Generación de Interfaces de Usuario para Múltiples Dispositivos

Una aproximación SPL basada en MDD

Ignacio Mansanet Benavent

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Director:

Dr. Joan Fons Cors

Centro de Investigación en Métodos de Producción de Software

Ignacio Mansanet Benavent

Generación de Interfaces de Usuario para Múltiples Dispositivos

Una Aproximación SPL Basada en MDD

Tesis de Máster. Julio de 2011

PROS
Centro de Investigación en Métodos de Producción de Software

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Generación de Interfaces de Usuario para Múltiples Dispositivos: Una Aproximación SPL Basada en MDD

Este documento ha sido preparado por
Ignacio Mansanet Benavent

Director
Dr. Joan Fons Cors

Miembros del Tribunal
Dr. Oscar Pastor López, Universidad Politècnica de València
Dr. Vicente Pelechano Ferragud, Universitat Politècnica de València
Dra. Alicia Villanueva García, Universitat Politècnica de València

Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València
Camí de Vera s/n, Edif. 1F
46022 - València, Spain

Tel: (+34) 963 877 007 (Ext. 83530)
Fax: (+34) 963 877 359
Web: http://www.pros.upv.es

Fecha: Julio-2011
Comentarios: Documento presentado en cumplimiento parcial de los requisitos para el grado de Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información por la Universitat Politècnica de València.

A mi hermana Marta y
a mi sobrina Clara.
Por recordarme día a día que la
vida merece ser vivida con alegría.

❝*Quan surts per fer el viatge cap a Ítaca,*
has de pregar que el camí sigui llarg,
ple d'aventures, ple de coneixences...❞
— Konstantinos Kavafis i Lluís Llach, 1975.

❝*You have to trust in something,*
your gut, destiny, life, karma, whatever.
This approach has never let me down,
and it has made all the difference in my life...❞
— Steve Jobs. Commencement address at Stanford University, 2005.

Prefacio

Al final parece que sí. Ha costado pero al final he terminado la tesis de máster pese a que ha habido momentos en que no estaba claro del todo...

— *"Joan, jo no ho veig..."*
— *"Natxet, tú ves fent que després ja voras que tot té sentit".*

Cuántas veces me ha dicho Joan esta frase. Y realmente tenía razón. Han pasado momentos de nerviosismo, momentos de risas o momentos de decaimiento pero al final parece que no ha terminado mal del todo.

Este trabajo no sólo me ha permitido conocer que es el mundo de la investigación desde dentro, sino conocer a gente maravillosa que siempre tendrán un sitio en mis recuerdos.

Lluís Llach dice que es necesario *"Trobar en el coratge tendressa, i en la humilitat fermesa"* [1]. Ésto que parece complejo, se hace sencillo cuando trabajas con tan buena gente, que no es lo mismo, sino mejor, que trabajar con gente buena.

València, julio de 2011 *Nacho Mansanet*

[1] "Encontrar en el coraje ternura, y en la humildad firmeza"

Agradecimientos

Este trabajo se ha llevado a cabo porque mucha gente ha estado involucrada. En primer lugar quiero agradecer a Joan Fons por todo el tiempo que me ha dedicado, sin el cual, nada de todo esto habría sido posible. Gracias Joan por creer en mí y darme la confianza que tantas veces no encontraba.

También a Vicente Pelechano. Pele, gracias por los consejos que nos das. Aunque a veces sean difíciles de entender, siempre son acertados y bienvenidos. También por la oportunidad que me diste hace 3 años de entrar en el centro ProS donde he conocido a tan buena gente.

Un agradecimiento especial es para Isma, por estar siempre ahí cuando le he necesitado, dándome buenos consejos y ayudándome cuanto estaba perdido, aportándome otro punto de vista, a veces muy particular, pero siempre acertado. Gracias Isma, todo ésto es en gran parte gracias a ti.

Quiero agradecer también especialmente a Miriam, por prácticamente empujarme a escribir mi primer artículo, aún cuando no las tenía todas conmigo, por ayudarme cuando lo he necesitado sin importarle su tiempo, y por enseñarme que a veces basta con un par de risas para solucionar los problemas.

También a Maria, Clara, Salva, Pablo y Mario; compañeros de laboratorio, pero por encima de todo esto, ya amigos. Por todos los momentos dentro y fuera del laboratorio y porque trabajar con vosotros

significa pasar buenos momentos.

Quiero también agradecer a Fani, José Luis, Raúl, Paco y al resto de compañeros y amigos del centro ProS por su colaboración y compañía.

A mi hermana, tíos, tías, abuela, primos, y demás familia, a la que tanto tiempo le ha quitado este trabajo, pero que al final parece que ha sido bien invertido.

Y a toda la gente que, aunque ya no está con nosotros, sigue cuidándonos desde algún lugar, y dándonos los mejores consejos para continuar haciendo camino. Papá, Mamá, no os olvido...

A todos vosotros, Gracias!

Nacho Mansanet

Resumen

Actualmente los dispositivos forman parte de nuestra vida cotidiana. La cantidad de dispositivos crece día a día ofreciendo nuevas posibilidades de interacción. Junto con los nuevos dispositivos, están proliferando las tiendas de aplicaciones como una forma fácil y cómoda de instalar aplicaciones que cubran nuestras necesidades. Se puede afirmar que el negocio del desarrollo de aplicaciones está creciendo a un ritmo alarmante en los últimos años.

Pero esta proliferación trae consigo la problemática planteada a las empresas de desarrollo de aplicaciones. Actualmente, al desarrollar una aplicación, se encuentran en la disyuntiva de decidir si se abarcan todas los dispositivos disponibles, con el aumento de coste que ello conlleva, o se opta por reducir el número de dispositivos destino, viendo así reducido su espectro de negocio y por tanto sus ingresos potenciales.

Esta tesis presenta GeMMINi, una solución para desarrollar interfaces especializadas para cada tipo de dispositivo, teniendo en cuenta la naturaleza de estos dispositivos. Se propone un enfoque de desarrollo dirigido por modelos, donde, a partir de modelos abstractos de interfaz y descripciones de dispositivos (especificadas mediante modelos de características utilizados en la definición de Líneas de Producto), se obtienen por transformación de modelos, prototipos de interfaces específicas para cada tipo de dispositivo que serán transformadas a implementaciones nativas en la plataforma destino.

Resum

Actualment els dispositius formen part de la nostra vida quotidiana. La quantitat de dispositius creix dia a dia oferint noves possibilitats d'interacció. Juntament amb els nous dispositius, estan proliferant les tendes d'aplicacions com una forma fàcil i còmoda d'instal·lar aplicacions que cobreixen les nostres necessitats. Pot afirmar-se que el negoci del desenvolupament d'aplicacions està creixent a un ritme alarmant en els últims anys.

Però aquesta proliferació comporta la problemàtica plantejada a les empreses de desenvolupament d'aplicacions. Actualment, a l'hora de desenvolupar una aplicació, es troben en la disjuntiva de decidir si es comprenen tots els dispositius disponibles, amb l'augment de cost que això comporta, o s'opta per reduir el nombre de dispositius destí, veient així reduït el seu espectre de negoci, i per tant els seus ingressos potencials.

Aquesta tesi presenta GeMMINi, una solució per a desenvolupar interfícies especialitzades per a cada tipus de dispositiu tenint en compte la natura d'aquestos dispositius. Es proposa un enfoc de desenvolupament dirigit per models, on a partir de models abstractes d'interfícies i descripcions de dispositius (especificats mitjançant models de característiques utilitzats en la definició de Línies de Producte), s'obtenen per transformació de models, prototipus d'interfícies específics per a cada dispositiu que seràn transformades a implementacions natives en la plataforma destí.

Abstract

Nowadays, computational devices play a key role in our daily life. Recent years have seen the increasing of many types of devices, offering new interaction techniques. With the appearing of new devices, application stores are gaining attention as an easy way to browse and download applications directly to a target device. This application development business is growing quickly in the last years.

However, development application companies have to face a significant problem. Right now, there is a wide diversity of platforms. If a business wants cross-platform applications, they must build many different copies of each application. This is time consuming, expensive, and often requires new developers. If a business decides to reduce the number of target devices, reducing the business spectrum, its incomes could be reduced too.

In this thesis we present GeMMINi, a method to develop native user interfaces for each type of device, taking into account the nature of these devices. Following the Model Driven Development principles, specific interface prototypes are obtained from abstract interface models and devices descriptions (by means of Feature Models used in the Product Lines definition). Then, these prototypes are translated to native implementations for each target platform.

Índice

Lista de Figuras

Lista de Tablas

Lista de Fragmentos de Código

— ❝*First, let me dispose of Socrates because I am sick and tired of this pretense that knowing you know nothing is a mark of wisdom.*❞

Isaac Asimov.

1

Introducción

Según el diccionario de la Real Academia Española de la lengua podemos definir el concepto de interfaz en el campo de la informática como:

> *"Cualquier conexión, física o lógica, entre un computador y el usuario, un dispositivo periférico o un enlace de comunicaciones."*

A partir de esta definición podemos extrapolar que, una interfaz es cualquier artefacto o mecanismo que permita a dos sistemas de naturaleza autónoma, establecer una comunicación que ambos entiendan. Cuando uno de esos sistemas es una persona y el otro un sistema informático, recibe el nombre de Interfaz de Usuario.

Ratón de Englebart

Todo nace cuando **Douglas Englebart** construye un sistema operativo para una máquina mas bien primitiva, con la cual interactuaba con un artefacto consistente en 2 ruedas perpendiculares y un botón. Este artefacto que mas tarde se vino a llamar ***ratón*** cambió los paradigmas de la computación. De repente ya no eran necesarios altos conocimientos de programación para manejar un ordenador. Nacía el **Ordenador Personal** y con él, la era PC.

En 1983, tras haberlo intentado con el Apple III, Apple Computer lanza al mercado el ***Apple Lisa***, en palabras de Steve Wozniak, padre de la idea junto a Steve Jobs: *"The first mass-marketed microcomputer with a graphical user interface"*.

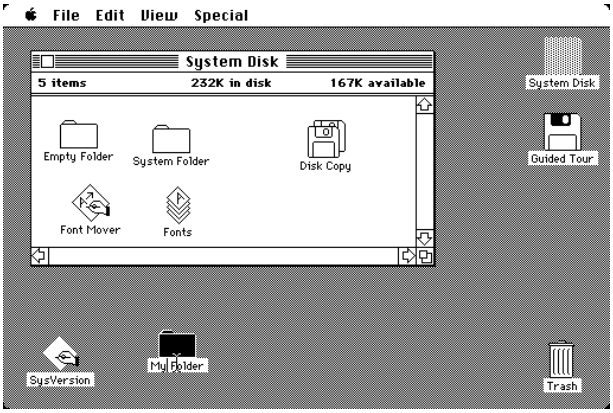


Figura 1.1: IGU del Macintosh

El Apple Lisa contaba con una memoria RAM de 1MB , un monitor de 12" en blanco y negro y una disquetera de $5\frac{1}{4}$. Fue un fracaso dado su precio, excesivamente elevado (10.000$). Pese a esto, en 1984 lanzan el ***Macintosh***. Fue el primer ordenador personal lanzado al mercado de manera exitosa, con ratón (además de teclado) como mecanismo de

interacción y con una ***interfaz gráfica WIMP***[1] (Window, Icon, Menú and Pointer).

Y este es el modelo que se ha seguido hasta hace unos pocos años. Pero ahora, casi 30 años después, la irrupción de nuevos dispositivos como los móviles y las tabletas, junto con la redefinición de otros ya existentes como la televisión, causa que tengamos que replantearnos los conceptos que teníamos asentados. Estos dispositivos traen consigo nuevos mecanismos de interacción que pueden ofrecer la posibilidad de eliminar los mecanismos conocidos hasta ahora como el teclado y el ratón. Estos dispositivos traen consigo la problemática de crear las nuevas interfaces de usuario que se adecuen a ellos y, a ser posible, a cuantos más, mejor.

Esta tesis propone un método para construir estas interfaces aplicando procesos de producción que satisfagan ciertos requisitos como la de los procesos, la reutilización, etc. El objetivo es centrarse en el proceso en la definición de la interfaz y que el proceso garantice la sistematización y la calidad del resultado final.

El resto de este Capítulo se estructura como sigue: la Sección 1.1 expone la motivación del trabajo para en la Sección 1.2 plantear formalmente el problema. En la Sección 1.3 se fijan los objetivos a cumplir y en la Sección 1.4 se expone la solución propuesta. Finalmente la Sección 1.7 presenta la estructura del presente documento.

1.1. Motivación

En los últimos años se ha podido observar la aparición de nuevos tipos de dispositivos móviles de pequeño y mediano tamaño, ofreciendo mecanismos de interacción novedosos. Así, por ejemplo, las interfaces táctiles o multi-táctiles se han consolidado como una solución más natural en el ámbito móvil, impulsando la penetración de estos dispositivos [31]. Estos dispositivos se suman a los ya existentes (ordenadores personales, portátiles, mini-portátiles,...), y a la redefinición de otros (como por ejemplo la TV), para ofrecer un mayor grado de interacti-

[1] Ventana, Icono, Menú, Puntero

vidad [42]. Dada esta situación, podemos asumir que en la actualidad estamos frente a una gran diversidad de dispositivos con diferentes capacidades y características que nos ofrecen múltiples mecanismos para la interacción persona-ordenador. Esta diversidad, presenta un reto a la comunidad de Interacción Persona-Ordenador (IPO), que requiere soluciones para construir y mantener versiones de aplicaciones para todos los dispositivos, manteniendo la consistencia entre versiones y el tipo de interacción [8].

1.2. Planteamiento de Problema

El desarrollo de aplicaciones multi-dispositivo sigue planteando retos a la comunidad científica. Aún hoy en día existen problemas que requieren de solución en este ámbito. El trabajo presentado en esta tesis de máster trata de ofrecer una solución que mejore el desarrollo de interfaces para múltiples dispositivos en un enfoque ingenieril.

Un ejemplo puede ser una aplicación como ***Remember the Milk***. La empresa desarrolladora de esta aplicación, la ha desarrollado para **nueve plataformas** distintas que podríamos agrupar en cuatro tipos de dispositivo: Dispositivos Móviles, Tabletas, Ordenadores y la Web. Además, dentro de la misma plataforma encontramos que tenemos también una división, así por ejemplo para la plataforma iOS encontramos versiones para iPhone e iPad. Además se lanzan **dos versiones** de la aplicación por cada *release*, para llegar a un total de unas **24 aplicaciones** individuales por cada versión de la aplicación que se lanza al mercado.

Es un número muy elevado para una sola aplicación, lo que pone de manifiesto la necesidad de aplicar procesos de desarrollo que faciliten la creación de este tipo de aplicaciones en entornos multi-dispositivo. Pero siguen existiendo preguntas que es necesario responder:

Pregunta 1: ¿Cómo se debe enfocar el desarrollo multi-dispositivo ante la actual fragmentación de dispositivos? Multi-Dispositivo vs Múltiples dispositivos

Pregunta 2: ¿Cómo abordar este desarrollo empleando procesos de producción, fomentando la reutilización y consiguiendo un nivel de abstracción alto?

En las siguientes secciones se trata de dar respuesta a estas preguntas.

1.3. Objetivos

El objetivo principal de este trabajo es definir un **método de desarrollo de interfaces para múltiples dispositivos**. Para ello, cabe responder primero a las preguntas planteadas en la sección anterior.

Con respecto a la **primera pregunta**, es de resaltar que en la actualidad existen diversas corrientes de las que vamos a resaltar: (1) aplicaciones capaces de adaptarse a múltiples dispositivos o (2) aplicaciones individuales para cada uno de los dispositivos. Ambas tienen su razón de ser y son utilizadas en diversas propuestas en función de los *requisitos previos* que se presenten, aunque la primera de ellas es más utilizada por tener un coste mas reducido.

En el primer caso tenemos una solución muy versátil y *"barata"* ya que se construye (o genera) una sola aplicación basada en un lenguaje común, típicamente HTML [4] o XML[14, 25]. Por contra, esta no suele aprovechar las características de interacción individuales que nos brinda cada dispositivo (capacidades multitáctiles, vibración, acelerómetros, etc.). La segunda aproximación nos permite aprovechar todas esas características pero a cambio debemos construir tantas aplicaciones como dispositivos tengamos. Es por ésto que es una solución menos versátil pero que permite explotar más las capacidades de los dispositivos de manera individual.

En la actualidad estamos presenciando la proliferación de las tiendas de aplicaciones como un nuevo modelo de negocio que reporta muchos beneficios no solo a los desarrolladores sino también a las empresas. Esta relación simbiótica se está dando gracias a las facilidades que tienen los desarrolladores para ofrecer sus productos a través de estas tiendas. Pero en estas tiendas la mayoría de las aplicaciones disponibles son nativas a

los dispositivos, por lo que se puede dar un cambio en la tendencia hacia los desarrollos individuales para cada dispositivo, dejando los desarrollos multi-dispositivo para las aplicaciones tipo web.

Con respecto a la **segunda pregunta**, uno de los objetivos de este trabajo es el de utilizar procesos que ayuden a fomentar la reutilización, al tiempo que se mantiene un nivel de abstracción que permita separar los aspectos de diseño de los de análisis. Para ello hemos optado por el uso de técnicas de líneas de producto como los modelos de características, que nos facilitan la reutilización.

También queremos eliminar la propensidad a errores típica de estos desarrollos. Por lo que se propone un método sistemático basado en principios del desarrollo software dirigido por modelos, utilizando técnicas como las transformaciones de modelos o la generación de código [21]. Esta aproximación metodológica nos permitirá obtener las interfaces a partir de sus definiciones abstractas, siguiendo una secuencia de pasos de desarrollo bien definidos.

1.4. Solución Propuesta

La Ingeniería Dirigida por Modelos (MDE) [40] propone el uso de modelos como la base del desarrollo de sistemas. Un modelo es una simplificación de un sistema, construido con un objetivo previsto en mente, que debería ser capaz de responder a las preguntas en lugar del sistema actual [5]. El uso de modelos (como el modelo de los aviones en un túnel de viento o modelos de sistemas informáticos) en la ingeniería tiene un beneficio doble. Por un lado, los modelos **guían el desarrollo de un sistema**. Por otro lado, los modelos permiten **razonar sobre el sistema** evitando hacer frente a los detalles técnicos.

El sistema especificado se puede generar automáticamente a partir de una descripción abstracta. Este enfoque abstrae los problemas de heterogeneidad tecnológica que se encuentran en el nivel de aplicación, que se agravan cuando se consideran múltiples dispositivos y/o modos de interacción. En este trabajo, presentamos un proceso de desarrollo basado en los fundamentos de MDE para especificar interfaces para

múltiples dispositivos atendiendo a criterios de interacción del usuario con el dispositivo. En concreto, este proceso de desarrollo proporciona las siguientes contribuciones:

Se va a definir un **método para el diseño** que permita especificar los aspectos referentes a la interfaz de usuario, como los mecanismos de interacción a utilizar o la información a mostrar.

Se va a definir también un **método de desarrollo** para guiar al diseñador en la construcción de interfaces de usuario de manera sistemática. El método abarcará desde la especificación inicial hasta la obtención del código resultante.

Con esto se obtiene una solución integral para la generación asistida (semi-automática) de interfaces para múltiples dispositivos, utilizando una aproximación en el marco del Desarrollo Software Dirigido por Modelos.

1.5. Método de Investigación

Con el fin de realizar el trabajo de esta tesis, hemos llevado a cabo un proyecto de investigación siguiendo la metodología de diseño para la realización de investigaciones en sistemas de información, como se describe en [26] y [44]. El diseño de la investigación consiste en el análisis del uso y el rendimiento de los artefactos diseñados para comprender, explicar, y con mucha frecuencia, para mejorar el comportamiento de los aspectos de los sistemas de información [44].

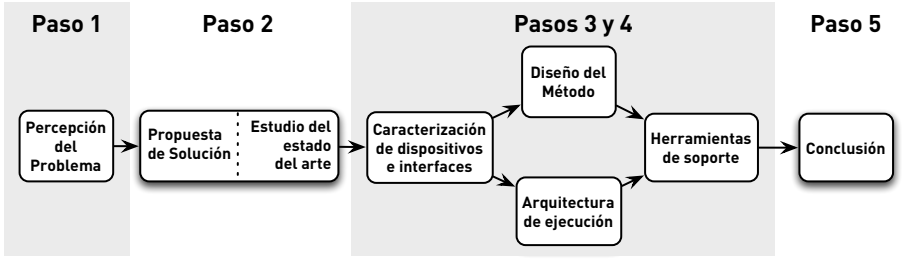


Figura 1.2: Método de investigación seguido en esta tesis de máster

El ciclo del proceso de diseño consta de 5 pasos: (1) percepción del problema, (2) propuesta de solución, (3) el diseño y desarrollo, (4) evaluación, y (5) conclusión. El ciclo de diseño es un proceso iterativo; el conocimiento producido en el proceso de construcción y evaluación de nuevos artefactos se utiliza como entrada para un mejor conocimiento del problema. Siguiendo el ciclo definido en la metodología de investigación, se inició con la percepción del problema (ver Figura 1.2). Se identificó el problema a resolver y se caracterizó claramente.

A continuación, se realizó la segunda etapa que comprende la propuesta de una solución al problema, y la comparación de las mejoras que introduce esta solución con las soluciones ya existentes. Para ello, han sido estudiados en detalle los enfoques más relevantes. Una vez descrita la solución al problema, la hemos desarrollado y validado (pasos 3 y 4). Estos dos pasos se llevan a cabo en varias fases (ver Figura 1.2). Con las tareas llevadas a cabo en estos pasos se ha pretendido caracterizar los dispositivos y las interfaces, definir las técnicas para su diseño, e implementar herramientas de soporte al método de desarrollo.

Por último, se analizan los resultados de nuestro trabajo de investigación con el fin de obtener varias conclusiones, así como para delimitar las áreas de investigación (paso 5).

1.6. Contexto del Trabajo

Esta tesis de máster se ha desarrollado en el Centro de Investigación en Métodos de Producción de Software[2] de la Universitat Politècnica de València[3]. Este trabajo se ha realizado gracias a los siguientes proyecto de investigación gubernamentales:

OSAMI Commons: Open Source Ambient Intelligence Commons. Proyecto ITEA 2 con referencia TSI-020400-2008-114.

OPEES: Open Platform for Engineering of Embedded Systems. Proyecto ITEA 2 con referencia TSI-020400-2010-36.

[2] http://www.pros.upv.es
[3] http://www.upv.es

EVERYWARE: Construcción de Software Adaptativo para la Integración de Personas, Servicios y Cosas usando Modelos en Tiempo de Ejecución. Proyecto del Ministerio de Ciencia e Innovación con referencia TIN-2010-18011, co-financiado con fondos FEDER.

1.7. Estructura del Documento

El resto de este documento se estructura en ocho Capítulos. Como guía de la estructura se puede consultar la lista siguiente:

Capítulo 2: discute otras propuestas que tratan los ámbitos en que se ha encuadrado este trabajo. Concretamente se comparan los trabajos en los ámbitos de la interacción persona-ordenador y el desarrollo dirigido por modelos centrado en las interfaces de usuario.

Capítulo 3: expone el contexto tecnológico y los fundamentos metodológicos en que se basa esta tesis. Presenta las técnicas existentes en el ámbito de las líneas de producto software y el modelado de interfaces de usuario que son aplicadas en el desarrollo de este trabajo y el conjunto de herramientas y lenguajes utilizados para darles soporte.

Capítulo 4: sienta las bases de esta tesis. Para ello presenta los diferentes artefactos necesarios para poner en práctica la propuesta presentada en este trabajo y ofrece una visión general de la aportación. Ademas justifica el enfoque elegido para el desarrollo de este trabajo exponiendo sus ventajas e inconvenientes.

Capítulo 5: detalla el modelo utilizado para especificar los dispositivos en términos de interacción y posición frente al usuario. Se justifica la necesidad de este nuevo modelo y se presenta junto a las primitivas utilizadas en su especificación.

Capítulo 6: presenta el catálogo de patrones de diseño de interfaz de usuario utilizado para establecer correspondencias entre conceptos abstractos y concretos de interfaz de usuario. Este capítulo

presenta tanto la semántica aportada por este modelo como la herramienta desarrollada a fin de dar soporte a su especificación.

Capítulo 7: expone el algoritmo mediante el cual se obtienen los bocetos específicos de plataforma tomando como entrada los modelos definidos en los capítulos anteriores. Se presentan algunos conceptos acuñados en el contexto de este trabajo de tesis que dan soporte al modelo. También se aportan ejemplos de aplicación de este algoritmo tanto en el propio capítulo como en el Anexo C

Capítulo 8: presenta el proceso de generación de la interfaz de usuario para el dispositivo destino. En este se presentan dos casos distintos de generación de las interfaces para distintos dispositivos que pretenden resolver el mismo problema a nivel abstracto: el acceso a un sistema de revisión de conferencias.

Capítulo 9: detalla la validación de la propuesta mediante un caso de estudio completo. Este caso de estudio esta basado en un escenario bien conocido: un sistema de revisión de conferencias.

Capítulo 10: resume las contribuciones y publicaciones principales de la presente tesis de máster y provee algunos indicios sobre los trabajos futuros relacionados con ésta.

Acrónimos Utilizados en este Documento

DSDM: (DDM, MDD) Desarrollo de Software Dirigido Por Modelos. Paradigma emergente para la construcción de software que utiliza los modelos para especificar programas y transformaciones de modelos para obtener los ejecutables.

XMI: *XML Metadata Interchange*. Estándar de la *Object Management Group* para el intercambio de metadatos de información vía XML.

UIM: *User Interface Modeling*. Lenguaje de alto nivel para especificar los requisitos de las interfaces de Usuario. También recibe este nombre la herramienta que le da soporte, MOSKitt UIM.

LSD: (DSL) Lenguaje Específico de dominio. Lenguaje de programación o de especificación ejecutable que ofrece la expresividad necesaria para especificar problemas referentes a un dominio, mediante la notación y abstracción propuesta.

IGU: (GUI) Interfaz Gráfica de Usuario. Es el tipo de interfaz de usuario que permite a los usuarios interactuar con los dispositivos mediante imágenes en lugar de comandos de texto.

LPS: (SPL) Línea de Productos Software. conjunto de sistemas software que comparten un conjunto de características y que satisfacen las necesidades de un segmento de mercado y que son desarrolladas a partir de un *framework* común según unas maneras preestablecidas.

M2M: Transformación Modelo-A-Modelo. Técnica en el entorno MDD mediante la cual se obtiene un modelo salida a partir de ciertos modelos de entrada, mediante la aplicación de patrones y reglas de transformación.

M2T: Transformación Modelo-A-Texto. Generación de código a partir de ciertos modelos de entrada.

— ❝*Murphy, O'Malley, Sod and Parkinson are alive and well - and working on your project...*❞

Anonymous.

2

Estado del Arte

Este trabajo se basa en diferentes conceptos y tecnologías. Para clarificar los conceptos en que basamos nuestra aproximación, en este capítulo se describen diferentes tecnologías y técnicas que serán utilizadas posteriormente, al tiempo que se introducen las aproximaciones más importantes en las áreas en que se encuadran.

Este trabajo se centra en el desarrollo de interfaces nativas para múltiples dispositivos. La propuesta presentada se encuentra en la intersección de dos áreas de investigación. Éstas son: El Desarrollo Software Dirigido por Modelos (DSDM, DDM, MDD) centrado en el desarrollo de interfaces y la Interacción Persona-Ordenador (IPO, HCI). Las áreas que están presentes en este trabajo tiene aspectos en común (i.e., la especificación de las interfaces, el modelado de la interacción, etc.). Pese

a estas similitudes, se van a explorar por separado tratando los aspectos mas relevantes de cada una de ellas.

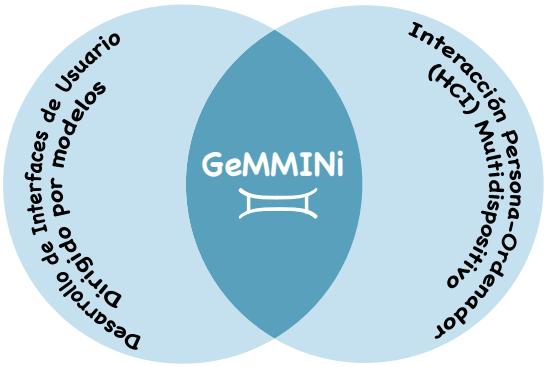


Figura 2.1: Dominios de aplicación involucrados en este trabajo

El resto de este capítulo se estructura como sigue: En la Sección 2.1 trataremos los trabajos en el área del desarrollo de interfaces de usuario dirigido por modelos. La Sección 2.2 tratará los relacionados con la interacción persona-ordenador en el ámbito del desarrollo multi-dispositivo. La Sección 2.3 expone un análisis de las propuestas estudiadas y finalmente concluye el capítulo.

2.1. Desarrollo de Interfaces Dirigido por Modelos.

El desarrollo de software dirigido por modelos se basa en la noción que podemos construir un modelo de un sistema software, el cual puede ser transformado en el sistema real [28]. Los modelos han sido utilizados desde hace mucho tiempo en los procesos de desarrollo de software. Desde lenguajes de especificación ejecutables como OASIS [39] a notaciones ampliamente usadas y aceptadas como UML [32], los modelos están muy presentes en el área del desarrollo de software. Según citan Agrawal et al. en [2]:

> *"Los modelos son creados no sólo para capturar requisitos, sinó también diseños e implementaciones. Los modelos no son sólo meros artefactos utilizados en la documentación, sinó documentos vivos que son transformados en implementaciones de sistemas".*

Los trabajos que se presentan en esta sección están íntimamente ligados con los que veremos en la Sección 2.2. Pese a esto se han separado y en esta Sección se muestran aquellos mas centrados en el uso de modelos para describir las interfaces de usuario, y los de la Sección 2.2 están mas centrados en la interacción.

Limbourg et al. en [25] presentan USIXML (USer Interface eXtensible Markup Language), un lenguaje para la descripción de interfaces de usuario en múltiples contextos de uso. Las aplicaciones descritas preservan la independencia entre diseño y plataforma.

USIXML consiste en un UIDL[1] que permite a los diseñadores desarrollar interfaces para múltiples plataformas. El proceso de desarrollo puede ser abordado desde cualquier nivel de abstracción. Es por esto que las transformaciones M2M se convierten en la *piedra angular* de la propuesta.

Uno de los aspectos más interesantes de este trabajo es la separación clara entre aspectos abstractos de interfaz (AIOs) y los aspectos concretos de interfaz (CIOs).

El problema de esta propuesta radica en la carencia de artefactos que permitan especificar las capacidades de interacción con que cuenta el dispositivo destino en que va a ejecutarse la interfaz.

Eisenstein et al. en [15] describen MANNA: *The Map ANNotation Assistant*, un software *hipotético* destinado a crear mapas anotados de áreas geográficas. Esta aplicación debe poder ejecutarse en varias plataformas y ser utilizada colaborativamente en internet. Para alcanzar estos objetivos se describen un conjunto de modelos y técnicas destinadas a facilitar el diseño de la interfaz de usuario.

[1] Lenguaje de descripción de interfaces de usuario

Se utiliza el modelado de la interfaz de usuario mediante MIMIC, descrito por Puerta en [37]. MIMIC es un lenguaje de modelado de interfaces de usuario basado en 3 componentes: la plataforma (entendida como tipo de dispositivo en función del tamaño de la pantalla), la presentación y las tareas a llevar a cabo. Se puede ver un esquema de la arquitectura propuesta por MANNA en la Figura 2.2.

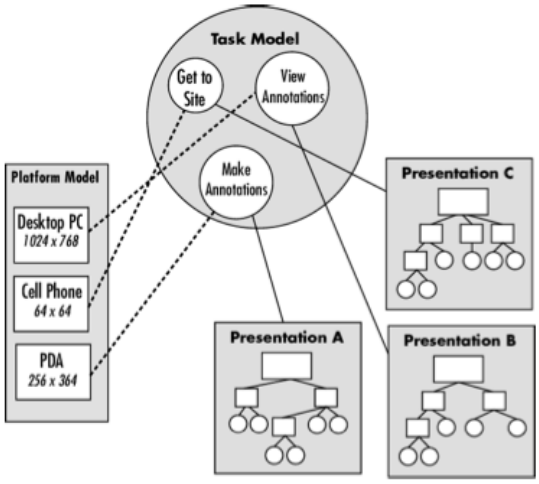


Figura 2.2: Esquema de la arquitectura propuesta por MANNA

Esta propuesta presenta el problema de la pobre expresividad proporcionada para realizar la representación de la plataforma. Como se puede ver en la Figura 2.2, la plataforma se define únicamente en función del tamaño de la pantalla del dispositivo, sin tener en cuenta otros mecanismos de interacción de entrada y/o salida.

Calvary et al. en [8] presenta el el *Marco de Referencia Unificado*. En éste se describe un proceso para la creación de interfaces sensibles al contexto. En este trabajo, el contexto se descompone en 3 factores: la plataforma, el entorno y el usuario final.

El proceso de desarrollo se realiza en 4 pasos: (1) creación de una especificación de las tareas a llevar a cabo, (2) modelado abstracto de la interfaz, (3) modelado concreto de la interfaz, y finalmente (4) creación

del sistema.

Como ocurría con USIXML, esta propuesta no contempla la posibilidad de representar las características del dispositivo destino de la interfaz.

En [29], Mori et al. presentan TERESA (*Transformation Environment for inteRactivE Systems representAtions*), una herramienta destinada al desarrollo de interfaces para múltiples plataformas, incluidos los dispositivos móviles. TERESA utiliza una solución basada en tres niveles de abstracción: un modelo de tareas, un modelo abstracto de interfaz y un modelo concreto.

En esta propuesta se acuña el concepto de aplicación "nómada" para definir las aplicaciones, cuyo funcionamiento está en función del dispositivo que las ejecuta. En la Figura 2.3 se muestra la instanciación del framework de Calvary et al. utilizando TERESA.

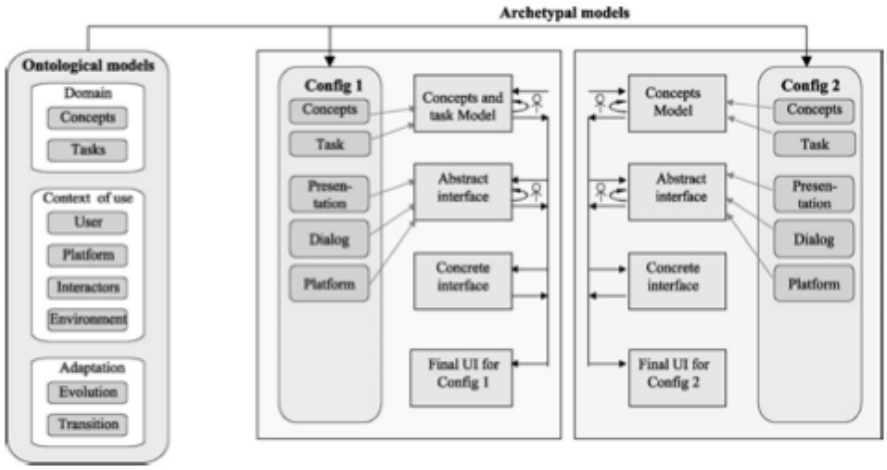


Figura 2.3: El marco de referencia unificado instanciado para TERESA

Clerckx et al. en [10] presentan DynaMo-AID (Dynamic Model-Based User Interface Development), un proceso de diseño y una arquitectura para el desarrollo de interfaces de usuario dinámicas en función del contexto. Es parte de Dygimes, descrito por Coninx et al. en [12]

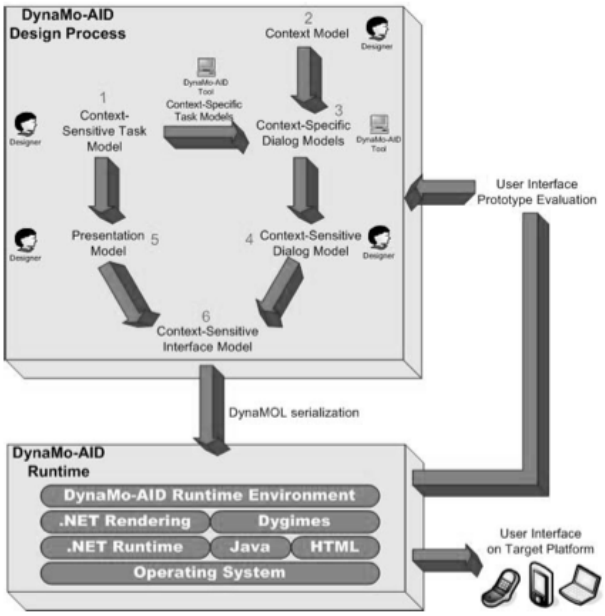


Figura 2.4: El proceso de desarrollo descrito por DynaMo-AID

La Figura 2.4 expone el proceso de desarrollo propuesto por DynaMo-AID. En primer lugar los diseñadores especifican los modelos que describen la interacción (1-6). Seguidamente estos modelos se serializan a un lenguaje descriptivo de interfaces de usuario basado en XML para exportar los modelos a la arquitectura de ejecución. Es en este momento cuando la interfaz se renderiza para la plataforma destino.

Valverde & Pastor en [45] presentan OOWS 2.0, una aproximación enfocada a la Ingeniería Web que utiliza técnicas de modelado para la generación de la interfaz. En este método se aplican las ideas de la comunidad HCI separando el modelado de la interfaz en un nivel abstracto, mediante un modelo conceptual basado en la interacción con el usuario, y en un nivel concreto. Respecto a este nivel concreto, se propone un modelo para interfaces RIA (Rich Internet Applications) que puede ser especializado en distintas tecnologías específicas.

Esta es una propuesta muy interesante por la separación que hacen en los niveles abstracto y concreto. No obstante y dado que su enfoque está dirigido a la ingeniería web, no se contempla la generación de aplicaciones nativas para múltiples dispositivos.

Aquino et al. en [3] presentan una aproximación basada en el desarrollo dirigido por modelos que generará interfaces de usuario multi-dispositivo y/o multi-plataforma y sobre esta generación se aplican criterios que garantizan la usabilidad.

Esta propuesta es muy interesante por la abstracción que realizan de la interfaz de usuario y por la aplicación de las técnicas MDE. No obstante, la generación que se realiza es de aplicaciones que se adaptan a los dispositivos y de estos solo se tiene en cuenta el tamaño de la pantalla para su descripción.

2.2. Interacción Persona-Ordenador

El problema del desarrollo de interfaces multi-dispositivo esta íntimamente ligado con la problemática conocida como *"fragmentación de dispositivos"*. Es por ello que surge la necesidad de etiquetar/catalogar los dispositivos sobre los que se trabaja.

Van Welie & Groot en [46] aportan una primera división de los dispositivos en características, atendiendo a criterios como su soporte a diferentes lenguajes de marcado, el navegador que utiliza y sus capacidades de renderizado, etc., teniendo en cuenta que el resultado final estará basado en HTML.

En [18], Gajos et al. se centran en la generación atendiendo a criterios del dispositivo, aunque sólo tienen en cuenta el tamaño de la pantalla.

Clerckx et al. en [11], proponen una ontología para identificar el dispositivo en función de ciertos parámetros de la modalidad de entrada y salida. En la misma línea, Gajos et al. en [18], presenta SUPPLE, una herramienta para la generación automática de interfaces que tiene en cuenta el tipo de dispositivo destino.

Blumendorf et al. en [6] presentan una aproximación creando modelos ejecutables que representen la interacción persona-ordenador modelando elementos que cambian con el tiempo y otros que permanecen inalterables, y realizando mappings entre ambos.

Paternò et al. en [35] presentan MARIA *"Un lenguaje declarativo, universal y con múltiples niveles de abstracción"*. Éste integra la aproximación TERESA con un lenguaje que describe de manera abstracta la interfaz que será refinada en lenguajes dependientes de plataforma en función de los recursos del dispositivo destino.

Otras aproximaciones como UIML, presentado por Abrams et al. en [1], o XIML, presentado por Puerta & Eisenstein en [38], definen lenguajes específicos de dominio diseñados para describir interfaces de usuario independientes de dispositivo. Estas propuestas tienen en común la generación de aplicaciones sobre un lenguaje independiente de la plataforma, como por ejemplo HTML o XML. Este enfoque permite abarcar gran número de dispositivos y plataformas, pero al tiempo no aprovecha las características particulares de los dispositivos y plataformas software en que se ejecutan (pantallas multi-táctiles, GPS, etc.).

Estos trabajos, no obstante, presentan el problema de atender sólo a criterios de modalidad y no otros como la interacción o la situación del usuario con respecto al dispositivo.

2.3. Análisis de las Propuestas Existentes y Conclusiones

Las propuestas expuestas en las secciones anteriores aportan soluciones en el campo del desarrollo de interfaces. Como hemos podido ver, algunas de ellas no contemplan la generación para múltiples dispositivos y otras, lo hacen utilizando para ello la base de un lenguaje común a todos ellos (típicamente HTML o XML).

Sólo una pequeña parte de ellas abordan criterios de interacción presentes en los dispositivos que están apareciendo actualmente y ninguna de ellas atiende a las características de la interacción del usuario respec-

	Nombre o Autor	¿Enfoque Metodológico?	¿Contempla el dispositivo?	¿Contempla al usuario?	¿Contempla la fragmentación?
1	USIXML	Si	No	No	No
2	MANNA	Si	Sólo la pantalla	No	No
3	Calvary	Si	No	No	No
4	MARIA-TERESA	No	Si	No	No
5	DynaMo-AID	Si	No	El contexto	No
6	OOWS 2.0	Si	No	No	N/A
7	Aquino	Si	Solo la pantalla	En términos de usabilidad	No
8	Van Welie	No	Si	No	Sólo por el navegador
9	Gajos	No	Si, la pantalla	No	No
10	Clerckx	Si	Si	No	No
11	SUPPLE	N/A	Si	No	En parte
12	UIML	N/A	No	No	No
13	XIML	N/A	No	No	No

Tabla 2.1: Comparación de las propuestas contempladas

to al dispositivo. Muy pocas de ellas aportan un enfoque metodológico para guiar al diseñador a lo largo del proceso de diseño y construcción de las interfaces de usuario.

Tampoco se aborda en ninguna de ellas la problemática introducida

por la gran cantidad de dispositivos existentes en el mercado, ya que en la especificación de los dispositivos destino, apenas si se tiene en cuenta el tamaño de la pantalla. En la Tabla 2.1 se expone un resumen de las propuestas estudiadas. Los items establecidos son los que mas afectan a la propuesta presentada en este documento.

La propuesta presentada en este documento trata de cubrir las necesidades que acarrea un desarrollo para múltiples dispositivos proporcionando mecanismos para especificar la interfaz y el dispositivo de manera independiente, así como provee de los mecanismos (en forma de transformaciones de modelos) mediante los cuales obtener las interfaces finales a partir de las citadas especificaciones.

— ❝*Don't reinvent the wheel, unless you plan on learning more about wheels.*❞

Head First Design Patterns.

3

Fundamentos y Contexto Tecnológico

En este trabajo se aplican técnicas del desarrollo de software dirigido por modelos y del desarrollo basado en líneas de producto software. Éstos son campos muy asentados en la informática, tanto a nivel académico como empresarial, lo que implica que existen propuestas ampliamente conocida y aceptadas.

La intención de este trabajo no es "reinventar la rueda" sino, en la medida de los posible, utilizar técnicas ampliamente aceptadas. Por ello, en el presente capítulo se presentan algunas de esas propuestas sobre las que se fundamenta la que se presenta en este trabajo y que sirven como base.

Se presentan también algunas herramientas utilizadas para dar soporte a estas técnicas y que por ello merecen ser estudiadas y, en la medida de lo posible, utilizadas.

El resto de este capítulo se estructura como sigue: En la Sección 3.1 se presentan los fundamentos y técnicas de las líneas de producto software. En la Sección 3.2 se realiza lo mismo para las técnicas del desarrollo dirigido por modelos, centrándose en las interfaces de usuario y las distintas técnicas en la subsección 3.2.1 de interfaces. También se exponen las herramientas utilizadas en esta propuesta para dar soporte a estas técnicas (Sección 3.2.2). En la Sección 3.3 se realiza una revisión de la situación actual en lo que a dispositivos móviles se refiere. Finalmente la Sección 3.4 concluye el capítulo.

3.1. Líneas de Producto Software

La producción en masa fue popularizada por Henry Ford a principios del Siglo XX. McIlroy acuñó el término *"Software Mass Production"* en 1968 [27]. Ese fue el principio de las **Líneas de Producto Software** (SPL).

Según Clements y Northrop [9], las líneas de producto software se definen como:

> *"Un conjunto de sistemas software que comparten un conjunto de características entre ellos y satisfacen las necesidades de un segmento del mercado y que son desarrollados a partir de ese conjunto de características comunes según un procedimiento preestablecido".*

El Instituto de Ingeniería del Software (SEI, Software Engineering Institute) propuso la Ingeniería de Líneas de Producto Software (SPLE)como una solución al crecimiento de la reutilización como técnica en el proceso de desarrollo de sistemas software.

Las SPL se convirtieron rápidamente en un paradigma en un paradigma de desarrollo de software que permitía a las compañías realizar

notables mejoras en los tiempos de lanzamiento a mercado, costes, productividad, calidad, etc.

Una característica fundamental de una SPL es el manejo de la **variabilidad**. La idea consiste en separar todos los productos derivados de una línea en tres partes diferenciadas y manejarlas cada una por separado:

Partes Comunes: son las partes que son comunes a todos los productos derivados de una misma línea. Desde el punto de vista de los requisitos, estos serían los requisitos comunes.

Partes Reutilizables: aspectos que son comunes solo a una parte de los productos de la línea. Proporcionando un mecanismo *plug-and-play*, resulta fácil construir nuevos productos (o parte de ellos), ensamblando piezas reutilizables.

Partes Específicas: Por muy bueno que sea el diseño de una SPL, siempre existirán requisitos que serán únicos para ciertos productos. En estos componentes no se debe invertir esfuerzo en la generalización ya que raramente los utilizaremos en otros productos.

Además de esta gestión de la variabilidad, un segundo principio clave de la SPLE es el uso de una aproximación basada en un ciclo de vida doble. En la Figura 3.1 podemos ver esta aproximación [23]. El desarrollo en este caso esta basado en la separación de dos procesos: La Ingeniería del Dominio y la Ingeniería de la Aplicación. A continuación se describen estos dos procesos:

Ingeniería del Dominio: Este proceso es el responsable de todo el análisis de la SPL visto como un todo que permita desarrollar cualquier parte común y variable. El uso de modelos de características para definir la Ingeniería de dominio en una línea de productos fue introducido por Kang et al. a principios de los años 1990 [24]. El objetivo de inferir un diseño o una implementación de los modelos de características es el punto común a varias de las contribuciones en el ámbito de las Líneas de Producto de Software.

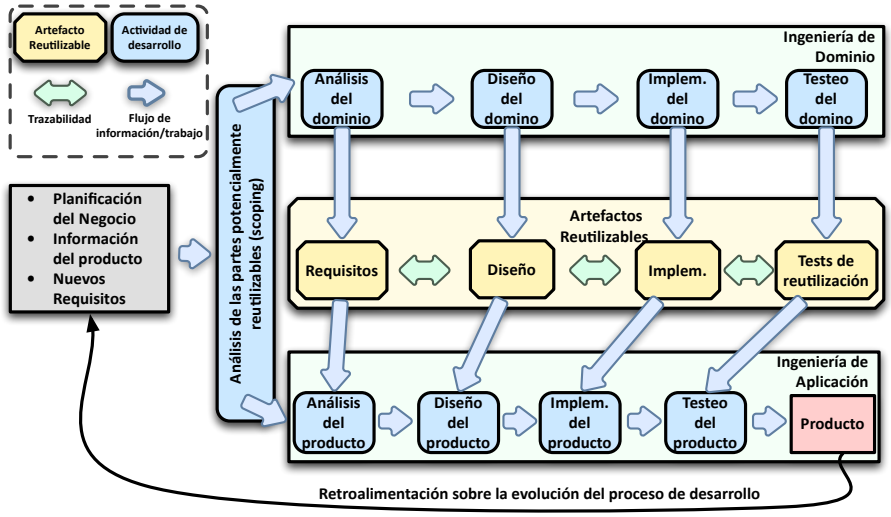


Figura 3.1: Ciclo de vida de las SPL.

Ingeniería de la aplicación: Este proceso se encarga tanto de la definición como de la integración dentro de los sistemas, de las partes específicas de cada uno de los productos.

Esta aproximación ha sido utilizada con éxito en muchos dominios que van desde la biomedicina, a la aviónica pasando por los sistemas de información y el desarrollo de software en general.

Centrado en el desarrollo de interfaces y de la interacción entre usuarios y dispositivos, Gil et al. en [19] utilizan modelos de características para definir los componentes de la interfaz de usuario disponibles en el sistema y su uso adecuado en función de la situación del usuario y su contexto.

La Figura 3.2 muestra un esquema de la aproximación propuesta por Gil et al. en que se puede observar que diferentes configuraciones del modelo de características (según el contexto de un mismo servicio) acarrean diferentes selecciones en el árbol de controles de interfaz y por ende, diferentes interfaces de usuario.

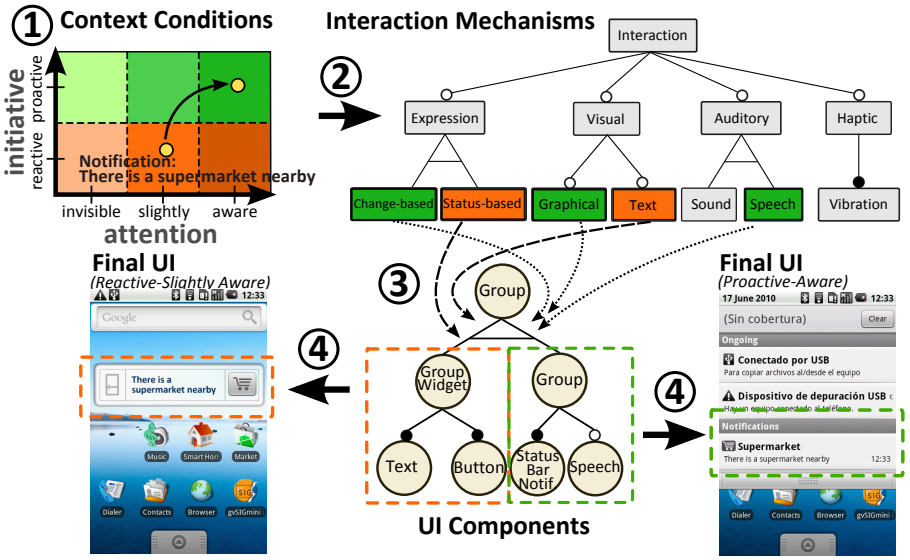


Figura 3.2: Aproximación propuesta por Gil et al.

Este trabajo es bastante completo en lo referido a la creación y modificación de las interfaces sensibles al contexto, pero no se contempla la posibilidad de introducir la capacidad de obtener interfaces para múltiples dispositivos. Resulta especialmente interesante el uso de diagramas de características para definir la interacción, y las correspondencias que se establecen entre éstos y los árboles de controles de interfaz de usuario.

3.2. Desarrollo Software Dirigido por Modelos

Este trabajo está enmarcado en el ámbito Model-Driven Development (MDD), que traducimos por Desarrollo de Software Dirigido por Modelos [28]. Este paradigma propone el uso de modelos conceptuales para representar los sistemas de forma abstracta. Gracias a una serie de transformaciones, previamente definidas, el analista puede generar el sistema representado a partir de los modelos conceptuales. La Object Management Group (OMG) ha propuesto un estándar MDD llamado

Model Driven Architecture (MDA) [5].

El Desarrollo de Software Dirigido por Modelos (DSDM) y más concretamente la propuesta MDA de OMG constituyen una aproximación para el desarrollo de sistemas software. Esta aproximación está basada en la separación entre la especificación de la funcionalidad esencial del sistema y la implementación de dicha funcionalidad usando plataformas de implementación específicas.

La iniciativa MDA cubre un amplio espectro de áreas de investigación (metamodelos, perfiles UML, transformaciones de modelos, definición de lenguajes de transformación (QVT), construcción de modelos PIM y PSM y transformaciones entre ellos, construcción de herramientas de soporte, aplicación en métodos de desarrollo y en dominios específicos, etc.). Algunos de estos aspectos están bien fundamentados y se están empezando a aplicar con éxito, otros sin embargo están todavía en proceso de definición. En este contexto son necesarios esfuerzos que conviertan a MDA, sus conceptos y sus técnicas relacionadas en una aproximación coherente, basada en estándares abiertos, y soportada por técnicas y herramientas maduras.

En una sesión plenaria celebrada en Montreal entre los días 23 y 26 de agosto de 2004, se acordó la siguiente definición de MDA:

> *"MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformations involved in MDA.*
>
> *MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modeling language must be used. Any modeling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations".*

La principal ventaja de MDD es que el analista es más eficiente en el desarrollo de sistemas, ya que *"sólo"* debe modelar el sistema y es el proceso de transformación el encargado de su implementación. El grado de automatización del proceso de transformación de modelos va desde el cien por cien de automatización, hasta que el analista tenga que aplicar las transformaciones de manera manual. Cuanto mayor sea la automatización del proceso, menor será la carga de trabajo a realizar por el analista.

Las Tecnologías de Transformación de Modelos surgen dentro del ámbito MDD proponiendo el desarrollo de sistemas completamente funcionales a partir de transformaciones automáticas entre modelos de distintos niveles de abstracción [34]. Por lo tanto abogan por el desarrollo de sistemas a partir de modelos conceptuales. Esta tesis se engloba dentro de esta tecnología ya que persigue el mismo fin: la generación del código que represente las interfaces especificadas.

3.2.1. Las Interfaces de Usuario en Entornos MDD

El diseño y la implementación de interfaces de usuario están entre las actividades que más esfuerzo y tiempo consumen en el proceso de desarrollo de software. Además, actualmente es muy común que las interfaces de usuario deban ser desarrolladas para diversas plataformas y dispositivos, a fin de que una aplicación pueda ser utilizada con las múltiples opciones que existen hoy en día. Ésto, junto con las preferencias y características propias de los usuarios finales y la diversidad de ambientes en los que las interfaces de usuario pueden ser utilizadas, añaden aún más complejidad al proceso de desarrollo de interfaces de usuario.

De entre las muchas técnicas de modelado de interfaces existentes, en nuestra propuesta utilizamos dos: (1) el modelo abstracto, con el fin de abstraer de problemas referentes a la tecnología, y (2) el modelo concreto como representación aproximada del resultado final de la interfaz.

El Modelado Abstracto de Interfaces

El primero de los modelos que vamos a estudiar es un Modelo Abstracto de Interfaz de Usuario. Con este modelo se especificarán qué datos serán visibles en nuestra interfaz y como estarán relacionados entre ellos.

El modelado abstracto de interfaz de usuario es una técnica ampliamente utilizada [10, 25, 29] por el alto nivel de abstracción que proporciona, y el alto grado de expresividad independiente de la plataforma destino.

Este concepto cobra una elevada importancia cuando el propósito es el desarrollo para múltiples dispositivos, como es nuestro caso, ya que se requiere de una descripción totalmente independiente de características referentes a la representación y que permita centrarse en la información.

La propuesta que presentamos dispone el uso de modelos abstractos de interfaz para especificar los aspectos de interacción de la interfaz, sin preocuparnos de las limitaciones de los dispositivos.

Este componente especifica qué información va a procesar nuestra interfaz, o lo que es lo mismo, que información va a tener disponible el usuario de la aplicación en función de su perfil. Este es un modelo conceptual que permite especificar el conjunto de componentes de interacción que definen la interfaz del usuario con el sistema de información.

En el contexto de este modelo definimos los componentes de interacción como elementos de modelado que describen el comportamiento de la interacción que espera el usuario, pero sin tener en cuenta cómo la interacción que abstraen será implementada.

En consecuencia, este modelo es abstracto ya que la definición de la interfaz se realiza de forma independiente a la tecnología utilizada para llevar cabo dicha interacción.

MOSKitt UIM (ver Sección 3.2.2) es el proyecto dentro de la herramienta MOSKitt que introduce primitivas de abstracción que permiten la representación de la navegación, filtrado de la información que se debe visualizar así como su tratamiento, la ejecución de servicios definidos en la capa de negocio, personalización de la información para los distintos usuarios, definición de patrones de interfaz para su reutilización...

Esta expresividad proporcionada por UIM nos permite abstraer totalmente la información que queremos mostrar,del tipo de interacción del dispositivo destino, o los controles que utilizaremos para mostrarla. Es por ésto que es la aproximación que utilizaremos en GeMMINi.

El Modelado Concreto de Interfaces

Las técnicas de modelado abstracto de interfaz tiene la ventaja del alto nivel de abstracción, pero tiene el inconveniente en la complejidad que dicho nivel de abstracción comporta. Por ello, cada vez más se utilizan técnicas de modelado concreto para especificar las interfaces de usuario.

Entre las técnicas existentes en este ámbito presentamos las siguientes:

Esbozado a *mano alzada*: Un boceto o *sketch* es un dibujo hecho a mano alzada, utilizando lápiz y papel, realizado generalmente sin instrumentos de dibujo auxiliares. Puede ser un primer apunte del resultado ideado que aún no está totalmente definido. Es un dibujo rápido de lo que luego llegará a ser un dibujo definido o la obra de arte final en sí.

Uso de plantillas: Un *stencil* o plantilla es un conjunto de representaciones más o menos aproximadas a la realidad de distintos controles que se utilizan en programas de dibujo (i.e., MS Visio, Omnigraffle, etc.). Componiendo los diferentes objetos que forman el stencil podemos obtener una representación muy aproximada a la realidad que nos permita validar con el cliente la interfaz.

Uso de Esquemas: Un *wireframe* es una representación esquemáti-

ca de una interfaz sin elementos gráficos que muestran contenido y comportamiento. Sirven como herramienta de comunicación y discusión entre arquitectos de información, programadores, diseñadores y clientes. También se pueden utilizar para comprobar la usabilidad de una interfaz.

La principal ventaja es que ofrece una perspectiva basada solamente en la arquitectura del contenido, obviando el diseño y evitando elementos accidentales que puedan distraer (colores, tipografías, imágenes, textos, etc.).

En la actualidad una de las técnicas más exitosas para representar interfaces de usuario es utilizar sketches. El sketching de interfaces de usuario permite representar las interfaces describiendo la apariencia que tendrán una vez desarrolladas. Existen numerosas herramientas que permiten este proceso de esbozado (Axure, Balsamiq, ForeUI...). Estas proporcionan elementos para construir las interfaces usando una notación cercana al usuario. Permiten expresar una representación inicial de como será la interfaz. Ésta se puede mostrar y validar con el cliente en etapas tempranas.

Sin embargo, actualmente la mayoría de los bocetos solo sirven como documentación y no se puede asegurar su validez y/o corrección. Uno de los problemas más graves es que no están enlazados con modelos conceptuales y por tanto no se pueden usar en entornos MDD.

MOSKitt Sketcher es el subproyecto dentro de la herramienta MOSKitt, desarrollado para permitir introducir técnicas de *sketching* en el proceso de desarrollo de software. Sketcher define un lenguaje específico de dominio para introducir estas técnicas en desarrollos con MOSKitt. Esta totalmente integrado en el entorno (gracias a su herramienta de soporte) por lo que podemos utilizar los bocetos tanto como documentación como de entrada para especificar otros modelos, generar código o simplemente diseñar la interfaz.Es por ello, que GeMMINi utiliza MOSKitt Sketcher para representar esos bocetos de interfaz.

3.2.2. Herramientas de Soporte: MOSKitt

MOSKitt[1] (MOdelling Software Kitt) es una herramienta CASE libre, basada en Eclipse que está siendo desarrollada por la *Consellería de Infraestructuras y Transporte* (CIT) de la *Generalitat Valenciana*, con la asesoría técnica del Centro de Investigación en Métodos de Producción de Software[2] de la Universidad Politécnica de Valencia[3], para dar soporte a la metodología gvMétrica (una adaptación de Métrica III a sus propias necesidades). GvMétrica utiliza técnicas basadas en el lenguaje de modelado UML.

Su arquitectura de plugins la convierte no sólo en una Herramienta CASE sino en una plataforma de modelado en software libre para la construcción de este tipo de herramientas. Está diseñada siguiendo una arquitectura modular para que pueda ser fácilmente extendida y/o adaptada en un futuro.

Las tareas principales a las que da soporte MOSKitt en el proceso de desarrollo de software son las siguientes:

- Edición gráfica de modelos.
- Soporte a la persistencia.
- Soporte al trabajo colaborativo y versionado de modelos.
- Transformación, trazabilidad y sincronización de modelos.
- Generación de documentación y de código DDL a partir de modelos.

De entre los muchos lenguajes de modelado y diagramas que maneja y soporta MOSKitt, destacamos UML (diagrama de clases, casos de uso...), diagramas de secuencia, UIM (User Interface Modelling), Sketcher, etc.

[1] http://www.moskitt.org
[2] http://www.pros.upv.es
[3] http://www.upv.es

MOSKitt UIM

Para llevar a cabo el modelado abstracto de la interfaz de usuario en GeMMINi utilizamos MOSKitt UIM. **MOSKitt UIM**[4] (en adelante UIM) es un lenguaje de alto nivel para especificar los requisitos de las interfaces de Usuario. La naturaleza anidada de este tipo de lenguajes permite que UIM se represente como un árbol.

El lenguaje propuesto por UIM introduce primitivas de abstracción que permiten la representación de la navegación, filtrado de la información que se debe visualizar así como su tratamiento, la ejecución de servicios definidos en la capa de negocio, personalización de la información para los distintos usuarios, definición de patrones de Interfaz para su reutilización, etc.

UIM está enfocado a la Capa de Presentación y se relaciona con otros modelos (tradicionalmente UML2, Base de Datos, BPMN etc.) utilizados para especificar las otras dos capas.

En UIM es muy importante el concepto de "visibilidad" entendida como que desde la Capa de Presentación se hacen visibles los datos y los servicios especificados en las otras dos capas. Así pues, en la definición del lenguaje encontraremos conceptos como:

- Propiedades o Atributos visibles
- Operaciones o Servicios visibles
- Parámetros visibles
- etc.

Entre las primitivas presentes en UIM, las que se utilizan en la descripción del caso de estudio (ver Capítulo 9) las podemos ver en la Tabla 3.1.

En la Figura 3.3 podemos ver un ejemplo de modelo UIM. Éste modela un fragmento de un sistema de revisión de conferencias, que es

[4] User Interface Modelling

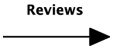
Figura	Nombre	Representación
	Usuario	Representa un Rol para el que se definirá un acceso al sistema. Normalmente representa responsabilidades detectadas durante el análisis del negocio.
	Vista	Define una interfaz de acceso al sistema en términos de Unidades de Interacción.
	Unidad de Interacción de Información	Recuperan y muestran información **no editable** de las estructuras de datos del sistema.
	Unidad de Interacción de Operación	Unidad de Interacción básica que permite ejecutar una operación (invocar un servicio) o un conjunto de operaciones de Negocio desde la interfaz.
	*Visualization Set*	Representa una vista sobre las estructuras de datos del sistema.
	Clase Visible (Atributo y Operación Visible)	**Clase Visible:** Determina qué entidades (clases UML, etc.) van a ser visibles a través del VS. **Atributo visible:** Propiedades visibles de la presente clase para esta UI. **Operación visible:** Operaciones accesibles de la presente clase para esta UI.
	Relación de recuperación	Se utilizan para visualizar información procedente de diferentes fuentes de datos relacionadas.
	Navegación	Enlaces a otras unidades de interacción.

Tabla 3.1: Notación UIM utilizada en el SRC

el caso de estudio que presentaremos en el Capítulo 9 En este ejemplo se modelan de manera abstracta las interfaces que permiten a un autor consultar las revisiones que han obtenido sus contribuciones a la

conferencia. En primer lugar (1) se consultan los datos del autor y las contribuciones que ha realizado;(2) el detalle de una contribución y (3) el detalle de una revisión.

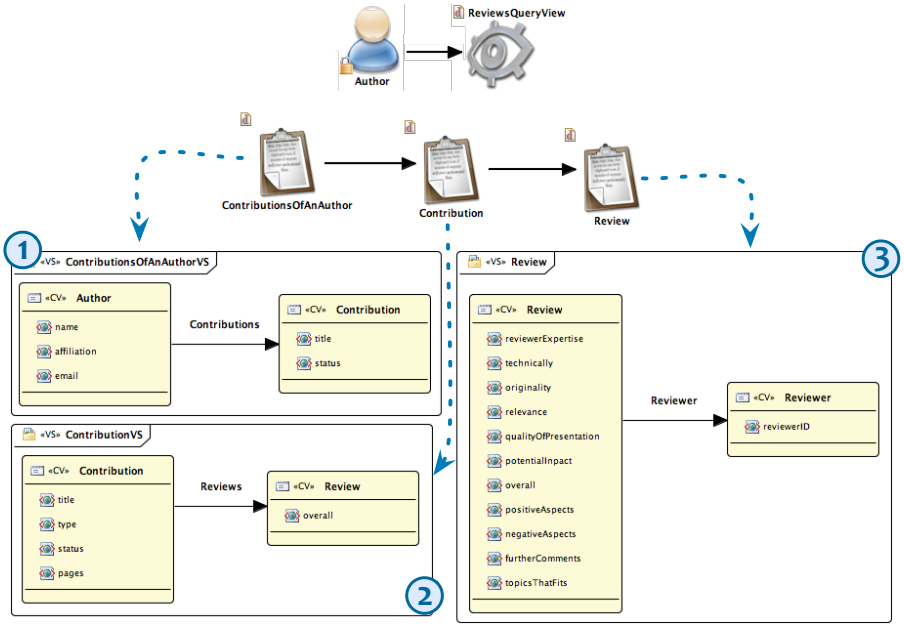


Figura 3.3: Reviews Query View

Este y otros modelos UIM se expondrán de manera mas amplia en el caso de estudio presentado en el Capítulo 9.

MOSKitt Sketcher

MOSKitt UIM nació debido a la carencia de estándares que formalicen el desarrollo de interfaces de usuario, pero tiene como puntos débiles la ausencia de mecanismos para representar aspectos estéticos o su alta complejidad en su notación, causada por su riqueza y flexibilidad.

Por ello en MOSKitt se han introducido técnicas de modelado concreto utilizando bocetos como modo para realizar la especificación de

las interfaces de usuario.

La herramienta utilizada para la manipulación de estos bocetos de Interfaz es **MOSKitt Sketcher**. Esta es otra herramienta dentro del entorno MOSKitt desarrollada para introducir técnicas para la creación y manipulación de bocetos en el desarrollo de interfaces de usuario.

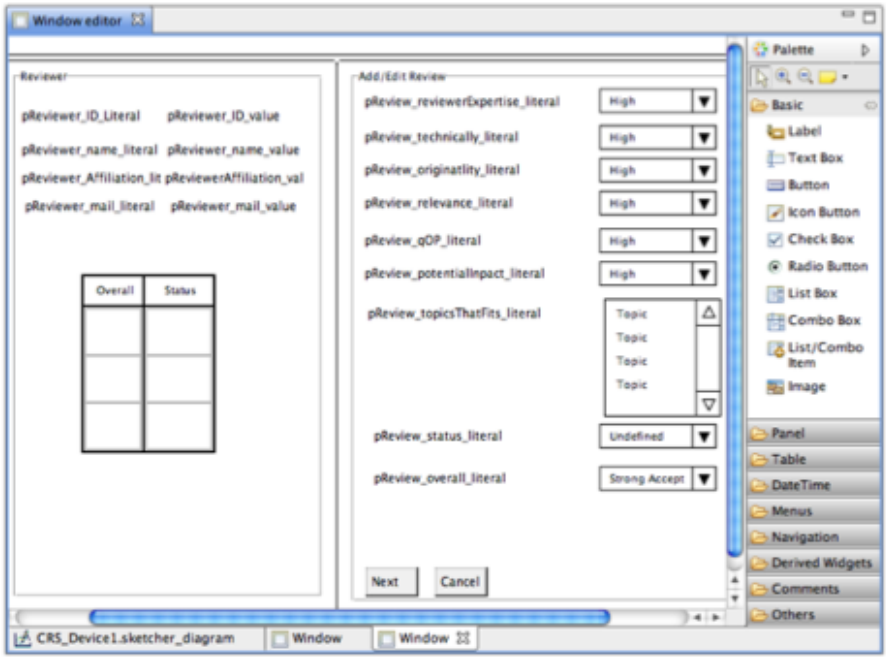


Figura 3.4: MOSKitt Sketcher

En la Figura 3.4 podemos ver la herramienta en funcionamiento.

Utiliza la notación estándar de las herramientas de sketching, por lo que resulta mas intuitiva y familiar a los desarrolladores. El listado concreto de controles soportados por esta herramienta se pueden consultar en el Anexo B.

MOSKitt Sketcher permite su uso con diferentes objetivos:

Diseñador de interfaces: Especifica los componentes a usar en el

desarrollo de la interfaz y su organización. En esta fase es habitual disponer *a priori* de diagramas tipo UML, UIM, etc.

Analista de sistemas: desde fases de desarrollo tempranas se podrá bocetar la interfaz y a partir de estas derivar otros modelos como UIM, UML, etc.

3.3. Estado Actual de los Dispositivos

En la actualidad estamos viendo aparecer prácticamente a diario nuevos dispositivos que se unen a los ya existentes creando una amplia oferta. La tendencia actual dicta que estos dispositivos, en lugar de tender a unificar tienden a aportar nuevos paradigmas como las emergentes tabletas o netbooks. Existe ademas una tendencia de actualización de estos dispositivos que hace que prácticamente cada 12-18 meses salga al mercado una nueva versión de los dispositivos.

Por otra parte tenemos a las empresas desarrolladoras de los sistemas operativos para estos dispositivos, las cuales, en su afán de crecimiento y diferenciación de la competencia, están inmersos en unos ciclos de evaluación continua que hacen que prácticamente cada 8-10 meses se lance una versión nueva del sistema operativo. Estas versiones nuevas no solo corrigen errores, sino que ofrecen nuevas funcionalidades e incluso cambian por completo el sistema.

Por otra parte tenemos la situación de los desarrolladores de aplicaciones, que se encuentran ante un volumen muy elevado de dispositivos con pocas capacidades de unificación y que ven como sus aplicaciones quedan prácticamente obsoletas en poco tiempo.

Todos estos factores hacen cada día mas necesarios mecanismos que faciliten la creación de estas aplicaciones de una manera rápida y reduciendo el esfuerzo mediante el fomento de técnicas que faciliten la reutilización de ciertos componentes.

3.4. Conclusiones

En el presente capítulo se han presentado los fundamentos metodológicos en que se basa este trabajo de tesis de máster. En concreto se han detallado las técnicas del desarrollo de software utilizando principios de líneas de producto y del desarrollo de software dirigido por modelos.

Se han presentado las técnicas de modelado tanto abstracto como concreto que utiliza GeMMINi así como el conjunto de herramientas que las soportan.

Estas técnicas suponen el contexto tecnológico del trabajo y por tanto serán utilizadas a lo lardo de todo el documento en los siguientes capítulos.

— ❝*Progress lies not in enhancing what is, but in advancing toward what will be.*❞

Kahlil Gibran

4

Visión General de la Propuesta

La cada vez más amplia variedad de dispositivos y la hetereogeneidad entre ellos, introduce nuevos retos y oportunidades en el diseño y desarrollo de interfaces para estos dispositivos. A la aparición de nuevos tipos de dispositivos (i.e., tabletas) se une el rediseño de algunos ya existentes (i.e., TV) ofreciendo nuevos modos de interacción [30] [31] lo que provoca que, en lugar de tender a la unificación, cada vez se producen más desigualdades entre los dispositivos [42].

Estas desigualdades han llevado a acuñar el concepto de *fragmentación de dispositivos*. Esta fragmentación de dispositivos ofrece nuevos retos a la comunidad de Interacción Persona-Ordenador (HCI), como

(1) la creación de aplicaciones para varios de los dispositivos existentes en el mercado, o (2) el mantenimiento de distintas versiones ofreciendo un mismo modo de interacción [8].

Este capítulo sienta las bases del trabajo presentado en esta tesis e introduce GeMMINi como una solución que sustenta estas bases. Con la aplicación de esta propuesta conseguimos **desarrollar interfaces para múltiples dispositivos utilizando un enfoque de líneas de producto software basado en el desarrollo dirigido por modelos**.

Este capítulo se estructura como sigue: en la Sección 4.1 se justifica la elección de un enfoque dirigido por modelos y en la Sección 4.2 se presentan las bondades de aplicar principios del desarrollo basado en SPL a la propuesta y en la Sección 4.3 se exponen las bondades de integrar MDD y SPL. En la Sección 4.4 se introduce la propuesta y finalmente la Sección 4.5 concluye el capítulo.

4.1. ¿Porqué un Enfoque Basado en Modelos?

La comunidad HCI hace mucho tiempo que considera el uso de modelos. En un contexto en que existen múltiples combinaciones de dispositivos, modos de interacción, etc., las implementaciones *ad-hoc* para cubrir todas las soluciones puede no resultar factible o rentable. Sottet et al. en [41] presenta este problema y destaca la relevancia de MDE para el modelado de la interacción .

MDE propone el uso de modelos para especificar los aspectos deseados de un sistema y obtener el código de manera automática para el dispositivo destino especificado.

La abstracción es un principio fundamental en la ingeniería del software. Con esta abstracción podemos describir la interfaz sin preocuparnos de los aspectos del dispositivo destino que podrían condicionar el desarrollo de la interfaz. Pinheiro da Silva en [36] presenta los siguientes beneficios que se obtienen al aplicar técnicas de modelado en el desarrollo de interfaces de usuario:

- Los modelos pueden aportar una descripción mas abstracta de la Interfaz de Usuario (IU) que las obtenidas con otras herramientas de desarrollo de IUs.
- Los modelos facilitan la creación de métodos para diseñar e implementar IU de manera sistemática ya que ofrecen: (1) diferentes niveles de abstracción a los modelos de interfaz de usuario; (2) la posibilidad de refinamiento de dichos modelos; y (3) brinda la posibilidad de reutilizar especificaciones de IUs.
- Los modelos proveen la infraestructura necesaria para automatizar las tareas relacionadas con los procesos de diseño y desarrollo de interfaces de usuario.

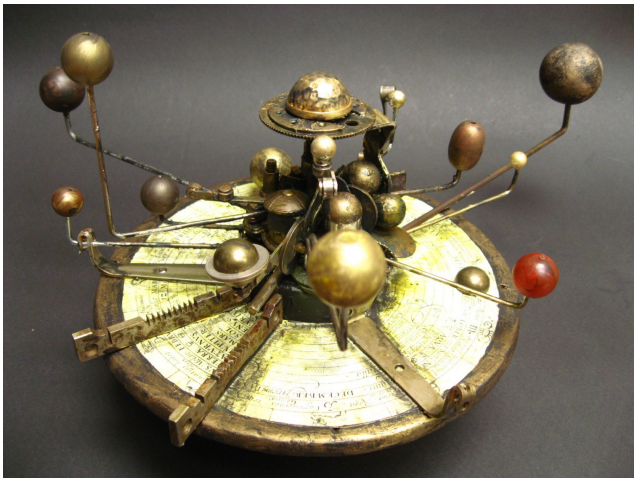


Figura 4.1: Abstracción de la estructura y el funcionamiento del sistema solar

4.2. ¿Porqué una Aproximación Basada en Líneas de Producto Software?

Utilizando el concepto de plataforma que veremos en la Sección 5.3, en la propuesta presentada en este documento se aplican principios y

técnicas de Líneas de Producto Software. La aplicación de estas técnicas proporciona una mejora en la productividad al gestionar las similitudes y diferencias de los productos, fomentando la reutilización y automatización en los procesos de desarrollo de software.

En la Figura 4.2 podemos ver una comparación esquemática entre los costes que acarrea el desarrollo convencional y los que acarrea el desarrollo aplicando principios y técnicas de Líneas de Producto Software.

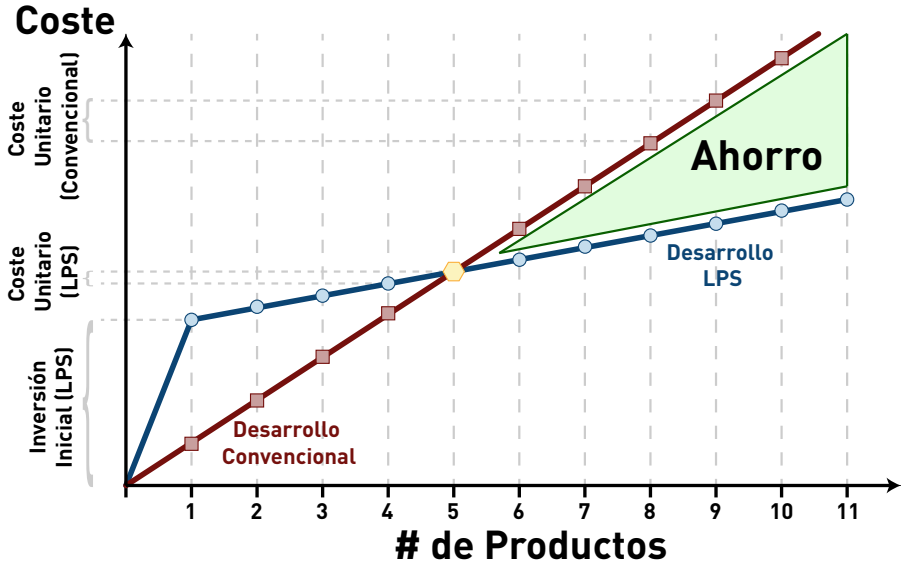


Figura 4.2: Desarrollo basado en LPS vs. Desarrollo convencional

La aplicación de estas técnicas supone una inversión inicial muy elevada que se ve subsanada dado el bajo coste unitario de los productos subsiguientes gracias a la automatización y la reutilización de componentes comunes. En cambio el desarrollo convencional, acarrea siempre los mismos gastos en una tendencia linealmente creciente.

4.3. Integrando el Desarrollo Dirigido por Modelos y las Líneas de Producto

Son bien conocidas las bondades del desarrollo de software dirigido por modelos en términos de aumento de la calidad de los productos y la reducción del tiempo de mercado, gracias a los altos niveles de abstracción que ofrecen. La ingeniería de líneas de producto provee de la habilidad para crear mantener y evolucionar un conjunto de productos similares utilizando los principios de variabilidad y reutilización como hemos visto anteriormente.

Con MDD, la creación de un paquete de productos similares, tradicionalmente se ha abordado mediante dos aproximaciones. La primera de ellas consiste en copiar y modificar los productos (modelos) ya existentes incorporando los nuevos componentes que sean necesarios. Esta aproximación fomenta la reutilización pero se puede dar el efecto bola de nieve causando grandes problemas.

La segunda aproximación consiste en crear una solución lo suficientemente genérica y abstracta que contemple "todas" las posibles variantes. Esta aproximación suele ser muy cara y en ocasiones imposible dado el amplio espectro que deberían cubrir las soluciones.

Con la integración de las técnicas del desarrollo dirigido por modelos y el desarrollo basado en líneas de producto obtenemos una solución sinérgica de ambas tecnologías.

Desde el punto de vista de MDD, la sinergia MDD-SPL nos aporta facilidades para expresar mediante modelos de datos como UML los puntos comunes y variables de la ingeniería del dominio en la etapa de diseño de una SPL. Así, cada punto de variación de la SPL es visible mediante los modelos. Esto permite realizar un modelado completo de la variabilidad de una línea de producto.

Desde el punto de vista de las SPL, la variabilidad nos permite introducir el uso de distintos modelos en función de los requisitos introducidos para los distintos productos que se puedan derivar de la línea, sin la necesidad de incorporarlos a la descripción de todos los productos.

4.4. GeMMINi: *Generation Method for Multiple Devices Interfaces*

Al enfrentarnos a un proceso de desarrollo de interfaces de usuario en múltiples dispositivos, existen tres preguntas que debemos plantearnos:

- ¿Qué **información** se va a mostrar en nuestra interfaz?
- ¿Qué **controles** utilizaremos para representar la interfaz?
- ¿En qué **dispositivos** se mostrará la interfaz?

Por tanto, la propuesta presentada debe proporcionar mecanismos que nos permitan especificar la información necesaria y requerida para responder a cada una de estas tres preguntas.

Necesitaremos pues un artefacto que nos permita representar la información a mostrar en nuestra interfaz de usuario de manera independiente a los dispositivos destino. Necesitaremos también un mecanismo que permita especificar cómo se va a representar esa información en la pantalla de los dispositivos. Por último necesitaremos especificar también las características de interacción con que cuentan los dispositivos que van a ejecutar la interfaz.

Esta tesis presenta **GeMMINi**, una solución al desarrollo de interfaces de usuario en múltiples dispositivos que utiliza técnicas de desarrollo dirigido por modelos (MDD) y conceptos y técnicas de Líneas de Producto Software (SPL). Este método sustenta las bases y requisitos presentados anteriormente.

La Figura 4.3 muestra los componentes (modelos y transformaciones) que utilizará GeMMINi para cubrir los requisitos planteados por las preguntas anteriores. Estos componentes se describen a continuación:

1. **Especificación abstracta de la Interfaz de Usuario:** Con esta especificación, obtenemos descripciones conceptuales de interfaces de usuario, que abstraen de problemas relacionados con las características de los dispositivos destino y se centran en describir la interfaz del usuario. El modelado abstracto de interfaces

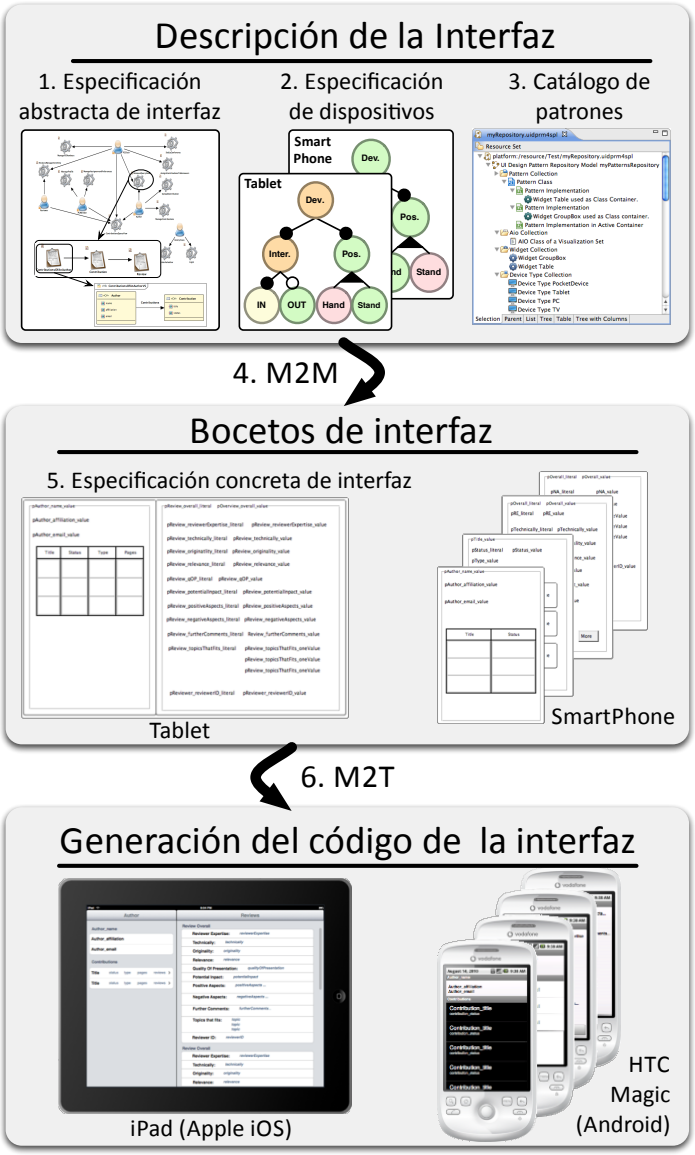


Figura 4.3: GeMMINi. Componentes del método y las relaciones entre éstos

de usuario es una técnica ya existente y ampliamente utilizada. En la mayoría de los casos, estos modelos se complementan con un modelo de datos (típicamente UML). Este artefacto junto con el modelo de datos, nos permitirá representar la información que manejará nuestra interfaz abstrayendo de problemas referidos a la tecnología o a los dispositivos. En el Capítulo 3 se puede ver la aproximación utilizada en esta tesis.

2. **Especificación de las características de los dispositivos:** Mediante este caracterización obtenemos descripciones de propiedades y variantes de dispositivos utilizando modelos de características. Está centrada en describir aspectos que afecten a la interacción con los usuarios. En el Capítulo 5 se puede consultar una explicación mas amplia de este componente. Éste nos ayudará a especificar las características de interacción con que cuenta el dispositivo, tanto a nivel de mecanismos de entrada y salida propia del dispositivo, como la interacción con el usuario (i.e., cerca o lejos del usuario.)

3. **Catálogo de patrones de interfaz:** Este modelo representa un catálogo de soluciones específicas para cada dispositivo a partir de conceptos del modelo abstracto de interfaz. Permite configurar la transformación que obtiene los bocetos, indicando las implementaciones válidas (en forma de objetos concretos de interfaz), para cada tipo de dispositivo. Con este artefacto conseguimos mantener una correspondencia entre los objetos abstractos de interacción (AIO) encontrados en el modelo abstracto de interfaz, y los Objetos Concretos (CIO) que utilizaremos para representar nuestro modelo Concreto de interfaz. Este catálogo se expone en profundidad en el Capítulo 6.

4. **Transformación M2M para obtener bocetos:** Esta transformación genera modelos de bocetos de interfaz de usuario específicos para cada dispositivo. Tomando como entrada los tres modelos anteriores, permite obtener un boceto como especificación concreta de la interfaz adaptada a las capacidades de interacción del dispositivo elegido. Este algoritmo se ejecutará de manera asistida

por el analista que tendrá la potestad de tomar algunas decisiones en lo que al diseño se refiere. Este algoritmo de transformación se estudiará en profundidad en el Capítulo 7, y en el Anexo C se puede consultar una ejecución completa del algoritmo para un caso de estudio.

5. **Especificación concreta de la interfaz de usuario:** Tomando como entrada el resultado de la transformación M2M anterior se obtiene un boceto de interfaz por cada dispositivo destino especificado. Este modelo se puede refinar para obtener un boceto mas adecuado o reconsiderar alguna decisión tomada en la transformación anterior. Estos bocetos son una representación aproximada del resultado final, y son muy útiles para validar con el usuario las interfaces que se van a obtener. En el Capítulo 3 se exponen algunas de las técnicas mas utilizadas para el bocetado así como la que utilizamos en este trabajo.

6. **Transformación M2T para generar el código de la interfaz:** A partir del boceto ya refinado, se procede a la generación de la interfaz de usuario adecuada a la plataforma destino. Este proceso se estudia en profundidad en el Capítulo 8.

4.4.1. Especificación de GeMMINi

Existen muchas métodos de desarrollo basados en ideas subyacentes que no están definidos de manera explícita (RUP, XP, M3, etc.) El formato en que se maneja la información en estos casos, suelen ser documentos de texto, que además son obtenidos manualmente.

Para evitar dicha situación, la OMG (Object Management Group) ha desarrollado y aprobado recientemente el estándar **SPEM** (*Software and Systems Process Engineering Metamodel*) versión 2.0, que pretende ser el estándar industrial para la representación de modelos de procesos de ingeniería del software e ingeniería de sistemas [22].

SPEM 2 es un estándar de metamodelado que sirve para representa procesos de ingeniería de software, entendiendo un Proceso Software

(PS) como *"Un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software"*.

SPEM 2 se encuadra dentro de la Ingeniería de Procesos de Software, el área de la ingeniería de software dedicada a *"la definición, implementación, medición y mejora de los procesos de ingeniería de software"*. Gracias al uso de SPEM 2, se puede disponer de modelos de PS, lo que proporciona capacidades para:

- Facilitar la comprensión y comunicación humana.
- Facilitar la reutilización.
- Dar soporte a la mejora de procesos.
- Dar soporte a la gestión de procesos.
- Guiar la automatización de procesos.
- Dar soporte para la ejecución automática.

En la Figura 4.4 se puede ver la especificación del proceso propuesto por GeMMINi utilizando SPEM.

Como podemos ver, la aplicación de GeMMINi empieza con dos **actividades** realizadas por el analista: el modelado abstracto de la interfaz de usuario y la especificación de las características de los dispositivos. A estos modelos se les une el **work product** representativo de la especificación del catálogo de patrones para, de manera automática y semi-asistida generar el boceto de interfaz de usuario.

Este boceto puede ser refinado por parte del diseñador en otra actividad. Una vez obtenido el boceto definitivo, queda la actividad automática de generar el código nativo de la interfaz de usuario.

Las primitivas utilizadas se pueden consultar en la Tabla 4.1.

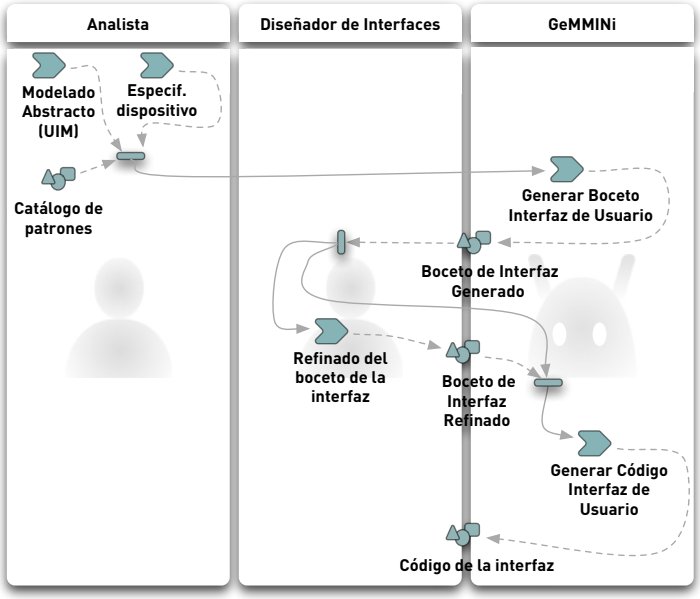


Figura 4.4: Descripción de GeMMINi con SPEM

Figura	Nombre	Significado
	***Activity***	Representa las actividades que se realizan dentro del proceso, y los productos, roles... asociados.
	***Work Product***	Representa un producto de trabajo de entrada o salida relacionado con una actividad o tarea
	***Parallel***	Representa la conjunción o disyunción en paralelo de dos o mas líneas de trabajo.
	***Role***	Representa al rol (persona o máquina) que realiza una actividad o tarea dentro de un proceso.

Tabla 4.1: Notación SPEM

4.5. Conclusiones

Este capítulo ha mostrado una visión general del método, los artefactos de que se compone y las relaciones entre éstos. GeMMINi es un propuesta de solución integral para la generación asistida (semi-automática) de interfaces para múltiples dispositivos. Ésta está basada en la aplicación de conceptos de Líneas de Producto, utilizando una aproximación en el marco del Desarrollo Software Dirigido por Modelos, lo que proporciona una aproximación que permite la reutilización de algunos de sus componentes, separando los artefactos de análisis de los de diseño e implementación, al tiempo que nos abstrae de errores típicos en este tipo de desarrollo.

Esta necesidad de abstracción junto con la posibilidad que nos brinda de aplicar un enfoque metodológico, justifican la aplicación de un enfoque basado en modelos. Se han presentado los artefactos de que está compuesto el método así como una especificación formal de éste utilizando SPEM. En los siguientes capítulos se van a explicar en profundidad estos artefactos y las relaciones entre ellos.

— ❝*Whatever the device you use for getting your information out, it should be the same information.*❞

Tim Berners-Lee

5

Especificación de las Características de los Dispositivos

Una vez detallado el contexto tecnológico y los fundamentos metodológicos utilizados en GeMMINi vamos a encargarnos de los artefactos mas importantes que componen el método.

Uno de los factores que debemos tener en cuenta en el proceso de desarrollo de la interfaz de usuario para una aplicación, es el dispositivo en que va a ejecutarse. El hecho de separar la especificación del dispositivo de la del modelo abstracto de interfaz (Capítulo 3), nos aporta un nivel de desacoplamiento necesario, dado que sean cuales sean los mo-

delos de dispositivos y/o de interfaz, éstos pueden ser reutilizables. Es decir, un mismo modelo de dispositivo lo podremos utilizar con varios modelos abstractos de interfaz y viceversa.

En el mercado de los dispositivos, existe una gran heterogeneidad, en la que prácticamente cada una de las principales marcas de dispositivos dispone de su propio sistema operativo (Windows Mobile, Symbian, RIM, MAC, Android, iOS, ...). Ésto se une a una visión de un mercado altamente volátil, con cambios muy significativos en periodos muy cortos.

Prácticamente cada evolución de los dispositivos, aporta nuevas características (i.e., el giroscopio del iPhone 4) que aportan la posibilidad de establecer nuevos mecanismos de interacción entre el usuario y el propio dispositivo. A esto podemos sumarle que esta heterogeneidad se esta trasladando al nivel "intra-dispositivo" por la gran cantidad de versiones que se lanzan en cortos períodos de tiempo. Esta heterogeneidad de no es una cuestión técnica sino una cuestión *política* entre compañías, que continuamente necesitan desmarcarse de la competencia, y no hay previsiones que sugieran un cambio.

Por ello se hace necesario algún mecanismo que nos facilite especificar estas características de interacción que poseen los dispositivos y que pueden ser utilizadas en nuestros desarrollos de interfaces de usuario.

El resto del capítulo se estructura como sigue: En la Sección 5.1 se analizan las características de interacción a tener en cuenta en los desarrollos y en la Sección 5.2 se expone el modelo de características [24] que utilizaremos para especificar estas características de interacción. En la Sección 5.3 se expone el concepto de plataforma entendido como tipo de dispositivo. En la Sección 5.4 se muestra la herramienta que va a dar soporte a este modelo para, finalmente en la Sección 5.5 concluir el capítulo.

5.1. Características Relativas a la Interacción con Dispositivos

Como decíamos en el Capítulo 2 este trabajo se enmarca en el área de la interacción persona-ordenador (HCI). Podemos definir interacción como una acción recíproca entre dos agentes, fenómenos, etc. En el caso de la HCI, estos dos agentes son una persona y un dispositivo.

Es por esto que debemos caracterizar al interacción con los dispositivos atendiendo a criterios no solo del propio dispositivo sino también del usuario.

Por una parte, atendiendo al usuario, necesitamos saber cual es su posición respecto al dispositivo. El tipo de interacción no será el mismo si el dispositivo se encuentra **cerca o lejos del usuario** o, en caso de utilizarlo con las manos, si el dispositivo requiere de **una o dos manos** para su manejo.

Por otra parte, atendiendo al dispositivo caracterizamos la interacción atendiendo a dos criterios: la entrada y la salida. Con respecto a la **interacción de salida**, Gil et al. en [20] la caracterizan como visual, auditiva o sensorial, lo cual referido a los dispositivos, los podemos entender como **vibración**, **pantalla**, **voz/audio**, etc.

La **interacción de entrada** es un tanto mas compleja. A los mecanismos de interacción clásicos como el **teclado** se deben añadir aquellos que están presentes en los dispositivos móviles (como la propia pantalla si es **táctil o multi-tactil**), la entrada por **voz**, etc. Además, los mecanismos de interacción novedosos han hecho que el **ratón**, como dispositivo apuntador, comparta ahora el protagonismo con otros como el ***joystick***, el ***stencil*** o el ***trackpad***.

A estos debemos unirles otros menos comunes como la **entrada mediante imágenes** para la lectura de códigos de barras o QRCodes[1] e incluso los sensores de magnitudes físicas que incorporan algunos dispositivos móviles como el **GPS**, **altímetros**, **acelerómetros**, etc.

[1] *Quick Response Code*: http://www.denso-wave.com/qrcode/qrstandard-e.html

5.2. Modelo de Características de la Interacción

Para agrupar estas características de interacción presentadas en la sección anterior GeMMINi propone el uso de Modelos de Características.

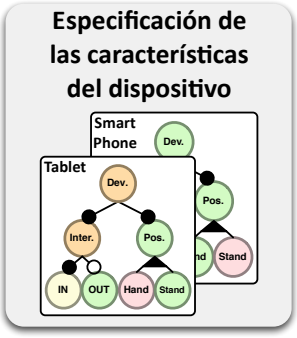
Estos modelos son ampliamente utilizados para describir el conjunto de productos derivados de una Línea de Producto en términos de características. En estos modelos, introducidos por Kang et al. en [24] se representan las características jerarquizadas en un árbol mediante relaciones de variabilidad (opcional, obligatorio, selección única o múltiple) y pueden estar conectadas mediante restricciones. La notación propuesta por Czarnecki & Kim en [13] para estos modelos la podemos ver en la Tabla 5.1[2].

Figura	Nombre	Representación
	Feature	f es la raiz
	Mandatory	$f_1 \Leftrightarrow f$
	Optional	$f_1 \Rightarrow f$
	Alternative	$(f_1 \vee f_2 \vee \ldots \vee f_n \Leftrightarrow f) \wedge \forall_{i<j}\ \nexists(f_i \wedge f_j)$
	Or	$f_1 \vee f_2 \vee \ldots \vee f_n \Leftrightarrow f$
	Requires	$f_i \Rightarrow f_j$
	Excludes	$\nexists(f_i \wedge f_j)$

Tabla 5.1: Notación de los modelos de características

Utilizando pues esta notación, se propone el uso de un componente que especifique las características de interacción con que cuenta el dispositivo destino de la interfaz y que por tanto podremos utilizar en nuestro desarrollo.

[2]Se asume que f_x es subcaracterística de f.



Para abordar la fase de Ingeniería del Dominio de las Líneas de Producto y describir las características de los dispositivos, nos hemos basado en una ontología propuesta por Clerckx et al. en [11] que especifica algunos aspectos de interacción, como la entrada por voz o por ratón o la salida por proyector o pantalla táctil/multi-táctil. GeMMINi propone una extensión que refina los modos de entrada y salida e incorpora otras características como el manejo del dispositivo (1 ó 2 manos) o su posición (cerca o lejos del usuario). El objetivo es poder describir de manera más precisa el entorno de interacción persona-ordenador.

La Figura 5.1 muestra este modelo de características con las extensiones propuestas (entre líneas punteadas) y la Figura 5.2 un ejemplo de éste, instanciado para especificar un iPad.

A partir de estos modelos de características, se inferirá la plataforma a la que pertenece el dispositivo en concreto.

5.3. El Concepto de Plataforma

En la actualidad están apareciendo nuevos dispositivos casi a diario. Estos dispositivos ofrecen mecanismos de interacción adecuados a su forma, tamaño, contexto, etc. Pero si se analizan estos dispositivos observando sus características, entre muchos de ellos no existen diferencias notables (a excepción del aspecto físico). Un ejemplo claro es la comparación entre un iPhone 4 y un Samsung Galaxy S II, que podemos ver en la Tabla 5.2. Atendiendo a sus características Hardware, las diferencias son mínimas.

Otro punto que observar son las aplicaciones. En la Figura 5.3 se pueden ver las aplicaciones de Páginas Amarillas para iPhone (izquierda) y para Android (derecha). Aunque parezcan distintas, los conceptos utilizados son los mismos, pero cada una está implementada según las guías de estilo de cada sistema operativo.

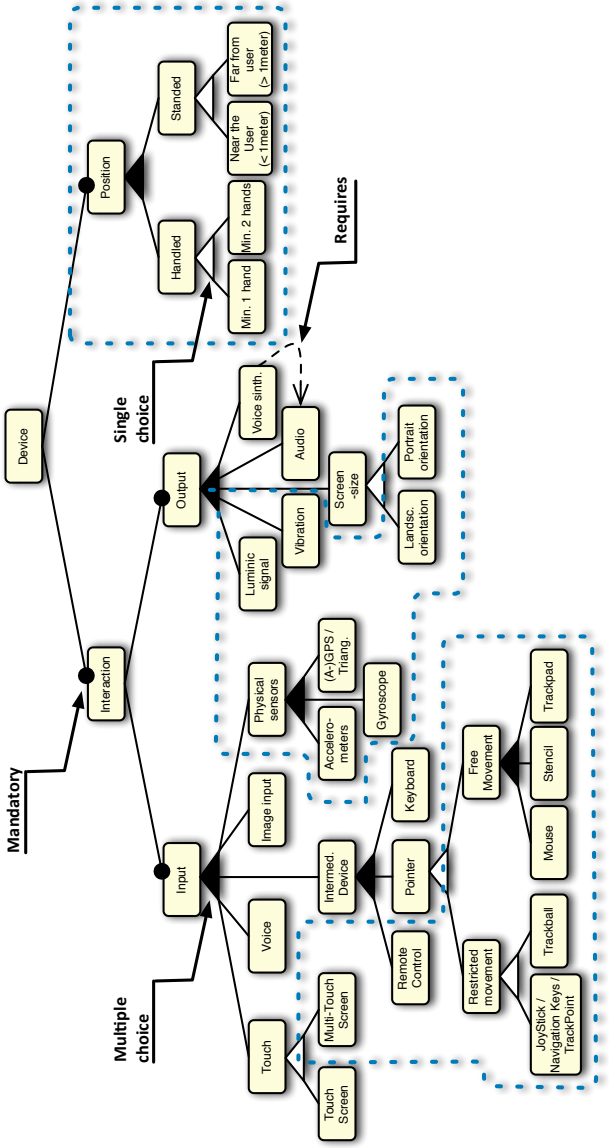


Figura 5.1: Modelos de características de dispositivos

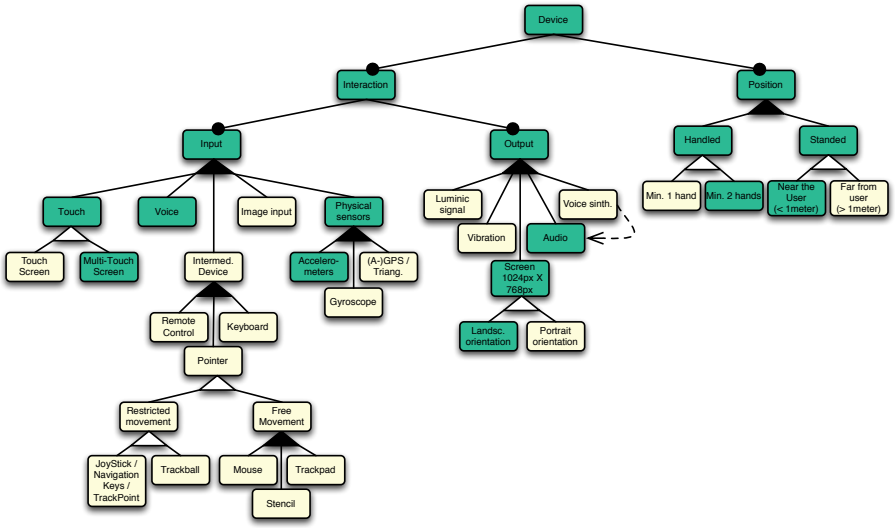


Figura 5.2: Modelos de características instanciado para el iPad

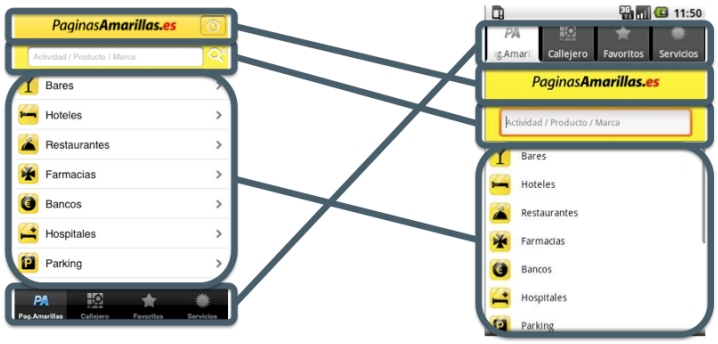


Figura 5.3: Comparación de las aplicaciones de *Páginas Amarillas*.

En la literatura existen definiciones del concepto de plataforma como agrupación de dispositivos que comparten entre ellos ciertas caracterí-sitcas. Por una parte tenemos los trabajos que definen plataforma como una tupla hardware-software que identifica a los dispositivos de manera individual [46]. Por otra parte, tenemos los trabajos que identifican la

	 **iPhone 4**	 **Samsung Galasy S II**
Sistema Operativo	iOS	Android
Procesador	Apple A4 1GHz	Samsung Orion 1GHz
RAM	512MB	1GB
Tamaño de Pantalla	3.5”	4.3”
Resolución	960 x 640	800 x 400
Cámara	Frontal y Trasera	Frontal y Trasera
Teclado	Virtual	Virtual
Conectividad	3G, GPS, 802.11 b/g/n	3G, GPS, NFC, 802.11 a/b/g/n

Tabla 5.2: Comparación entre el iPhone 4 y el Sansung Galaxy S II

plataforma atendiendo a criterios hardware (físicos) sin tener en cuenta criterios de interacción con el usuario [12]. Finalmente tenemos las aproximaciones que lo único que tienen en cuenta es el tamaño de la pantalla [17].

Estos enfoques por separado no atienden a todos los criterios necesarios para aplicar GeMMINi ya que no tienen en cuenta factores como la posición del usuario respecto al dispositivo o si el usuario maneja el dispositivo con las manos o se utilizan dispositivos intermediarios (como el teclado, el ratón o un lápiz). Esto nos llevan a redefinir el concepto de **plataforma**. Para ello nos basamos en las definiciones existentes de este concepto, tomando partes de cada una e incorporando nuevos conceptos que entendemos necesarios. Así pues, entendemos plataforma como tipo de dispositivo, es decir, bajo este término se agruparan todos aquellos

dispositivos que compartan la mayoría de sus características de interacción y posición. En GeMMINi se especificarán las plataformas mediante el modelo de características, y será definido en el catálogo de patrones de diseño como el subconjunto mínimo de características representativas del tipo de dispositivo.

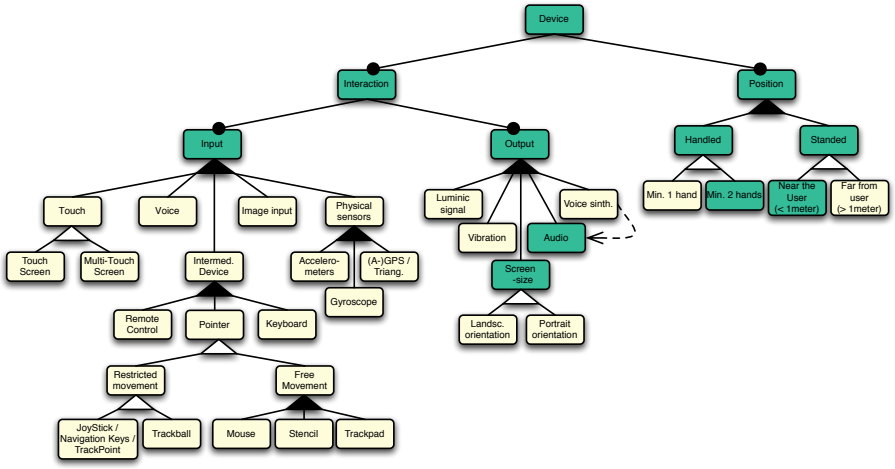


Figura 5.4: Especificación de la plataforma "tableta".

En la Figura 5.4 podemos ver un ejemplo de definición de plataforma. En este caso corresponde a una tableta. Por un lado podemos ver que para utilizarlo con las manos se requieren las dos, y si el dispositivo esta sobre una superficie, éste estará cerca del usuario.

Por otro lado vemos que se requiere de una pantalla y de salida de sonido. En cuanto a la entrada, puede resultar extraño que no se requiera en si de ningún mecanismo de entrada, pero por la notación inherente al modelo de características, esto no se puede dar. Vemos que tenemos una relación *Or* que pende de las características de la interacción de entrada, y esto implica que *al menos una* de ellas deberá estar activa.

El concepto de plataforma cobra importancia en GeMMINi, ya que lo utilizaremos tanto en el Capítulo 6 para establecer la adecuación de los controles a los dispositivos, como en el Capítulo 7 donde los bocetos

generados serán dependientes de plataforma y no de dispositivo.

5.4. Herramienta de Soporte

Aprovechando la arquitectura de plugins que proporciona MOSKitt, se ha desarrollado un paquete de herramientas, para dar soporte a las partes del método propuesto por GeMMINi (que no tienen soporte en MOSKitt) y otros procesos de desarrollo de software basado en líneas de productos dinámicas. Entre ellas encontramos el editor que da soporte a la especificación de modelos de características y también el editor que permite establecer configuraciones en términos de estados de características (activo, inactivo, descartado) sobre los modelos especificados. Ambos se exponen a continuación.

5.4.1. Feature Model 4 SPL

FM4SPL es una herramienta dentro de M4SPL que permite especificar modelos de características. Ofrece la expresividad completa de la notación gráfica de los modelos de características propuesta por Czarnecki & Kim en [13].

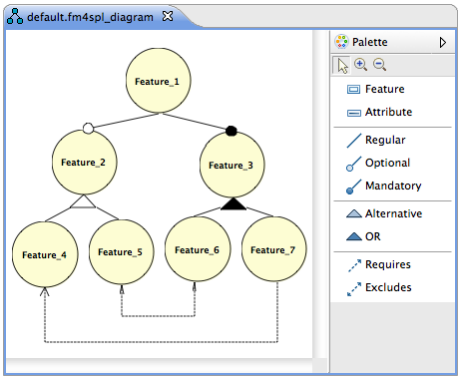


Figura 5.5: Feature Model Editor

En la Figura 5.5 podemos ver el editor de modelos de características

en la que vemos un ejemplo de éstos junto con la paleta. La notación propuesta permite especificar características (*Feature*) con atributos (*Attribute*) que son jerarquizadas mediante relaciones binarias (*Optional* o *Mandatory*), o de grupo (*Alternative* u *Or* con *Regular*). Entre las características se pueden especificar restricciones como requerir de otra característica (*Require*) o excluirla (*Excludes*).

Las principales funcionalidades ofrecidas para la manipulación de estos modelos de características son:

- **Flexibilidad en la notación:** De forma dinámica puede alternase la notación gráfica utilizada para representar los elementos de un modelo.
- **Layout automático en forma de árbol:** FM4SPL puede reorganizar los elementos de un modelo para recrear un árbol gráfico, donde los elementos inferiores son las hojas, el superior la raíz y el resto de elementos están ordenados por niveles.
- **Explosión de características:** Cualquier característica puede explotarse en otro modelo de características. Además la representación gráfica de las características explotadas cambia para ofrecer información sobre la población de su *submodelo*.

5.4.2. Configuration Model 4 SPL

CM4SPL es otra herramienta dentro de M4SPL que permite la creación y edición de modelos de configuraciones. Este modelo toma como entrada un modelo de características y permite especificar en este, ciertas configuraciones para cada característica del modelo (activa, inactiva, descartada, etc.).

En la Figura 5.6 se muestra este modelo donde se pueden ver marcadas como activas las características 1 y 3, como inactiva la 6 y descartada la 5. El resto de características están sin marcar.

En lo que a la visualización y manipulación de los modelos se refiere, comparte todas y cada una de las características con que cuenta FM4SPL.

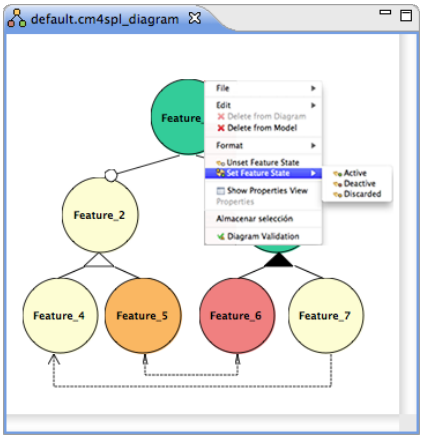


Figura 5.6: Configuration Model Editor

5.5. Conclusiones

En el presente capítulo se ha expuesto el modelo de características utilizado para especificar los dispositivos. Este modelo atiende a dos factores: (1) los mecanismos de interacción que ofrece el dispositivo (tanto de entrada como de salida), y (2) la posición del dispositivo respecto del usuario (ya sea si se coge con las manos o si está apoyado en una superficie).

Se han presentado además dos herramientas. La primera de ellas nos ofrece la posibilidad de especificar modelos de características siguiendo la notación propuesta por Czarnecki & Kim. Con la segunda podemos especificar configuraciones de estos modelos, marcando las características como activas o inactivas, en función del estado en que se presentan en nuestro dispositivo.

En el siguiente capítulo se muestra el catálogo de patrones de diseño. Este catálogo será utilizado para realizar correspondencias entre Objetos Abstractos de Interfaz (AIO) y objetos concretos de Interfaz (CIO) en función de los tipos de dispositivo destino.

— ❝*Elegance is not a dispensable luxury but a factor that decides between success and failure.*❞

Edsger Wybe Dijkstra.

6

Catálogo de Patrones de Interfaz

Las técnicas de modelado abstracto de interfaces de usuario, como las introducidas en el Capítulo 3, son muy útiles cuando nos enfrentamos a un reto como el de desarrollar interfaces de usuario para múltiples dispositivos. El uso de modelos abstractos de interfaz permite al analista abstraerse de problemas relativos al dispositivo o la tecnología que este utiliza.

Éstos modelos, especifican la información que se va a manejar (mediante los modelos de datos) y la agrupan atendiendo a criterios organizativos utilizando primitivas como las unidades de interacción [7]. Pero este nivel de abstracción, que resulta muy útil en etapas tempranas del

proceso de desarrollo, comporta que a medida que se avanza en el proceso se deban tomar las decisiones acerca de cómo se va a representar y manejar esa información. Y cuando hablamos de interfaces de usuario, los mecanismos para representar y manejar la información suelen ser los widgets o controles.

Cuando se deben elegir controles de interfaz de usuario se deben tener en cuenta una serie de factores que pueden condicionar la elección y afectar a la claridad y usabilidad de la aplicación. Se debe hallar un equilibrio entre la adecuación y la usabilidad en función de los dispositivos destino de la aplicación [43]. Entre estos factores destacamos los siguientes:

Espacio disponible: En determinados dispositivos, la cantidad de espacio disponible marca los controles que se pueden utilizar. Algunos, pese a ser más adecuados en ciertas situaciones, no es muy recomendable su utilización dada la porción del espacio disponible en pantalla que requieren.

El usuario con respecto al dispositivo: El usuario final de la aplicación puede que no esté familiarizado con el uso de ciertos controles muy específicos.

El usuario respecto al dominio de la aplicación: El conocimiento del usuario sobre el dominio de la aplicación es un factor importante. Si nos encontramos sólo con usuarios expertos, podemos utilizar unos controles diferentes a si los usuarios no son expertos, ya que en este último caso, los controles deberán ser más intuitivos.

Otras Aplicaciones: Es importante respetar las guías de desarrollo de los dispositivos y las convenciones establecidas. En este caso la curva de aprendizaje requerida para utilizar la aplicación será mas pronunciada y a los usuarios les costará menos *aprender* a utilizar la aplicación.

Por tanto elegir un control para representar o manejar cierto tipo de información no es una decisión trivial, y acarrea ciertas consecuencias.

Los factores que influyen son muy heterogéneos. Unos se pueden conocer de antemano (el dispositivo, otras aplicaciones parecidas,. . .) y otros no (los referentes a las características del usuario). Por ello debemos centrarnos en aquellos que podemos conocer y controlar con anterioridad.

El mismo tipo de información puede ser manejado por controles muy distintos entre si (i.e., podemos manejar una fecha con un calendario o con un campo de texto). Pero todos los controles no son adecuados en todas las situaciones [43]. Por tanto, en la decisión de qué controles utilizar en cada situación, influyen factores como los mecanismos de interacción que ofrece el dispositivo (pantalla táctil, teclado, ratón, lápiz, etc.), la posición del usuario respecto al dispositivo (el usuario suele tener mas cerca la pantalla de su móvil que la de su televisión) o el espacio disponible para representar esos controles (una pantalla de ordenador brinda mas espacio que la de una tableta).

También se debe tener en cuenta que el mercado de los dispositivos está en constante cambio y continuamente aparecen nuevos dispositivos que ofrecen nuevos mecanismos de interacción, al tiempo que aparecen nuevos controles que pueden resultar más adecuados que los existentes para cierto tipo de información.

Por ello, surge la necesidad de almacenar todo ese conocimiento mediante un mecanismo para poder utilizarlo los procesos de desarrollo. Este mecanismo debe permitir establecer correspondencias entre los conceptos abstractos y sus posibles representaciones en forma de objetos concretos. Al tiempo, debe permitir establecer condiciones para estos objetos concretos atendiendo a los criterios expuestos anteriormente. Por ultimo, y dada la constante evolución del mercado, este mecanismo debe poder ser fácilmente modificable y ampliable.

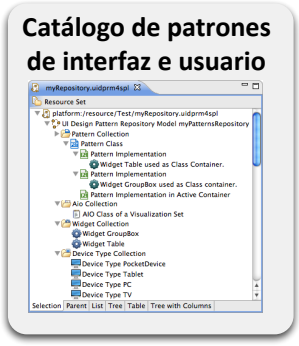
El resto del Capítulo se estructura como sigue: En la Sección 6.1 se expone el modelo dispuesto para almacenar el catálogo de correspondencias entre conceptos abstractos y concretos. En la Sección 6.2 se define el proceso a seguir para su uso y en la Sección 6.3 la herramienta que lo soporta. Finalmente la Sección 6.4 concluye el Capítulo.

6.1. Patrones de Diseño de Interfaz de Usuario

Como veíamos en la introducción de este capítulo, los factores que influyen en la elección de los controles de una interfaz son muchos. En primer lugar debemos atender al concepto abstracto que queremos representar. Este concepto abstracto ya marca la naturaleza del control que se utilizará en la interfaz de manera clara. Por ejemplo, aunque es posible, no se debe utilizar el mismo control para introducir un texto y para seleccionar un elemento de una lista.

En segundo lugar debemos atender al dispositivo, ya que los mecanismos de interacción que ofrece pueden limitar el tipo de control a utilizar. Por ejemplo, en dispositivos con pantallas táctiles o multitáctiles se debe evitar el uso de botones pequeños ya que, al utilizarlo con las manos, es mas difícil pulsarlos.

En tercer lugar se debe atender a los propios controles. Ante un conjunto de controles válidos para representar un concepto abstracto, puede que existan algunos cuyo uso esté mas extendido para el fin dado. Por tanto, si de antemano no se conoce el tipo de usuario al que dirigimos nuestra interfaz, se debe optar por controles fáciles de utilizar e intuitivos.



Para llevar a cabo esta selección se ha definido un componente que tendrá en cuenta todos estos factores. Éste permitirá establecer correspondencias entre los conceptos abstractos de interfaz y sus posibles implementaciones en forma de controles. Además, estos controles podrán incorporar el conjunto de plataformas (ver Sección 5.3) para los que son adecuados, en caso que sea necesario.

La Figura 6.1 muestra el modelo de este catálogo, en el cual se diferencian 4 tipos de entidades: los **controles o widgets** para construir los bocetos, los **objetos abstractos de interfaz** (AIO), los **dispositivos**, y los **patrones** que proponen una solución (en forma de widgets) para cada AIO, aportando diferentes

implementaciones en función del dispositivo. Cada tipo de dispositivo tiene asociada una configuración del modelo de características (ver Capítulo 5) que representará la plataforma. Mediante el uso del concepto de plataforma (ver Sección 5.3) podremos inferir, dada la especificación de un dispositivo a que plataforma pertenece, y por tanto, si existen controles mas adecuado que otros.

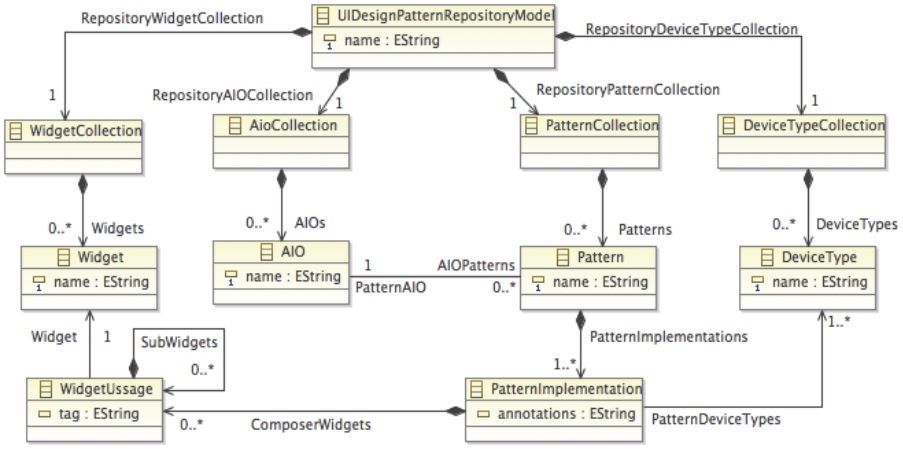


Figura 6.1: Modelo del Catálogo de patrones

Así pues, hemos definido patrón como una estructura de información la cual permite realizar una correspondencia entre un concepto del modelo abstracto de interfaz (especificado en UIM) y los diversos conjuntos de objetos concretos que permiten realizar una implementación concreta de ese concepto abstracto. Cada una de esas implementaciones la expresamos en forma de controles de interfaz, pero como decíamos anteriormente, existen distintos factores que afectan a esta elección. Es por esto que a cada uno de los conjuntos que expresan una implementación concreta, se le puede asignar si se desea un conjunto de plataformas (ver Sección 5.3) para las que es mas adecuada. Por ejemplo, una vista multicolumna como implementación de una navegación en profundidad, puede no ser muy adecuada para dispositivos con pantallas pequeñas.

6.2. Utilización del Catálogo de Patrones

Este catálogo será utilizado en el proceso de transformación (ver Capítulo 7) para ofrecer al diseñador las implementaciones disponibles para los patrones detectados en el modelo abstracto de interfaz, según el tipo de dispositivo.

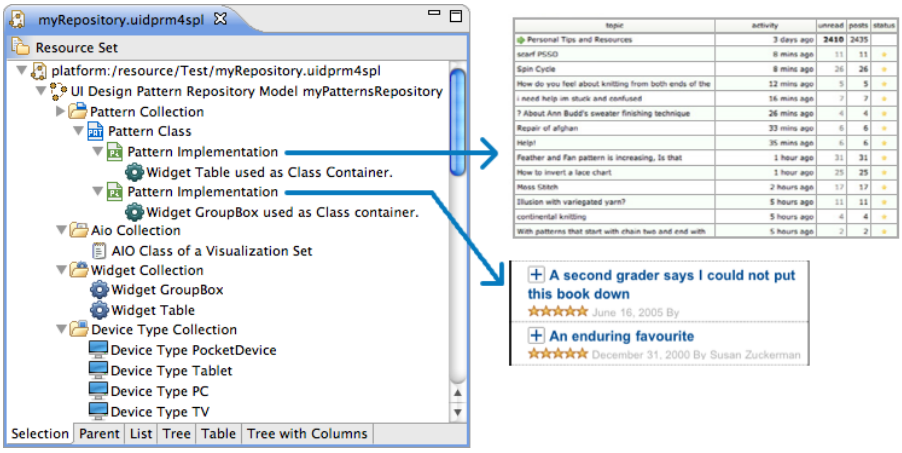


Figura 6.2: Instanciación del Catálogo de patrones. Ejemplos de patrones

Un ejemplo de su utilización lo podemos ver en la Figura 6.2. En ella se presentan dos implementaciones para un mismo concepto abstracto de interfaz (clase visible). Éstos son: una tabla y una lista de registros. Ambas implementaciones están marcadas como válidas para cualquier dispositivo, aún cuando las tablas con muchas columnas pueden no ser adecuadas para dispositivos con pantallas pequeñas. Por ello si el asistente de transformación (ver Sección 7.3.1), detecta que se excede el número de columnas recomendable, advertirá al analista por si desea reconsiderar la selección de la implementación del patrón.

Gracias a la estructura del catálogo, se fomenta la reutilización de sus partes. Como decíamos anteriormente, utilizamos el catálogo para establecer correspondencias entre conceptos abstractos y concretos. Dado que no tenemos información específica de ningún proyecto, el pro-

pio catálogo se convierte en una pieza reutilizable en todos nuestros procesos de desarrollo.

Un mismo concepto abstracto puede tener varias implementaciones igualmente válidas (i.e., para mostrar una fecha podemos utilizar una label o un calendario) y elegir entre ellas en función de nuestras necesidades. Igualmente ocurre a la inversa, un mismo control lo podemos utilizar para representar diferentes conceptos abstractos (i.e., una label como salida de texto o de números).

También es de reseñar que nuestro catálogo es ampliable de 4 maneras: (1) incorporando nuevas definiciones de plataforma para nuevos tipos dispositivos, (2) incorporando nuevos controles que no existían en el momento de la creación de este catálogo, (3)añadiendo nuevos conceptos o patrones abstractos detectados en los modelos, y (4) incorporando nuevas correspondencias entre conceptos abstractos y concretos.

Con la utilización de este catálogo conseguimos plasmar el conocimiento que antes debía poseer el diseñador, en un modelo reutilizable en todos nuestros procesos de desarrollo. Esta es una ventaja frente a la mayoría de propuestas de la literatura, que no ofrecen esta variedad de implementaciones para un mismo concepto abstracto, sino que las correspondencias son uno a uno.

No obstante, esta variabilidad introduce una sobrecarga en el proceso de obtención de bocetos que veremos en el Capítulo 7, ya que debe ser el diseñador el que opte por una de las implementaciones posibles para un mismo concepto.

Para representar este catálogo se ha utilizado un editor EMF (ver Sección 6.3) que está integrado con los modelos de características y de configuraciones utilizados anteriormente para describir las características de interacción de los dispositivos.

6.3. Herramienta de Soporte: M4GeMMINi

MOSKitt 4 GeMMINi es una herramienta basada en MOSKitt 4 SPL (y por ende en MOSKitt) que incorpora y aglutina todas las características presentes en M4SPL y MOSKitt, a las que se añaden los

modelos, transformaciones y generaciones automaticas necesarias para dar soporte a GeMMINi. En este Capítulo se va a presentar UIDPRe, el editor que da soporte al catálogo de patrones de diseño presentado anteriormente.

6.3.1. *UI Design Pattern Repository Editor*

UIDPRe es un editor que forma parte de M4GeMMINi y que permite especificar instancias del catálogo de patrones de diseño de interfaz de usuario. Éste permite utilizar toda la notación para expresar los conceptos necesarios para realizar la asignación de cada tipo de AIO a un conjunto de controles en función de un tipo de dispositivo.

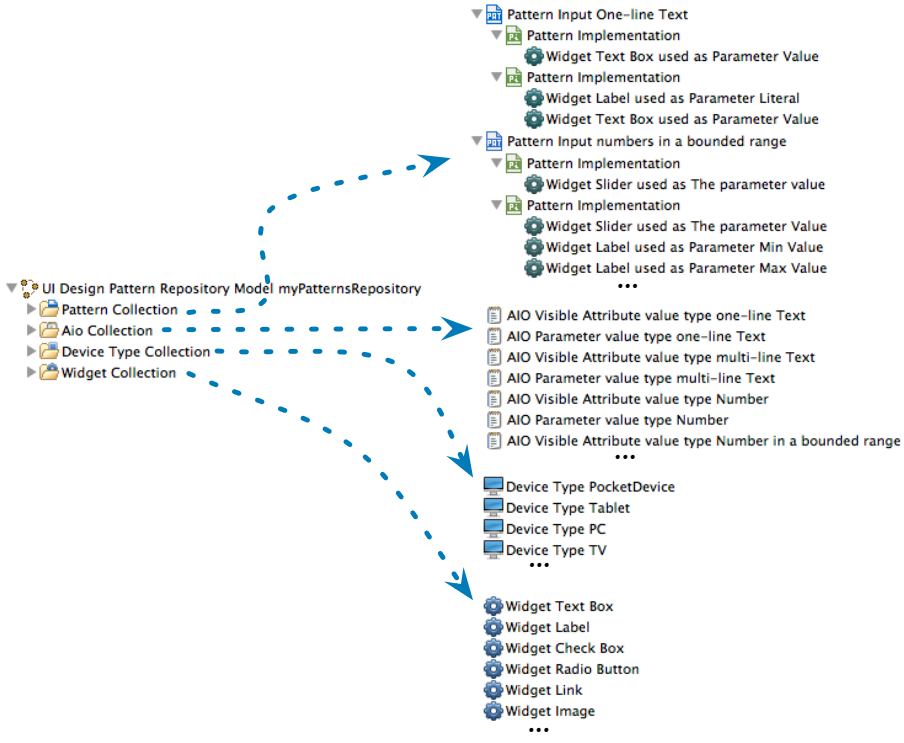


Figura 6.3: UI Design Pattern Repository Editor

En la Figura 6.3 podemos ver un ejemplo de este modelo. Cabe señalar que cada tipo de dispositivo tiene asociado un modelo de configuraciones (ver Sección 5.4) accesible mediante un menú emergente que permite especificar las características activas que delimitan este tipo de dispositivo en función del modelo de características que expresa los modos de interacción de los dispositivos y que hemos visto en el Capítulo 5.

Las especificaciones realizadas mediante este editor se guardarán en un fichero en formato XMI. Este tipo de ficheros utilizan una notación basada en XML fácilmente accesible mediante el código que proporciona EMF.

Tanto las instancias del catálogo como las clases que se han generado y que ayudan a manejarlas serán muy importantes en el proceso de obtención de los bocetos de interfaz (ver Capítulo 7) ya que, como decíamos anteriormente, este catálogo establece las correspondencias entre los conceptos abstractos del modelo de interfaz abstracto (una de las entradas del proceso de transformación) y los conceptos concretos que formarán los bocetos de interfaz.

6.4. Conclusiones

Este capítulo presenta un catálogo de patrones de diseño para interfaces de usuario. Este artefacto es utilizado para establecer correspondencias entre objetos abstractos y concretos de interfaz. Al diseñar interfaces de usuario partiendo de un modelo abstracto, se da que un mismo concepto abstracto puede ser implementado de distintas maneras, pero no todas son válidas para todas las ocasiones.

Este catálago trata de almacenar toda esa variabilidad así como las posibles restricciones que pudiesen darse frente a algunas implementaciones en función de la plataforma destino. También se ha presentado una herramienta en forma de editor que da soporte a la especificación y mantenimiento de este catálogo y que está integrada con el resto de herramientas utilizadas en este trabajo.

Esta información almacenada en el catálogo será muy útil en el pro-

ceso de transformación que veremos en el capítulo siguiente donde se obtendrán los bocetos específicos de plataforma que implementan la interfaz especificada de manera abstracta.

— ❝*You can't do sketches enough. Sketch everything and keep your curiosity fresh.*❞

John Singer Sargent

7

Generación de Bocetos Específicos de Dispositivo

En los capítulos anteriores hemos definido los artefactos que especifican la información a mostrar de manera abstracta e independiente de dispositivo, los dispositivos dónde presentarla y las correspondencias entre la representación abstracta y los objetos concretos que utilizaremos para representar la interfaz de usuario. En el presente capítulo se presenta el proceso de obtención de los bocetos de interfaz de usuario, específicos para cada plataforma.

Cada tecnología empleada en los dispositivos destino, implementa una apariencia distinta de los controles. Como ejemplo podemos ver en la Figura 7.1 dos implementaciones distintas de un mismo concepto, un

Figura 7.1: Campo de fecha en iOS y Android

campo de entrada de una fecha, en formato *mes-día-año*.

Pese a sus diferencias en lo que a apariencia se refiere, su funcionalidad y finalidad es la misma. Es por esto que en la propuesta presentada, se introduce un paso intermedio entre la definición de las interfaces y la generación de su código. Este paso intermedio consiste en la obtención de bocetos de interfaz.

Estos bocetos serán una representación aproximada del resultado que obtendremos tras la generación del código, utilizando concepto generales de las técnicas de creación de bocetos. Con esto, conseguimos abstraer al diseñador, de implementaciones particulares que de estos conceptos haga cada tecnología.

La representación de estos bocetos se realizará utilizando las técnicas que se han expuesto en el Capítulo 3. En concreto se utilizará la notación que provee MOSKitt Sketcher para la creación de bocetos de interfaz.

El resto del Capítulo se estructura como sigue: en la Sección 7.1 se presenta la generación de los bocetos. En la Sección 7.2 se exponen una serie de conceptos previos necesario para en la Sección 7.3 exponer el proceso de transformación que se lleva a cabo. Finalmente la Sección 7.4 concluye el Capítulo.

7.1. Generación Asistida de Bocetos

Modelar interfaces de usuario no es una tarea sencilla dada la naturaleza compleja de la interacción Persona-Ordenador. Existen factores

muy influyentes como los diferentes tipos de interacción, la gran cantidad de conceptos a representar, la especificación del comportamiento de la interfaz, o la falta de estándares que hacen que los lenguajes de especificación existentes sean cada vez menos expresivos y más complejos.

La creación de "bocetos" es una técnica ampliamente aceptada para representar ideas u objetos de una manera preliminar. En la actualidad, una de las técnicas más exitosas para representar interfaces de usuario es utilizar bocetos o "sketches" ya que permite representar las interfaces, describiendo cual será su apariencia final una vez desarrolladas. Esta técnica es más cercana al usuario ya que permite expresar una representación inicial de cómo será la interfaz e incluso validarla con el cliente en etapas tempranas del proceso de desarrollo. Lamentablemente estos bocetos sólo sirven como documentación ya que no se asegura validez ni corrección, ni tampoco están enlazados con modelos conceptuales, por lo que es difícil utilizarlos en entornos MDD.

Por ésto GeMMINi utiliza la notación propuesta por MOSKitt Sketcher para generar los bocetos a partir de una transformación modelo a modelo asistida. Esta transformación toma como entrada un modelo abstracto de interfaz (UIM), una especificación de dispositivo (sobre el modelo de características) y un catálogo de patrones de interfaz.

La transformación analiza el modelo UIM en busca de los constructores abstractos (AIO). Para cada AIO se buscan las implementaciones disponibles en el catálogo que sean adecuadas para el dispositivo destino. En caso que exista más de una implementación válida, se consultará al analista, por medio de un asistente, para que elija una implementación. Este proceso se analiza en profundidad en las secciones siguientes. Como resultado se obtendrá un modelo Sketcher que estará enlazado con el resto de los artefactos que componen la propuesta.

7.2. Conceptos Previos

La utilización de modelos abstractos de interfaz como descriptores de la información es muy útil dado el alto grado de abstracción que otorgan frente a cuestiones tecnológicas o dependientes de presentación. Pero estos modelos abstractos deben traducirse a modelos concretos o al propio código y es entonces cuando surge la necesidad de estructurar la información. Esta estructuración puede traer consigo problemas si no se interpretan correctamente los modelos abstractos.

Uno de los problemas que puede surgir es la duplicidad o la carencia de información. En los modelos abstracto se realiza una estructuración de la información en unidades de interacción, pero estas no tienen una traducción literal en interfaces de usuario. Así pues, puede darse el caso de tener en una misma interfaz dos o mas unidades de interacción, e incluso el caso contrario, que por restricciones de los dispositivos tengamos que dividir la unidad de interacción en dos o mas interfaces distintas.

En el primero de los casos nos podemos encontrar con que dos unidades de interacción que manejan los mismos datos (o muy parecidos) se muestren al mismo tiempo en la misma interfaz conduciendo a una situación en que los mismos datos están visibles dos (o mas) veces. En este caso se debería realizar un procesado de estas unidades de interacción que permita detectar estos elementos duplicados para evitar mostrarlos repetidos.

El segundo de los casos es justo el contrario; la estructuración inicial contemplaba que todos los datos de la unidad de interacción se mostrasen juntos, pero por restricciones ajenas se deben separar. En este caso se debe procurar mantener una trazabilidad que permita que los datos por separado mantengan el sentido.

7.2.1. Estructurando la Información

Con el fin de aportar mayor estructuración a nuestro bocetos y facilitar el proceso de transformación evitando la duplicidad de información, se han definido dos conceptos: (1) el concepto de contenedor y (2) la fusión de clases visibles.

Estos son conceptos muy importantes por estar presentes a lo largo de todo el algoritmo de transformación y se estudian en las siguientes secciones.

El Concepto de Contenedor

Con el fin de ordenar y estructurar la información dentro de las interfaces e incluso las relaciones existentes entre las interfaces, se ha definido el concepto de **Contenedor**. Para definir un contenedor, en primer lugar cabe diferenciar los controles utilizados en 2 tipos: **Atómicos** y **No Atómicos**. Los controles atómicos serán aquellos que no puedan contener controles en su interior (i.e., Label, TextBox, etc.) y los no atómicos el resto (i.e., Tabla, Panel, GroupBox, Ventana, etc.) que serán los contenedores. Así pues, el algoritmo irá marcando que contenedor está activo en cada momento de la transformación dando así la posibilidad de insertar información en éste. Un ejemplo son las relaciones y las navegaciones. Éstas tienen 3 posibles implementaciones: (1) en el Contenedor Activo, (2) en otro Contenedor de la misma ventana (2) (lo que crea un contenedor y lo marca como activo), o (3) en otra ventana (lo que crea una ventana con un contenedor dentro y lo marca como activo).

Fusión de Dos Clases

Es de reseñar también el proceso de **fusión de dos clases visibles**. Este puede darse cuando las unidades de interacción que las contienen se implementan en la misma ventana. Puede darse el caso que los atributos visibles en una de ellas, sean un subconjunto de los visibles en otra. En este caso, la segunda subsumirá a la primera adoptando la subsumiente la implementación de la subsumida o viceversa. En caso de no compartir los atributos, la segunda clase asumiría la implementación de la primera pero se debería consultar la implementación deseada para los atributos de esta segunda.

Ejemplo de fusión de dos clases con subsumción de atributos Supongamos 2 clases visibles A y A' representadas en las unidades de interacción UI_1 y UI_2 respectivamente. ambas son representativas de la misma clase del modelo de datos. Los atributos visibles de A serán el conjunto $\{VAttr_A\}$ y los de A' $\{VAttr_{A'}\}$. Supongamos que: $\{VAttr_{A'}\} \subset \{VAttr_A\}$

Si se decidiese representar UI_1 y UI_2 en la misma ventana del boceto, se podrían fusionar A y A'. el proceso sería el siguiente:

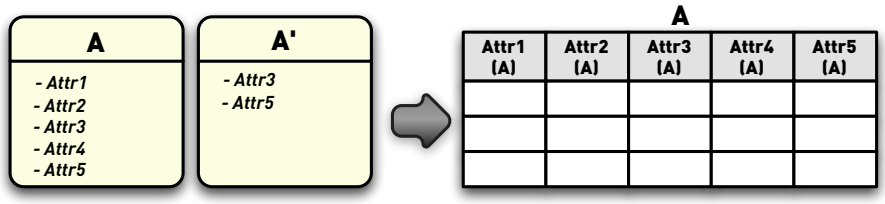


Figura 7.2: Ejemplo de fusión de dos clases con subsunción de atributos implementado como una tabla

UI_1 y UI_2 en la misma ventana. ¿Fusionar Clases? $\rightarrow$ ***SI***
A = A'. ¿Fusionar? $\rightarrow$ ***SI***
$\{VAttr_A\} \not\subset \{VAttr_{A'}\}$
$\{VAttr_{A'}\} \subset \{VAttr_A\}$
A y A' se fusionan.
A subsume a A'
Marcar $\{VAttr_{A'}\}$ *como visitados*
A' asume la implementación de A.

Ejemplo de fusión de dos clases sin subsumción de atributos Supongamos 2 clases visibles B y B' representadas en las unidades de interacción UI_1 y UI_2 respectivamente. ambas son representativas de la misma clase del modelo de datos. Los atributos visibles de B serán el conjunto $\{VAttr_B\}$ y los de B' $\{VAttr_{B'}\}$. Supongamos que: $\{VAttr_{B'}\} \not\subset \{VAttr_B\} y \{VAttr_B\} \not\subset \{VAttr_{B'}\}$

Si se decidiese representar UI_1 y UI_2 en la misma ventana del boceto, se podrían fusionar B y B'. el proceso sería el siguiente:

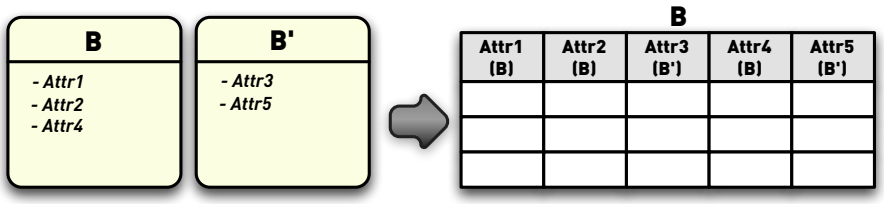


Figura 7.3: Ejemplo de fusión de dos clases sin subsunción de atributos implementado como una tabla

UI_1 y UI_2 en la misma ventana. ¿Fusionar Clases? $\rightarrow$ ***SI***
B = B'. ¿Fusionar? $\rightarrow$ ***SI***
$\{VAttr_B\} \not\subset \{VAttr_{B'}\}$
$\{VAttr_{B'}\} \not\subset \{VAttr_B\}$
B y B' se fusionan.
B' asume la implementación de B.
Consultar los atributos de B'

Otros ejemplos de fusión de dos clases aplicados a un caso de estudio se pueden consultar en el Anexo C.

7.3. Obteniendo los Bocetos de Interfaz

Una vez definidos todos los artefactos y conceptos necesarios para la obtención de los bocetos de interfaz, podemos proceder a realizar esta transformación. La generación de estos bocetos nos permitirá comprobar que la estructuración realizada por el proceso de transformación ha sido la mas adecuada para el dispositivo destino.

7.3.1. El Proceso de Transformación

El proceso de aplicación de este algoritmo es un punto clave en GeMMINi. Este proceso toma como entrada los tres artefactos defini-

dos anteriormente: (1) un modelo UIM, (2) la especificación de los dispositivos, y (3) el catálogo de patrones. El proceso seguido, aprovecha la naturaleza de árbol del modelo UIM, y se recorre éste utilizando un algoritmo DFS[1] (ver Figura 7.4). Éste recorre el árbol en profundidad (y pre-orden) recordando los nodos anteriormente visitados.

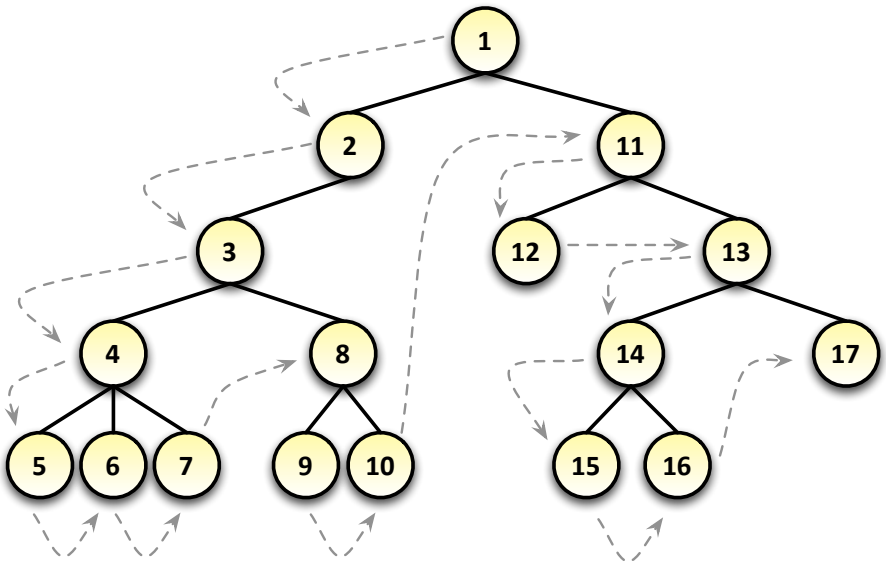


Figura 7.4: Árbol explorado utilizado un algoritmo DFS

Es importante que el algoritmo utilizado recorra el árbol en profundidad, ya que dada la naturaleza de UIM, utilizando este tipo de algoritmos, nos permite acceder a la información por bloques, representando así conjuntos completos de información y pudiendo aplicar conceptos como la fusión de clases que hemos visto con anterioridad.

En cada nodo visitado se consulta si éste es una Unidad de Interacción (UI) de UIM:

- Si lo es, se consulta si en el mismo contenedor ya existe otra UI

[1] Depth-first search

y si es posible aplicar el proceso de ***Fusión de Clases*** (ver Sección 7.2.1).

- Si no lo es, directamente se consulta el catálogo de patrones para obtener las posibles implementaciones para el AIO en la plataforma representativa del dispositivo destino.

Dependiendo del tipo de AIO consultado, la implementación puede consistir en un widget complejo o en uno atómico (ver Sección 7.2.1). En caso de tratarse de un widget complejo (***Contenedor***), se marcará este nuevo contenedor como activo y los nuevos AIO encontrados a partir de ese momento se representarán dentro de éste, hasta encontrar una relación, ya sea entre clases o entre UI.

En el momento de procesar una clase, se podrán tomar 3 decisiones distintas: seguir en el mismo contenedor, crear uno nuevo en la misma ventana o crear una nueva ventana con un contenedor nuevo.

En caso de decidir seguir en el mismo contenedor o en la misma ventana pero en diferente contenedor, se consultará si es posible aplicar la fusión de clases (ver Sección 7.2.1). Si es posible y se decide fusionar las clases, los atributos nuevos pasarán a formar parte del contenedor activo en ese momento. En caso contrario, se implementarán como si de una clase diferente se tratase.

En caso de decidir crear una nueva ventana, se creará una nueva con un contenedor, cuya implementación será la que se decida para la clase que esta siendo procesada.

Una vez implementada la clases, se procederá a implementar los atributos y operaciones de esta sin posibilidad de aplicar la fusión de clases ya que hemos creado una nueva ventana, y esta solo contiene la clase que estamos procesando

Una vez se llegue a las hojas del árbol se aplicará *backtracking* hasta el siguiente nodo no visitado hasta tener todos los nodos del árbol marcados como visitados.

En la Figura 7.5 se muestra el diagrama de flujo representativo del algoritmo de transformación empleado para obtener los bocetos de interfaz partiendo del modelo abstracto de interfaz (UIM), la especificación

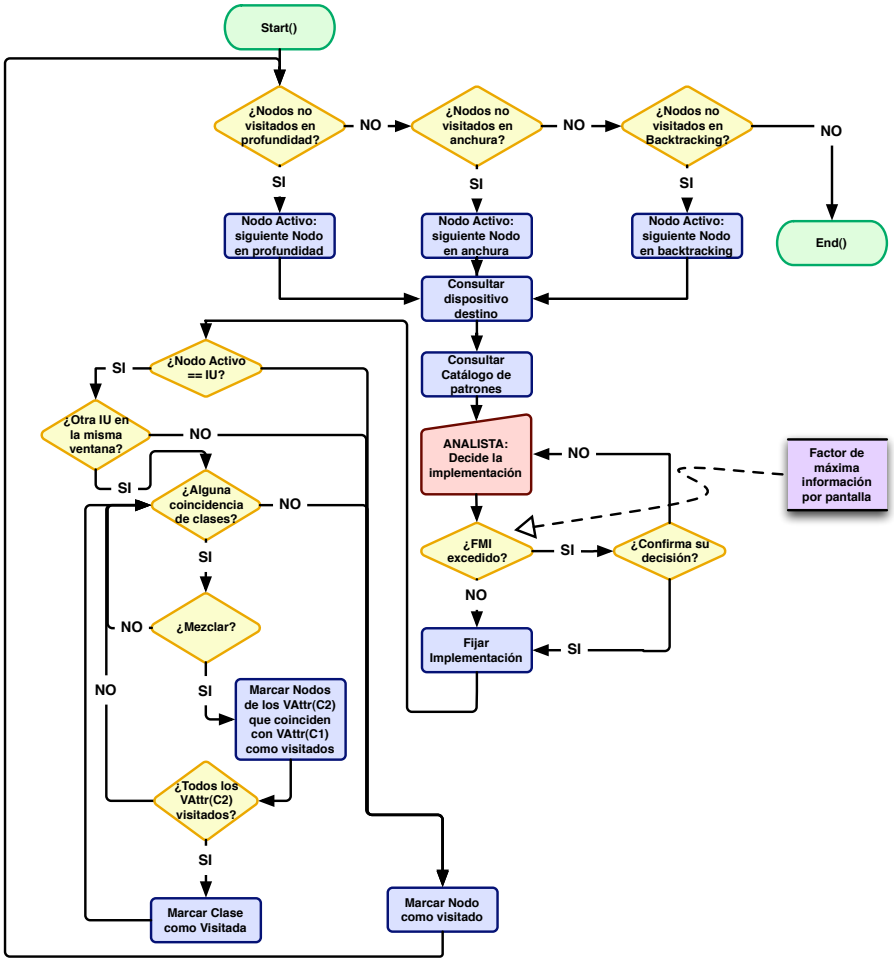


Figura 7.5: Diagrama de flujo representativo del algoritmo de transformación

de los dispositivos y el catálogo de patrones. En el Anexo C se muestra una instanciación completa del algoritmo para el caso de estudio presentado en el Capítulo 9.

7.4. Conclusiones

En este capítulo se ha presentado el algoritmo de transformación mediante el cual se obtienen los bocetos de interfaz específicos de plataforma. Estos bocetos son un paso intermedio entre la definición abstracta de la interfaz y su implementación en código nativo.

Se ha presentado el algoritmo de transformación así como una serie de conceptos previos definidos para dar soporte y facilitar este proceso. También se ha presentado la herramienta que da soporte a este proceso.

En cuanto a los bocetos, hemos utilizado la notación propuesta por MOSKitt Sketcher (Capítulo 3) y que está integrada con el resto de artefactos del proceso.

Se ha aportado también el Anexo C en que se expone de manera secuencial la aplicación de este algoritmo de transformación paso por paso para un escenario del caso de estudio que veremos en el Capítulo 9.

En el siguiente capítulo veremos la generación del código nativo de las interfaces a partir de sus bocetos, loq ue supone el último paso en la aplicación de GeMMINi.

— ❝*Be as you wish to seem.*❞
Socrates

8

Generación de Interfaces Nativas de los Dispositivos

La creación de bocetos es una de las técnicas más utilizadas para concretar ideas en los procesos industriales de construcción de productos. Una idea similar se puede aplicar al desarrollo de software, donde tradicionalmente se ha usado para la definición preliminar de la interfaz de usuario.

Estas técnicas se basan en la idea de exponer una representación inicial, incompleta, normalmente gráfica y representativa de ideas abstractas, con el objetivo de *visualizar* una aproximación a lo que será la solución. Es decir, un boceto no es más que una ***representación aproximada del resultado*** y no el resultado final en sí.

En el ámbito del desarrollo de interfaces, estos bocetos, al ser independientes de cualquier tipo de implementación, nos ofrecen una representación lo suficientemente abstracta, como para ser aplicable a cualquier tecnología destino. Esta abstracción nos facilita el desarrollo de interfaces en múltiples dispositivos, ya que, como decíamos en anteriores capítulos, nos encontramos ante una gran heterogeneidad en los dispositivos, y esto no ocurre sólo a nivel hardware, sino también a nivel software.

En esta heterogeneidad, las empresas buscan oportunidades para diferenciarse de la competencia y es por ello que cada una ha creado una apariencia o *look&feel* que prácticamente la identifica.

Es por ésto que los usuarios están acostumbrados al uso de los controles propuestos por las empresas de desarrollo y su uso no les comporta el esfuerzo extra de aprender cómo se manejan. Es por ello que es importante respetar estos principios ya sentados y propuestos por las guías de desarrollo y apariencia propuestas por las empresas desarrolladoras de dispositivos.

Existen, no obstante, casos en que la empresa desarrolladora del software y el hardware no son la misma. El ejemplo más claro es Android. Por una parte Google, como empressa desarrolladora del sistema operativo, propone un SDK[1] oficial con unos controles "standar". Por otra parte, las empresas desarrolladoras de los dispositivos (HTC, Samsung, etc.) crean una versión distinta en que cambian la apariencia de los controles y pasan a dar soporte a ambas versiones (la original y la propietaria).

En este caso se debe decidir ante las opciones que se presentan: (1) generar el código sobre utilizando los controles de la versión oficial del SDK que provee la empresa desarrolladora del sistema operativo, o (2) generar sobre cada versión de los distintos SDK que proveen las distintas empresas desarrolladoras de los dispositivos.

Sea cual sea la decisión, el objetivo es el mismo: crear una traducción lo mas fiel posible de los bocetos obtenidos en el paso anterior, a las diferentes implementaciones de los controles, o lo que es lo mismo,

[1] *Software Development Kit* - Kit de Desarrollo de Software

obtener productos concretos en las tecnologías destino a partir de un mismo boceto.

El resto de este capítulo se estructura como sigue: en la Sección 8.1 se presentan los principios de las técnicas de obtención de productos a partir de bocetos y en la Sección 8.2 se discute sobre la generación multi-dispositivo frente a la generación en múltiples dispositivos. En la Sección 8.3 se presenta la generación de las interfaces para en la Sección 8.5 exponer el proceso de generación que se lleva a cabo. En la Sección 8.7 se expondrán las herramientas que van a dar soporte a este proceso de generación y finalmente la Sección 8.8 concluye el Capítulo.

8.1. Del Boceto al Producto

El proceso de prototipado de productos no siempre se basa en bocetos que especifiquen la *apariencia*. Existen otras técnicas basadas en crear productos que implementen una funcionalidad casi idéntica a la que tendrá el producto final pero sin atender a cuestiones estéticas (i.e., `Cubelets`[2]).

Pese a ésto, el uso de bocetos como técnica de aproximación entre las especificaciones abstractas y los productos finales es una de las técnicas más utilizadas dado que estos bocetos pueden ser utilizados incluso para validar los requisitos del producto con los usuarios.

La concreción de un boceto en un producto final (una interfaz en nuestro caso) exige una instanciación de los conceptos expresados en el boceto a conceptos utilizados en la plataforma final.

En la Figura 8.1 podemos ver cómo a partir de un mismo boceto, se han obtenido dos productos. El boceto (centro), especifica que una mesa debe tener 4 patas y encima de éstas un tablero. Vemos que uno de los productos derivados del boceto (producto 1, izquierda) tiene las 4 patas de metal y el tablero de conglomerado mientras que el otro (producto 2, derecha) tiene tanto las patas como el tablero de madera.

Estos productos, aunque distintos entre sí, mantienen las similitudes

[2]http://www.modrobotics.com/

Figura 8.1: Dos productos a partir de un mismo boceto

estructurales y/o funcionales y materializan las condiciones especificadas en el boceto con diferentes posibles implementaciones.

Algo parecido ocurre en los procesos de producción de software, aplicados al desarrollo de interfaces de usuario. El boceto es un *modelo* representativo del resultado final de la interfaz que debe asemejarse lo más posible a este, pero manteniendo su naturaleza genérica [33].

Una vez obtenido este boceto, formado por representaciones abstractas de controles como las utilizadas en MOSKitt Sketcher (ver Anexo B), la instanciación que se haga de éste en las distintas plataformas software destino dependerá de la implementación que en dichas plataformas se haya dado para los controles genéricos con que hemos modelado nuestro boceto de interfaz.

8.2. Multi-Dispositivo vs Múltiples Dispositivos

Al enfrentarnos a un proceso de obtención de interfaces para varios dispositivos se plantea una pregunta: *¿Se generará una sola interfaz válida para todos los dispositivos, o se generará una interfaz distinta por cada dispositivo?*. Esta es una cuestión no trivial. En primer lugar recuperemos las definiciones que de ambos términos dábamos en el capítulo de introducción:

Multi-dispositivo: aplicación capaz de adaptarse a múltiples dispositivos

Múltiples Dispositivos: aplicaciones individuales para cada uno de los dispositivos.

Atendiendo a estos conceptos, entendemos pues que una solución **Multi-Dispositivo** es aquella capaz de ejecutarse en todos los dispositivos destino. En cambio una solución para **Múltiples Dispositivos** la entendemos como aquella que, partiendo de la misma especificación se derivan aplicaciones individuales para los distintos dispositivos. En la Tabla 8.1 se puede ver una comparación más amplia de las aproximaciones

En el presente trabajo nos hemos decantado por una solución en Múltiples Dispositivos por dos razones:

1. Entendemos que este tipo de soluciones permiten una mejor adecuación al dispositivo destino en términos de usabilidad y capacidades de interacción.

2. Pese a tratarse de una solución para múltiples dispositivos, esto no quita que podamos adoptar la postura de realizar la generación final sobre un lenguaje común e independiente de plataforma como HTML o XML.

Esta segunda razón brinda una posibilidad que tenemos a nuestro alcance gracias a que GeMMINi realiza la separación entre los conceptos abstractos y concretos. También gracias a que al tratarse de un método basado en modelos, podemos explotar todas las características y técnicas que nos ofrece el Desarrollo Software Dirigido por Modelos.

8.3. Generación Automática de Interfaces Nativas

La generación de interfaces nativas para los dispositivos es un problema ampliamente abordado desde diversas perspectivas. La actual

	Multi-Dispositivo	Múltiples Dispositivos
Adaptación a características concretas	Baja	Alta
Coste incorporar nuevos dispositivos	Bajo	Medio-Alto
Reutilización	Baja	Alta
Versatilidad	Alta	Baja
Adecuación Usuario - Dispositivo	Baja	Alta
Orden del coste inicial de una aplicación para `n` dispositivos	`O(1)`	`O(n)`
Dificultad en el desarrollo	Depende de la tecnología	Depende de la tecnología
Evolución	Sólo se evoluciona una aplicación, pero las novedades deben soportarlas todos los dispositivos	Se evoluciona cada aplicación por separado, pero esto incrementa el coste.

Tabla 8.1: Multi-Dispositivo vs Múltiples Dispositivos

tendencia de crecimiento tanto en el número de dispositivos como de plataformas hace que esta tarea sea cada vez mas compleja.

Dado que en cada plataforma destino, los controles se han implementado de maneras distintas en función de la apariencia que se desea dar a las interfaces (*look&feel*), el resultado que se obtenga de la generación de interfaces a partir de bocetos, debería diferir únicamente en la estética pero no en la funcionalidad siempre que se trate de dispositivos

parecidos o pertenecientes a la misma plataforma (ver Sección 5.3).

El proceso de generación propuesto en GeMMINi se aprovecha de la naturaleza XMI (XML) de la notación utilizada en MOSKitt Sketcher lo que facilita la generación de interfaces en las plataformas mas novedosas que están apareciendo en los últimos tiempos (Android, iOS, RIM, etc.), las cuales, en su mayoría, están también basadas en lenguajes de marcado.

En el contexto de esta tesis de máster se han desarrollado dos generadores para dos tecnologías distintas: (1) Sktecher→Android y (2) Sketcher→iOS. En la Sección 8.5 se muestra con mas detalle el proceso que se sigue para la generación de estas interfaces en los distintos dispositivos.

8.4. El Objetivo de la Transformación

En los métodos basados en el desarrollo dirigido por modelos con generación de código, **el objetivo final es producir distintos tipos de ficheros de texto** que son procesados por las distintas herramientas de implementación como los compiladores [16].

Existen otros casos, como la refactorización de modelos, podría resultar suficiente con la obteción de otro modelo, pero en el caso que ocupa esta tesis, esto no es suficiente. Además, esta implementación debe cumplir con las reglas de formato/sintaxis que las diferentes tecnologías finales requieran.

El desarrollo de interfaces plenamente operativas implica que no sólo debe generarse la estructura, sino también parte de la funcionalidad. Esto significa que los modelos del método deben contener la suficiente información como para hacer que esto sea posible. Sin embargo, estos requisitos aumentan la complejidad de las reglas de transformación.

8.4.1. Los Resultados Esperados de la Transformación

El objetivo final de nuestra transformación es el de obtener los ficheros suficientes como para expresar las interfaces de usuario en las distintas tecnologías destino. como decíamos en el capítulo anterior, este código generado debe hacer uso de los SDK oficiales de los distintos sistemas operativos.

Esto implica que en ocasiones no será suficiente un fichero que especifique la interfaz. Por ejemplo, Android basa sus interfaces en un conjunto formado por tres tipos de ficheros: (1) la propia especificación de la interfaz, (2) el fichero `R` de recursos y (3) el código java necesario. Con esto se consigue una separación que permite que la misma aplicación en dispositivos distintos, puedan utilizar distintos recursos. (ver Figura 8.2)

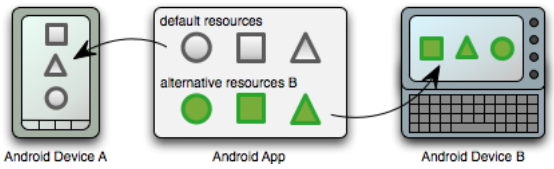


Figura 8.2: Dos dispositivos distintos, cada uno con sus recursos

Los Recursos

La externalización de recursos, como imágenes, cadenas de texto o secuencias de código de la aplicación, permite mantenerlos de forma independiente y también proporcionar recursos alternativos dependiendo de las distintas configuraciones que admiten los dispositivos específicos, tales como idiomas o la orientación de pantalla. Con el fin de proporcionar compatibilidad con diferentes configuraciones, se deben organizar los recursos en una estructura de ficheros dentro del proyecto, agrupándolos según tipo y configuración

Por ejemplo, puede que la orientación por defecto de nuestra interfaz sea vertical. Esta especificación, la guardaremos en el directorio especí-

fico `/res/layout`. Pero si queremos incluir un orientación horizontal alternativa, esta la guardaremos en el directorio `/res/layout-land` y Android automáticamente aplicará la configuración correspondiente en función de la configuración actual del dispositivo.

La interfaz

En Android, las unidades básicas de expresión de interfaces de usuario son las Vistas (Views). Estas vistas contienen widgets que ofrecen CIOs completamente funcionales.

Estas vistas sirven como base en la implementación de la interfaz y están compuestas por un `layout` que estructura los distintos controles en la pantalla. Existen distintos tipos de layouts:

- **Linear Layout:** Implementa una vista con los elementos apilados uno encima del otro.
- **Table Layout:** Implementa una vista en forma de lista o tabla.
- **Grid View:** Implementa una vista en parrilla. Muy útil para imágenes.
- **Tab View:** Implementa un a vista por pestañas.

En este trabajo los layout utilizados han sido los `Absolute Layout` porque son los que mas se ajustan a trabajo realizado con MOSKitt Sketcher, ya que se pueden fijar las coordenadas x e y de los controles. Un ejemplo de `Absolute Layout` se puede ver en el Fragmento de Código 8.1.

Fragmento de Código 8.1: Ejemplo de Absolute Layout en Android (código)

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <AbsoluteLayout
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     xmlns:android="http://schemas.android.com/apk/res
          /android"
```

```
 6     >
 7     <Button
 8         android:layout_width="188px"
 9         android:layout_height="wrap_content"
10         android:text="Button"
11         android:layout_x="126px"
12         android:layout_y="361px"
13         />
14     <Button
15         android:layout_width="113px"
16         android:layout_height="wrap_content"
17         android:text="Button"
18         android:layout_x="12px"
19         android:layout_y="361px"
20         />
21</AbsoluteLayout>
```

El código anterior daría como resultado la interfaz que podemos ver en la Figura 8.3

Figura 8.3: Ejemplo de Absolute Layout en Android (interfaz)

El Código

Por último, se debe vincular a la actividad de la interfaz el contenido de esta. Una actividad en Android es una clase Java que especifica una de las operaciones que puede realizar el usuario y que por tanto requieren de una interfaz y en su caso, de un fichero de recursos.

En el Fragmento de Código 8.2 podemos ver el método de la actividad `onCreate()` donde se vincula el layout a la actividad.

Fragmento de Código 8.2: Código de una actividad

```
1 public void onCreate(Bundle savedInstanceState) {
2     super.onCreate(savedInstanceState);
3     setContentView(R.layout.main);
4 }
```

8.5. El Proceso de Generación

Las tendencias actuales en muchas tecnologías se decantan por separar la especificación interfaz de la lógica de la aplicación, tendiendo así a acercarse a paradigmas como el modelo-vista-controlador. iOS o Android son claros ejemplos de aplicación de este paradigma.

En muchos casos se tiende a especificar la interfaz utilizando un lenguaje de marcado (típicamente XML) identificando cada componente de la interfaz para su posterior manejo en el controlador. Así, en un XML se define la estructura de la interfaz, y en código se pueblan los controles que lo requieran, y se manejan los posibles eventos que estos lancen. Puede ocurrir que sean necesario además de estos, algunos ficheros de recursos como cadenas de texto, iconos, etc.

Esta separación se adecua perfectamente a la propuesta que presentamos, ya que nuestro propósito es el de generar la estructura de las interfaces y en caso que sea necesario, el código que pueble algunos controles (i.e., un ComboBox)

La estructura de las especificaciones en MOSKitt Sketcher es un

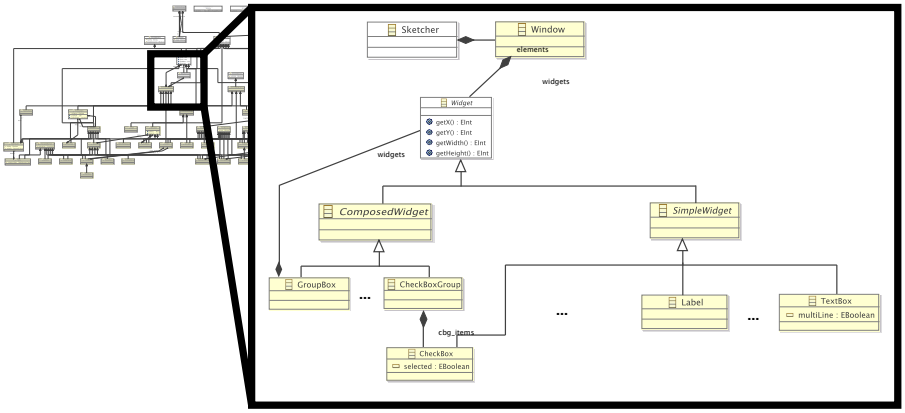


Figura 8.4: Fragmento del metamodelo de Sketcher

árbol en que la raíz es el propio modelo y los nodos inmediatamente siguientes son especificaciones de ventanas. Posteriormente, cada ventana tiene en su interior los nodos correspondientes a los controles. En caso de ser controles complejos (ver Sección 7.2.1), éstos a su vez contendrán los controles que lo forman, como se puede ver en la Figura 8.4.

Para la generación de código se ha optado por utilizar la tecnología XPand2 propuesta por openArchitectureWare[3]. XPand2 es un motor de plantillas que permite la generación a partir de modelos EMF. La salida de la generación puede ser un lenguaje de programación o cualquier otra cosa. Xpand requiere que se defina su metamodelo EMF y una o más plantillas que especifiquen la traducción entre el modelo y el resultado. Una vez especificada esta definición, se puede ejecutar el generador de código mediante la definición de un modelo de EMF, obteniendo como resultado la traducción tal y como se ha especificado en la plantilla.

Para llevar a cabo el proceso de generación nos hemos aprovechado de la estructura de árbol que nos proporciona MOSKitt Sketcher y generaremos cada ventana por separado.

Como decíamos anteriormente, en el contexto de este trabajo de

[3]http://www.openarchitectureware.org

tesis se han desarrollado dos generadores para dos tecnologías distintas: iOS y Android. Ambas implementan sus interfaces mediante ficheros XML apoyados por código (Objective-C y Java respectivamente) que las puebla de datos y maneja sus eventos. Así pues, las estructuras de las plantillas serán muy parecidas.

Fragmento de Código 8.3: Estructura de las plantillas de generación

```
1  «IMPORT sketcher»
2  «IMPORT es::cv::gvcase::mdt::sketcher»
3  «DEFINE genXml FOR Sketcher»
4    «FILE "res/drawable/fondonegro.xml"-»
5      <shape xmlns:android="http://schemas.android.com/
           apk/res/android" type="OvalShape" >
6            <solid android:color="#FF000000"/>
7      </shape>
8    «ENDFILE»
9    «EXPAND genXml FOREACH elements»
10 «ENDDEFINE»
11
12 «DEFINE genXml FOR Window»
13   «FILE "res/layout/"+id.toLowerCase()+".xml"  -»
14     <?xml version="1.0" encoding="utf-8"?>
15       <AbsoluteLayout
16         xmlns:android="http://schemas.android.com/apk
             /res/android"
17         android:orientation="vertical"
18         android:layout_width="«getWidth()»px"
19         android:layout_height="«getHeigth()»px">
20         «FOREACH widgets AS e»
21           «EXPAND genXml (id) FOR e»
22         «ENDFOREACH»
23       </AbsoluteLayout>
24   «ENDFILE»
25 «ENDDEFINE»
26
27 «DEFINE genXml (String parent) FOR Button»
28 //CODE FOR BUTTON
29 «ENDDEFINE»
30
```

```
«DEFINE genXml (String parent) FOR CheckBox»
 //CODE FOR CHECK BOX
«ENDDEFINE»

«DEFINE genXml (String parent) FOR RadioButton»
 //CODE FOR RADIO BUTTON
«ENDDEFINE»

«DEFINE genXml (String parent) FOR CheckBoxGroup»
 //CODE FOR CHECK BOX GROUP
  «FOREACH checkBoxes AS e ITERATOR it»
    //CODE FOR CHECK BOX GROUP ITEM
  «ENDFOREACH»
«ENDDEFINE»

...
```

El fragmento de código 8.3 muestra la estructura general de las plantillas de generación en XPand. Vemos que se genera un fichero por cada ventana, y por cada control (element) que contiene en su interior, se invoca a la plantilla de generación correspondiente.

Fragmento de Código 8.4: Generación de una label en iOS

```
«DEFINE wid FOR Label»
          <object class="IBUILabel" id="«id»">
            <reference key="NSNextResponder" ref="
               Window1"/>
            <int key="NSvFlags">1316</int>
            <string key="NSFrame">{{«getX()», «getY()
               »}, {«IF getWidth()==-1» «42» «ELSE»
               «getWidth()» «ENDIF», «IF getHeight()
               ==-1» «21» «ELSE» «getHeight()» «ENDIF
               »}}</string>
            <reference key="NSSuperview" ref="Window1
               "/>
            <bool key="IBUIOpaque">NO</bool>
            <bool key="IBUIClipsSubviews">YES</bool>
            <int key="IBUIContentMode">7</int>
```

```
10              <bool key="IBUIUserInteractionEnabled">NO
                    </bool>
11              <string key="targetRuntimeIdentifier">
                    IBCocoaTouchFramework</string>
12              <string key="IBUIText">«text»</string>
13              <object class="NSColor" key="
                    IBUITextColor" id="«id»text">
14                <int key="NSColorSpace">1</int>
15                <bytes key="NSRGB">MCAwIDAAA</bytes>
16              </object>
17              <object class="NSColor" key="
                    IBUIHighlightedColor" id="«id»high">
18                <int key="NSColorSpace">3</int>
19                <bytes key="NSWhite">MQA</bytes>
20              </object>
21              <int key="IBUIBaselineAdjustment">1</int>
22              <float key="IBUIMinimumFontSize">10</
                    float>
23            </object>
24 «ENDDEFINE»
```

En el fragmento de código 8.4 se muestra la plantilla de generación de una label en iOS. Vemos que se fijan las coordenadas (`getX()` y `getY()`) para que la representación de las interfaces sea lo mas parecida posible al boceto (en lo que a estructura y presentación se refiere).

Pero como decíamos anteriormente, algunos controles requieren de ficheros de soporte que los pueblen de datos. Este es el caso, por ejemplo, de un combobox. En el fragmento de código 8.5 vemos que se genera tanto el fragmento de código XML que representa en sí al *combobox*, como el fragmento de código java que le incorpora los datos (*combobox items*) correspondientes y el fragmento que especifica los items en el fichero de recursos.

Fragmento de Código 8.5: Generación de un Combobox en Android

```
1 //INTERFACE XML
2 «DEFINE genXml (String parent)  FOR ComboBox»
3 <Spinner
```

```
4   android:id="@+id/«parent»_«id»"
5   android:layout_width=«IF getWidth()>0»"«getWidth()
        »px"«ELSE»"100px"«ENDIF»
6   android:layout_height=«IF getHeight()>0»"«getHeight
        ()»px"«ELSE»"21px"«ENDIF»
7   android:layout_x="«getX()»px"
8   android:layout_y="«getY()»px"></Spinner>
9 «ENDDEFINE»
10
11 //RESOURCES XML
12 «DEFINE genStrings (String parent) FOR ComboBox»
13   <string-array name="«parent»_«id»_array">
14   «FOREACH items AS e»  <item>«e.sampleText»</item>
15   «ENDFOREACH»</string-array>
16 «ENDDEFINE»
17
18 //JAVA Code
19 «DEFINE genInit (String parent) FOR ComboBox»
20         Spinner spinner = (Spinner) findViewById(R.id
               .«parent»_«id»);
21         ArrayAdapter<CharSequence> adapter =
               ArrayAdapter.createFromResource(
22           this, R.array.«parent»_«id»_array, android.
                 R.layout.simple_spinner_item);
23         adapter.setDropDownViewResource(android.R.
               layout.simple_spinner_dropdown_item);
24         spinner.setAdapter(adapter);
25 «ENDDEFINE»
```

8.6. Resultados de la Generación

El siguiente ejemplo muestra los resultados de la generación de código para un sketch de una interfaz para el Android HTC Magic. En primer lugar podemos ver en la Figura 8.5

Vemos que se trata de una interfaz para introducir una serie de datos personales de una persona. Esta compuesta por cuatro `textfield` con sus respectivas etiquetas para introducir el nombre, telefono, dirección y

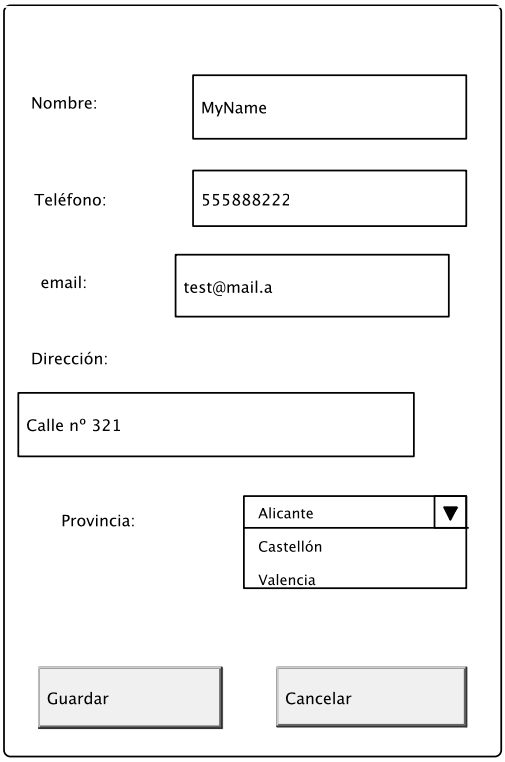


Figura 8.5: Ejemplo de transformación

correo electrónico. También tenemos una lista de selección única implementada con un `combobox` para introducir la provincia de residencia del usuario. finalmente tenemos dos botones para guardar y cancelar respectivamente.

Para implementar esta interfaz se han generado tres ficheros que pasamos a mostrar a continuación. En primer lugar mostramos el XML que implementa la interfaz en el Fragmento de Código 8.6

Fragmento de Código 8.6: Ejemplo de generación. XML de la interfaz

```
<?xml version="1.0" encoding="utf-8"?>

//LAYOUT
<AbsoluteLayout
  xmlns:android="http://schemas.android.com/apk/res/
      android"
  android:orientation="vertical"
  android:layout_width="320px"
  android:layout_height="480px">

  //NOMBRE
  <TextView
    android:id="@+id/EditWindow_nameLabel"
    android:text="Nombre:"
    android:layout_width="82px"
    android:layout_height="34px"
    android:layout_x="17px"
    android:layout_y="42px">
 </TextView>
    <EditText
    android:id="@+id/EditWindow_nameTextBox"
    android:text="MyName"
    android:layout_width="173px"
    android:layout_height="41px"
    android:layout_x="119px"
    android:layout_y="42px"
    android:enabled="true">
 </EditText>

  //TELEFONO
  <TextView
    android:id="@+id/EditWindow_labelTfno"
    android:text="Telefono:"
    android:layout_width="69px"
    android:layout_height="36px"
    android:layout_x="19px"
    android:layout_y="102px">
 </TextView>
  <EditText
```

```
39      android:id="@+id/EditWindow_tfnoTextBox"
40      android:text="555888222"
41      android:layout_width="173px"
42      android:layout_height="36px"
43      android:layout_x="119px"
44      android:layout_y="102px"
45      android:enabled="true">
46  </EditText>
47
48   //DIRECCION
49   <TextView
50      android:id="@+id/EditWindow_Label2"
51      android:text="Direccion:"
52      android:layout_width="82px"
53      android:layout_height="25px"
54      android:layout_x="17px"
55      android:layout_y="208px">
56  </TextView>
57   <EditText
58      android:id="@+id/EditWindow_directionTextBox"
59      android:text="Calle n 321"
60      android:layout_width="250px"
61      android:layout_height="41px"
62      android:layout_x="9px"
63      android:layout_y="242px"
64      android:enabled="true">
65  </EditText>
66
67   //PROVINCIA
68   <TextView
69      android:id="@+id/EditWindow_provinciaComboBox"
70      android:text="Provincia:"
71      android:layout_width="85px"
72      android:layout_height="31px"
73      android:layout_x="36px"
74      android:layout_y="307px">
75  </TextView>
76   <Spinner
77      android:id="@+id/EditWindow_provinciaComboBox"
78      android:layout_width="141px"
```

```
79      android:layout_height="59px"
80      android:layout_x="151px"
81      android:layout_y="307px">
82  </Spinner>
83
84   //MAIL
85   <TextView
86      android:id="@+id/EditWindow_Label3"
87      android:text="email:"
88      android:layout_width="54px"
89      android:layout_height="35px"
90      android:layout_x="34px"
91      android:layout_y="157px">
92  </TextView>
93   <EditText
94      android:id="@+id/EditWindow_mailTextBox"
95      android:text="test@mail.a"
96      android:layout_width="173px"
97      android:layout_height="40px"
98      android:layout_x="119px"
99      android:layout_y="157px"
100     android:enabled="true">
101 </EditText>
102
103  //BOTONES
104  <Button
105     android:id="@+id/EditWindow_saveButton"
106     android:layout_width="116px"
107     android:layout_height="39px"
108       android:text="Guardar"
109       android:layout_y="415px"
110       android:layout_x="22px"
111     android:enabled="true">
112 </Button>
113  <Button
114     android:id="@+id/EditWindow_cancelButton"
115     android:layout_width="120px"
116     android:layout_height="38px"
117       android:text="Cancelar"
118       android:layout_y="415px"
```

```
119       android:layout_x="172px"
120     android:enabled="true">
121   </Button>
122</AbsoluteLayout>
```

A continuación en el Fragmento de Código 8.7 se muestra el fichero de recursos de la interfaz. Como vemos, contiene las cadenas que pueblan el `combobox`. Esta separación permite que por ejemplo, añadir mas provincias, supondría añadir items a la lista.

Fragmento de Código 8.7: Ejemplo de generación. Fichero de recursos

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <string-array name="
       EditWindow_provinciaComboBox_array">
4     <item>Alicante</item>
5     <item>Castellon</item>
6     <item>Valencia</item>
7     <item>Otra</item>
8   </string-array>
9 </resources>
```

Finalmente, el Fragmento de Código 8.8 muestra la clase Actividad que da soporte a la interfaz.

Fragmento de Código 8.8: Ejemplo de generación. Clase *Actividad*

```
1 package com.testsketcher2android;
2 import android.app.Activity;
3 import android.os.Bundle;
4 import android.view.Window;
5 import android.view.WindowManager;
6 //combo box
7 import android.widget.Spinner;
8 //Button
9 import android.widget.Button;
10 //ListBox y ComboBox
11 import android.widget.ArrayAdapter;
```

```
12
13
14 import android.content.Intent;
15 import android.view.View;
16
17 public class EditWindow extends Activity {
18
19 @Override
20 public void onCreate(Bundle savedInstanceState) {
21   super.onCreate(savedInstanceState);
22     requestWindowFeature(Window.FEATURE_NO_TITLE);
23     getWindow().setFlags(WindowManager.LayoutParams.
          FLAG_FULLSCREEN,
24                                  WindowManager.
                                       LayoutParams.
                                       FLAG_FULLSCREEN);
25     setContentView(R.layout.editwindow);
26     init();
27     }
28
29 public void init() {
30   Button boton_EditWindow_saveButton = (Button)
       findViewById(R.id.EditWindow_saveButton);
31         boton_EditWindow_saveButton.
              setOnClickListener(new View.
              OnClickListener() {
32              public void onClick(View view) {
33                   Intent myIntent = new Intent(view.
                        getContext(), ViewWindow.class);
34                   startActivityForResult(myIntent, 0);
35              }
36
37     });
38   Button boton_EditWindow_cancelButton = (Button)
       findViewById(R.id.EditWindow_cancelButton);
39         boton_EditWindow_cancelButton.
              setOnClickListener(new View.
              OnClickListener() {
40              public void onClick(View view) {
41                   Intent myIntent = new Intent(view.
```

```
                    getContext(), ViewWindow.class);
42                startActivityForResult(myIntent, 0);
43            }
44
45    });
46   Spinner spinner = (Spinner) findViewById(R.id.
        EditWindow_provinciaComboBox);
47        ArrayAdapter<CharSequence> adapter =
             ArrayAdapter.createFromResource(
48          this, R.array.
              EditWindow_provinciaComboBox_array,
              android.R.layout.simple_spinner_item);
49        adapter.setDropDownViewResource(android.R.
             layout.simple_spinner_dropdown_item);
50        spinner.setAdapter(adapter);
51
52    }
53 }
```

La Figura 8.6 muestra la comparación entre el boceto que deseábamos generar y la interfaz obtenida finalmente. Podemos ver que el parecido es notable, pese a que hay algunas posiciones de los controles que no se corresponden. Esto es debido a las diferentes implementaciones que de los widgets abstractos hace el SDK oficial de Android que provee Google.

Atendiendo a los **resultados** expuestos, podemos afirmar que en el caso de android somos capaces de generar el mas del **90 % del código necesario** para obtener las interfaces (la lógica de la aplicación queda fuera del ámbito de este trabajo).

Esto lo consideramos un éxito teniendo en cuenta que el 10 % restante son ficheros de infraestructura (imágenes, iconos, librerías específicas, etc.) o componentes con una alta frecuencia de cambio entre versiones y que por tanto hemos decidido dejar al margen

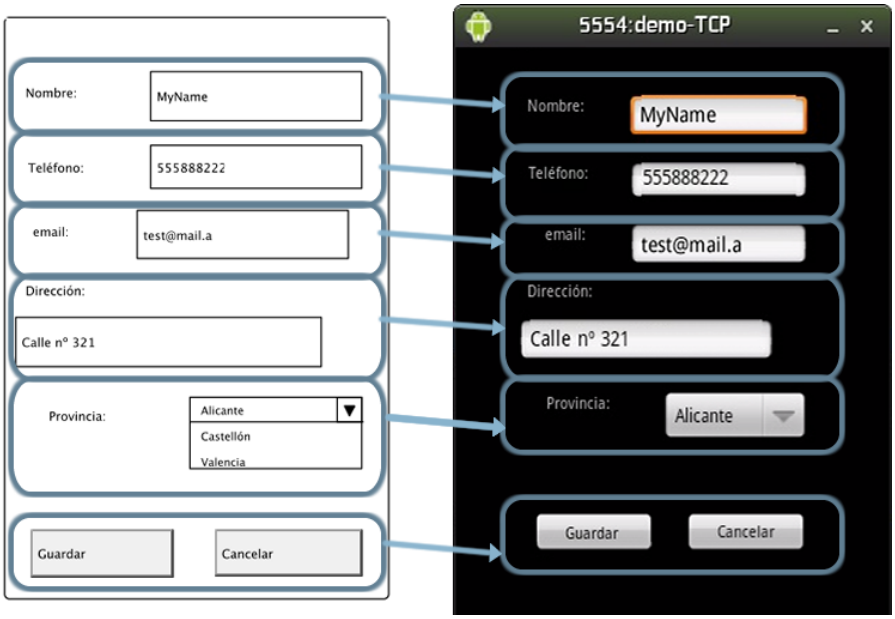


Figura 8.6: Ejemplo de transformación. Comparación entre boceto y resultado

8.7. Herramienta de Soporte: M4GeMMINi

Aprovechando los puntos de extensión que nos brinda MOSKitt y sobre los que hemos construido la herramienta MOSKitt 4 GeMMINi, se ha incorporado la generación de código a los distintos dispositivos. Esta generación toma como entrada un modelo MOSKitt Sketcher y genera la interfaz sobre una plataforma destino preestablecida (se deben crear tantos transformadores como tecnologías destino necesitemos abarcar).

Este trabajo que podría parecer complejo, se torna relativamente sencillo ya que, como hemos visto en la Sección 8.5, para el proceso de generación hemos utilizado la tecnología XPand2 (Modelling Workflow Engine). Esta tecnología esta basada en el uso de plantillas de transformación, las cuales toman como entrada un modelo y para cada primitiva se genera el fragmento de código correspondiente.

Como decíamos anteriormente, en el contexto de esta tesis de máster, se han implementado dos generadores de código (Android e iOS). Hemos observado que las plantillas de generación desarrolladas en ambos casos siguen una estructura casi idéntica. Es por esto que, observando las tendencias actuales para la especificación de interfaces en las tecnologías emergentes, podemos intuir que estas semejanzas se darán en mas casos.

Precisamente para estos casos se ha pensado en un trabajo futuro (que queda fuera del ámbito de esta tesina) consistente en crear una infraestructura de generación que facilite la creación de estos generadores y así poder llegar a mas tecnologías destino.

De ésto ser así, la dificultad en la creación del generador radicaría en poseer un conocimiento suficiente de la tecnología que nos permitiese conocer los fragmentos de código necesarios para implementar los controles especificados en nuestro modelo Sketcher.

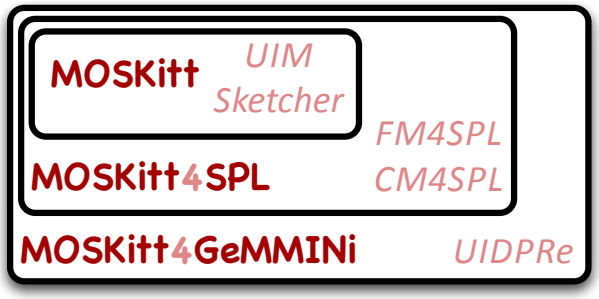


Figura 8.7: Estructura de las herramentas utilizadas en este trabajo

En la Figura 8.7 se puede ver la estructura de las herramientas utilizadas/desarrolladas en el contexto de esta tesis. Como podemos ver, se encuentran todas dentro de un mismo entorno, al tiempo que todos parten de una misma raíz (MOSKitt) lo que nos ofrece grandes beneficios y posibilidades de ampliación dada su arquitectura basada en plugins y en puntos y mecanismos de extensión.

8.8. Conclusiones

El presente capítulo ha presentado el último paso en la aplicación de GeMMINi: la generación de código nativo de las interfaces.

Se han analizado distintas propuestas de generación y se ha planteado la que se lleva a cabo en la propuesta exponiendo el proceso de transformación, y algunos fragmentos de las plantillas de generación desarrolladas para los dos transformadores realizados en el contexto de esta tesis: (1) Sktecher→Android y (2) Sketcher→iOS.

Se ha expuesto también la herramienta que da soporte al proceso de generación (MOSKitt 4 GeMMINi) así como la tecnología utilizada para llevara a cabo (XPand2).

Hemos visto que las plantillas de generación serán muy parecidas independientemente de la tecnología utilizada como destino. Por esto se plantea un trabajo futuro en esta línea que no es otro que el de implementar estas generaciones utilizando principios de líneas de producto con el fin de fomentar aún mas si cabe la reutilización de componentes y facilitando el trabajo de los desarrolladores.

Una vez presentados todos los componentes y transformaciones que forman parte de GeMMINi, en el siguiente capítulo se expone su aplicación a un caso práctico.

— ❝*Setting an example is not the main means of influencing another, it is the only means.*❞

Albert Einstein.

9

Caso de Estudio

Este capítulo describe la aplicación de nuestra propuesta a un caso práctico. Poniendo en práctica nuestra propuesta, queremos ilustrar como GeMMINi puede ser aplicado en un escenario conocido como es un Sistema de Revisión de Conferencias.

Basándonos en este caso de estudio, validamos si el método propuesto en esta tesis puede hacer frente a las necesidades de un proceso de desarrollo de una aplicación para múltiples dispositivos.

Con la aproximación seguida para validar la propuesta, en primer lugar queremos verificar que el modelado abstracto es adecuado para describir interfaces de usuario. En este caso, expondremos 2 escenarios basados en el SRC para los que modelaremos los requisitos de interfaz de manera abstracta.

Por otra parte necesitamos validar si obtenemos suficiente expresividad utilizando el catálogo del patrones de diseño, para representar el conocimiento necesario para realizar las correspondencias entre los Objetos Abstractos de Interfaz (AIOs) y los Objetos Concretos de Interfaz (CIOs) para obtener los bocetos. Para ello se ha elaborado un catálogo completo que establece las correspondencias entre los AIOs detectados en MOSKitt UIM y los CIOs disponibles en MOSKitt Sketcher.

Por último evaluaremos si nuestro modelo de características que utilizamos para definir nuestros dispositivos tiene la granularidad suficiente como para describir los tipos de dispositivos necesarios de manera que los conjuntos de características seleccionados sean una distribución total y disjunta.

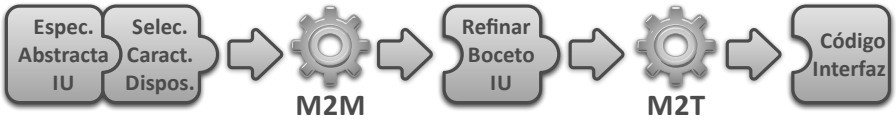


Figura 9.1: GeMMINi. Fases del Método.

En la Figura 9.1 se puede ver el método especificado por GeMMINi en el que se diferencian las 3 fases que lo componen. Éstas son las que se van a aplicar en los siguientes apartados al Sistema de Revisión de Conferencias.

El resto del capítulo se estructura como sigue: En la Sección 9.1 se exponen dos escenarios del caso de estudio del Sistema de Revisión de Conferencias y la Sección 9.2 concluye el capítulo.

9.1. Sistema de Revisión de Conferencias

Para clarificar la propuesta, en este artículo se aplicará GeMMINi al desarrollo de un Sistema de Revisión de Conferencias (en adelante SRC). Este es el tipo de sistemas utilizados para dar soporte al proceso de envío, revisión y notificación de resultados para las contribuciones de investigación en conferencias.

Del sistema completo se van a exponer 2 escenarios. Por una parte se van a modelar y generar las interfaces que dan soporte a los autores para que accedan a consultar los resultados que han obtenido sus contribuciones tras el proceso de revisión.Por otra parte, se van a exponer las interfaces que permitan a los revisores acceder al sistema para introducir las revisiones a las contribuciones que les han sido asignadas.

Como dispositivos destino se van a establecer un iPad (Apple iOS) y un HTC Magic (Android).

9.1.1. Modelado Abstracto de la Interfaz de Usuario

El modelado abstracto de la interfaz de usuario está enfocada a la interacción definida en términos de presentación de la información entendida como conjunto de elementos de la interfaz de usuario. Este es un modelo genérico independiente de cualquier dispositivo o contexto de uso.

El modelo abstracto representa una expresión canónica de los conceptos del dominio independiente de la plataforma de computación. En nuestro caso, como decíamos en la Sección 3.2.2, hemos utilizado UIM para especificar este modelo abstracto de interfaz.

En lo referente a nuestro caso de estudio en la Figura 9.2 se puede ver el **modelo completo** de vistas en UIM para todo el sistema. En concreto nos vamos a centrar en los dos escenarios que nos ocupan: ***ReviewsQueryView*** y ***ReviewsManagementView*** (entre líneas punteadas). Vemos que todos los actores a excepción del anónimo deben estar registrados y haber ingresado en el sistema para acceder a las vistas específicas de cada uno. Por ello en este caso vamos a dar por supuesto el proceso de registro en el sistema ya que será común a todo ellos.

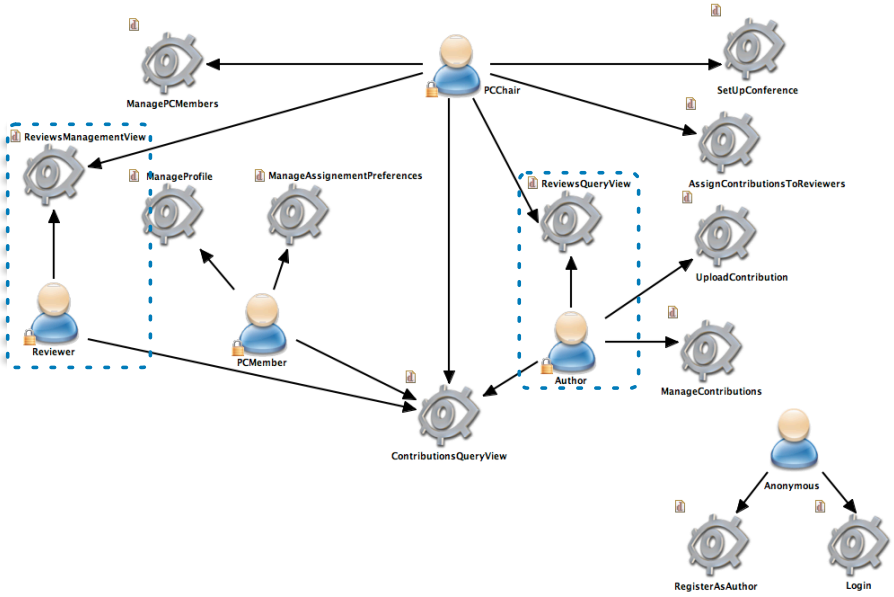


Figura 9.2: Diagrama UIM de vistas por actor del SRC

Especificación abstracta de "*ReviewsQueryView*": vista de consulta de revisiones

En la Figura 9.3 se puede ver el modelo abstracto para las interfaces que permiten a un autor consultar las revisiones que han obtenido sus contribuciones a la conferencia. En primer lugar (1) se consultan los datos del autor y las contribuciones que ha realizado; tras esto (2) vemos el detalle de una contribución y los resultados generales de las revisiones que ha obtenido. Por último, (3) el detalle de una revisión y el identificador del revisor que la ha realizado.

Vemos que en este caso no existen operaciones visibles pues se trata de unas interfaces para la consulta y no son necesarias. Se pude observar también que existen atributos que no se muestran por tratarse del actor que se trata (i.e: *pCComments* de la clase *Review*).

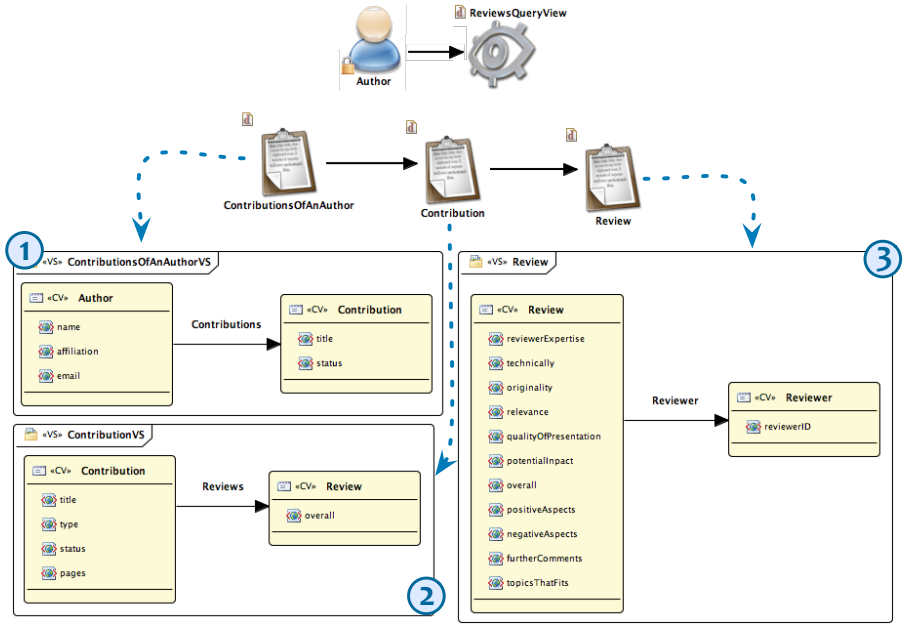


Figura 9.3: Reviews Query View

Especificación abstracta de "*ReviewsManagementView*": vista de gestión de revisiones

En la Figura 9.4 se observa el modelo abstracto de las interfaces que permiten a los revisores introducir las revisiones a las contribuciones que les han sido asignadas. En primer lugar (1) pueden verse los datos del revisor y las revisiones que han realizado así como la opción de crear una nueva revisión (2). Tras esto puede acceder al detalle de una revisión realizada (3) pudiendo modificarle el estado (4) o editar otros campos (5). Por último puede consultar algunos datos de la contribución a que se refiere la revisión (6).

En este caso, al tratarse de interfaces de gestión, vemos que si tenemos disponibles operaciones (i.e: *createReview*). Éstas se implementarán mediante un formulario cuyos campos podremos implementar según las posibilidades que nos ofrezca el catálogo de patrones que hemos visto

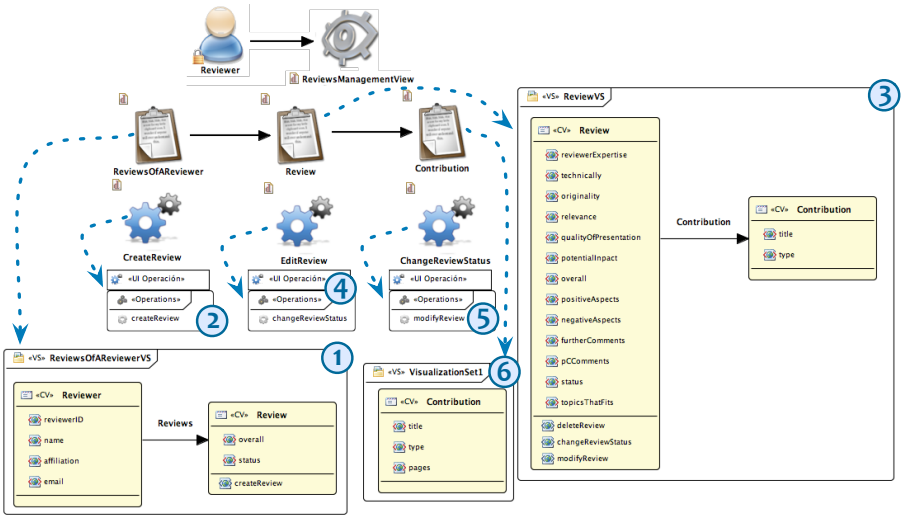


Figura 9.4: Reviews Management View

anteriormente en el Capítulo 6.

9.1.2. Selección de las Características de los Dispositivos

En este punto se van a especificar las características de interacción de los dispositivos que hemos especificado como destino. Estos son un iPad y un HTC Magic.

Cabe destacar que, el fin de estas especificaciones es incorporarlas a un repositorio de descripciones de dispositivos que podremos utilizar en todos nuestros procesos de desarrollo. Por ello, la descripción de un dispositivo la deberemos realizar una sola vez.

Especificación de las características del iPad

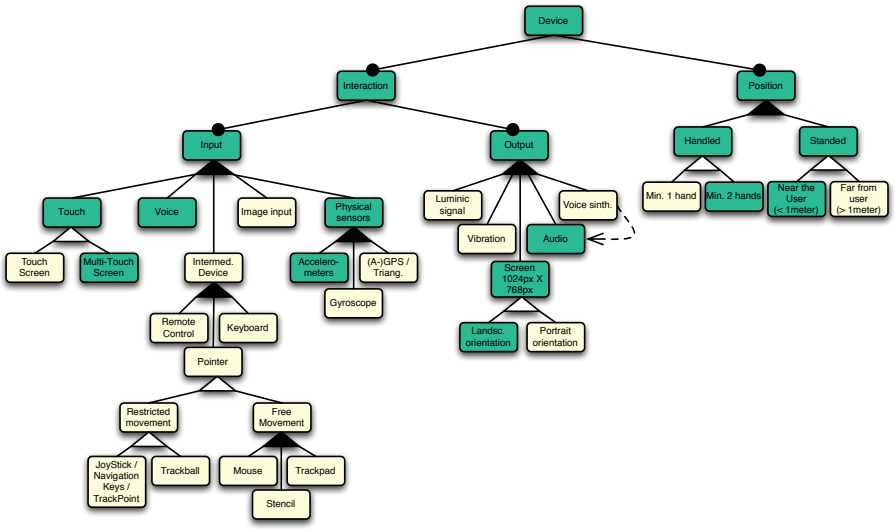


Figura 9.5: Especificación del iPad.

En la Figura 9.5 se observa el modelo de características que exponíamos en la Sección 5.2 instanciado para especificar las características de interacción que ofrece el iPad. En este caso se ha especificado la pantalla en horizontal como posición natural para nuestra aplicación (*Device*→*Interaction*→*Output*→*Screen*→*Landscape Orientation*).

Vemos que en este caso se ha especificado que como mínimo se requieren las 2 manos (*Device*→*Position*→*Handled*→*Min. 2 hands*) para manejar el dispositivo y que este se encontrará cerca del usuario (*Device*→*Position*→*Standed*→*Near the user*).

Especificación de las características del HTC Magic

En la Figura 9.6 se observa el modelo de características instanciado para especificar la interacción que ofrece el HTC Magic. En este caso se ha especificado la pantalla en vertical como posición natural (*Portrait*

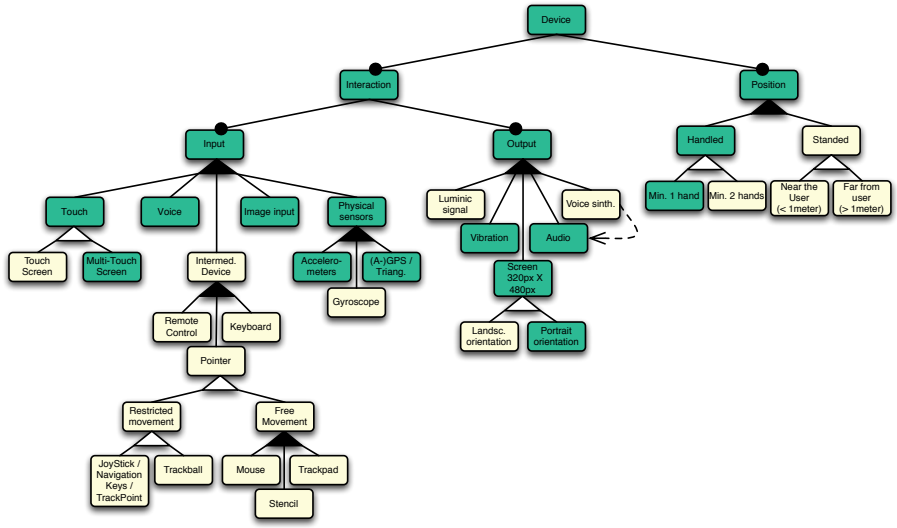
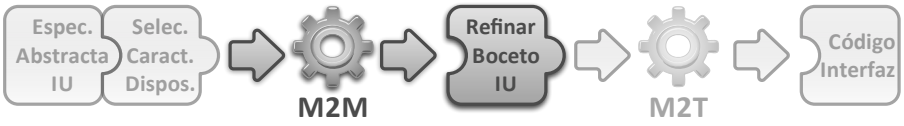


Figura 9.6: Especificación del HTC Magic.

Orientation).

Hemos especificado en este caso que solo se requiere 1 mano (*Min. 1 hand*) para manejar el dispositivo y en este caso no se ha especificado posición para el dispositivo sobre una superficie.

9.1.3. Generación de Bocetos de Interfaz



En este punto se aplica el proceso de transformación M2M expuesto en la Sección 7.1.

En el Anexo C se puede ver el proceso completo de la aplicación del algoritmo de transformación para el escenario *ReviewsQueryView*

Tras el proceso de transformación obtenemos un boceto completo

de las interfaces de usuario, y podemos definir las trazas de decisiones tomadas que justifican tales modos de interacción. Como se puede apreciar en las figuras siguientes, se han obtenido dos interfaces de usuario diferentes, una por cada tipo de dispositivo, adaptadas a la naturaleza y capacidades de interacción de cada uno de ellos.

"*ReviewsQueryView*": bocetos de interfaz para la consulta de revisiones

En este caso podemos ver que, por las características del iPad, hemos podido condensar toda la información en una sola unidad de interacción (Figura 9.7), pero superando el factor de máxima cantidad de información por pantalla.

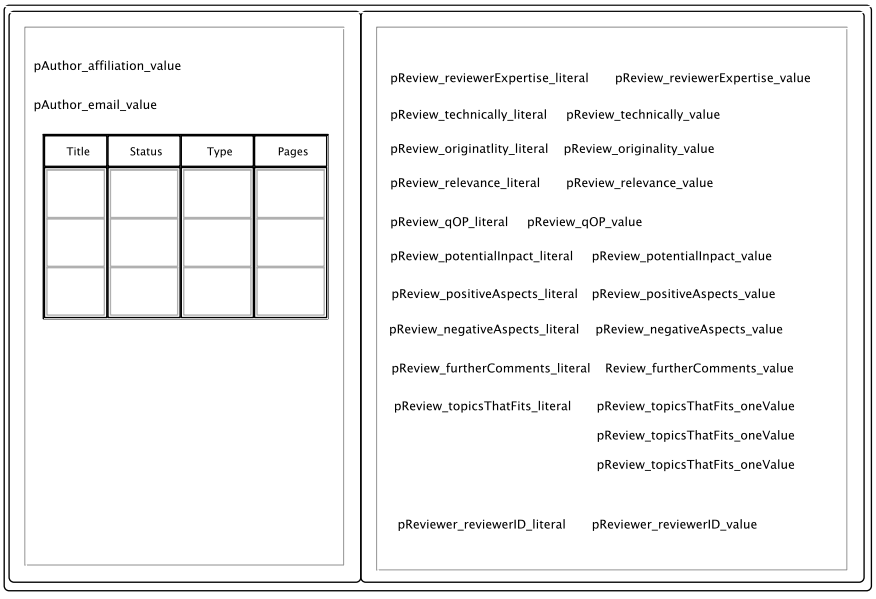


Figura 9.7: Sketch de interfaz para la consulta de revisiones en iPad

En cambio para el HTC Magic hemos requerido 4 (Figura 9.8), en las que no se ha llegado a superar el FMI. Las decisiones tomadas para

llegar a este resultado se pueden consultar en el Anexo C.

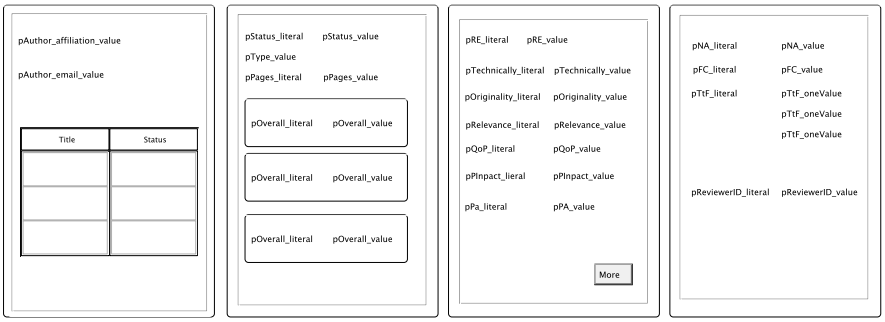


Figura 9.8: Sketch de interfaz para la consulta de revisiones en Android

En este caso se han fusionado las clases Review y Contribution, y las navegaciones han sido todas del tipo "En contenedor Activo" o "Misma Ventana y Distinto Contenedor".

Se puede apreciar también que la información del detalle de una revisión en el HTC Magic se ha dividido en 2 unidades de interacción pues en el proceso de transformación se detectó el *factor de máxima cantidad de información por unidad de interacción*. Este factor indica la máxima cantidad de atributos que pueden ser mostrados en una unidad de interacción sin necesidad de *scroll*. Cuando éste es alcanzado se advierte al analista por si desea cambiar de unidad de interacción (botón more).

"*ReviewsManagementView*": bocetos de interfaz para la gestión de revisiones

En este caso, tras aplicar el algoritmo de transformación, vemos que hemos requerido 3 unidades de interacción para el iPad (Figura 9.9), 1 para la información de la revisión y 2 para el modo de edición.

Para el HTC Magic en cambio, se han requerido 7 unidades de interacción (Figura 9.10), 3 para la información y 4 para la edición.

Se puede apreciar que pese a tratarse de AIOs diferentes en los casos de la selección de la contribución (selección única) y selección de topics

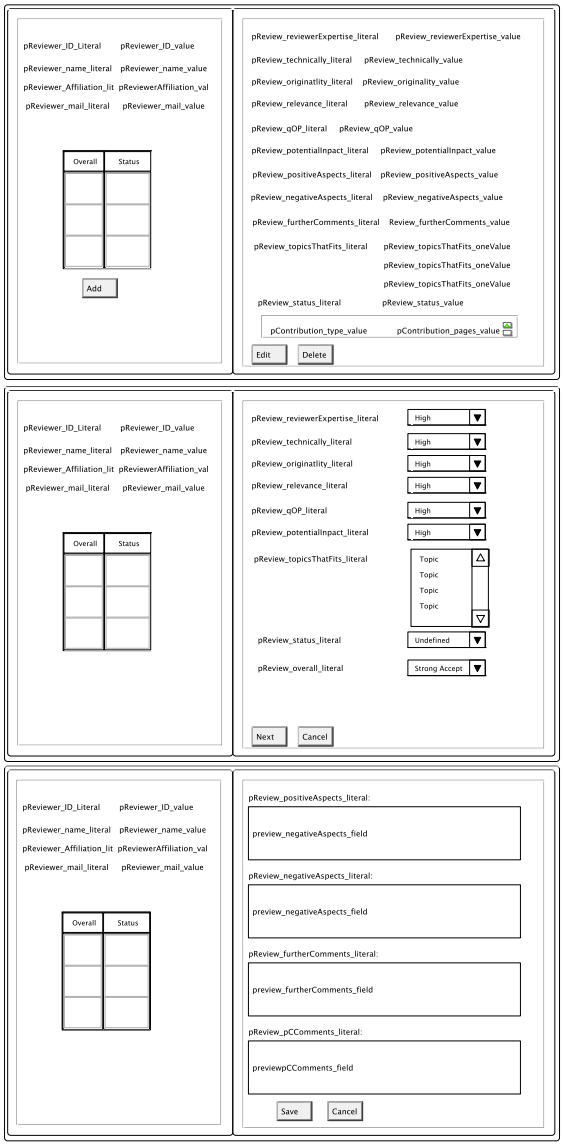


Figura 9.9: Sketch de interfaz para la gestión de revisiones en iPad

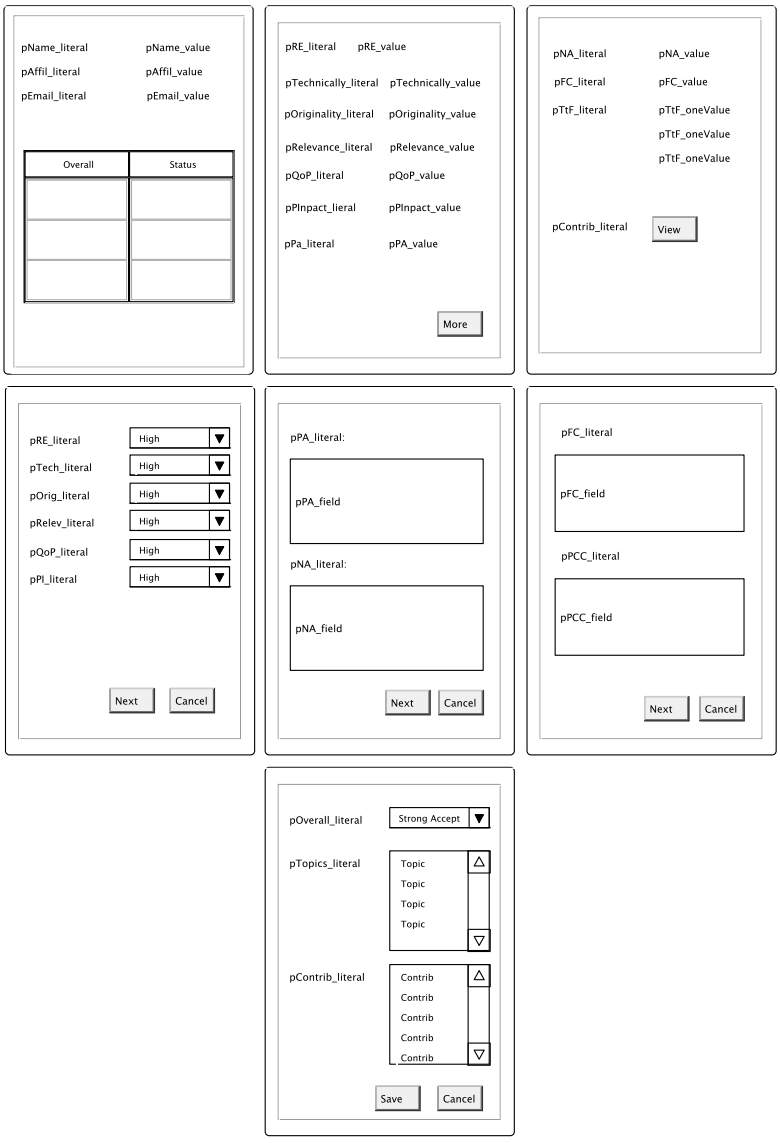


Figura 9.10: Sketch de interfaz para la gestión de revisiones en Android

(selección múltiples), se ha utilizado el mismo CIO pero como se podrá ver en la Sección 9.1.4 la implementación de este será diferente en cada caso.

En total se han generado 14 bocetos distintos (3+7 para Android y 1+3 para iPad). Se trata de un número bastante elevado de interfaces teniendo en cuenta que los escenarios aplicados son solo dos y bastante sencillos. La complicación que acarrearía la construcción de las interfaces de estos sistemas adhoc pone de manifiesto la necesidad de aplicar métodos sistemáticos como GeMMINi. Mas aún si estos sistemas afrontan su desarrollo en múltiples dispositivos.

9.1.4. Generación del Código de la Aplicación

Esta última fase supone la generación del código para la interfaz nativa dependiente del dispositivo destino. Este paso implica la transformación de estas especificaciones sobre las tecnologías y plataformas destino (iOS y Android, respectivamente). Se aporta el resultado de esta transformación para los distintos tipos de dispositivo y escenarios relatados.

Se debe tener en cuenta que los bocetos son una representación aproximada del resultado final y que por tanto, cada CIO representado en el boceto puede tener una implementación distinta en función de la plataforma destino. No hay que olvidar que se generan aplicaciones nativas sobre la API estándar del fabricante y por tanto cada uno facilita su implementación.

"*ReviewsQueryView*": interfaces para la consulta de revisiones

En la Figura 9.11 se puede ver el resultado de la generación de las interfaces para el iPad utilizando iOS. Como decíamos anteriormente se

ha podido condensar toda la información en una interfaz pero a cambio, ésta necesita de un *scroll* lateral para poder acceder a toda la información.

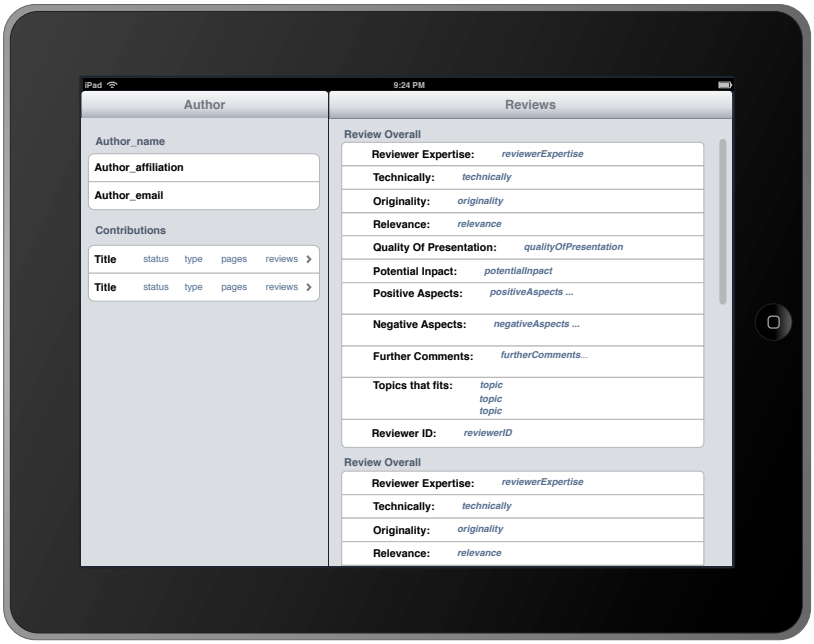


Figura 9.11: Interfaz para la consulta de revisiones en iPad

En cambio vemos que las cuatro interfaces generadas para el HTC Magic disponibles en la Figura 9.12, son capaces de mostrar toda la información solicitada sin utilizar ninguna barra de desplazamiento.

"*ReviewsManagementView*": interfaces para la gestión de revisiones

En este caso vemos como las interfaces son apoyadas por menús emergentes que habitualmente no estan en pantalla pero que siguen el estilo propuesto por el look&feel de los sistemas operativos.Vemos que en iOS (Figura 9.14) los controles de selección única son un *spinner*

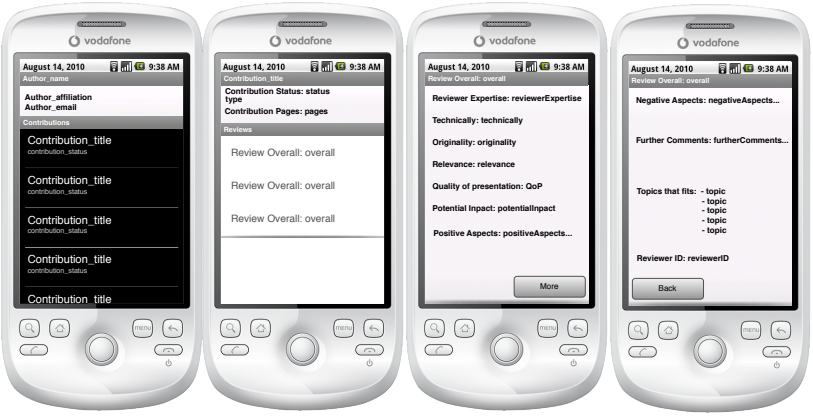


Figura 9.12: Interfaz para la consulta de revisiones en Android

que aparecerá al pie de la página, mientras que las selecciones múltiples se implementan con *pop-ups* emergentes con las listas de elementos seleccionables.

En Android (Figura 9.13) las opciones de añadido, edición y borrado se muestran mediante menú emergente al pulsar el botón físico "menú" del dispositivo. En éste caso las listas de selección única se implementan utilizando un *RadioButton Group* emergente mientras que las de selección múltiple lo hacen mediante un *CheckBox Group*.

Figura 9.13: Interfaz para la gestión de revisiones en Android

Figura 9.14: Interfaz para la gestión de revisiones en iPad

9.2. Conclusiones

En este capítulo se ha aplicado GeMMINi a dos escenarios de un caso práctico: un sistema de revisión de conferencias. En este caso, se han obtenido las interfaces para la consulta y la gestión de las revisiones por parte de autores y revisores respectivamente. En total hemos generado 14 interfaces distintas (3+7 para Android y 1+3 para iPad). Se trata de un número bastante elevado de interfaces teniendo en cuenta que los escenarios aplicados son solo dos y bastante sencillos.

Esto pone de manifiesto la necesidad de aplicar métodos de desarrollo como GeMMINi que nos ayuden a llevar a cabo todo el proceso, mas cuando hablamos de desarrollos para múltiples dispositivos, ya que con la aplicación del método, tan sólo hemos tenido que definir la interfaz abstracta ya que el resto del proceso puede ser totalmente automatizado.

— ❝*I think and think for months and years. Ninety-nine times, the conclusion is false. The hundredth time I am right.*❞

Albert Einstein.

10 Conclusiones

El trabajo expuesto en este documento presenta un método de desarrollo dirigido por modelos para obtener interfaces de usuario para múltiples dispositivos, partiendo de una misma especificación abstracta.

Se ha presentado una revisión del estado del arte y se ha centrado el trabajo en las áreas de estudio y comunidades correspondientes, presentando los principios mas importantes de cada una de ellas (Capítulo 2) y una visión global de la solución propuesta (Capítulo 4) así como los fundamentos metodológicos y el contexto tecnológico en que se encuadra la propuesta (Capítulo 3).

Tras esto, por una parte se ha presentado el método de diseño con los modelos a especificar por parte del diseñador para la obtención de dichas interfaces (Capítulos 3, 5 y 6). Por otra parte se ha expuesto la aplica-

ción del método de desarrollo para la generación (semi-)automática de las interfaces especificadas (Capítulos 7 y 8).

Finalmente se ha aplicado el método propuesto a un caso de estudio ampliamente conocido: *un sistema de revisión de conferencias* (Capítulo 9).

Este último capítulo presenta las conclusiones sobre el trabajo desarrollado en esta tesis. La Sección 10.1 presenta las contribuciones mas importantes presentadas en este trabajo. La Sección 10.2 provee la lista de publicaciones relacionadas con esta tesis y, finalmente, la Sección 10.3 resume las líneas de trabajo presentes y futuras que pueden extender la línea de investigación presentada en esta tesis.

10.1. Contribuciones

La contribución principal de este trabajo es un proceso de desarrollo de interfaces de usuario para múltiples dispositivos. El proceso comprende tanto aspectos arquitectónicos como metodológicos. Por ello, el presente trabajo presenta las siguientes contribuciones:

Método de diseño. Se han definido los pasos para obtener el diseño de las interfaces que puedan ser llevadas a múltiples dispositivos, atendiendo a las características de interacción que estos presenten.

Por una parte se utiliza MOSKitt UIM para modelar de manera abstracta la interfaz especificando solo los aspectos concernientes a la información y no a su representación en los distintos dispositivos. Esto se realiza mediante (1) un modelo de datos (típicamente UML) y (2) el propio modelo abstracto de interfaz, que agrupa y estructura la información especificada en el modelo de datos utilizando conceptos como *unidad de interacción* o *clase visible*.

Por otra parte se ha especificado un modelo de características que permite expresar las capacidades de los distintos dispositivos atendiendo a criterios tanto de interacción (entrada y salida) como de posición de los dispositivos respecto al usuario (cerca o lejos del usuario, una o dos manos).

Método de desarrollo. Se ha definido un método sistemático para la obtención (semi-)automática de interfaces de usuario, utilizando un enfoque basado en líneas de producto bajo el marco del desarrollo de software dirigido por modelos. Este método comprende desde la especificación hasta la obtención de las distintas interfaces.

Dado que el método esta completamente soportado por modelos, se han podido aplicar las técnicas del desarrollo dirigido por modelos (como transformaciones de modelos o generación de código).

10.2. Publicaciones

A continuación se listan las publicaciones relacionadas con esta trabajo de tesis:

1. Carlos Cetina, **Ignacio Mansanet**, Ismael Torres, Joan Fons. *Una arquitectura para la integración de tecnologías en sistemas domóticos sensibles al contexto.* KNX International Forum 2010, Madrid, Spain, 2010. pp. 96—102. Premio al mejor artículo presentado por un joven investigador.

2. **Ignacio Mansanet**, Joan Fons, Ismael Torres, Vicente Pelechano. *GeMMINi: Prototipado de interfaces de usuario sobre múltiples dispositivos. Una estrategia basada en Líneas de Producto y MDD.* XII Congreso de Interacción Persona-Ordenador (Interacción'11). Lisboa, Portugal, 2011.

El primero de los artículos fue publicado en el *KNX International Forum*, foro internacional destinado a la discusión de temas relacionados con la domótica y la Inteligencia Ambiental. En el centro ProS se está desarrollando sistemas de esta naturaleza que tienen como base la integración de los dispositivos en la vida cotidiana [47].

En el desarrollo de estos sistemas y en el marco del proyecto *OSAmI Commons* surgió la necesidad de implementar interfaces de usuario en

diversos dispositivos (PC, Móvil, TV, etc.) que permitiesen el control de estos sistemas de inteligencia ambiental, y que pudieran aprovechar al máximo las capacidades de interacción de cada dispositivo. Fruto de esta necesidad, se inició esta línea de investigación en el desarrollo de aplicaciones para múltiples dispositivos.

Además se ha enviado y estamos pendiente de recibir la evaluación de la siguiente:

3. **Ignacio Mansanet**, Miriam Gil, Joan Fons, Vicente Pelechano. *A Feature-based Approach for Context-Aware Interactions over Multiple Devices*. V International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI'11). Riviera Maya, Mexico, 2011.

10.3. Trabajo Futuro

La investigación presentada en este documento no está cerrada. Existen muchas direcciones interesantes que tomar para dotar a la propuesta de mayor aplicabilidad. A continuación se resumen las mas interesantes:

Ampliación del catálogo de patrones. El catálogo de patrones es una de las piezas claves de nuestra propuesta ya que permite establecer correspondencias entre conceptos abstractos e implementaciones concretas de interfaz. Actualmente este catálogo está limitado a correspondencias *atómicas*, es decir, se trata cada unidad de interacción de manera individual.

Se pretende ampliar este catálogo (y por ende el algoritmo de transformación) para detectar y tener también en cuenta patrones de diseño a través de unidades de interacción y detectar así nuevos patrones (i.e., listas anidadas).

Obtención automática de bocetos. Relacionado con el punto anterior y dado que se pretende ampliar el catálogo de patrones, esto puede causar que el proceso de obtención de los bocetos de interfaz resulte algo tedioso e incluso difícil de entender al perderse la trazabilidad.

Se esta trabajando en la mejora de los algoritmos para la obtención de bocetos de manera que presenten el mayor grado de automatización posible, siempre preservando la capacidad que se brinda al usuario para establecer las primitivas concretas a utilizar.

Sensibilidad al contexto. Mark Weiser describió la computación ubicua (UbiComp) como un mundo donde la computación y la comunicación sería distribuida mediante servicios en todo nuestro entorno cotidiano [47]. Estos servicios son ofrecidos a los usuarios mediante dispositivos. La promesa de la interacción natural de UbiComp implica adaptarse a un gran número de condiciones de contexto para hacer la vida mas agradable a los usuarios.

Con este fin, se esta trabajando actualmente en la integración de la presente propuesta con la presentada por Gil et al. en [20], con el fin de dotar a las interfaces generadas de adaptación automática a las condiciones del entorno y/o del usuario. Se quiere obtener un mecanismo mediante el cual, las interfaces obtenidas mediante la aplicación del método presentado en este documento, sean sensibles al contexto en que son utilizadas y, en la medida de lo posible, no *molesten* al usuario.

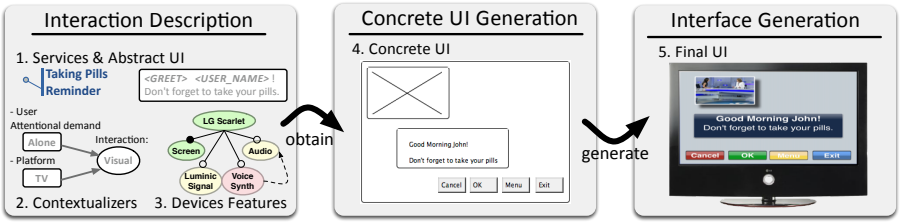


Figura 10.1: Generación de interfaces sensibles al contexto.

Desarrollo por parte del usuario final. El uso de técnicas de modelado para formalizar conceptos, permite la automatización de los procesos del desarrollo software. En el presente trabajo se parte de una especificación abstracta de la interfaz de usurario que es difícil de entender por los usuarios pero que nos brinda la posibilidad de separar los conceptos concernientes a la información de

los dependientes de dispositivos. El siguiente paso es el de proveer de herramientas que permitan especificar esos modelos abstractos mediante un lenguaje fácil de entender por cualquier usuario.

Bibliografía

[1] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., & Shuster, J. E. (1999). UIML: an appliance-independent XML user interface language. In *Proceedings of the eighth international conference on World Wide Web*, WWW '99, (pp. 1695–1708). New York, NY, USA: Elsevier North-Holland, Inc.

[2] Agrawal, A., Levendovszky, T., Sprinkle, J., Shi, F., & Karsai, G. (2002). Generative programming via graph transformations in the model-driven architecture. In *OOPSLA 2002 Workshpo in Generative Techniques in the context of Model Driven Architecture*.

[3] Aquino, N., Vanderdonckt, J., Condori-Fernández, N., Dieste, O., & Pastor, O. (2010). Usability evaluation of multi-device/platform user interfaces generated by model-driven engineering. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, (pp. 30:1–30:10). New York, NY, USA: ACM.

[4] Berti, S., Mori, G., Paternò, F., & Santoro, C. (2005). Teresa: an environment for designing multi-device interactive services. In *4th Simposio su Human-Computer-Interaction*, (pp. 40 – 44). Hcitaly 2005.

[5] Bézivin, J., & Gerbé, O. (2001). Towards a Preci-

se Definition of the OMG/MDA Framework. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, (p. 273). Washington, DC, USA: IEEE Computer Society.

[6] Blumendorf, M., Roscher, D., & Albayrak, S. (2010). Dynamic user interface distribution for flexible multimodal interaction. In *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*, ICMI-MLMI '10, (pp. 20:1–20:8). New York, NY, USA: ACM.

[7] Bodart, F., & Vanderdonckt, J. (1996). Widget standardisation through abstract interaction objects. In *Advances in Applied Ergonomics*, (pp. 300–305). USA Publishing.

[8] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, *15*, 289–308.

[9] Clements, P., & Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc.

[10] Clerckx, T., Luyten, K., & Coninx, K. (2005). DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In R. Bastide, P. Palanque, & J. Roth (Eds.) *In The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems*, vol. 3425 of *Lecture Notes in Computer Science*, (pp. 77–95). Springer-Verlag.

[11] Clerckx, T., Vandervelpen, C., & Coninx, K. (2008). Engineering Interactive Systems. chap. Task-Based Design and Runtime Support for Multimodal User Interface Distribution, (pp. 89–105). Berlin, Heidelberg: Springer-Verlag.

[12] Coninx, K., Luyten, K., Vandervelpen, C., Bergh, J., & Creemers, B. (2003).

Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In *Human-Computer Interaction with Mobile Devices and Services*, vol. 2795 of *Lecture Notes in Computer Science*, (pp. 256–270). Springer Berlin / Heidelberg.

[13] Czarnecki, K., & Kim, P. (2005). Cardinality-based feature modeling and constraints: A progress report. In *Proceedings of the International Workshop on Software Factories At OOPSLA 2005*.

[14] Di Santo, G., & Zimeo, E. (2007). Reversing guis to ximl descriptions for the adaptation to heterogeneous devices. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, (pp. 1456–1460). ACM.

[15] Eisenstein, J., Vanderdonckt, J., & Puerta, A. (2001). Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the 6th international conference on Intelligent user interfaces*, IUI '01, (pp. 69–76). New York, NY, USA: ACM.

[16] Ferrara, J. M. (2008). *Model Driven Development of Pervasive Systems. Building a Software Factory*. Ph.D. thesis, Universidad Politécnica de Valencia.

[17] Gajos, K., & Weld, D. S. (2004). SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*, (pp. 93–100). New York, NY, USA: ACM.

[18] Gajos, K. Z., Weld, D. S., & Wobbrock, J. O. (2010). Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, *174* (12-13), 910 – 950.

[19] Gil, M., Giner, P., & Pelechano, V. (2010). Service obtrusiveness adaptation. In *Proc. AmI 2010*, (pp. 11–20). Springer-Verlag.

[20] Gil, M., Giner, P., & Pelechano, V. (2011). Personalization for unobtrusive service interaction. *Personal and Ubiquitous Computing*, *15*, 1–19.

[21] Giner, P. (2010). *Automating the development of Physical Mobile Workflows: a Model Driven Engineering approach*. Ph.D. thesis, Universidad Politécnica de Valencia.

[22] Group, O. M. (2008). Software Process Engineering Metamodel (SPEM) 2.0. Tech. rep., Object Management Group,

[23] Hallsteinsen, S., Stav, E., Solberg, A., & Floch, J. (2006). Using product line techniques to build adaptive systems. In *Proceedings of the 10th International on Software Product Line Conference*, (pp. 141–150). Washington, DC, USA: IEEE Computer Society.

[24] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. rep., Carnegie-Mellon University Software Engineering Institute,

[25] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., & López-Jaquero, V. (2005). USIXML: A Language Supporting Multi-path Development of User Interfaces. In R. Bastide, P. Palanque, & J. Roth (Eds.) *Engineering Human Computer Interaction and Interactive Systems*, vol. 3425 of *Lecture Notes in Computer Science*, (pp. 200–220). Springer Berlin / Heidelberg.

[26] March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decis. Support Syst.*, *15*, 251–266.

[27] McIlroy, D. (1968). Mass-Produced Software Components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, (pp. 88–98). Springer-Verlag.

[28] Mellor, S. J., Clark, A. N., & Futagami, T. (2003). Guest editors' introduction: Model-driven development. *IEEE Software*, *20*(5), 14–18.

[29] Mori, G., Paterno, F., & Santoro, C. (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Softw. Eng.*, *30*, 507–520.

[30] Moscovich, T. (2006). Multi-touch interaction. In *CHI '06 extended abstracts on Human factors in computing systems*, CHI EA '06, (pp. 1775–1778). New York, NY, USA: ACM.

[31] Moscovich, T. (2007). *Principles and applications of multi-touch interaction*. Ph.D. thesis, Providence, RI, USA. AAI3272023.

[32] Object Management Group (2007). Unified Modeling Language (UML).

[33] Panach, J. I., España, S., Pederiva, I., & Pastor, O. (2007). OO-Sketch: una herramienta para la captura de requisitos de interacción. In *X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS 2007)*.

[34] Panach Navarrete, J. I. (2010). *Incorporación de Mecanismos de Usabilidad en un Entorno de Producción de Software Dirigido por Modelos*. Ph.D. thesis, Universidad Politécnica de Valencia.

[35] Paternò, F., Santoro, C., & Spano, L. D. (2009). MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, *16*, 19:1–19:30.

[36] Pinheiro da Silva, P. (2001). User Interface Declarative Models and Development Environments: A Survey. In P. Palanque, & F. Paternò (Eds.) *Interactive Systems Design, Specification, and Verification*, vol. 1946 of *Lecture Notes in Computer Science*, (pp. 207–226). Springer Berlin / Heidelberg.

[37] Puerta, A. (1996). The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. In *In Computer-Aided Design of User Interfaces*, (pp. 5–7). Namur University Press.

[38] Puerta, A., & Eisenstein, J. (2002). XIML: A Universal Language for User Interfaces. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, (pp. 214–215). New York, NY, USA: ACM.

[39] Rohs, M., & Bohn, J. (2003). Entry Points into a Smart Campus Environment " Overview of the ETHOC System. In *ICDCSW '03. Proceedings of the 23rd International Conference on Distributed Computing Systems*, (p. 260). IEEE Computer Society.

[40] Schmidt, D. C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, *39*, 25–31.

[41] Sottet, J.-S., Calvary, G., & J.-M., F. (2005). Towards model-driven engineering of plastic user interfaces. Tech. rep., MDDAUI,

[42] Thielmann, B., & Dowling, M. (1999). Convergence and innovation strategy for service provision in emerging Web-TV markets. *International Journal on Media Management*, *1*(1), 4–9.

[43] Tidwell, J. (2011). *Designing interfaces. Patterns for effective interaction design*. O'Reilly, 2nd ed.

[44] Vaishnavi, V., & Kuechler, W. (2007). Design Research in Information Systems.

[45] Valverde, F., & Pastor, Ó. (2009). Facing the technological challenges of web 2.0: A ria model-driven engineering approach. In *Web Information Systems Engineering - WISE 2009*, vol. 5802 of *Lecture Notes in Computer Science*, (pp. 131–144). Springer Berlin / Heidelberg.

[46] Van Welie, M., & Groot, B. d. (2002). Consistent Multi-device Design Using Device Categories. In *Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*, Mobile HCI '02, (pp. 315–318). London, UK: Springer-Verlag.

[47] Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 66–75.

ANEXOS

A

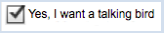
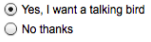
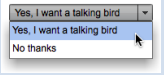
Descripción de Controles de Interfaz

En el proceso de diseño y construcción de una interfaz de usuario, es tan importante la decisión de qué información mostrar, como la de cómo mostrarla. Es por esto que la elección de los controles que utilizaremos para representar nuestra interfaz, cobra una gran importancia. Pero, según cada situación, se deben elegir los controles que sean más adecuados.

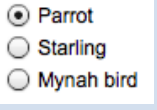
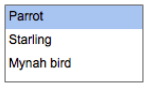
En este anexo se presentan unas listas de controles agrupadas por el tipo de entrada (o salida) para el que son utilizados, acompañados de un conjunto de pros y contras de su utilización según determinadas circunstancias.

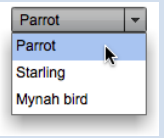
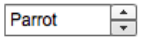
A.1. Selecciones

Selección binaria

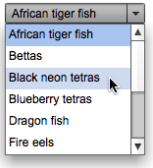
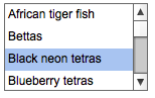
Control	Ejemplo	Pros	Contras
CheckBox		Simplicidad; Ocupa poco espacio.	Solo expresa un valor de la dupla lo que puede llevar a confusión.
2 radio buttons		Las dos opciones son visibles.	Puede consumir mucho espacio.
DropDown list box		Todas las opciones disponibles; Fácil de ampliar *a posteriori*; Ocupa poco espacio.	Solo hay visible una opción a la vez en visualización.
Botón Toggle		Ídem Checkbox.	Ídem Checkbox.

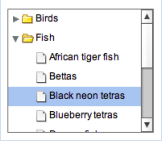
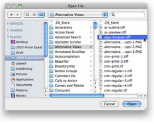
Selección única entre pocos elementos

Control	Ejemplo	Pros	Contras
RB Group		Todas las opciones son visibles.	Consume mucho espacio.
Lista/Tabla de selección única		Todas las opciones son visibles.	Consume mucho espacio.

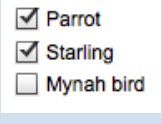
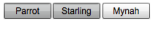
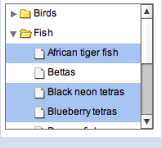
DropDown List Box		Todas las opciones son visibles; Fácil de ampliar *a posteriori*; Ocupa poco espacio.	Solo hay una opción visible en estado normal.
Botones Toggle excluyentes		Ídem checkbox.	Ídem checkbox.
Spinner		Ocupa poco espacio.	Solo una opción es visible; Puede ser difícil manejar botones pequeños en determinados dispositivos.

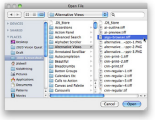
Selección única entre muchos elementos

Control	Ejemplo	Pros	Contras
DropDown List Box		Todas las opciones son visibles; Fácil de ampliar *a posteriori*; Ocupa poco espacio.	Solo hay una opción visible en estado normal.
Lista/Tabla de selección única		Todas las opciones son visibles.	Consume mucho espacio.

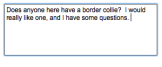
Listas en cascada		Muchas opciones son visibles; La ordenación facilita el acceso.	Ocupa mucho espacio; Requiere conocimientos por parte del usuario.
Emergente		Permite navegación.	Difícil de diseñar y utilizar; Requiere más interacción.

Selección múltiple de elementos

Control	Ejemplo	Pros	Contras
CB's		Todas las opciones son visibles.	Consume mucho espacio.
botones Toggle		Todas las opciones son visibles; Ocupa poco espacio.	Solo para pocos elementos; Puede confundir con los excluyentes.
Listas en cascada		Muchas opciones son visibles; La ordenación facilita el acceso.	Ocupa mucho espacio; Requiere conocimientos por parte del usuario; Puede confundir con la selección única.

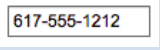
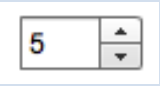
Emergente		Permite navegación.	Difícil de diseñar y utilizar; Requiere más interacción; Puede confundir con la selección única.
Lista/Tabla de selección múltiple		Muchas opciones visibles; Puede reducirse en tamaño (scroll).	No todas las opciones son visibles sin scroll; A más opciones visibles más espacio ocupa.
Lista/Tabla de CB's		Muchas opciones visibles; Puede reducirse en tamaño (scroll); Fácil de entender.	No todas las opciones son visibles sin scroll; A más opciones visibles más espacio ocupa.
List Builder		La selección es fácilmente visible y ordenable.	Consume mucho espacio; Si ambos conjuntos son grandes, se hace difícil de manejar.

A.2. Texto

Control	Ejemplo	Pros	Contras
Text Field		Ocupa poco espacio.	Solo una línea de texto.
botones Toggle		Puede albergar muchas lineas de texto.	Puede ocupar mucho espacio.

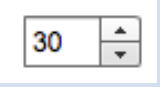
A.3. Valores Numéricos

Números

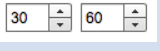
Control	Ejemplo	Pros	Contras
Text Field		Ocupa poco espacio; Permite varios formatos.	La ausencia de formato puede confundir; Requiere validación.
Text Field Formateado		Expresa el formato; Evita algunas validaciones.	Consume más espacio; Un cambio de formato requiere un cambio del control, no solo de la validación.
Spin Box		Ocupa poco espacio; Permite interacción tanto en teclado como en puntero.	Solo adecuado para enteros o tramos; Puede ser difícil manejar botones pequeños en determinados dispositivos; Puede requerir muchos clics hasta llegar al valor.

Números en un rango

Control	Ejemplo	Pros	Contras
Slider		Muy obvio; Expresa el valor y la escala; Limita al rango especificado.	Consume mucho espacio.

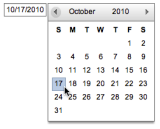
Text Field con ayuda		Ocupa poco espacio.	No hay restricción ante el input *a priori*; Requiere más validaciones.
Spinner		Ocupa poco espacio; Permite interacción tanto con teclado como con puntero.	No se ve el rango; Puede ser difícil manejar botones pequeños en determinados dispositivos; Puede requerir muchos clics hasta llegar al valor.
Slider con text field		Permite visualización tanto gráfica como textual.	Requiere validación.

Especificación de subrango

Control	Ejemplo	Pros	Contras
Doble Slider		Muy obvio; Expresa el valor y la escala; Limita al rango especificado.	No es muy familiar a los usuarios.
Dos sliders		Familiar y obvio.	Ocupa mucho espacio.
Dos Spinner		Ocupa poco espacio; Permite interacción tanto en teclado como en puntero; Valores limitados en rango.	Puede ser difícil manejar botones pequeños en determinados dispositivos; Puede requerir muchos clics hasta llegar al valor; Necesita validación.

Dos text fields	30 60	Consume muy poco espacio.	Requiere validación; No hay limitación, por lo que requiere indicar el rango con etiquetas.

A.4. Fechas y Horas

Control	Ejemplo	Pros	Contras
Text Field	Oct 17, 2010	Muy simple; Permite muchos formatos; Ocupa poco espacio.	Requiere validación; La ausencia de formato implica que debe indicarse el requerido.
Text Field con formato	Oct 17 , 2010	Muy simple; Formato claro; Ocupa poco espacio.	Puede ocupar mucho espacio; un cambio de formato requiere un cambio del control, no solo de la validación.
Calendar / Reloj		Muy obvio; Restringido a los valores deseados.	Consume mucho espacio.
Drop down con calendar		Combina las ventajas del calendario y el text field.	La interacción no es obvia; Requiere validación.

Spin Box	Ocupa poco espacio; Los valores están restringidos	Puede ser difícil manejar botones pequeños en determinados dispositivos; Puede requerir muchos clics hasta llegar al valor.
Spin Box dividido	Ocupa poco espacio; Los valores están restringidos; El formato es obvio.	Puede ser difícil manejar botones pequeños en determinados dispositivos.

B

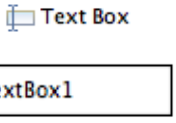
Controles Disponibles en MOSKitt Sketcher

En este anexo se presenta una selección de los controles disponibles en MOSKitt Sketcher para llevar a cabo las técnicas de creación de bocetos.

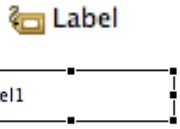
Se han agrupado en dos conjuntos: (1) los controles simples o atómicos que podemos ver en la Sección B.1, y (2) los complejos que podemos ver en la Sección B.2.

B.1. Widgets Simples

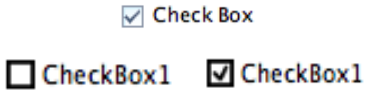
CONTROL #1

Nombre:	***TextBox***
Representación en Sketcher:	
Observaciones:	Se suele utilizar como input para cualquier tipo de dato aunque en ciertas ocasiones puede ser output (un dato derivado)

CONTROL #2

Nombre:	***Label***
Representación en Sketcher:	
Observaciones:	Su uso es eminentemente como output, aunque en algunos casos es meramente informativo

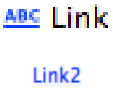
CONTROL #3

Nombre:	***Check Box***
Representación en Sketcher:	
Observaciones:	Ocurre lo mismo que con el TextBox, aunque en este caso se utiliza solo para datos bivaluados

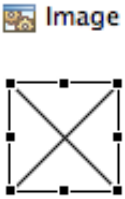
CONTROL #4

Nombre:	***Radio Button***
Representación en Sketcher:	
Observaciones:	Ocurre lo mismo que con el TextBox, aunque en este caso se utiliza solo para datos bivaluados

CONTROL #5

Nombre:	***Link***
Representación en Sketcher:	
Observaciones:	

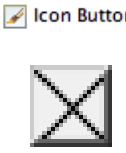
CONTROL #6

Nombre:	***Image***
Representación en Sketcher:	
Observaciones:	

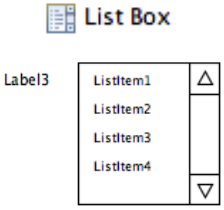
CONTROL #7

Nombre:	***Button***
Representación en Sketcher:	
Observaciones:	

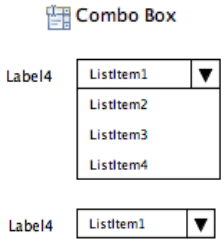
CONTROL #8

Nombre:	***Icon Button***
Representación en Sketcher:	
Observaciones:	

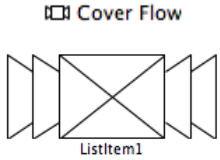
CONTROL #9

Nombre:	***List Box***
Representación en Sketcher:	
Observaciones:	Selección de múltiples elementos de una lista con muchos elementos

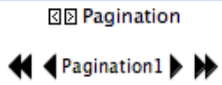
CONTROL #10

Nombre:	***Combo Box***
Representación en Sketcher:	
Observaciones:	Selección única de una lista con muchos elementos

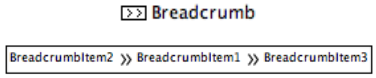
CONTROL #11

Nombre:	***Cover Flow***
Representación en Sketcher:	
Observaciones:	

CONTROL #12

Nombre:	***Pagination***
Representación en Sketcher:	
Observaciones:	

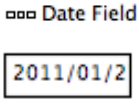
CONTROL #13

Nombre:	***Breadcrumb***
Representación en Sketcher:	
Observaciones:	

CONTROL #14

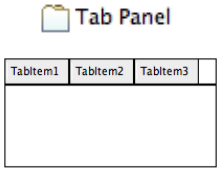
Nombre:	***Calendar***
Representación en Sketcher:	
Observaciones:	Se puede utilizar tanto para input como output de datos de tipo fecha

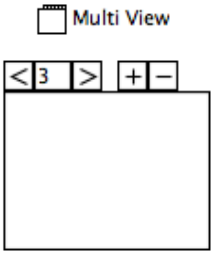
CONTROL #15

Nombre:	***DateField***
Representación en Sketcher:	
Observaciones:	Ídem al Text Box pero para datos de tipo fecha

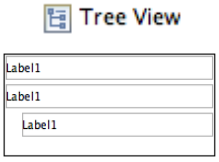
CONTROL #15	
Nombre:	***Time field***
Representación en Sketcher:	
Observaciones:	Ídem al Text Box pero para datos de tipo hora

B.2. Widgets Complejos

CONTROL #16	
Nombre:	***Tab Panel***
Representación en Sketcher:	
Observaciones:	Se suele utilizar para implementar un menú con pocos items

CONTROL #17	
Nombre:	***Multi View***
Representación en Sketcher:	
Observaciones:	

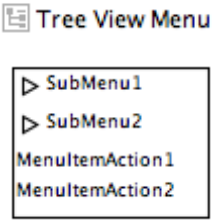
CONTROL #18

Nombre:	***Tree view***
Representación en Sketcher:	
Observaciones:	

CONTROL #19

Nombre:	***Table***
Representación en Sketcher:	
Observaciones:	

CONTROL #20

Nombre:	***TreeView Menu***
Representación en Sketcher:	
Observaciones:	Menú con muchas entradas incluso anidadas

CONTROL #21

Nombre:	***Bar Menu***
Representación en Sketcher:	
Observaciones:	Menú con pocas entradas

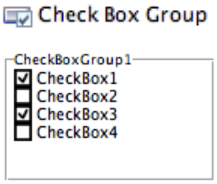
CONTROL #22

Nombre:	***ToolBar Menu***
Representación en Sketcher:	
Observaciones:	Menu con pocas entradas. Idem al BarMenu pero típicamente con imágenes en lugar de textos

CONTROL #23

Nombre:	***Radio button Group***
Representación en Sketcher:	
Observaciones:	Selección única de una lista con pocos elementos

CONTROL #23

Nombre:	***Check Box Group***
Representación en Sketcher:	
Observaciones:	Selección múltiple de una lista con pocos elementos

C

Proceso de Ejecución del Algoritmo de Transformación

En el presente anexo se expone la aplicación del algoritmo de transformación de GeMMINi mediante el cual se obtiene el modelo MOSKitt Sketcher. Éste, como decíamos en el Capítulo 7, toma como entrada el modelo de características de los dispositivos, el catálogo de patrones y el modelo UIM representativo de la interfaz abstracta. El algoritmo recorre éste último en profundidad y pre-orden recordando los nodos visitados consultando para cada AIO el catálogo de patrones proponiendo las opciones adecuadas al dispositivo (consultar Capítulo 7).

Cada paso del proceso presentado en este anexo presenta el AIO afectado, las opciones del catálogo para éste, y las elecciones tomadas

paar los dispositivos que nos ocupan (iPad y HTC Magic) mostrando los cambios que provoca esta elección en la interfaz de usuario.

C.1. Pasos del Algoritmo

Paso 1

Nodo	Catálogo
Empezar () → Nueva Ventana	N/A

iPad

Elección	Cambios en el Boceto
[**CA:**V1C1]	

HTC Magic

Elección	Cambios en el Boceto
CA:V1C1	

Paso 2

Nodo	Catálogo
	

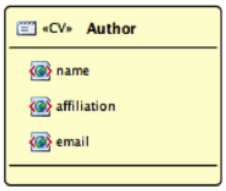
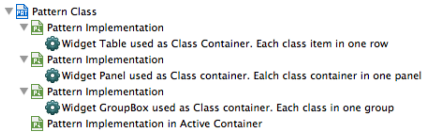
iPad

Elección	Cambios en el Boceto
En **CA**	

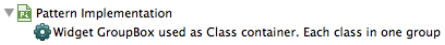
HTC Magic

Elección	Cambios en el Boceto
En **CA**	

Paso 3

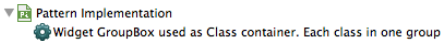
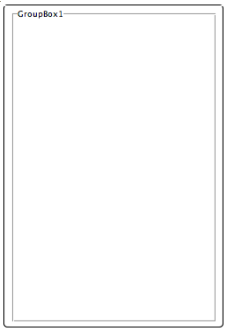
Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
	

CA: Contenedor de Registro

HTC Magic

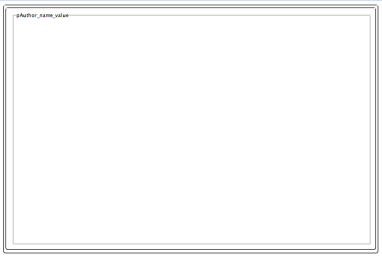
Elección	Cambios en el Boceto
	

CA: Contenedor de Registro

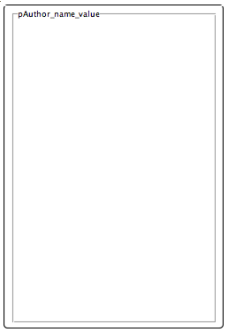
Paso 4

Nodo	Catálogo
	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

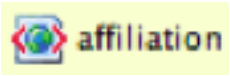
iPad

Elección	Cambios en el Boceto
Pattern Implementation Register Field without literal Widget Label used as Value	

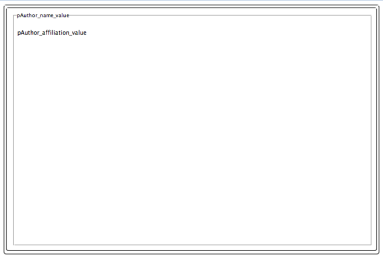
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register Field without literal Widget Label used as Value	

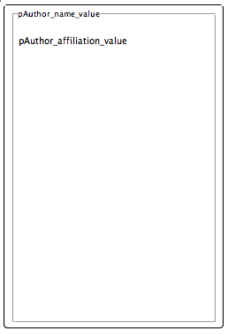
Paso 5

Nodo	Catálogo
	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register Field without literal Widget Label used as Value	

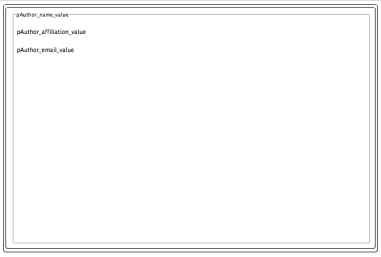
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register Field without literal Widget Label used as Value	

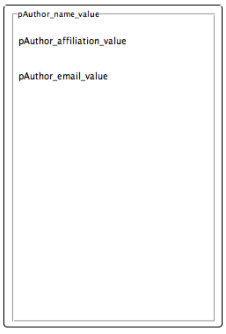
Paso 6

Nodo	Catálogo
	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register Field without literal Widget Label used as Value	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register Field without literal Widget Label used as Value	

Paso 7

Nodo	Catálogo
Contributions →	Pattern Relationship Pattern Implementation Active Container Pattern Implementation Same Window & Different Container Widget GroupBox used as New container Pattern Implementation Different Window Widget Window used as New Window Widget GroupBox used as New container

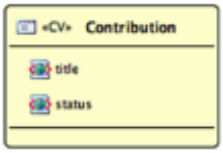
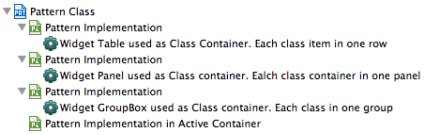
iPad

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

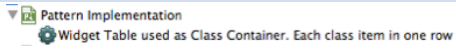
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

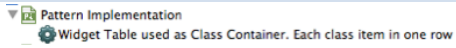
Paso 8

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
	[**CA:** Tabla] **SUBSUMIDO**

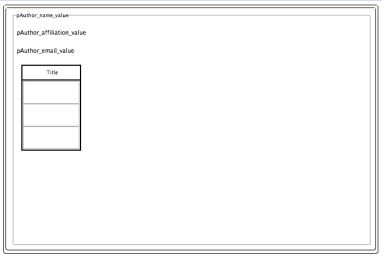
HTC Magic

Elección	Cambios en el Boceto
	[**CA:** Tabla]

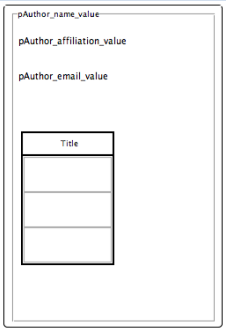
Paso 9

Nodo	Catálogo
	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

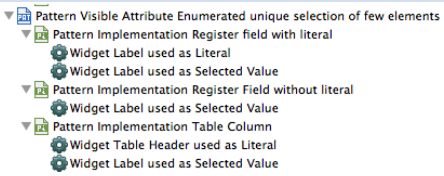
iPad

Elección	Cambios en el Boceto
Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Value	

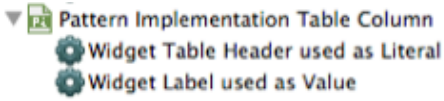
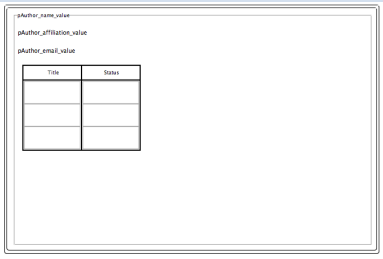
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Value	

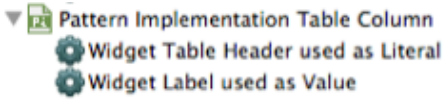
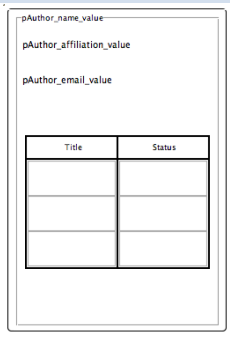
Paso 10

Nodo	Catálogo
	

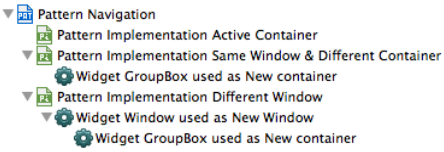
iPad

Elección	Cambios en el Boceto
	

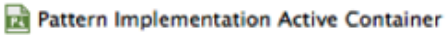
HTC Magic

Elección	Cambios en el Boceto
	

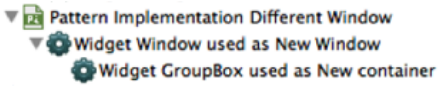
Paso 11

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
	

HTC Magic

Elección	Cambios en el Boceto
 [**CA:**V2C1]	

Paso 12

Nodo	Catálogo
 Contribution	

iPad

Elección	Cambios en el Boceto
En **CA**	

Contribution IU y *ContributionsOfAnAuthor IU* están en la misma ventana

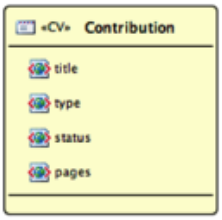
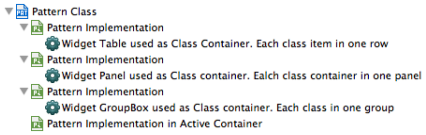
¿Coincidencia de Clases? → SI
¿Fusionar? → SI
VAttr(Contribution$_C$) $\not\subset$ VAttr(Contribution$_{COAA}$)
VAttr(Contribution$_{COAA}$) $\subset$ VAttr(Contribution$_C$)

Contribution$_{COAA}$ se subsume y *Contribution*$_C$ adopta la implementación de *Contribution*$_{COAA}$ (Tabla)

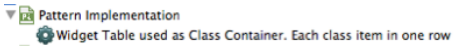
HTC Magic

Elección	Cambios en el Boceto
En **CA**	

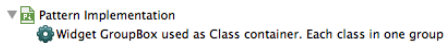
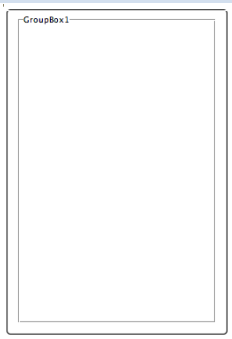
Paso 13

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
ADOPTADO 	

HTC Magic

Elección	Cambios en el Boceto
	

CA: Contenedor de Registro

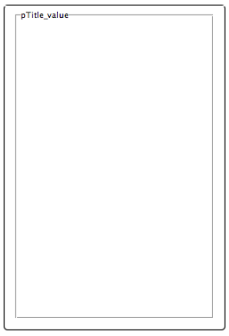
Paso 14

Nodo	Catálogo
	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

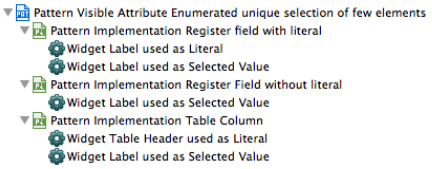
iPad

Elección	Cambios en el Boceto
SUBSUMIDO	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register Field without literal Widget Label used as Value	

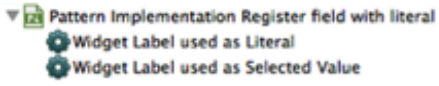
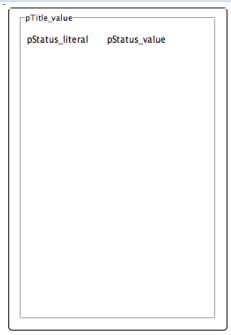
Paso 15

Nodo	Catálogo
	

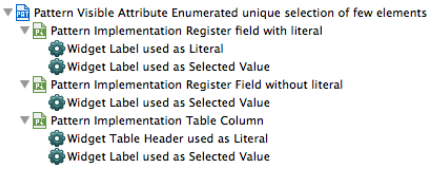
iPad

Elección	Cambios en el Boceto
SUBSUMIDO	

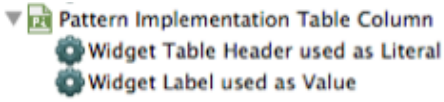
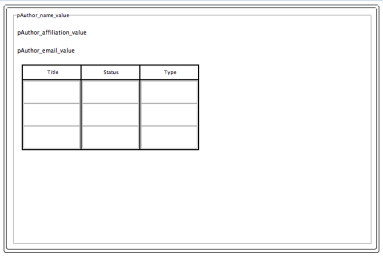
HTC Magic

Elección	Cambios en el Boceto
	

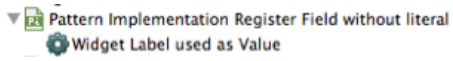
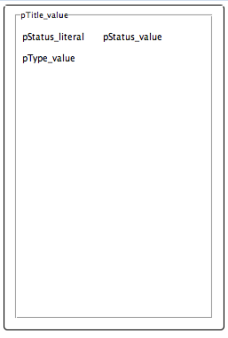
Paso 16

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
	

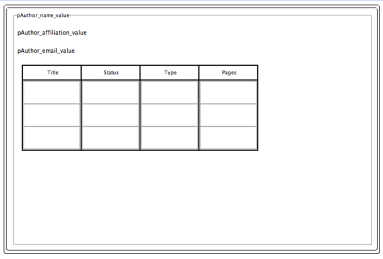
HTC Magic

Elección	Cambios en el Boceto
	

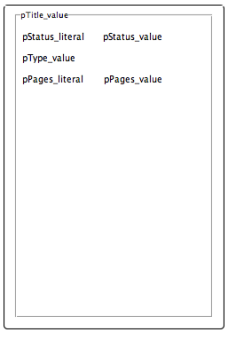
Paso 17

Nodo	Catálogo
	Pattern Visible Attribute Number Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Value Pattern Implementation Register Field without literal Widget Label used as Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Value	

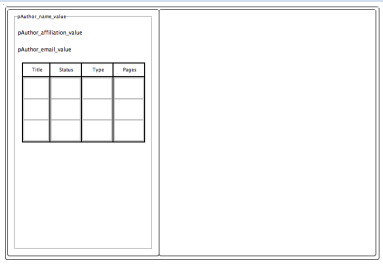
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

Paso 18

Nodo	Catálogo
Reviews →	Pattern Relationship Pattern Implementation Active Container Pattern Implementation Same Window & Different Container Widget GroupBox used as New container Pattern Implementation Different Window Widget Window used as New Window Widget GroupBox used as New container

iPad

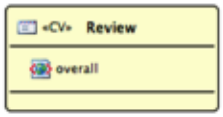
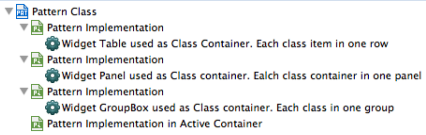
Elección	Cambios en el Boceto
Pattern Implementation Same Window & Different Container Widget GroupBox used as New container	

CA: V1C2

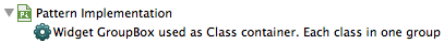
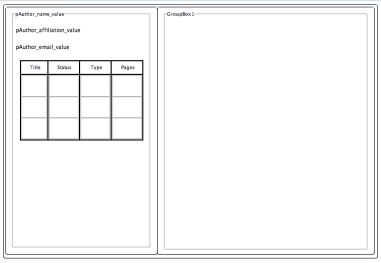
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

Paso 19

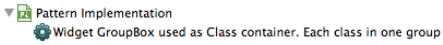
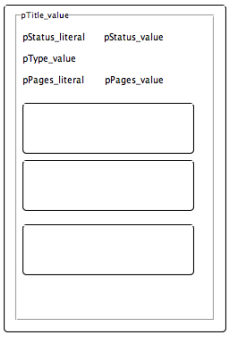
Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
	

CA: Contenedor de Registro
SUBSUMIDO

HTC Magic

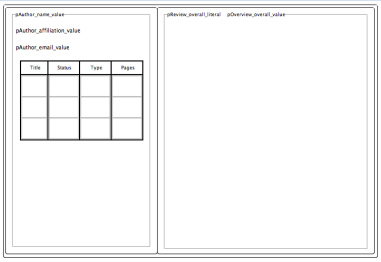
Elección	Cambios en el Boceto
	

CA: Contenedor de Registro

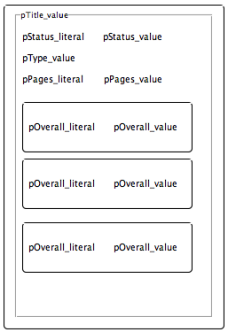
Paso 20

Nodo	Catálogo
	Pattern Visible Attribute Enumerated unique selection of few elements Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value Pattern Implementation Register Field without literal Widget Label used as Selected Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Selected Value

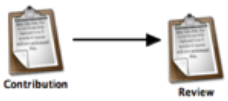
iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

Paso 21

Nodo	Catálogo
	Pattern Navigation Pattern Implementation Active Container Pattern Implementation Same Window & Different Container Widget GroupBox used as New container Pattern Implementation Different Window Widget Window used as New Window Widget GroupBox used as New container

iPad

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Different Window Widget Window used as New Window Widget GroupBox used as New container	
CA: V3C1	

Paso 22

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
En **CA**	

Esta decisión podría causar el sobrepasamiento del FMI[1]. ¿Seguro que desea continuar? → SI

Review IU y *ContributionsOfAnAuthor IU* están en la misma ventana

¿Coincidencia de Clases? → NO

Review IU y *Contribution IU* están en la misma ventana

¿Coincidencia de Clases? → SI
¿Fusionar? → SI
VAttr(Review$_R$) $\not\subset$ VAttr(Review$_C$)
VAttr(Review$_C$) $\subset$ VAttr(Review$_R$)

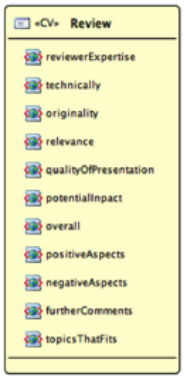
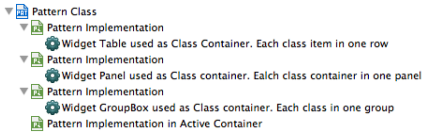
Review$_C$ se subsume y *Review*$_R$ adopta la implementación de *Review*$_C$ (Lista de Registros)

HTC Magic

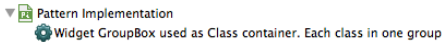
Elección	Cambios en el Boceto
En **CA**	

[1]Factor de máxima cantidad de información por unidad de interacción

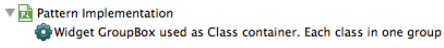
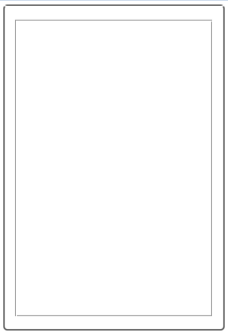
Paso 23

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
ADOPTADO 	

HTC Magic

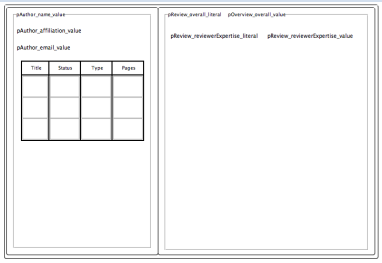
Elección	Cambios en el Boceto
	

CA: Contenedor de Registro

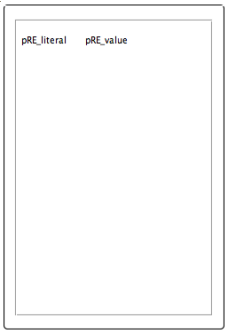
Paso 24

Nodo	Catálogo
reviewerExpertise	Pattern Visible Attribute Enumerated unique selection of few elements Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value Pattern Implementation Register Field without literal Widget Label used as Selected Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Selected Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

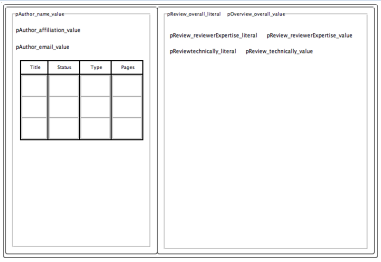
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

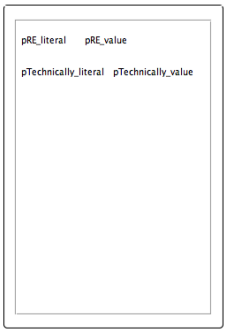
Paso 25

Nodo	Catálogo
technically	Pattern Visible Attribute Enumerated unique selection of few elements Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value Pattern Implementation Register Field without literal Widget Label used as Selected Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Selected Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

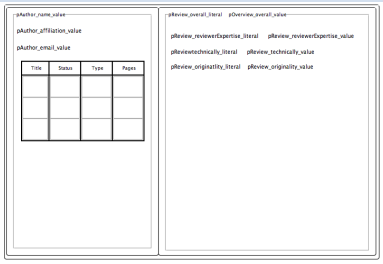
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

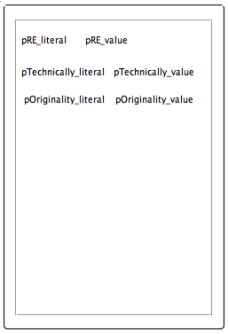
Paso 26

Nodo	Catálogo
originality	Pattern Visible Attribute Enumerated unique selection of few elements Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value Pattern Implementation Register Field without literal Widget Label used as Selected Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Selected Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

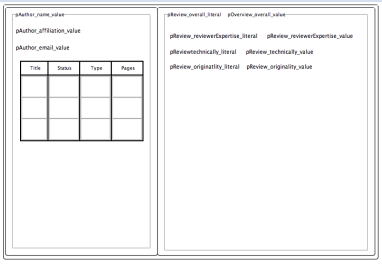
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

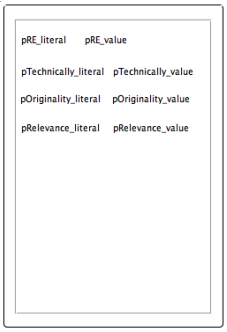
Paso 27

Nodo	Catálogo
relevance	Pattern Visible Attribute Enumerated unique selection of few elements Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value Pattern Implementation Register Field without literal Widget Label used as Selected Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Selected Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

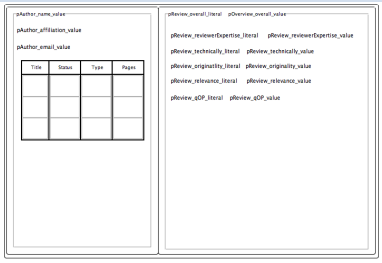
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

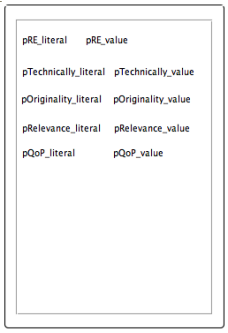
Paso 28

Nodo	Catálogo
qualityOfPresentation	Pattern Visible Attribute Enumerated unique selection of few elements Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value Pattern Implementation Register Field without literal Widget Label used as Selected Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Selected Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

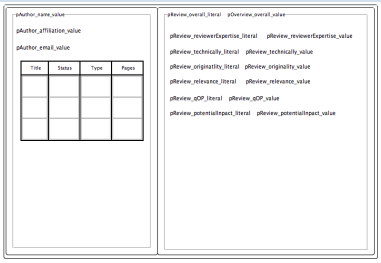
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

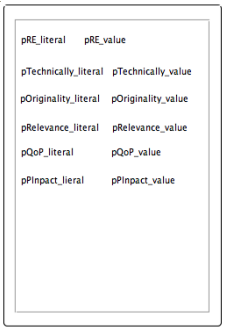
Paso 29

Nodo	Catálogo
potentialInpact	Pattern Visible Attribute Enumerated unique selection of few elements Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value Pattern Implementation Register Field without literal Widget Label used as Selected Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Selected Value

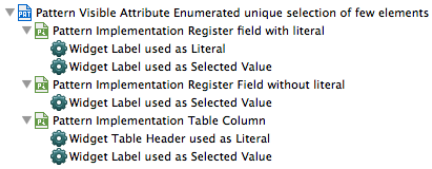
iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

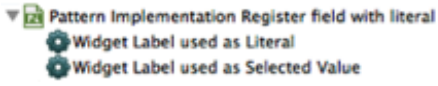
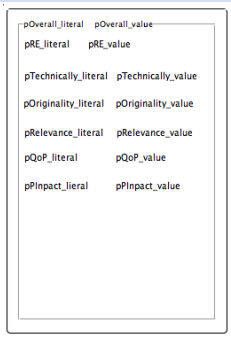
Paso 30

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
SUBSUMIDO	

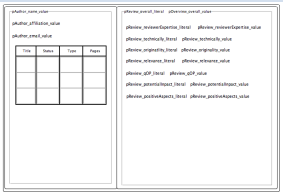
HTC Magic

Elección	Cambios en el Boceto
	

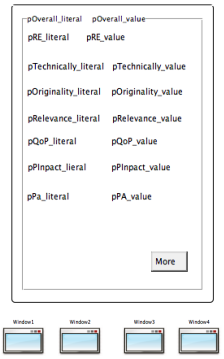
Paso 31

Nodo	Catálogo
positiveAspects	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

FMI alcanzado. ¿Desea continuar en la misma ventana o crear una ventana nueva?
→ CREAR

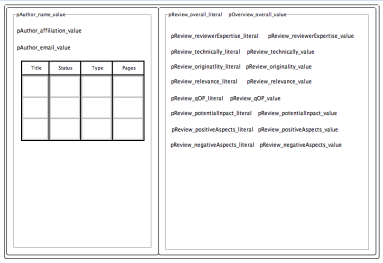
`Boton more en` **CA**

`Nueva Ventanta→`**CA:**`V4C1↪Contenedor de Registros`

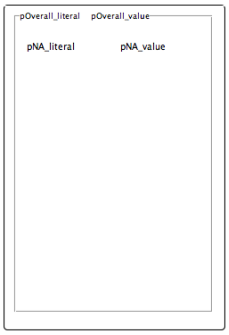
Paso 32

Nodo	Catálogo
negativeAspects	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

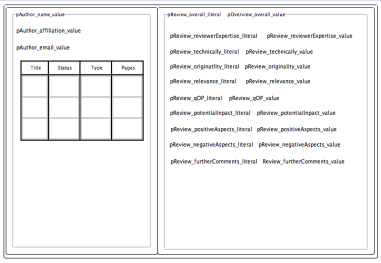
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

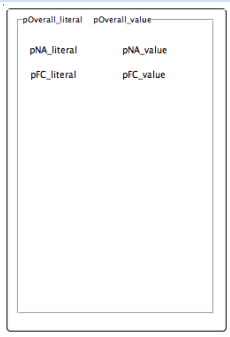
Paso 33

Nodo	Catálogo
furtherComments	Pattern Visible Attribute text Pattern Implementation Register field with literal Widget Ussage Literal -> Label Widget Ussage Value -> Label Pattern Implementation Register Field without literal Widget Ussage Value -> Label Pattern Implementation Table Column Widget Ussage Literal -> Table Header Widget Ussage Value -> Label

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

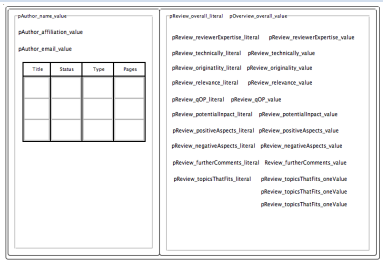
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

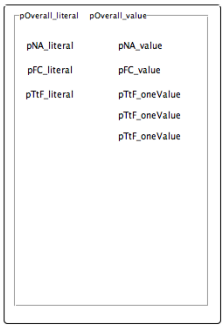
Paso 34

Nodo	Catálogo
topicsThatFits	Pattern Visible Attribute Enumerated multiple selection Pattern Implementation Register field with literal Widget Label used as List Header Widget List used as Selected Value Widget Label used as List Value Pattern Implementation Register Field without literal Widget List used as Selected Value Widget Label used as List Value Pattern Implementation Table Column Widget Table Header used as Literal Widget List used as Selected Value Widget Label used as List Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as List Header Widget List used as Selected Value Widget Label used as List Value	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as List Header Widget List used as Selected Value Widget Label used as List Value	

Paso 35

Nodo	Catálogo
Reviewer →	Pattern Relationship Pattern Implementation Active Container Pattern Implementation Same Window & Different Container Widget GroupBox used as New container Pattern Implementation Different Window Widget Window used as New Window Widget GroupBox used as New container

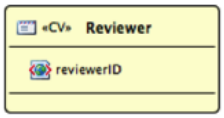
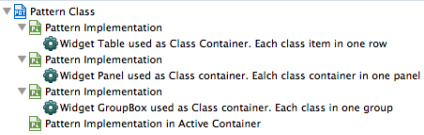
iPad

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

Paso 36

Nodo	Catálogo
	

iPad

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

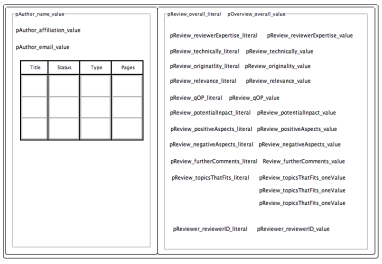
HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Active Container	

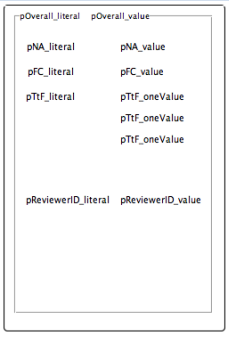
Paso 37

Nodo	Catálogo
reviewerID	Pattern Visible Attribute Number Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Value Pattern Implementation Register Field without literal Widget Label used as Value Pattern Implementation Table Column Widget Table Header used as Literal Widget Label used as Value

iPad

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

HTC Magic

Elección	Cambios en el Boceto
Pattern Implementation Register field with literal Widget Label used as Literal Widget Label used as Selected Value	

Epílogo

Al principio de este documento hay dos citas. Ambas son citas importantes para mí porque están enmarcadas en textos que me han ayudado en momentos difíciles de mi vida, cuando todo parecía desmoronarse.

La primera de ellas es de Konstantinos Kavafis, poeta griego del siglo XIX. En concreto la cita pertenece a la primera estrofa del poema *Itaca*, famoso por haber sido musicado por Lluís Llach. Itaca, habla de la utopía, las ambiciones; esos sueños que hacen que enfrentemos todas las adversidades de la vida y también todas sus maravillas para alcanzarlos. Si los alcanzamos, pueden desilusionarnos porque no corresponden a lo que habíamos soñado, pero siempre valdrá la pena, porque gracias a esos sueños hemos marchado siempre hacia adelante, enriqueciéndonos con cada experiencia vivida. No se puede vivir sin ambiciones y cada persona tiene las suyas. Cada cual quiere llegar a su *Itaca*. Cada vez que vivimos algo nuevo o diferente deberíamos encontrar otra visión acerca de lo que hemos vivido antes y lo que viviremos después de cada experiencia vivida.

La segunda cita es un extracto del célebre discurso de Steve Jobs, en la ceremonia de graduación de la Universidad de Stanford en 2005. En el Jobs nos habla sobre la fe en uno mismo, sobre la necesidad de confiar en nuestras posibilidades ya que serán ellas las que nos conduzcan al éxito. Si confiamos en nosotros mismos, si nos creemos capacitados para seguir adelante, siempre podremos superar esos obstáculos que aparecen

en el camino (que siempre aparecen). Jobs termina esta conferencia con una frase de una revista: ***Stay hungry. Stay foolish.***

El trabajo en equipo es como un castillo de naipes. Debemos apoyarnos los unos en los otros. Debes siempre dar lo mejor de ti para el equipo y confiar. Confiar en que el resto del equipo hará lo mismo para juntos llegar al éxito, sabiendo que puedes contar con ellos en los momentos de flaqueza, de duda, o de confusión. Si una sola pieza del equipo falla, puede que la estructura se venga abajo, pero eso sólo pasará si no existe la suficiente confianza en los otros, si no creemos en nuestras posibilidades, o si nos falta ambición o afán de superación.

El equipo que me rodea está formado por mucha gente. Se que puedo confiar en ellos sin la menor duda. Se que no me defraudarán o que como mínimo intentarán dar lo mejor de si para no fallarme, y al tiempo, eso me obliga siempre a dar lo mejor de mi para tampoco fallarles a ellos. Y así, juntos, seguiremos avanzando hasta lograr lo que nos propongamos.

Gracias a todos por vuestra ayuda.

Nacho Mansanet

València. Julio de 2011

www.pros.upv.es

Centro de Investigación en Métodos
de Producción de Software
Universitat Politècnica de València
Camí de Vera s/n, Edifici 1F, Dept. DSIC
46022 - València
Spain
Tel: (+34) 963 877 007 (Ext. 83530)
Fax: (+34) 963 877 359

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

PROS

Centro de Investigación en Métodos de Producción de Software