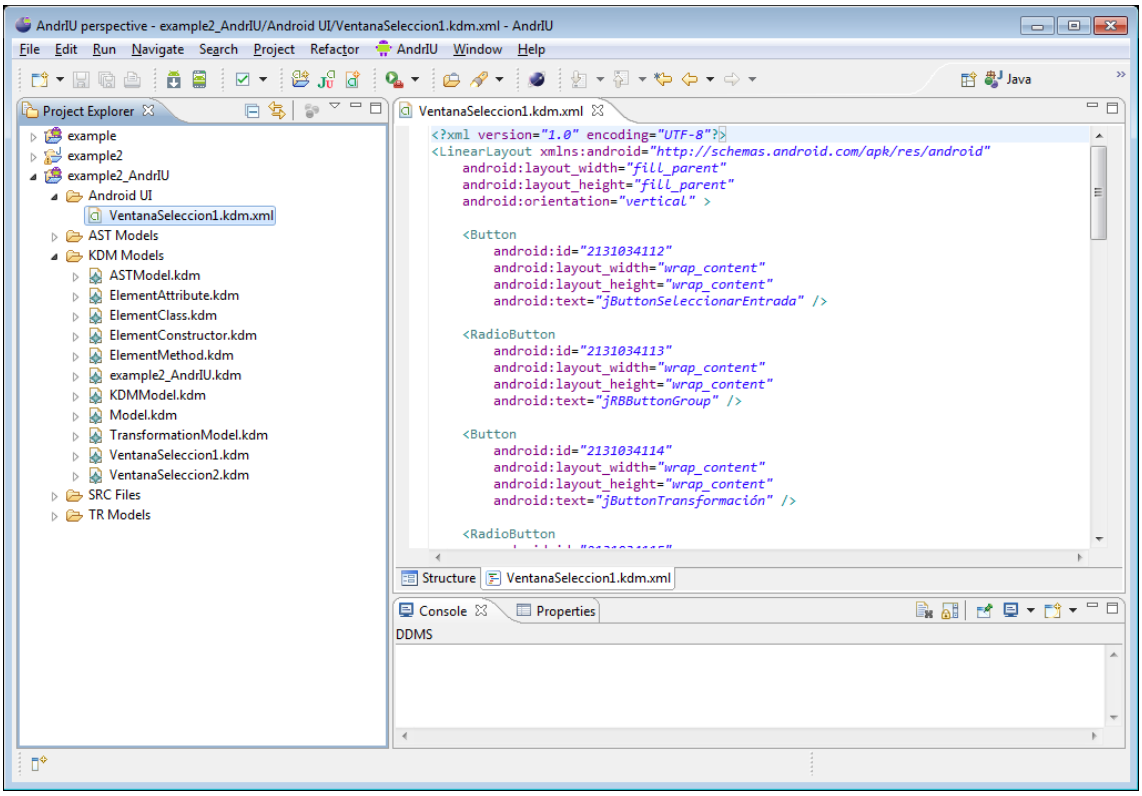


Figura 8.23. GUI para Android generada en el proyecto AndrIU

8.1.5.2. Generando UI Android en un proyecto Android

Para generar ficheros de GUI para Android en un proyecto Android, hay que disponer primero de un proyecto Android en el workspace. Si no se dispone de uno, se puede crear de la forma habitual pulsando *"Ctrl+N"* o seleccionando *"New>Other…"* y eligiendo *"new Android Project"* (véase Figura 8.24) . A continuación seleccionar el modelo KDM del que se quieren generar los ficheros de GUI y a continuación seleccionar la opción del menú o del menú contextual *"Generate Android UI in Android Project"* (véase Figura 8.25). Tras esto, se pedirá que se seleccione el proyecto Android donde se van a generar los ficheros (véase Figura 8.26) y a continuación el fichero .java donde se definen las declaraciones de los elementos que conforman la interfaz de usuario (por defecto se llama *R.java*) (véase Figura 8.27). Finalmente, los ficheros se habrán generado en el directorio *"res/layout"* del proyecto Android, que es donde se almacenan las interfaces de usuario y los elementos de la interfaz de usuario se habrán definido en el fichero java seleccionado (véase Figura 8.28).

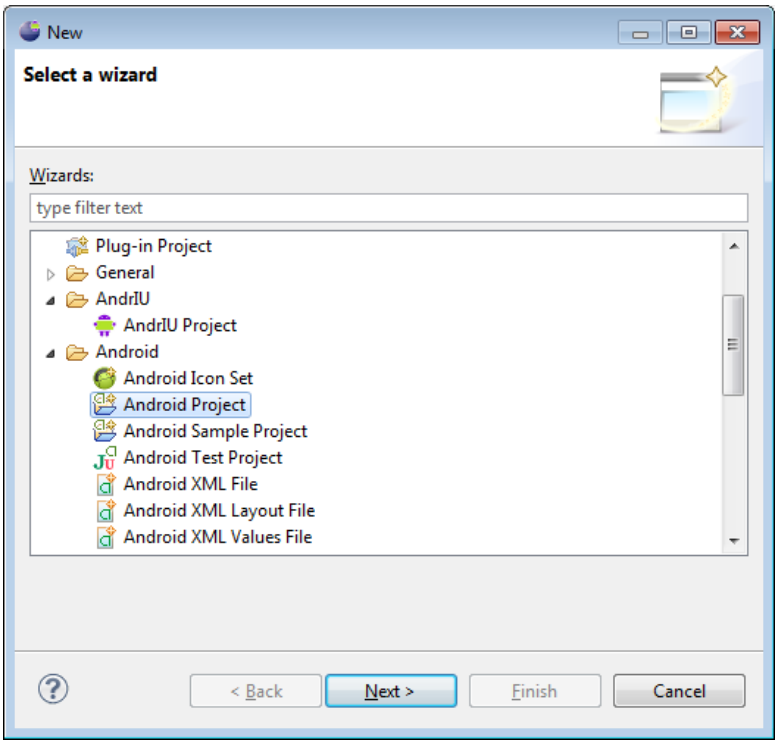


Figura 8.24. Ventana ge selección de Wizard: Android Project

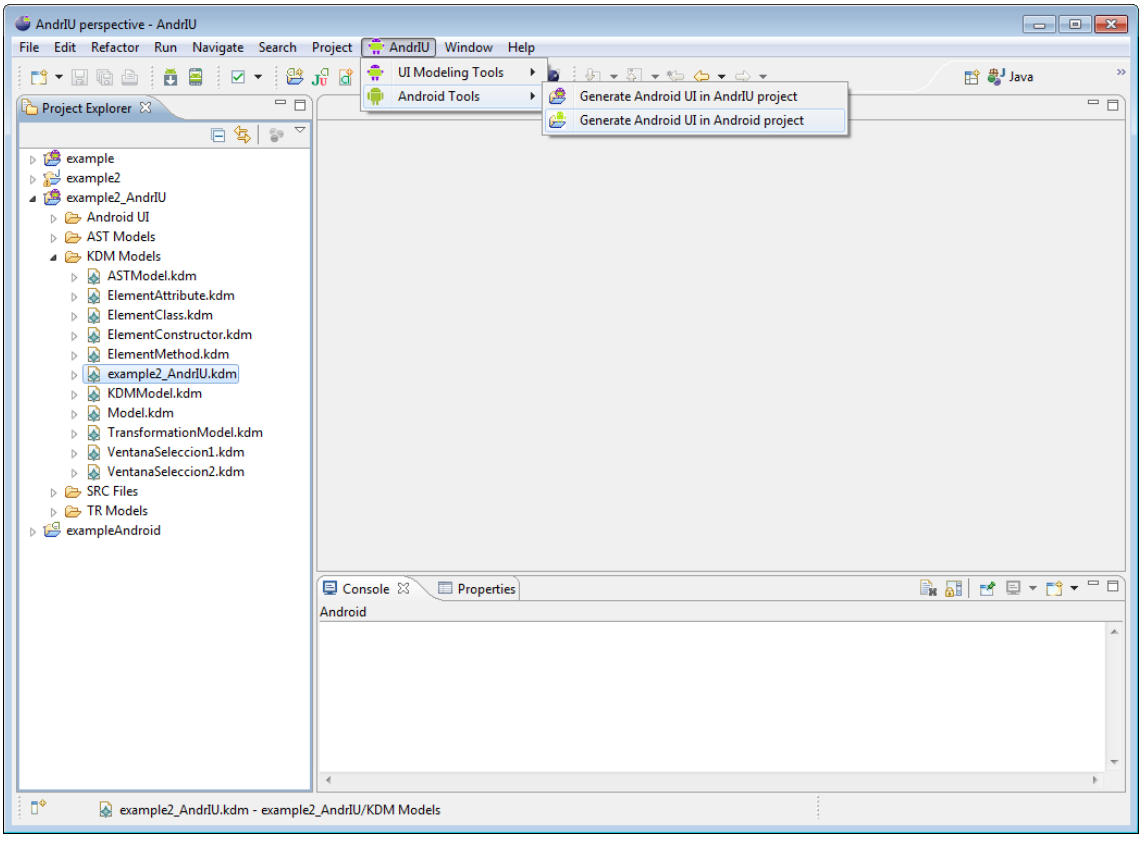


Figura 8.25. Opción de generar Android UI en proyecto Android

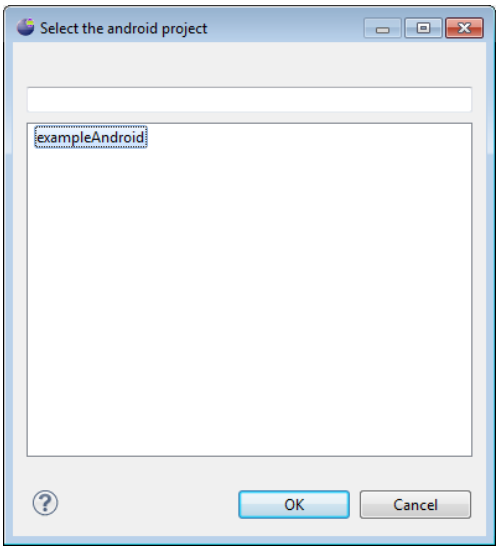


Figura 8.26. Ventana de selección de proyecto Android

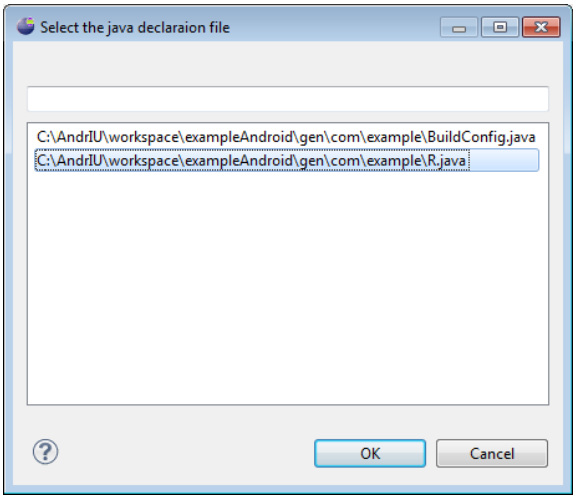


Figura 8.27. Ventana de selección del fichero de declaración de elementos (R.java)

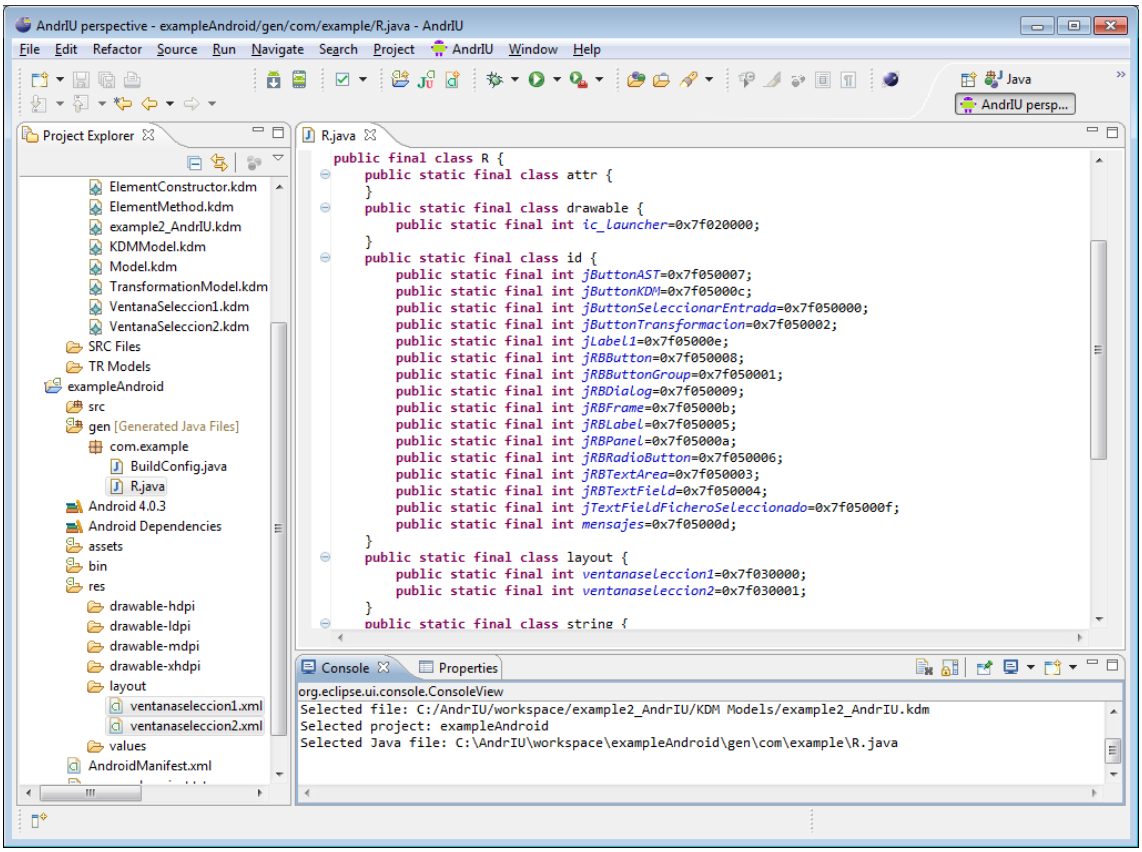


Figura 8.28. Vista de AndrIU con los ficheros generados en el proyecto Android

8.1.6. Mostrando y editando los modelos KDM generados

Para utilizar el editor no gráfico que ofrece *AndrIU*, únicamente hay que hacer doble clic sobre el modelo KDM que se desee, ya que es el editor por defecto para estos modelos. También se puede hacer clic derecho sobre el modelo y utilizar la opción *"Open with > KDM Model Editor"*.

Para utilizar el editor gráfico de *AndrIU*, hay que generar primero el fichero del diagrama del modelo KDM. Para ello hay que hacer clic derecho sobre el modelo KDM y seleccionar *"Initialize kdm_diagram diagram file"* (véase Figura 8.29). A continuación aparecerá la ventana de la Figura 8.30, donde podemos cambiar el nombre al fichero del diagrama que se va a generar, y pulsar *"Next"*. En la siguiente ventana hay que seleccionar el *"UI Model"* (véase Figura 8.31), y finalmente se creará el diagrama.

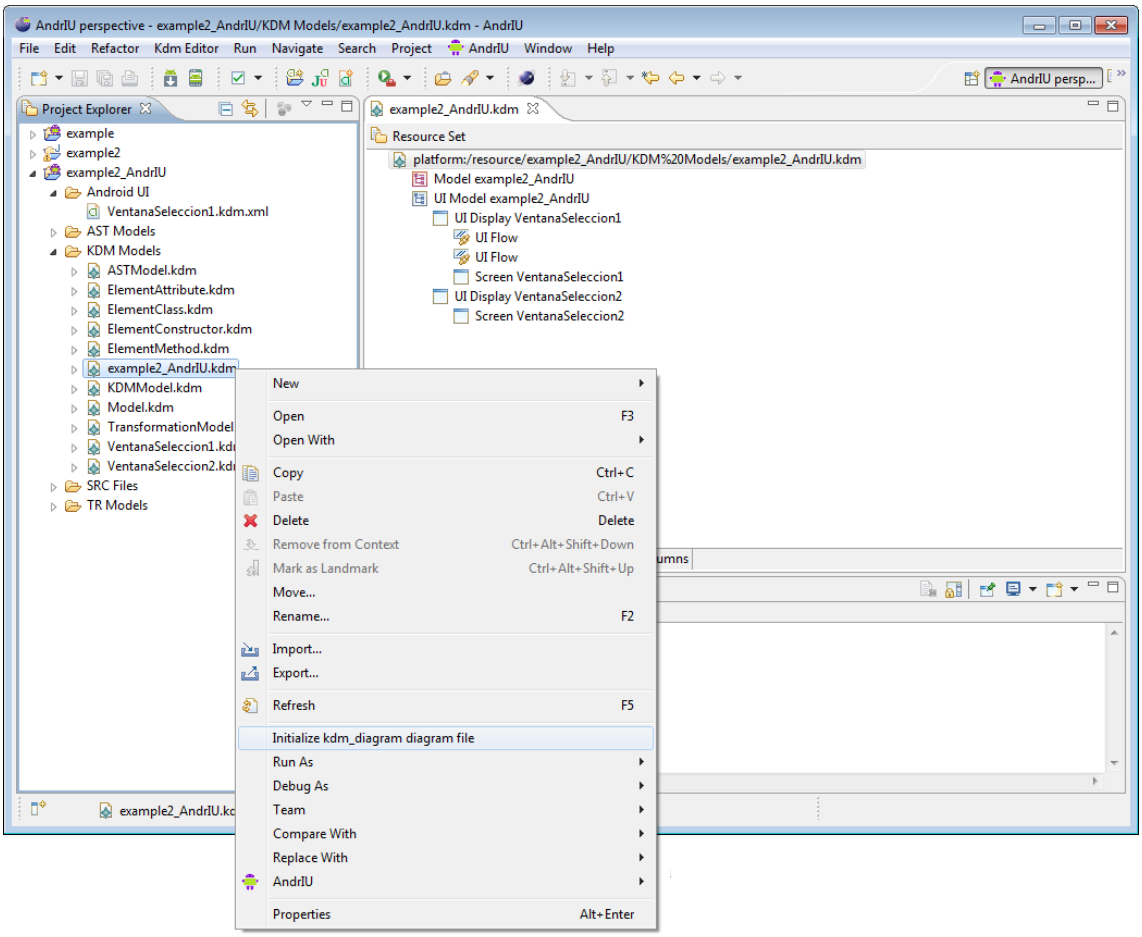


Figura 8.29. Opción del menú contextual para inicializar el diagrama

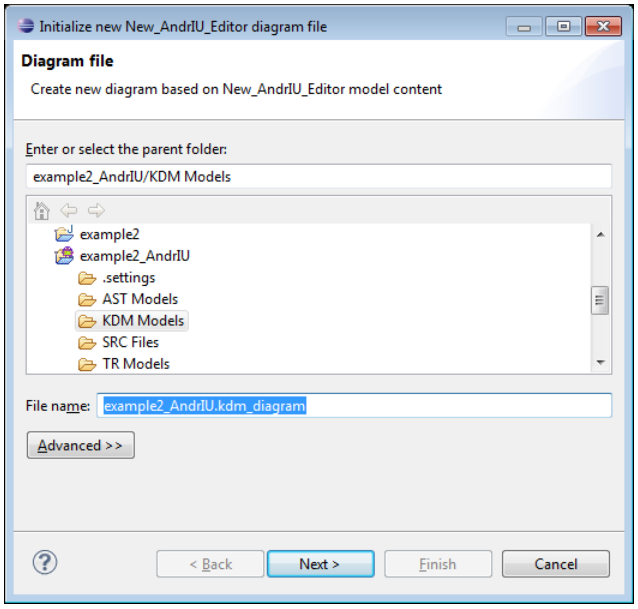


Figura 8.30. Ventana de inicialización de diagrama I (nombre del fichero)

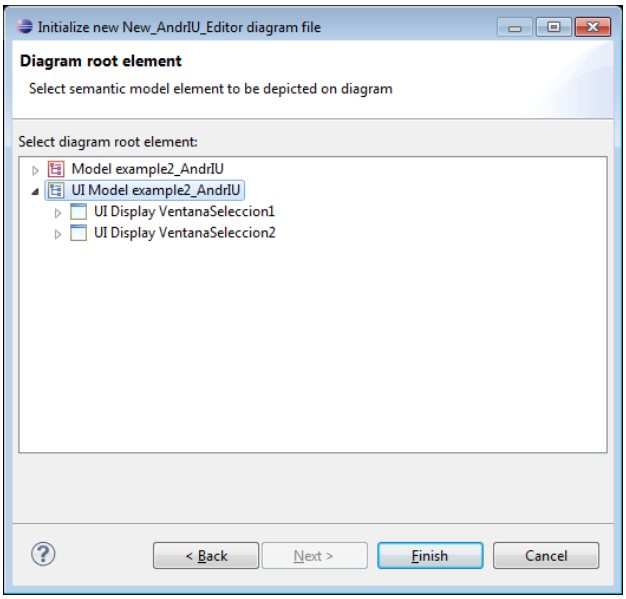


Figura 8.31. Ventana de inicialización de diagrama II (selección del UI Model)

Al hacer doble clic sobre el fichero de diagrama, se abrirá el editor gráfico de *AndrIU* (véase Figura 8.32), en el cual se puede editar el diagrama.

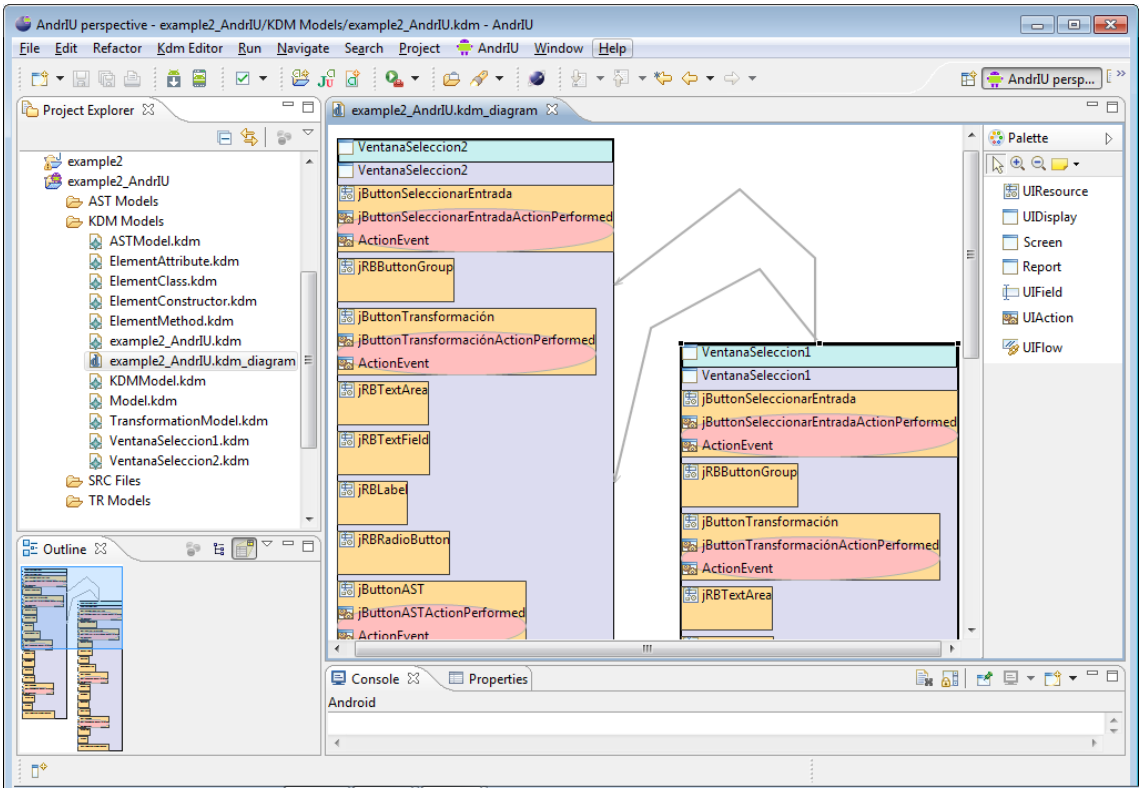


Figura 8.32. Vista del editor gráfico de AndrIU

8.2. Anexo II. Caso de Estudio

En este anexo se presenta un caso de estudio donde se han aplicado las distintas utilidades que ofrece *AndrIU*. Este ejemplo de aplicación se encuadra dentro de las pruebas de aceptación del proyecto (véase apartado 5.9.4.3), como contribución a la prueba de todas las funcionalidades de la herramienta así como del análisis de los resultados obtenidos.

8.2.1. Descripción del sistema *AELG-Socios*

Este caso de estudio corresponde a la aplicación *AELG-Socios*, una aplicación Java de escritorio que gestiona el directorio de contacto, registro y pago de tasas de la asociación de escritores en lengua gallega (http://www.aelg.org/). La GUI de esta aplicación consta de varias ventanas, la mayoría de ellas a modo de formulario. Como ejemplo, la Figura 8.33 muestra una de las interfaces gráficas de usuario para la gestión de las categorías y tasas de los diferentes escritores. La Figura 8.34 muestra la ventana principal de edición de los datos de cada escritor de la asociación.

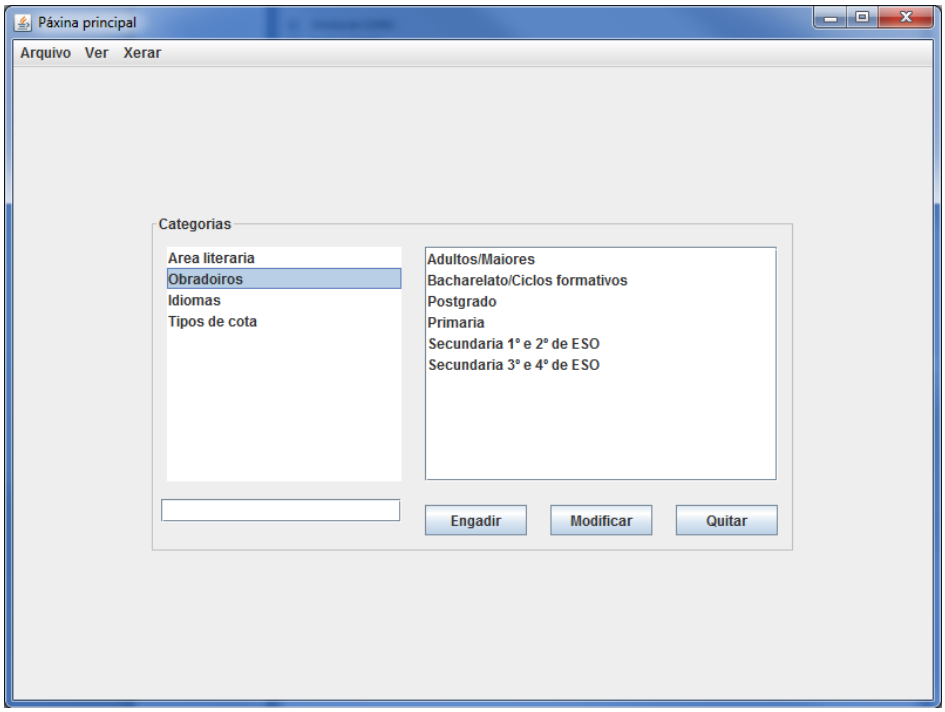


Figura 8.33. Ejemplo de GUI de *AELG-Socios (I)*

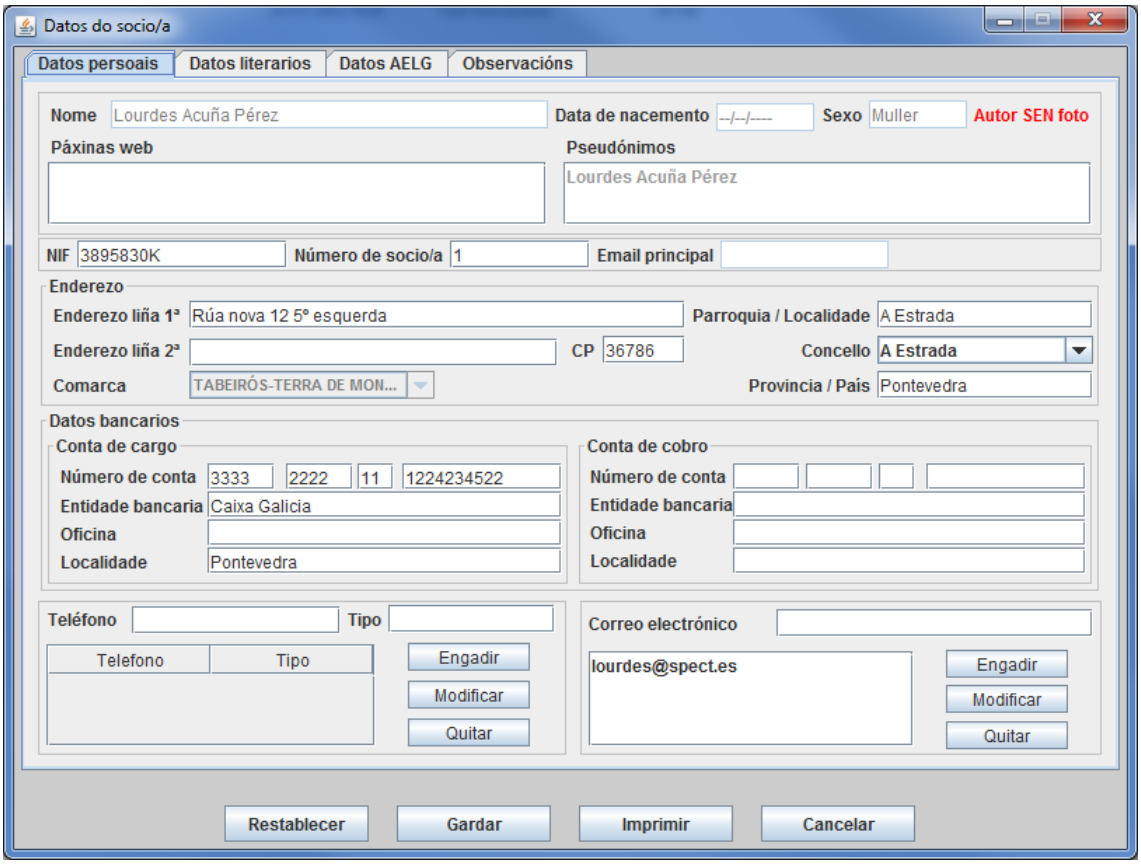


Figura 8.34. Ejemplo de GUI de *AELG-Socios (II)*

8.2.2. Extracción de los modelos de GUI

Para realizar la extracción de los modelos de la aplicación *AELG-Socios*, se ha creado un proyecto Java y se han importado los recursos necesarios. Los pasos a seguir son los siguientes:

8.2.2.1. Crear un proyecto AndrIU a partir del proyecto Java.

El primer paso es la creación de un proyecto AndrIU a partir del proyecto Java que contiene la aplicación. Como se puede observar en la Figura 8.35, el proyecto AndrIU se ha generado correctamente, y contiene en el directorio *"SRC Files"* una copia de todos los ficheros de código fuente que tiene el proyecto Java original en el directorio *"src"*, respetando además toda la jerarquía de paquetes del proyecto original. También se observa el directorio *"TR Models"* con el modelo de transformación por defecto *"default.xml"*.

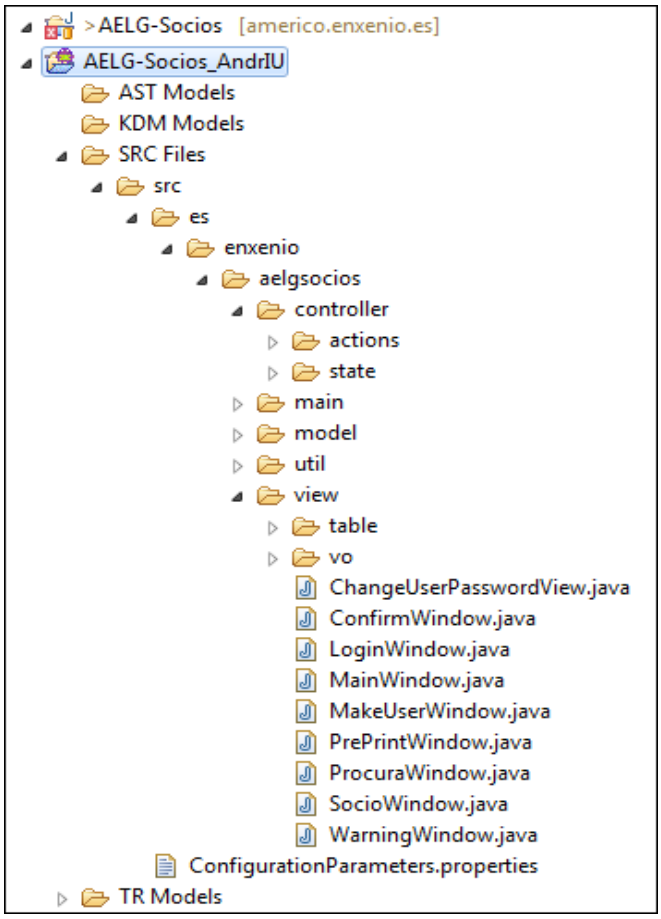


Figura 8.35. Proyecto *AELG-Socios* generado

8.2.2.2. Generar los modelos KDM de todos los ficheros contenidos en el proyecto AndrIU por separado.

Tras la generación del proyecto AndrIU, se generan los modelos KDM por separado de todos los ficheros de código fuente almacenados en el directorio *"SRC Files"*. Tras la generación, se puede observar que se han generado correctamente en el directorio *"AST Models"* todos los modelos AST de los ficheros de código fuente (véase Figura 8.36),y en la Figura 8.37 se puede observar un fragmento de uno de los modelos AST generados. Del mismo modo se observa que los modelos KDM se han generado correctamente en el directorio *"KDM Models"* (véase Figura 8.38) y en la Figura 8.39 se puede observar uno de los modelos KDM generados. Como se esperaba, los modelos KDM que no corresponden a ficheros que contienen elementos de la GUI tienen el modelo *UIModel* vacío, y únicamente se definen en el *CodeModel* la definición de las clases y los métodos.

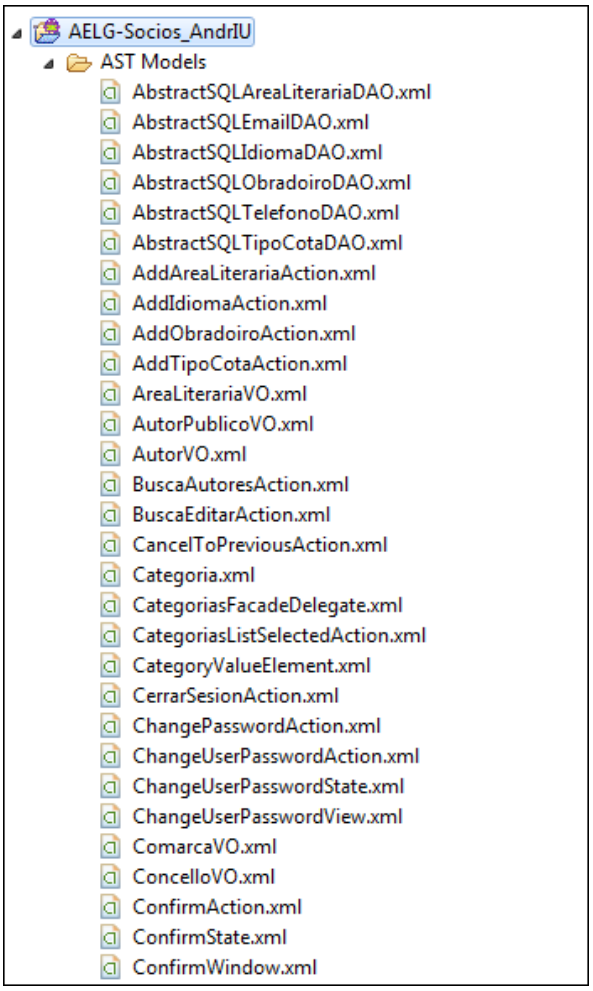


Figura 8.36. Modelos AST generados

```
<Attribute xmi:id="ObservacionsTextArea" name="ObservacionsTextArea" type="JTextArea" />
<Attribute xmi:id="jScrollPane5" name="jScrollPane5" type="JScrollPane" />
<Attribute xmi:id="jLabel36" name="jLabel36" type="JLabel" />
<Attribute xmi:id="TipoCotaComboBox" name="TipoCotaComboBox" type="JComboBox" />
<Attribute xmi:id="jLabel37" name="jLabel37" type="JLabel" />
<Attribute xmi:id="EmailPrincipalField" name="EmailPrincipalField" type="JTextField" />
<Attribute xmi:id="jLabel38" name="jLabel38" type="JLabel" />
<Attribute xmi:id="jComarcasComboBox" name="jComarcasComboBox" type="JComboBox" />
<Attribute xmi:id="jLabel39" name="jLabel39" type="JLabel" />
<Attribute xmi:id="ConcelloComboBox" name="ConcelloComboBox" type="JComboBox" />
<Attribute xmi:id="jPanelCargo25" name="jPanelCargo25" type="JPanel" />
<Attribute xmi:id="jPanelCobro25" name="jPanelCobro25" type="JPanel" />
<Attribute xmi:id="jLabel151" name="jLabel151" type="JLabel" />
<Attribute xmi:id="NumeroConta2Field1" name="NumeroConta2Field1" type="JTextField" />
<Attribute xmi:id="NumeroConta2Field2" name="NumeroConta2Field2" type="JTextField" />
<Attribute xmi:id="NumeroConta2Field3" name="NumeroConta2Field3" type="JTextField" />
<Attribute xmi:id="NumeroConta2Field4" name="NumeroConta2Field4" type="JTextField" />
<Attribute xmi:id="jLabel121" name="jLabel121" type="JLabel" />
<Attribute xmi:id="EntidadeBancaria2Field" name="EntidadeBancaria2Field" type="JTextField" />
<Attribute xmi:id="jLabel141" name="jLabel141" type="JLabel" />
<Attribute xmi:id="Oficina2Field" name="Oficina2Field" type="JTextField" />
<Attribute xmi:id="jLabel131" name="jLabel131" type="JLabel" />
<Attribute xmi:id="LocalidadeBanco2Field" name="LocalidadeBanco2Field" type="JTextField" />
<Constructor xmi:id="SocioWindow" name="SocioWindow">
  <Parameters />
  <Body Class="class domain.javaParser.ast.stmt.ExplicitConstructorInvocationStmt" Code="super();">
    <Statement TIPO="ExpressionStatement">
      <Expression TIPO="MethodCallExpression" name="initialize" />
    </Statement>
  </Body>
</Constructor>
<Method xmi:id="getPreviousState" name="getPreviousState" returnType="DefaultState">
  <Parameters />
  <Body Class="class domain.javaParser.ast.stmt.ReturnStmt" Code="return previousState;" />
</Method>
<Method xmi:id="setPreviousState" name="setPreviousState" returnType="void">
  <Parameters>
    <Attribute xmi:id="previousState" name="previousState" type="DefaultState" />
  </Parameters>
  <Body>
    <Statement TIPO="ExpressionStatement">
```

Figura 8.37. Fragmento de uno de los modelos AST generados

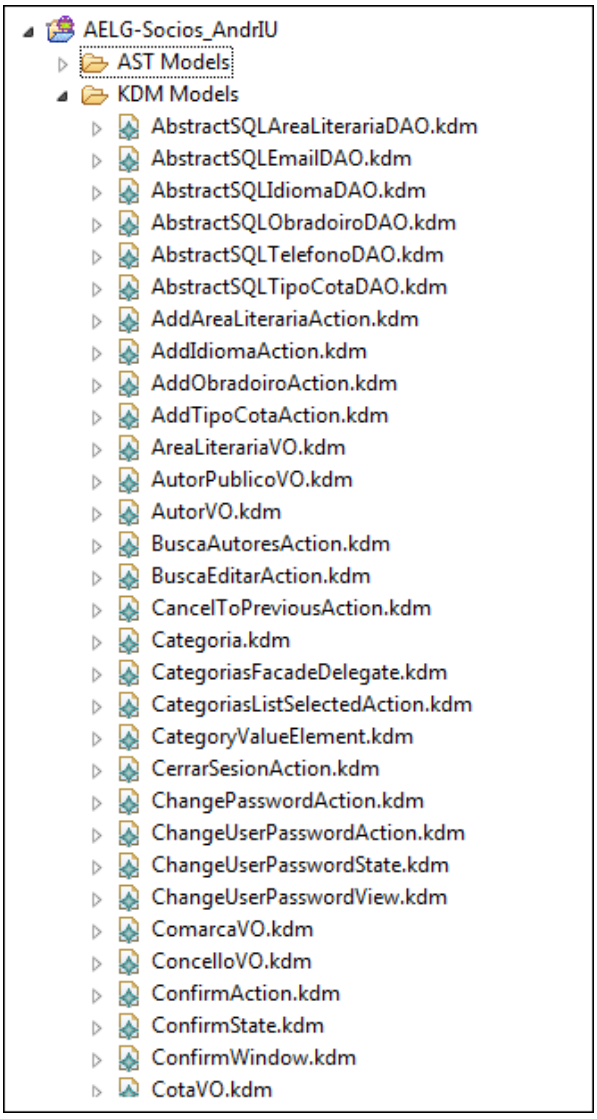


Figura 8.38. Modelos KDM generados

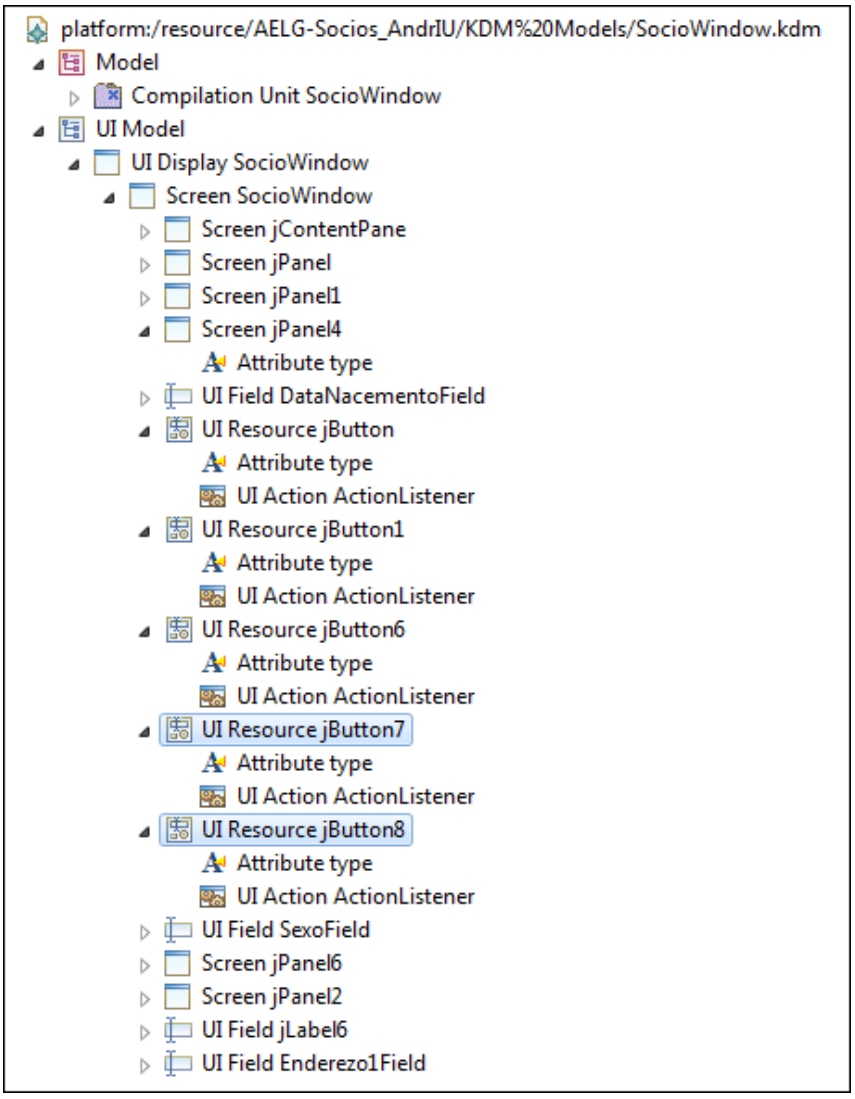


Figura 8.39. Fragmento de uno de los modelos KDM generados

8.2.2.3. Generar el modelo KDM unificado de todos los ficheros contenidos en el proyecto AndrIU.

A continuación se genera el modelo KDM unificado. Como se puede observar en la Figura 8.40, el modelo se ha generado correctamente, conteniendo todos los UIDisplay que conforman la GUI de la aplicación.

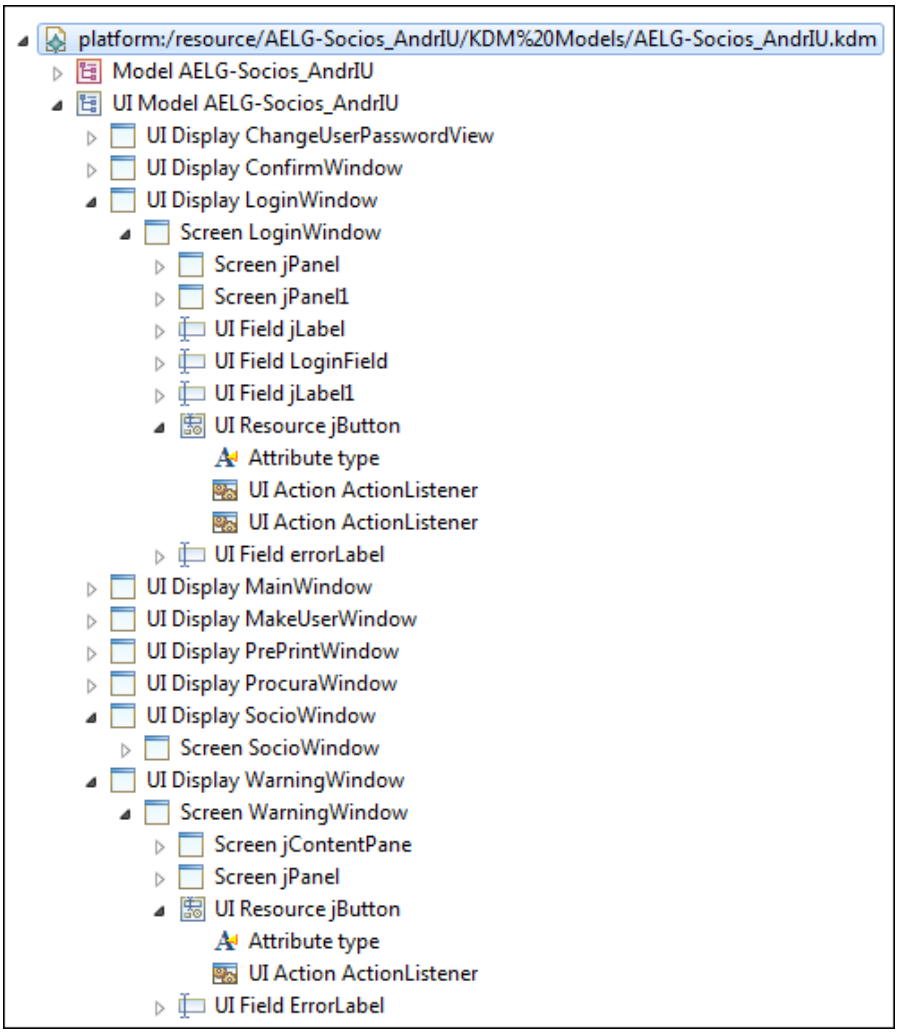


Figura 8.40. Fragmento del modelo KDM unificado generado

8.2.2.4. Generar el fichero de diagrama del modelo KDM unificado y observar los resultados.

Tomando este último modelo KDM, se genera el diagrama para poder observar el modelo con el editor gráfico. Como se observa en la Figura 8.41, el diagrama se genera correctamente y el contenido del mismo se corresponde con el contenido del modelo KDM generado.

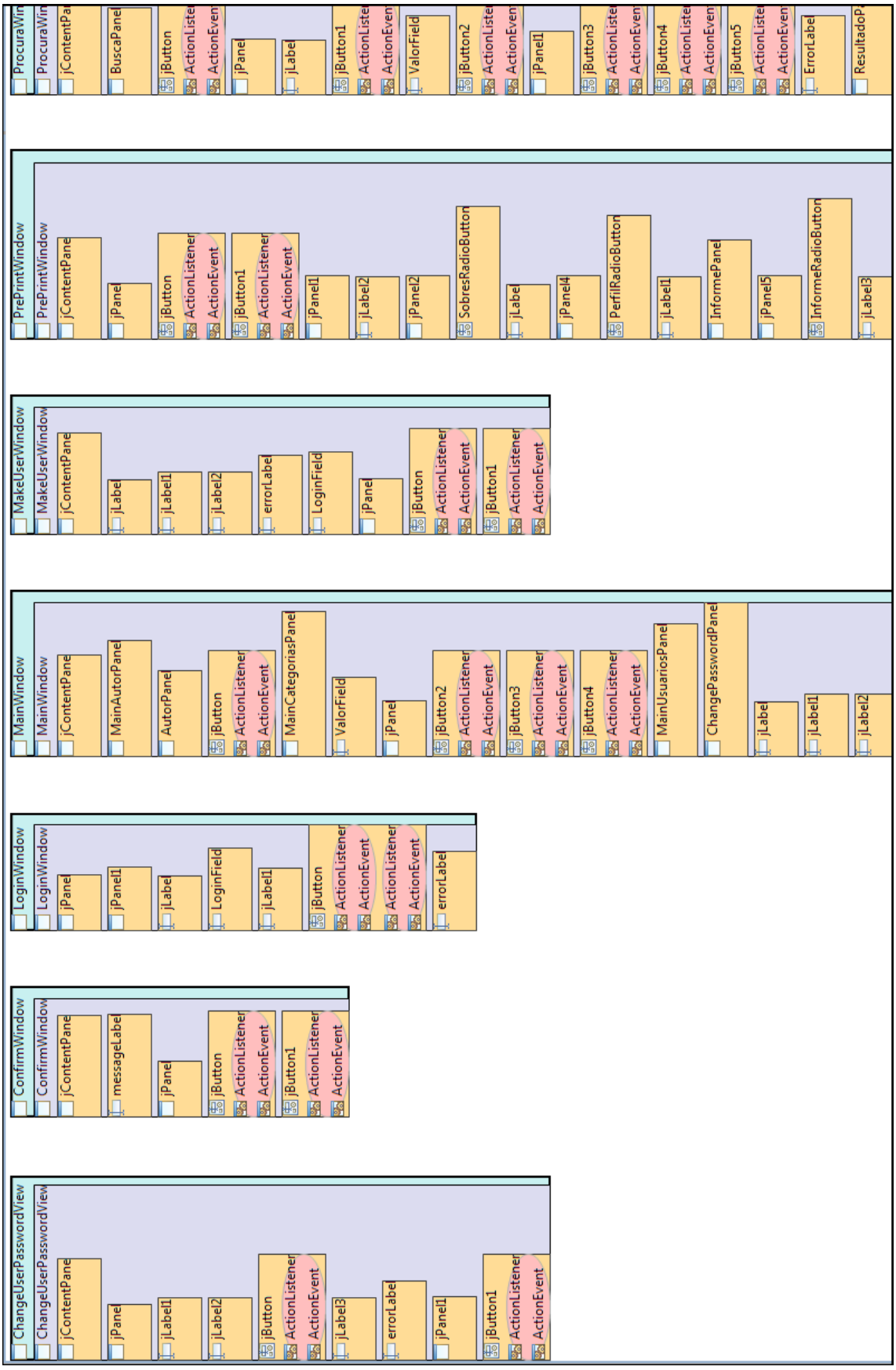


Figura 8.41. Fragmento del diagrama generado

8.2.3. Generación de GUI para Android en el propio proyecto AndrIU

Para la generación de GUI para Android en el propio proyecto AndrIU se ha tomado el modelo KDM unificado. Como se observa en la Figura 8.42, el fichero de GUI se ha generado correctamente en el directorio *"Android UI"*. En la Figura 8.43 se puede observar un fragmento de este fichero.

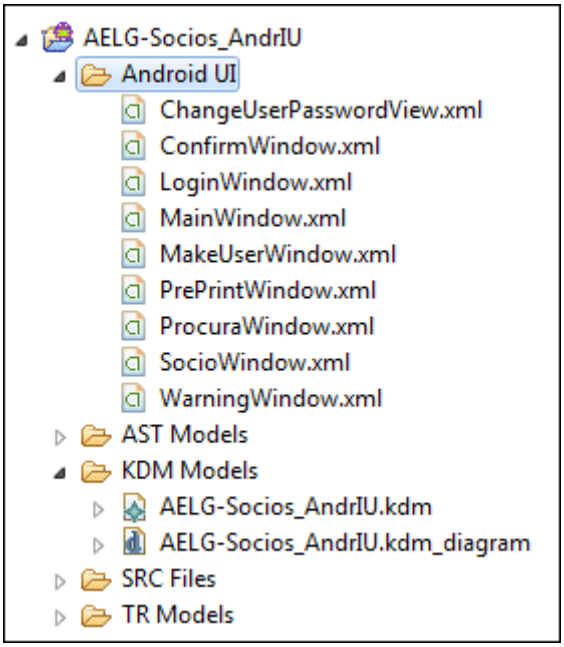


Figura 8.42. Ficheros de GUI para Android generados en el proyecto *AELG-Socios_AndrIU*

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/DataNacementoField"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <EditText
        android:id="@+id/SexoField"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/jLabel6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+string/jLabel6"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/Enderezo1Field"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/jLabel9"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+string/jLabel9"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/CPField"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/jLabel10"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+string/jLabel10"
        android:textAppearance="?android:attr/textAppearanceMedium" />
```

Figura 8.43. Fragmento de uno de los ficheros de GUI generados

8.2.4. Generación de GUI para Android en un proyecto Android

Para la generación de GUI para Android en un proyecto Android se ha generado un nuevo proyecto Android *"AELGSociosForAnroid"*.

8.2.4.1. Generación de los ficheros de GUI en el proyecto Android

Una vez creado el proyecto Android, se procede a la generación del fichero de GUI a partir del modelo KDM del proyecto *AELG-Socios_AndrIU*. Al igual que en el caso anterior, se seleccionan el proyecto Android creado el fichero *R.java* para la definición de los elementos de la GUI. Como se observa en la Figura 8.44, los ficheros se ha generado correctamente (en la Figura 8.45 puede observarse un fragmento del código de uno de estos ficheros y en la Figura 8.46 el mismo fichero con el editor de Android), así como la definición de los elementos en el fichero *R.java* (véase Figura 8.47).

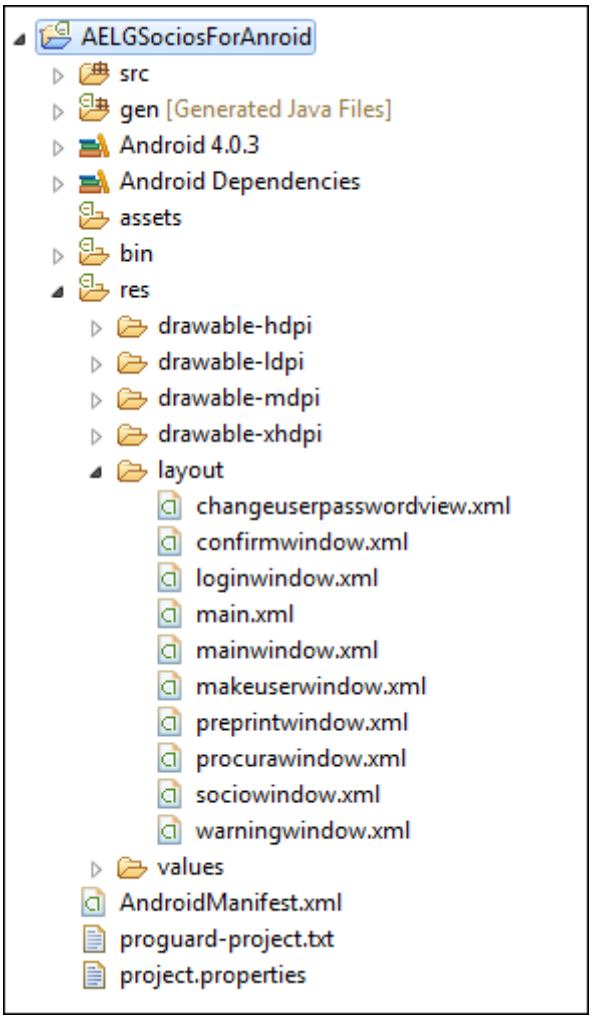


Figura 8.44. Proyecto *AELGSociosForAndroid* con los ficheros de GUI generados

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/jLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+string/jLabel"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/LoginField"
        android:layout_width="168dp"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/jLabel1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+string/jLabel1"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <Button
        android:id="@+id/jButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+string/jButton" />

    <TextView
        android:id="@+id/errorLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@+string/errorLabel"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</LinearLayout>
```

Figura 8.45. Código de uno de los ficheros de GUI generados

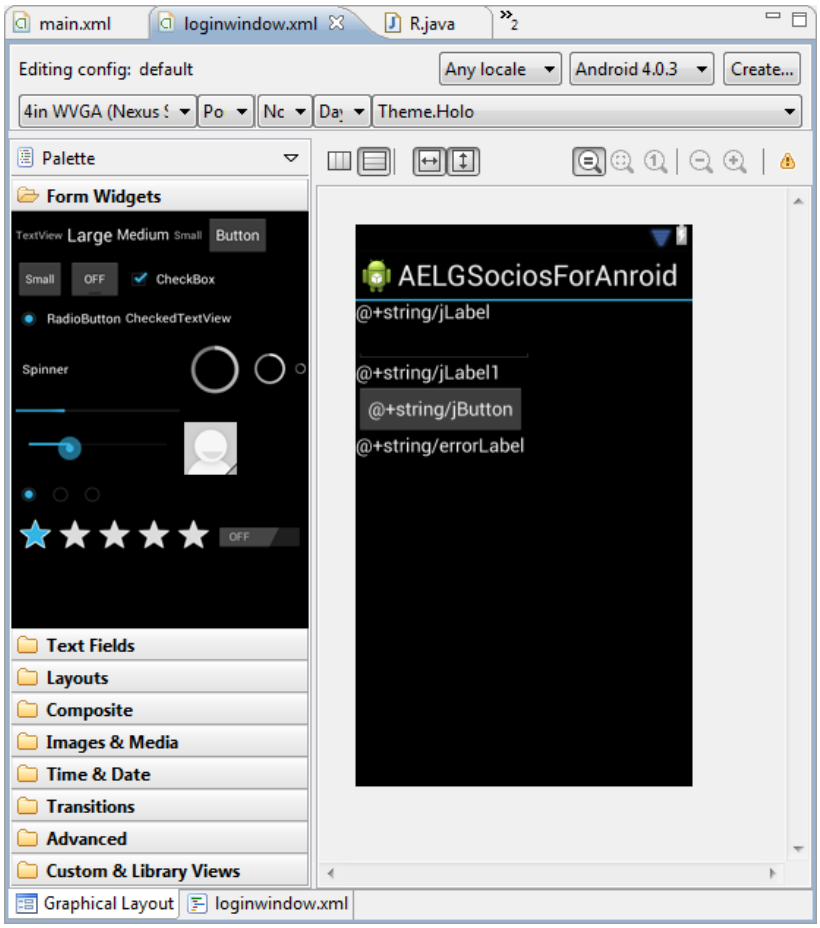


Figura 8.46. Vista con el editor de ADT de uno de los ficheros generados

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int AAnoField=0x7f05003d;
        public static final int AnoCotaField=0x7f050047;
        public static final int BAnoField=0x7f050042;
        public static final int CPField=0x7f05002f;
        public static final int DataNacementoField=0x7f05002c;
        public static final int DesmarcarButton=0x7f050023;
        public static final int EmailField=0x7f050055;
        public static final int EmailPrincipalField=0x7f050063;
        public static final int Enderezo1Field=0x7f05002e;
        public static final int Enderezo2Field=0x7f050050;
        public static final int EntidadeBancaria2Field=0x7f05006c;
        public static final int EntidadeBancariaField=0x7f050031;
        public static final int ErrorLabel=0x7f050029;
        public static final int EtiquetasRadioButton=0x7f050024;
        public static final int FotoLabel=0x7f05005d;
        public static final int ImporteCotaField=0x7f050049;
        public static final int ImporteField=0x7f050010;
        public static final int InformeRadioButton=0x7f050015;
        public static final int LocalidadeBanco2Field=0x7f050070;
        public static final int LocalidadeBancoField=0x7f050032;
        public static final int LocalidadeField=0x7f050030;
        public static final int LoginField=0x7f050008;
        public static final int MarcarButton=0x7f05001f;
        public static final int NIFField=0x7f05005e;
        public static final int NSocioField=0x7f05005f;
        public static final int NameField=0x7f05004e;
        public static final int NumeroConta2Field1=0x7f050067;
        public static final int NumeroConta2Field2=0x7f050068;
        public static final int NumeroConta2Field3=0x7f050069;
        public static final int NumeroConta2Field4=0x7f05006a;
        public static final int NumeroContaField1=0x7f050034;
        public static final int NumeroContaField2=0x7f050051;
        public static final int NumeroContaField3=0x7f050052;
        public static final int NumeroContaField4=0x7f050053;
        public static final int ObservacionsTextArea=0x7f050060;
        public static final int Oficina2Field=0x7f05006e;
        public static final int OficinaField=0x7f050033;
        public static final int PAAnoField=0x7f050038;
        public static final int PerfilRadioButton=0x7f050014;
        public static final int ProvinciaField=0x7f050035;
```

Figura 8.47. Fragmento de código del fichero *R.java*

8.2.5. Resultados y Conclusiones

Tras la generación de los modelos que se han explicado en los apartados anteriores, se presentan a continuación los resultados obtenidos. En primer lugar, en la Tabla 8.1 se muestra de manera resumida el número de clases, clases que contienen informaciones referentes a la GUI del sistema, modelos UI de KDM generados y Layout de Android generados.

Resultados Caso de Estudio 2	
Clases (Java)	163
Cases de GUI (Java)	9
***UIDisplays* creados (KDM)**	9
***Layouts* creados (Android)**	9

Tabla 8.1. Resumen de modelos del Caso de Estudio

De este modo se puede observar que de 163 clases de que consta la aplicación *AELG-Socios*, únicamente 9 de ellas contienen información referente a la GUI. Se han creado 9 *UIDisplays* en los modelos de UI del modelo KDM del sistema, es decir un *UIDisplay* por cada clase Java que define una ventana. Y de la misma manera, por cada *UIDisplay* se ha generado un *Layout* para Android.

A continuación, en la Tabla 8.2 se muestran más en detalle los elementos generados por cada una de las 9 clases de GUI. De este modo se pueden observar el número de elementos de GUI que se definen en cada una de las clases, el número de elementos de UI generados el modelo KDM (diferenciando entre elementos contenedores, no contenedores y el total) y elementos de GUI generados en cada *Layout* de Android.

Java		KDM			Android
Nombre del archivo	GUI	no contenedores	contenedores	total	GUI
ChangeUserPasswordView	11	6	3	9	6
ConfirmWindow	5	3	2	5	3
LoginWindow	8	5	2	7	5
MainWindow	34	17	12	29	17
MakeUserWindow	11	7	2	9	7
PrePrintWindow	54	28	10	38	28
ProcuraWindow	30	14	7	21	14
SocioWindow	159	99	28	127	99
WarningWindow	4	2	2	4	2

Tabla 8.2.Resultados por fichero Caso de Estudio

De acuerdo a los resultados descritos en la Tabla 8.2, se presenta en la Figura 8.48 un gráfico para mostrar de manera gráfica dichos resultados.

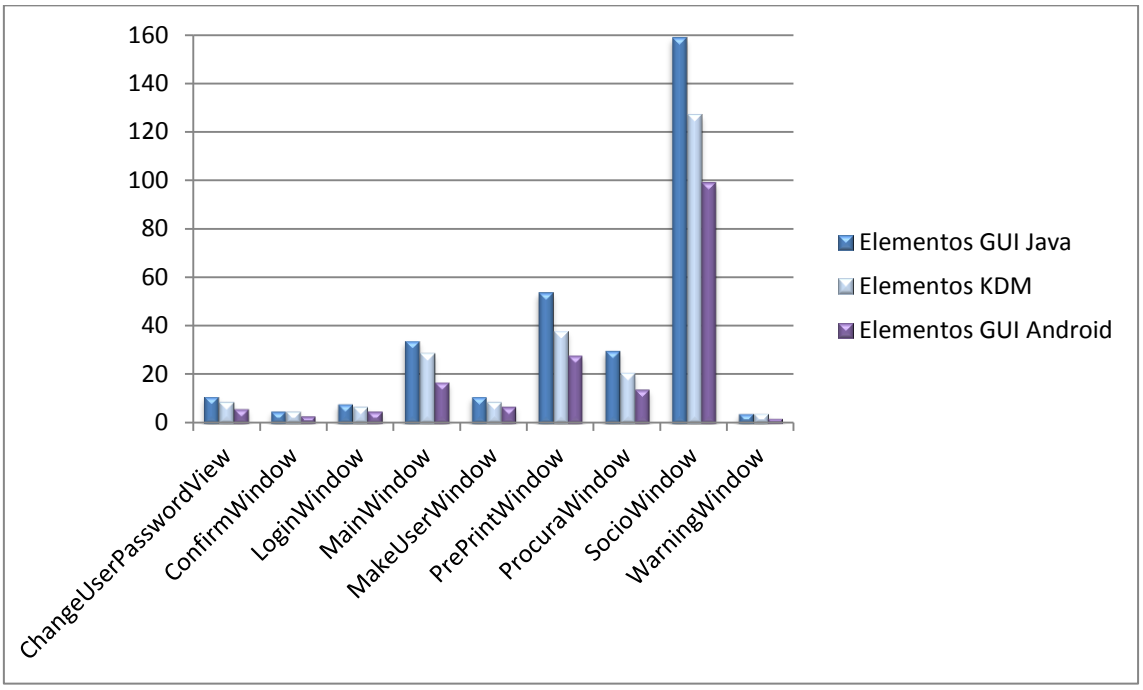


Figura 8.48. Resultados Caso de Estudio

Tras analizar los resultados obtenidos al realizar las tareas descritas en los apartados anteriores, puede concluirse que los resultados obtenidos han sido satisfactorios. Por cada clase Java se ha generado correctamente un *UIDisplay* en el modelo KDM como se esperaba. Además, el número de elementos generados en cada *UIDisplay* se acerca al número de elementos de GUI de la clase Java correspondiente. El porcentaje de elementos generados no es el 100% debido a que únicamente se tiene un conjunto de elementos de GUI en los modelos de transformación. De este modo existen algunos tipos de elementos de GUI que se omiten en la generación, pero que pueden tenerse en cuenta en versiones posteriores de AndrIU.

El número de elementos de GUI generados en los *Layout* de Android coincide con el número de elementos no contenedores generados en los modelos KDM. Los elementos no contenedores hacen referencia a los *Panel* de java, que se representan como *Screen* en KDM y su función es la de agrupar elementos. En este caso no se han tenido en cuenta en la generación de GUI de Android, pero en futuras versiones de AndrIU se pueden utilizar para agrupar los elementos generados en diferentes *Layout* u otro tipo de agrupación de elementos que ofrezca Android.

Por último, en la Figura 8.49 se muestra la conversión de una de las ventanas que componen la GUI de la aplicación *AELG-Socios* al *Layout* para Android generado automáticamente. Se ha cambiado manualmente el texto de las etiquetas y botones generados.

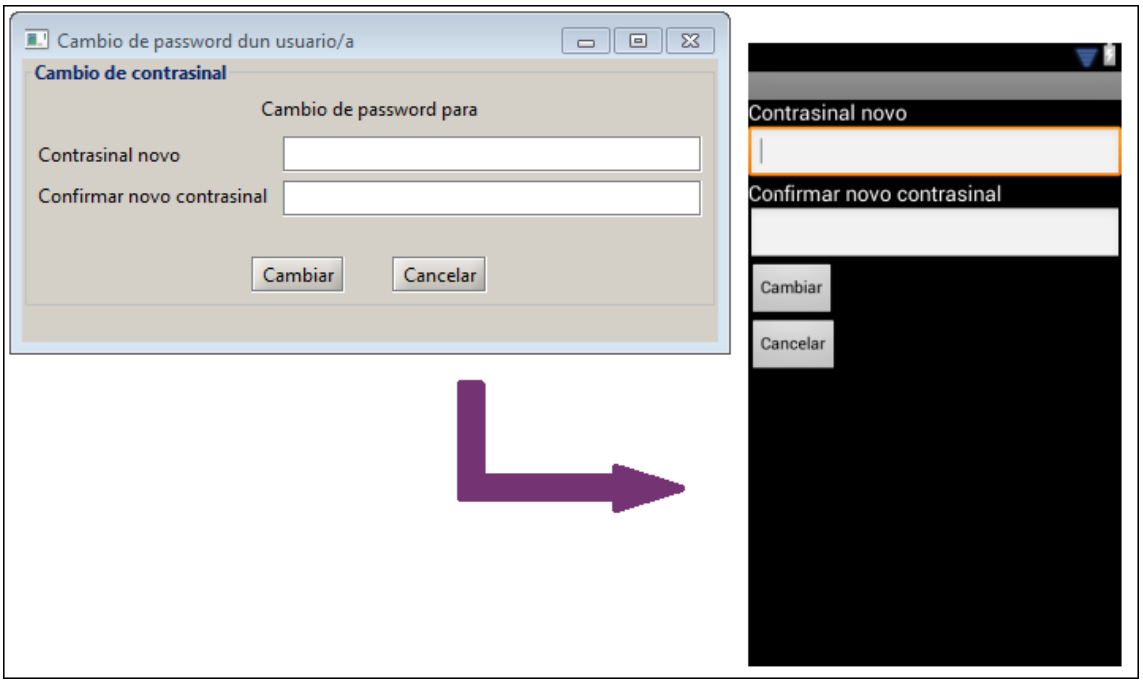
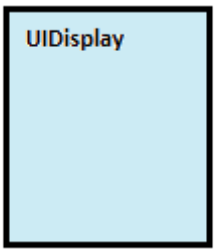
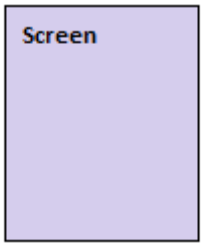


Figura 8.49. Resultado de conversión de GUI

8.3. Anexo III. Notación del editor gráfico UI-KDM

En este anexo se presenta la propuesta de notación para la representación de modelos de GUI siguiendo el estándar KDM (ISO/IEC 19506). Esta notación es la que se ha seguido en los modelos generados por la herramienta *AndrIU*, y la representación que se explica a continuación (véase Tabla 8.3) coincide con los diagramas generados con el editor gráfico de *AndrIU*.

Elemento	Descripción	Notación
UIDisplay	El *UIDisplay* es una unidad de visualización de la GUI de la aplicación. Habitualmente representará una ventana, y es el elemento contenedor principal del resto de elementos de la GUI. Se representa como un rectángulo con borde grueso y el nombre del elemento en la esquina superior izquierda.	
Screen	Es uno de los elementos contenedores que se pueden encontrar dentro del *UIDisplay*. El *Screen* hace referencia a un conjunto de elementos de GUI que ofrecen funciones de entrada-salida de datos para la aplicación. Se representa como un rectángulo con borde continuo delgado y el nombre del *Screen* en la esquina superior izquierda.	
Report	Es el otro tipo de elemento contenedor que se puede encontrar dentro del *UIDisplay*. El *Report* es prácticamente igual al *Screen*, salvo que hace referencia a unidades de visualización con funciones únicamente de salida de datos (como ventanas de notificaciones, muestra de resultados, etc.). Se representa como un rectángulo con borde discontinuo delgado y el nombre del *Report* en la esquina superior izquierda.	
UIResource	Representa a los elementos de control de la GUI que no son textuales. A este grupo pertenecen los botones, casillas de selección, etc. Se representan como un rectángulo de color naranja con el nombre del *UIResource* en la esquina superior izquierda.	

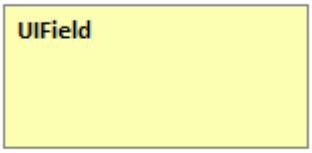
UIField	Representa a los elementos textuales de la GUI. A este grupo pertenecen los campos de texto, las áreas de texto, las etiquetas, etc. Se representan como un rectángulo de color amarillo con el nombre del *UIResource* en la esquina superior izquierda.	
UIAction	Representan a los manejadores de eventos que tienen los elementos de control de la GUI. Un *UIResource*, por ejemplo, puede tener uno o más *UIAction* que representan a la acción que se realizará cuando se produzca cierto evento sobre el elemento de control. Se representan como una elipse con el nombre del *UIAction* en el centro.	
UIFlow	Representan a los flujos de control que se producen entre ventanas. Cuando se produce una acción que implica que se "abra" una nueva ventana, un *UIFlow* lo representa. Un *UIFlow* toma como nodo origen y como nodo destino elementos del tipo *UIDisplay*, ya que son las unidades principales de visualización. Se representa como una línea continua con una flecha en el nodo destino.	

Tabla 8.3. Descripción de la notación UI-KDM

ABREVIATURAS Y ACRÓNIMOS

A

ADM	Architecture Driven Modernization
ADT	Android Development Tools
API	Application Programming Interfaces
AST	Abstract Syntax Tree

C

CASE	Computer Aided Software Engineering
CdU	Caso de Uso
CIM	Computation Independent Model

E

EMF	Eclipse Modeling Framework
EPL	Eclipse Public License

G

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

I

IDE	Integrated Development Environment
ISO	International Organization for Standardization
IU	Interfaz de Usuario

K

KDM	Knowledge Discovery Metamodel

L

LIS	Legacy Information System

M

MDA	Model Driven Architecture
MDD	Model Driven Development

O

OMG	Object Management Group

P

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

R

RUP	Rational Unified Process
RF	Requisito Funcional
RnF	Requisito no Funcional

S

SDK	Software Development Kit
SI	Sistema de Información

T

TIC	Tecnologías de la Información y la Comunicación

U

UCLM	Universidad de Castilla-La Mancha
UI	User Interface
UML	Unified Modeling Language

W

W3C	World Wide Web Consortium

X

XMI	*XML Metadata Interchange*
XML	Extensible Markup Language