Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla

Propuestas para la reutilización en el Desarrollo de Interfaces de Usuario Basado en Modelos

Antonio Luis Delgado González

Dirigida por los doctores
D. José A. Troyano y D. Antonio J. Estepa
Universidad de Sevilla

Memoria de Tesis Doctoral

Esta tesis está dedicada a mi familia:

a mis hijas Mª Ángeles, Trinidad y Mercedes,
a mi esposa Mª Ángeles,
a mis hermanas Ana, Mª Jesús y Rocío
y a mis padres Antonio y Trinidad.

Sin la ayuda, dedicación y amor de todos ellos
nunca habría llegado hasta aquí
y este trabajo no habría sido posible.

Agradecimientos

A mis tutores de tesis:

A José Antonio Troyano. Por tus indicaciones siempre certeras y por tu ayuda en la revisión y corrección de documentos. Gracias por allanarme un camino que para mi no era fácil.

A Antonio Estepa, además de tutor, amigo. Gracias por tu ayuda indispensable en la redacción de artículos, por tu paciencia, por tu talante y buen humor. Sin tu apoyo, seguramente habría abandonado esta empresa.

Gracias a los dos.

Resumen

Esta tesis versa sobre la reutilización en el desarrollo de *interfaces de usuario* basado en modelos.

La baja reusabilidad de las especificaciones de *Modelos de la IU* ha sido identificada como un posible obstáculo para la adopción del desarrollo de *interfaces de usuario* basado en modelos por parte de la industria. El objetivo de esta investigación es aportar conocimiento sobre la reutilización en este ámbito para mejorar la situación actual de esta tecnología.

En este trabajo se propone el uso de ciertas técnicas de reutilización empleadas habitualmente en la ingeniería del software para aplicarlas al desarrollo de *interfaces de usuario* basado en modelos y éstas son implementadas sobre un entorno de desarrollo concreto. Este entorno de desarrollo de *interfaces de usuario* basado en modelos con características de reutilización potenciadas es utilizado para construir varias aplicaciones que poseen aspectos comunes.

A través de este caso práctico y siguiendo un método empírico, se analiza el impacto que las técnicas de reutilización han tenido en el desarrollo de las *interfaces de usuario*. Los resultados son importantes: además del alto grado de reutilización general (entorno al 46,2 %) se alcanzan picos de reutilización del 56,9 % para algunos casos particulares y se obtiene un ahorro en el tamaño de las especificaciones del 71 %. Aunque los resultados obtenidos están condicionados por el contexto (entorno de desarrollo, proyectos desarrollados, etc.) se extraen algunas conclusiones aplicables a la generalidad del desarrollo de *interfaces de usuario* basado en modelos.

Índice general

Índice de listados **1**

1. Introducción **1**
1.1. Hipótesis y metodología 2
1.2. Resumen de las aportaciones de la tesis 3
1.3. Estructura del documento 4

2. Estado del arte **5**
2.1. Introducción al MB-UID 5
2.1.1. Modelos de la interfaz de usuario 6
2.1.2. Lenguajes de descripción de la interfaz de usuario 8
2.1.3. Evolución histórica 10
2.1.4. Framework de Referencia Cameleon 12
2.2. Reutilización 16
2.2.1. Soporte para la reutilización en los UIDLs 22
2.2.2. Desarrollo de interfaces de usuario multi-dispositivo basado en modelos 25
2.2.3. Métodos de reutilización basados en patrones 26
2.3. Conclusiones 28

3. Técnicas de reutilización en el *MB-UID* **31**
3.1. Técnicas de reutilización en especificaciones 32
3.1.1. Técnicas soportadas por el lenguaje de especificación 32
3.1.2. Inclusión (I) 34
3.1.3. Paquetes (P) 34
3.1.4. Bibliotecas de reutilización (B) 35
3.2. Técnicas de reutilización en repositorios de modelos 36
3.2.1. Paquetes (P) 36
3.2.2. Repositorios centralizados (C) 36
3.2.3. Repositorios federados (F) 37
3.3. Factores a considerar en la selección y aplicación de las técnicas propuestas 39
3.3.1. Arquitectura del MB-UIDE 39
3.3.2. Estructura de los recursos reutilizables 41
3.3.3. Dominio de reutilización y otros aspectos 43
3.4. Conclusiones 45

4. WAINE **47**
4.1. Introducción a WAINE . . . 47
4.1.1. Modelos de la interfaz de usuario . . . 48
4.1.2. Arquitectura . . . 51
4.1.3. Lenguaje de descripción de la interfaz de usuario . . . 53
4.1.4. Proceso de desarrollo y generación de la interfaz de usuario . . . 55
4.1.5. Herramientas de desarrollo . . . 63
4.2. Técnicas de reutilización soportadas por WAINE . . . 64
4.2.1. Subespecificación y adaptación (S,A) . . . 64
4.2.2. Herencia (H) . . . 67
4.2.3. Inclusión y Bibliotecas de reutilización (I,B) . . . 68
4.2.4. Paquetes (P) . . . 71
4.2.5. Repositorios compartidos: centralización y federación (C,F) . . . 73
4.3. Conclusiones . . . 75

5. Experiencia de reutilización en el MB-UID **77**
5.1. Caso de estudio . . . 77
5.2. Experiencia cualitativa en la reutilización . . . 81
5.2.1. Subespecificación y adaptación (S,A) . . . 81
5.2.2. Herencia (H) . . . 83
5.2.3. XInclude y Bibliotecas (I,B) . . . 83
5.2.4. Paquetes (P) . . . 84
5.2.5. Repositorios compartidos (C,F) . . . 85
5.3. Resultados cuantitativos de la reutilización . . . 90
5.3.1. Recursos de la interfaz de usuario y cantidad de reutilización . . . 90
5.3.2. Reutilización en el modelo de presentación . . . 92
5.4. Conclusiones . . . 99

6. Conclusiones y trabajos futuros **101**
6.1. Lecciones aprendidas . . . 102
6.1.1. Sobre la reusabilidad de los modelos y la subespecicación . . . 102
6.1.2. Sobre el tamaño y adaptabilidad de los recursos . . . 103
6.1.3. Sobre las técnicas empleadas . . . 103
6.2. Líneas de investigación abiertas . . . 104
6.3. Publicaciones . . . 104

A. DTD del lenguaje ASL **117**

B. Desarrollo de una aplicación **125**

C. Componente ASL **135**

D. Líneas de código por aplicación **137**

E. Obtención de los datos de la tabla 5.2 **141**

Índice de figuras

2.1. Desarrollo de la interfaz de usuario basado en modelos 6
2.2. UIDLs: modelos soportados, año de aparición e impacto 8
2.3. Framework de referencia Cameleon 14
2.4. Arquitecturas para la generación de la interfaz de usuario 15
2.5. Comparativa de propuestas en el framework de referencia Cameleon . . 16
2.6. Componente XICL 23

3.1. Nueva ventana obtenida por adaptación 33
3.2. Arquitectura centralizada para compartir objetos de la interfaz de usuario 37
3.3. Arquitectura federada para compartir objetos de la interfaz de usuario 38
3.4. Recursos reutilizables de la interfaz de usuario 42

4.1. Diagrama informal del tipo de aplicación soportada por WAINE 48
4.2. Modelos y lenguajes de WAINE relacionados con el framework Cameleon 49
4.3. Modelos de la interfaz de usuario en WAINE 50
4.4. Arquitectura de WAINE 51
4.5. Modelo conceptual del repositorio de la interfaz de usuario 52
4.6. Arquitectura del run-time 54
4.7. Proceso de desarrollo de una aplicación con WAINE 56
4.8. Proceso de desarrollo de la interfaz de usuario 57
4.9. Ejemplo de anotación del diagrama entidad-relación 58
4.10. Generación de la interfaz de usuario final 61
4.11. Interfaz de usuario de la aplicación de ejemplo 63
4.12. Contexto de uso de las técnicas de reutilización implementadas 64
4.13. Subespecificación y adaptación en ASL. 65
4.14. Ejemplo de adaptación en ASL 66
4.15. Relación de herencia de algunos widgets 67
4.16. Relación de herencia entre grupos y usuarios 69
4.17. Validación, inclusión y transformación de una especificación ASL 71
4.18. Contenido de un fichero wpk 72

5.1. Aplicación de gestión de reservas 78
5.2. Aplicación de planificación de contenidos para los paneles de información 79
5.3. Infraestructura para las aplicaciones desarrolladas 81
5.4. Ejemplo de reutilización por adaptación 82
5.5. Ejemplo de formulario compartido por dos aplicaciones 85

5.6. Presentación centralizada - Dominio centralizado 86
5.7. Presentación centralizada - Dominio distribuido 87
5.8. Modelo de presentación federado 88
5.9. Unidad de interacción federada 89
5.10. Frecuencia por tipo de recurso según su tamaño en líneas de código . . 94
5.11. Sobrecarga de adaptación por tamaño del formulario 95

6.1. Fragmentación en un modelo de la interfaz de usuario 102

B.1. Diagrama informal de la aplicación 125
B.2. Bocetos de las interfaces de usuario 126
B.3. Diagrama entidad-relación y código SQL de la aplicación 127
B.4. Diagrama entidad-relación anotado 128
B.5. Formulario para el secretario. 133

Índice de tablas

2.1. Evolución histórica de los MB-UIDEs 11
2.2. Elementos potencialmente reutilizables 17
2.3. Dimensiones de la reutilización 18
2.4. Propuestas para la reutilización en el MB-UID 29

3.1. Resolución de la reutilización por arquitectura y técnica 39
3.2. Aplicación de las técnicas según el tipo de recurso 43
3.3. Técnicas por arquitectura y composición del recurso 44
3.4. Ámbito de aplicación más apropiado para cada técnica 44

4.1. Comparativa de UIDLs: modelos soportados y etiquetas empleadas . . 55
4.2. Tabla Rol/funcionalidad 57
4.3. Aspectos de presentación concreta tratados en ASL 62
4.4. Técnicas de reutilización implementadas en WAINE 76

5.1. Lista de aplicaciones desarrolladas 80
5.2. Recursos definidos y reutilizados por aplicación y modelo 91
5.3. Composición del modelo de presentación por aplicación 92
5.4. Tamaño de los recursos reutilizados en el modelo de presentación 94
5.5. Sobrecarga de adaptación por proyecto 95
5.6. Nivel de reutilización del modelo de presentación de cada aplicación . . 97
5.7. Beneficio de la reutilización por aplicación 98

B.1. Tabla rol/funcionalidad de la aplicación de ejemplo 127

D.1. Líneas de código por aplicación 137

F.1. Relación de actuaciones de mantenimiento realizadas 146

Índice de Listados

4.1. Extracto de un documento ASL 58
4.2. Conf. widgets 61
4.3. Conf. de colores y estilos 61
4.4. Widget específico 62
4.5. Estilo para un formulario 62
4.6. Ejemplo de nuevo widget creado por adaptación 67
4.7. Ejemplo de nuevo widget definido por herencia 68
4.8. DTD para las etiquetas definidas por el estándar XInclude 69
4.9. Modularizando una especificación con XInclude 70
4.10. Archivo meta.xml del paquete Percon 72
4.11. Contenido del archivo dsource.cfg 74
A.1. Sintaxis del lenguaje ASL en formato DTD 117
B.1. Código ASL de la aplicación 128
B.2. Configuración del origen de datos del repositorio de la interfaz de usuario 132
B.3. Código del evento event_BeforeDelete 132
C.1. Código de un componente ASL 135
D.1. Cuenta de las líneas de código de archivos SQL 137
D.2. Cuenta de las líneas de código de los documentos ASL 138
D.3. Script cpmstat 138
E.1. Consulta para obtener tablas y vistas del repositorio de la interfaz de usuario 141
E.2. Consulta para obtener formularios y su origen de datos (tablas o vistas) 141
E.3. Consulta para contabilizar grupos de una aplicación 142
E.4. Consulta para contabilizar menus de una aplicación 142
E.5. Consulta para contar referencias a contenedores y formularios 142
E.6. Script para contabilizar referencias jerárquicas 143

Capítulo 1

Introducción

En el ámbito industrial y empresarial existe una necesidad continua de incrementar la competitividad para lograr un mejor posicionamiento en el mercado. La productividad es un factor clave de la competitividad y puede ser definida como la relación entre los resultados y los recursos utilizados para obtenerlos [97, 85]. La industria del software está obligada a mejorar su productividad constantemente [52] y para conseguirlo emplea diversos métodos que permiten reducir el tiempo empleado en las distintas fases de construcción de un producto software (análisis, diseño e implementación). Estos métodos tratan, con mayor o menor éxito, de reutilizar recursos existentes [49, 57] o de generar automáticamente productos completos [41, 53, 64] (o al menos partes de los mismos). Hoy en día, gracias a las aportaciones en distintos campos de la informática, podemos considerar recursos reutilizables requisitos, diseños, modelos o conocimiento [55, 109, 73] y se generan de forma automática documentación o código entre otros productos [33, 86].

Como es conocido desde hace décadas, la construcción de la *interfaz de usuario* (IU) de un proyecto software representa entorno al 48 % del código de la aplicación y cerca del 50 % del tiempo dedicado al desarrollo [82]. Posiblemente en la actualidad la situación sea aún peor por el número de tecnologías involucradas en el desarrollo de la *interfaz de usuario* y la eclosión de dispositivos y modalidades de uso (p.ej. gráficas, táctiles o vocales). Es lógico, por lo tanto, que diversas aportaciones en el ámbito del desarrollo de la *interfaz de usuario* se hayan orientado a conseguir que el desarrollo sea más productivo [81, 72].

De entre estas técnicas, el desarrollo de *interfaces de usuario* basado en modelos (*Model-based User Interface Development*, *MB-UID*) [95, 42, 70, 69] describe la *interfaz de usuario* usando un conjunto de modelos abstractos que constituyen la base del proceso de desarrollo. Al igual que en el paradigma de la ingeniería dirigida por modelos (*Model-Driven Engineering*, MDE [112]), en el *MB-UID* los modelos son utilizados para la generación (semi)automática de diversos productos *software*, entre ellos el código de la *interfaz de usuario*. Esto redunda en un proceso de desarrollo de la *interfaz de usuario* más productivo, reduciendo el esfuerzo necesario para su desarrollo y mejorando la calidad del código generado [95, 125, 69].

Durante las pasadas décadas han proliferado los entornos de desarrollo de la interfaz de usuario basados en modelos [95, 70] (MB-UID *environments*, *MB-UIDEs*), dando

lugar a diferentes herramientas y lenguajes de especificación [45, 70]. Sin embargo, a pesar del número de propuestas disponibles, ningún *MB-UIDE* ha experimentado una adopción amplia por parte de la industria del software [119, 74]. Algunos autores [95, 4] argumentan que la pobre reusabilidad de las especificaciones de *Modelos de la IU* puede ser un factor que explique parcialmente esta falta de interés por parte de la industria. Más recientemente, Meixner *et al.* [70] han apoyado esta idea, apuntando también a la falta de armonización entre la MDE y el *MB-UID*, a la ausencia de referencias de aplicación con éxito en el mundo industrial y a la escasez de casos de estudio, como retos que deben ser afrontados por los sistemas de *MB-UID* en el futuro. Entre esos aspectos, esta tesis se enfoca hacia la reutilización.

La comunidad del *MB-UID* ha realizado avances en la reutilización de la *interfaz de usuario*. Algunos lenguajes de especificación han contemplado mecanismos para la reutilización [50, 18, 19, 2]; existen *MB-UIDEs* que generan código para varias plataformas partiendo de una única especificación [7, 100, 125] y también pueden encontrarse propuestas para la reutilización basadas en patrones de diseño [4, 29, 113]. Sin embargo, no hay literatura específicamente orientada a analizar la experiencia de reutilizar elementos de la *interfaz de usuario*, ya sea de forma interna en un proyecto o entre proyectos distintos. Los métodos y problemas relacionados con la reutilización de los componentes de la *interfaz de usuario* y su coste/beneficio asociado, no han sido tratados por la comunidad del *MB-UID*. Una posible razón de ello puede ser la complejidad del asunto que implica varias cuestiones relacionadas como: (a) qué fragmentos de la *interfaz de usuario* o qué modelos pueden ser objeto de reutilización, (b) qué técnicas son apropiadas para la reutilización, (c) cómo evaluar los beneficios de la reutilización en la *interfaz de usuario*. La respuesta a esas cuestiones se vuelve incluso más complicada en la situación actual, ya que como se pone de manifiesto en [70], ciertos estándares emergentes del W3C sobre determinados *Modelos de la IU* [91, 122] no han sido aún ampliamente adoptados y no hay casos de estudio suficientes para permitir analizar los potenciales beneficios de la reutilización con cierto rigor.

1.1. Hipótesis y metodología

Teniendo en consideración la situación actual en el desarrollo de la *interfaz de usuario* y la necesidad constante de incrementar la productividad y la competitividad, esta investigación se ha centrado en la reutilización en el *MB-UID*. Las hipótesis de este trabajo son las siguientes:

- Algunas de las técnicas de reutilización usadas frecuentemente en la ingeniería del software pueden ser aplicadas con éxito en el *MB-UID*. Recordemos que la pobre reusabilidad de las especificaciones de *Modelos de la IU* es uno de los factores que puede explicar la falta de adopción del *MB-UID* por parte de la industria, como ha sido expuesto por algunos autores [95, 4]. Existe por lo tanto un interés por parte de la comunidad del *MB-UID* en incrementar el conocimiento sobre cómo reutilizar los elementos de los *Modelos de la IU* y sobre cuáles son las implicaciones de las técnicas empleadas.

- Las técnicas de reutilización anteriores aplicadas al *MB-UID* reducirán considerablemente el esfuerzo del desarrollo de la *interfaz de usuario*. En general, la reutilización incrementa la productividad en el desarrollo y la calidad de los productos generados como ha quedado de manifiesto en varios estudios [73]. Estos beneficios deberían estar presentes también en el contexto del *MB-UID*.

- Pensamos también que las técnicas propuestas pueden aportar nuevas vías para la reutilización de los elementos de la *interfaz de usuario* a distintos niveles de abstracción. Las técnicas permiten reutilizar, desde elementos simples y concretos, a grandes entidades abstractas de una interfaz ofreciendo a los desarrolladores nuevas posibilidades a la hora de construir las *interfaces de usuario*.

En esta tesis se emplea el método empírico para demostrar las hipótesis de partida. Se desarrolla un nuevo *MB-UIDE*, WAINE [20, 21], que implementa las técnicas de reutilización propuestas. Este *MB-UIDE* potenciado para la reutilización será empleado en el desarrollo de varias aplicaciones relacionadas. Finalmente se mide el impacto que las técnicas de reutilización han tenido en el desarrollo de estas aplicaciones. Aunque los resultado obtenidos están condicionados por el entorno (*MB-UIDE*, proyectos, etc.) se extrerán algunas ideas aplicables a la generalidad del *MB-UID*.

1.2. Resumen de las aportaciones de la tesis

Son resultados de la investigación descrita en esta tesis las siguientes aportaciones:

1. Una relación de técnicas de reutilización potencialmente aplicables al *MB-UID*. La aplicación de algunas de las técnicas propuestas es inédita y nunca habían sido empleadas en este ámbito [22].

2. Un *MB-UIDE* con *Lenguaje de Descripción de la IU* propio [20]. Se trata de un sistema que pretende simplificar y acelerar el desarrollo de la *interfaz de usuario* a estudiantes y programadores sin experiencia. El *MB-UIDE* implementa las técnicas de reutilización propuestas en la tesis. Esta implementación sirve como validación a la aplicabilidad de las técnicas de reutilización en este ámbito.

3. Un caso de estudio en el que el *MB-UID* es aplicado con éxito al desarrollo de varias aplicaciones actualmente en producción y que son usadas por cientos de usuarios. Al tratarse de aplicaciones que manejan aspectos comunes, diversas técnicas de reutilización han sido puestas en práctica en la construcción de las mismas. Tengamos en cuenta que en [70] se pone de manifiesto que existen pocas referencias a sistemas en funcionamiento y que hay una carencia importante de casos de estudio. En nuestro caso práctico, además, se valoran los beneficios obtenidos midiendo el impacto de la reutilización en el desarrollo de las *interfaces de usuario* de las distintas aplicaciones. También se analizan consecuencias no cuantitativas de la reutilización, estudiando las distintas posibilidades que ha generado la aplicación de las técnicas propuestas en el desarrollo de la *interfaz de usuario*.

4. Un análisis cuantitativo de la reutilización en la *interfaz de usuario* en sistemas de *MB-UID*. Hasta el momento no se ha publicado ningún análisis similar al presentado en este estudio. Se han aplicado métricas empleadas de forma usual a la reutilización en la ingeniería del software a los *Modelos de la IU*. El análisis presentado en este documento puede servir como guía o modelo para futuros trabajos de reutilización en la *interfaz de usuario*.

Es necesario tener en cuenta que no todas las conclusiones de esta tesis pueden ser extendidas a la generalidad del *MB-UID*. Sin embargo, sirven al propósito de ilustrar los potenciales beneficios de la reutilización de los elementos de la *interfaz de usuario* en ámbitos internos (en el mismo proyecto) y externos (entre distintos proyectos), aportando indicios para la construcción de futuros *Lenguajes de Descripción de la IU* y *MB-UIDEs* con características de reutilización potenciadas.

1.3. Estructura del documento

Este apartado presenta la estructura del documento con el fin de facilitar al lector la comprensión de este trabajo. La tesis sigue la siguiente estructura.

- Capítulo 1. Estado del arte. Se tratan en este apartado las principales propuestas sobre el *MB-UID*, se comenta su evolución histórica y se citan algunos de los sistemas más relevantes. Posteriormente, se enumeran las propuestas relacionadas más directamente con la reutilización de la *interfaz de usuario* en el *MB-UID*.

- Capítulo 2. Presentación de una colección de técnicas de reutilización potencialmente aplicables al *MB-UID*. Para cada técnica se nombran experiencias de uso en otros campos y se propone una aplicación determinada en el *MB-UID*.

- Capítulo 3. Explicación detallada de WAINE, el *MB-UIDE* empleado en las experiencias de esta tesis. Se describe su arquitectura, *Lenguaje de Descripción de la IU*, proceso de desarrollo, etc. Posteriormente se presenta la implementación de las técnicas de reutilización propuestas en el capítulo anterior sobre este sistema.

- Capítulo 4. Descripción de un caso real en el que nuestro *MB-UIDE* y las técnicas de reutilización propuestas han sido aplicados al desarrollo de seis aplicaciones. En primer lugar se transmite la experiencia en la aplicación de cada una de las técnicas para posteriormente presentar un análisis cuantitativo de la reutilización.

- Capítulo 5. Para finalizar el documento se enumeran las conclusiones de la tesis, se exponen algunas ideas que pueden ser aplicables a la generalidad del *MB-UID* y se proponen ciertas líneas de investigación como continuación de este trabajo.

Capítulo 2

Estado del arte

En este capítulo se describen los trabajos que guardan relación con los aspectos tratados en esta tesis. Se ha considerado dividir el capítulo en dos secciones. La primera está dedicada al *MB-UID* y en ella se presenta una panorámica sobre esta tecnología. En la segunda nos centramos en la reutilización y más específicamente en aquellas aportaciones dentro del *MB-UID* relacionadas con la reutilización de *Modelos de la IU*.

2.1. Introducción al MB-UID

El *MB-UID* ofrece un entorno en el que los desarrolladores pueden diseñar e implementar *interfaces de usuario* de una forma profesional, consistente y sistemática [95, 42, 70, 69]. Está basado en la idea de que la *interfaz de usuario* puede ser completamente definida por un conjunto de modelos declarativos, cada uno de los cuales trata facetas particulares de la *interfaz de usuario* como las tareas, la presentación, etc. La especificación de cada modelo consiste en una descripción abstracta de los aspectos pertenecientes a su dominio a través de un Lenguaje Descriptivo. Estas especificaciones guían el ciclo de vida del desarrollo de la *interfaz de usuario* y son la base para la generación automática de diversos productos software, entre ellos, el código fuente de la *interfaz de usuario*. En la figura 2.1 se puede apreciar como en el *MB-UID* los *Modelos de la IU* son el núcleo del desarrollo de la *interfaz de usuario*. Los desarrolladores trabajan sobre esos modelos realizando distintas transformaciones directamente con un editor o bien empleando herramientas software especializadas. Finalmente de los *Modelos de la IU* se extraen de forma automática (o semiautomática) diversos productos como el código de la *interfaz de usuario*, documentación o diagramas.

Los *MB-UIDEs* son *suites* de herramientas software que soportan el diseño y desarrollo de *interfaces de usuario* empleando *Modelos de la IU* [42]. De acuerdo con Schlungbaum [111], existen dos criterios básicos para que una herramienta pueda ser considerada un *MB-UIDE*:

1. Un *MB-UIDE* debe incluir un modelo abstracto de alto nivel y declarativo sobre el sistema interactivo a desarrollar (p.ej. un *Modelo de Tareas*, un *Modelo de Dominio* o ambos).

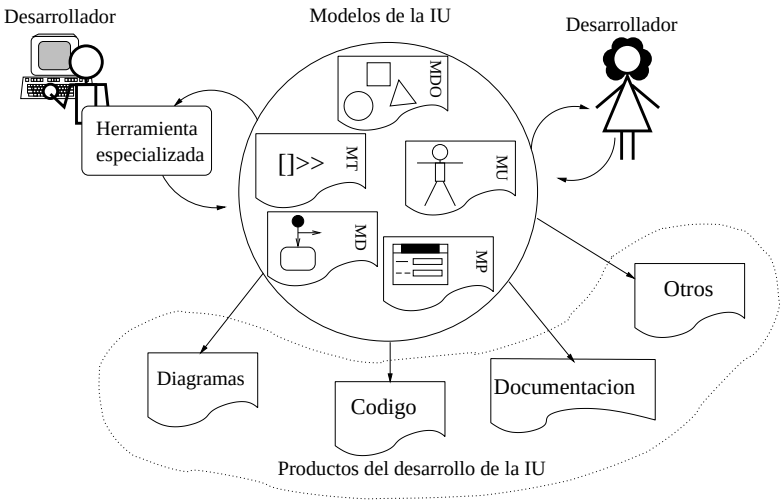


Figura 2.1: Desarrollo de la interfaz de usuario basado en modelos

2. El *MB-UIDE* debe explotar una relación clara y soportada por un proceso computacional desde (1) hasta la *interfaz de usuario* ejecutable.

2.1.1. Modelos de la interfaz de usuario

Puerta [99] define los *Modelos de la IU* como una representación computacional de los aspectos relevantes de una *interfaz de usuario*. Szekely [117] define un *Modelo de la IU* como una especificación declarativa de alto nivel de algún aspecto determinado de la *interfaz de usuario*, como su apariencia, estructura o comportamiento dinámico.

Históricamente no ha existido consenso sobre el conjunto de lenguajes y modelos necesarios para describir la *interfaz de usuario*, por lo tanto, la mayoría de sistemas han definido sus propios lenguajes [115, 45] y modelos [95, 70]. Sin embargo, algunos modelos usados recurrentemente en el *MB-UID* (ver figura 2.1) fueron identificados sobre la primera década del 2000 [95, 120]:

- El *Modelo de Usuario* (MU) recoge las características de los usuarios finales clasificándolos de acuerdo a sus habilidades, su conocimiento de la aplicación o la información que deben procesar. Algunos sistemas como ADEPT [128], MECANO [102] y TADEUS [25] han utilizado este modelo. El *User Modeling Markup Language* (UserML) [48] ha representado la propuesta más avanzada sobre este modelo.

- El *Modelo de Dominio* (MDO) se usa para definir los objetos que son accesibles a los usuarios a través de la *interfaz de usuario*. Según Silva [95] son constructores usuales de este modelo: clases, atributos, relaciones y operaciones. Para especificar el *Modelo de Dominio*, se han empleado diversas notaciones: diagramas de clases (OVID [108], WISDOM [83], IDEAS [94], TEALLACH [27], etc.), *Diagramas Entidad-Relación* (TRIDENT [10], GENIUS [51]), especificaciones algebraicas (ITS), etc.

- El *Modelo de Tareas* (MT) describe el conjunto de tareas que los usuarios pueden realizar, su descomposición en subtareas y sus relaciones temporales y condiciones. Se trata de un modelo de naturaleza jerárquica, donde se describen diferentes tipos de tareas y las acciones involucradas en su desarrollo. Son elementos frecuentes de este modelo [95]: tareas, objetivos, acciones, precondiciones y postcondiciones.

  El *Modelo de Tareas* ha ido adquiriendo gran importancia en la descripción de la *interfaz de usuario* al más alto nivel de abstracción [91]. En esta evolución se han empleado varias notaciones en la especificación del modelo (HTA, GOMS, TKS, UAN, etc.) [24]. Hoy en día la notación *Concurrent Task Trees* (CTT [87, 88, 91]) es ampliamente adoptada.

- El *Modelo de Presentación* (MP), está dedicado a aspectos de presentación de la *interfaz de usuario*. Puede ser descompuesto en dos submodelos:

  - *Modelo de Presentación Abstracta* (MPA) que trata, a un nivel de descripción abstracto la estructura y comportamiento de los objetos de la *interfaz de usuario*. Son elementos de este modelo los Objetos de Interacción Abstracta [123] (*Abstract Interaction Objects*) que son objetos sin representación gráfica e independientes de cualquier entorno y las *unidades de interacción* (o espacios de trabajo), una colección de *Objetos Abstractos de Interacción* agrupados de forma lógica para tratar con las entradas/salidas de una tarea.
  - *Modelo de Presentación Concreta* (MPC) que describe en detalle partes de la *interfaz de usuario* empleando Objetos de Interacción Concreta [123] (*Concrete Interaction Objects*), elementos "manipulables" de una *interfaz de usuario* que pueden ser empleados para mostrar o introducir información relativa a las tareas del usuario, ventanas (*windows*), una representación manejable y visible de una vista y *layouts*, un algoritmo que distribuye los *Objetos Concretos de Interacción* en una ventana.

- El *Modelo de Diálogo* (MD) [68] captura aspectos dinámicos de la interacción del usuario con el sistema, incluyendo una especificación de la relación entre unidades de presentación (p.ej. entre ventanas) así como entre elementos de la *interfaz de usuario* (p.ej. activar/desactivar objetos de interacción). También soporta la integración con el núcleo funcional de la aplicación, lo que requiere un mapeo entre eventos y acciones de acuerdo a restricciones predefinidas [62]. Incluye el conjunto de acciones que el usuario puede llevar a cabo en los distintos estados del sistema y las transiciones entre esos estados. El *Modelo de Diálogo* enlaza tareas con elementos de interacción formando un puente entre el *Modelo de Tareas* y el de *Presentación* [70].

Entre los modelos anteriores, los de *Tareas*, *Presentación* y *Diálogo*, pueden ser considerados los principales según Meixner [70] ya que son los que tienen una influencia directa en el contenido y apariencia de la *interfaz de usuario*.

2.1.2. Lenguajes de descripción de la interfaz de usuario

Los *Lenguajes de Descripción de la IU* (*User Interface Description Languages*, UIDLs [115, 45]) son lenguajes de alto nivel empleados en algunas etapas del desarrollo de la *interfaz de usuario* para describir las características interesantes de la interfaz con respecto al resto de la aplicación interactiva.

A lo largo de los años han surgido varias propuestas. Por un lado aquellas destinadas a cubrir la mera descripción de *interfaces de usuario*, como es el caso de UIML, XUL, XAML, Xforms, o AUIML. Por otro, aquellas desarrolladas para dar soporte al *MB-UID* y que por lo tanto, no se limitan a describir la interfaz sino que facilitan mecanismos para describir, dependiendo de los casos, al usuario, las tareas, el dominio, etc. como XIML, UsiXML o Teresa-XML. En la figura 2.2 se resumen las características principales de los *Lenguajes de Descripción de la IU* que han tenido mayor difusión. En dicha figura aparece en el eje vertical el número de modelos soportados por el lenguaje un indicador de su completitud y en el eje horizontal su año de aparición. El radio de la burbuja representa el impacto del lenguaje[1].

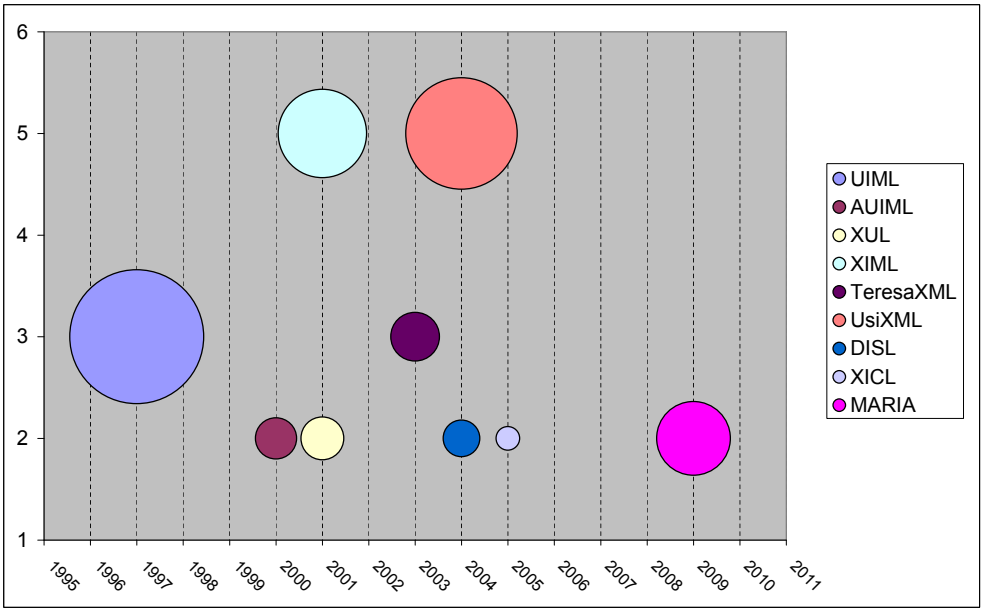


Figura 2.2: UIDLs: modelos soportados, año de aparición e impacto

A continuación, pasamos a resumir las propuestas que han tenido mayor difusión por orden cronológico:

[1]Para estimar el impacto de cada lenguaje se utilizado el número de citas según *Google Académico*. Se ha buscado el artículo con mayor número de referencias que contiene en su título el nombre del lenguaje en cuestión. Si no hay artículos que en su título contengan el nombre del lenguaje, se ha empleado el artículo de mayor número de referencias para ese lenguaje en la bibliografía de esta tesis.

- UIML (*User Interface Markup Language*) [3, 2, 6] es un lenguaje XML que ofrece un método para describir una *interfaz de usuario* independiente del dispositivo, la modalidad y el lenguaje de implementación. También permite describir la apariencia, la interacción y la conexión de la *interfaz de usuario* con la lógica de la aplicación. Aunque UIML es ostensiblemente independiente del dispositivo y del medio utilizado en la presentación, no parece que se tuviera en cuenta en su diseño los trabajos de investigación previos sobre *interfaces de usuario* basados en modelos. Por ejemplo, el lenguaje no provee la noción de tarea y principalmente intenta definir una estructura abstracta asociada a la presentación que ofrecerá la *interfaz de usuario*.

- AUIML (*Abstract User Interface Markup Language*) [7] es un lenguaje que separa la presentación de la semántica de la interacción, es decir, permite definir el propósito de la interacción con el usuario en lugar de centrarse en la presentación. Ha sido utilizado en OVID [108] y WISDOM [83] para proporcionar la especificación abstracta de una *interfaz de usuario*.

- XUL (*XML User Interface Language*) [50, 103] es la tecnología para *interfaces de usuario* multiplataforma basada en XML de Mozilla. Esta tecnología permite a los desarrolladores definir una interfaz gráfica multiplataforma para el usuario utilizando una mezcla de XML, HTML, CSS y JavaScript. Un usuario puede modificar cualquier aspecto en la *interfaz de usuario* de una aplicación Mozilla basada en XUL, simplemente modificando archivos que utilizan la sintaxis estándar de una página web. En Java existen gran cantidad de APIs para la generación de *interfaces de usuario* gráficas basadas en XUL, por ejemplo: Luxor XUL, XWT, Thinlets o SwingML.

- XIML (*eXtensible Interface Markup Language*) [100, 101] es un lenguaje extensible basado en XML para soportar múltiples modelos de especificación de *interfaces de usuario*. Una especificación XIML puede conducir a una interpretación en tiempo de ejecución o a una fase de generación de código en tiempo de diseño.

  XIML surgió como evolución de una propuesta anterior MIMIC [102]. El objetivo de este lenguaje es describir la interfaz de usuario de forma abstracta y posteriormente, obtener una especificación concreta a partir de ella. El desarrollo basado en modelos propuesto por XIML define la *interfaz de usuario* mediante cinco modelos dedicados a las tareas, dominio, usuario, diálogo y presentación.

- TeresaXML [90] es un *Lenguaje de Descripción de la IU* que permite producir *interfaces de usuario* para diferentes plataformas en tiempo de diseño. Es un lenguaje basado en XML dieñado por F.Paternò para la especificación de interfaces que además, enlaza con su propuesta anterior para el modelado de tareas CTT [87]. Actualmente se centra en aplicaciones Web. Este lenguaje especifica cómo se organizan varios *Objetos Abstractos de Interacción* que componen una *interfaz de usuario*, junto con la especificación del diálogo de la propia interfaz.

- UsiXML (*User Interface eXtensible Markup Language*) [58] propone un lenguaje de descripción de *interfaces de usuario* con el que se asegura poder trabajar

a diferentes niveles de abstracción (tareas, dominio, interfaz abstracta, interfaz concreta, interfaz final). Esta propuesta garantiza la trazabilidad, tanto hacia arriba como hacia abajo entre niveles de abstracción. También propone mecanismos para gestionar las asociaciones que se establecen entre los diferentes modelos contemplados.

- DISL (*Dialog and Interface Specification Language*) [110] es una extensión de UIML para soportar descripciones de diálogos genéricas e independientes de la modalidad. Las modificaciones realizadas se han enfocado principalmente en la descripción de *widgets* genéricos y en mejorar aspectos sobre la descripción del comportamiento.

  La estructura global de DISL tiene una parte opcional de cabecera para definir metainformación y otra con una colección de *templates* e interfaces de los cuáles sólo uno puede estar activo cada vez.

  Las interfaces se usan para describir la estructura del diálogo, su estilo y su comportamiento mientras que los *templates* sólo describen estructura y estilo, lo que permite que puedan reutilizarse por otros diálogos. También se ha desarrollado un editor interactivo de DISL [70].

- XICL (*eXtensible user-Interface Markup Language*) [18, 19] ofrece una forma sencilla de desarrollar componentes para la *interfaz de usuario*. Los componentes se crean partiendo de definiciones HTML y otros componentes XICL. Las especificaciones XICL son interpretadas en tiempo de ejecución y traducidas a HTML dinámico.

- MariaXML [92] es el sucesor de TeresaXML. Soporta comportamientos dinámicos, eventos, aplicaciones de internet enriquecidas (*Rich Internet Applications*[2]) e *interfaces de usuario* multidispositivo permitiendo que sean enlazadas con servicios web.

2.1.3. Evolución histórica

Hasta el momento se han identificado cuatro generaciones de sistemas de *MB-UID* [95, 42, 70, 26]. En la tabla 2.1, se muestran diversos sistemas clasificados según la generación a la que pertenecen. Pasamos a describir las particularidades principales de cada una de estas generaciones:

1. La primera generación (1990-1996), se centró en encontrar y abstraer aspectos relevantes de una *interfaz de usuario*. Las herramientas empleadas usaban un modelo universal que integraba todos los aspectos de interés para la *interfaz de usuario*. Estas propuestas estaban más dirigidas a la generación automática de

[2]Las aplicaciones de internet enriquecidas [36] combinan la arquitectura ligera de la web con la potencia computacional e interactividad de la *interfaz de usuario* de las aplicaciones de escritorio. El resultado mejora todos los elementos de una aplicación web (datos, lógica de negocio, comunicación y presentación).

la *interfaz de usuario* que a la definición de un proceso de desarrollo alternativo para el *MB-UID*. Ejemplos de esta generación son UIDE [31], AME [66] o HUMANOID [60].

2. La segunda generación (1995–2000) se centró en la extensión de los modelos de la *interfaz de usuario* a través de la integración de nuevos modelos en el *MB-UID* y en la expresión de la semántica de la *interfaz de usuario* a alto nivel. El modelado de la *interfaz de usuario* se comienza a estructurar en varios modelos: *Modelo de Tareas*, *Modelo de Diálogo*, *Modelo de Presentación*, etc. Se hace énfasis en la integración del *Modelo de Tareas* y se propone el uso del desarrollo centrado en el usuario [126]. Pertenecen a esta generación ADEPT [128], TRIDENT [10] y MASTERMIND [117].

3. La tercera generación (2000-2004) se dedicó a tratar la problemática de la explosión de nuevas plataformas de interacción que estaban surgiendo (*smartphones*, PDAs, etc.). El reto era desarrollar *interfaces de usuario* para distintos dispositivos con diferentes restricciones (p.ej. el tamaño de la pantalla). La propuesta principal era que los sistemas usaran técnicas para la especificación de *interfaces de usuario* independientes del dispositivo. Pertenecen a esta generación TERESA [77] y Dygimes [124].

4. La cuarta generación (aproximadamente desde 2004 a nuestros días) se centra en

Año	1ª Generación	2ª Generación	3ª Generación	4ª Generación
1990	ITS, DON			
1991				
1992	HUMANOID, TRIDENT			
1993	GENIUS			
1994	ADEPT, TADEUS			
1995		AIDE, MASTERMIND		
1996		AME, JANUS, TLIM		
1997		AMULET, CTT, UIML		
:	:	:	:	:
2001			TEALLACH, XIML	
2002			CAMELEON	
2003			TERESA, useML	
2004				DISL, UsiXML
2005				
2006				SerCHo
2007				EMODE
2008				Gummy
2009				MARIA

Tabla 2.1: Evolución histórica de los MB-UIDEs

el desarrollo de *interfaces de usuario* sensibles al contexto[3] para distintas plataformas, dispositivos y modalidades y en la integración de aplicaciones web. Los elementos básicos de la mayoría de propuestas actuales son modelos especificados en lenguajes XML que pueden ser importados y exportados desde herramientas específicas de desarrollo. Las propuestas prefieren emplear actualmente el término *model-driven* [106, 5] a *model-based*. El *Model-driven UI development* [5] pone en el centro del proceso de desarrollo a los modelos, que son transformados en código ejecutable o son renderizados por un intérprete. UsiXML [58] y MARIA [92] son propuestas de esta generación.

2.1.4. Framework de Referencia Cameleon

Para finalizar esta introducción al *MB-UID* trataremos el *framework* de referencia *Cameleon* (*Cameleon Reference Framework*, CRF) [12, 13]. En la última década ha existido una gran proliferación de dispositivos (PDAs, *smartphones*, *tablets*, etc.) y modalidades interactivas (vocales, táctiles, etc.). Los usuarios desean acceder a sus sistemas desde cualquier tipo de dispositivo y que las *interfaces de usuario* se adapten a éstos incluso de forma dinámica en función del entorno, es decir, las aplicaciones actuales deben adaptarse como un camaleón a un contexto cambiante.

Para los desarrolladores, esto introduce nuevas dificultades como construir múltiples versiones de una aplicación o dotar a las aplicaciones de la capacidad de adaptarse dinámicamente a cambios del contexto. Esto implica mayores costes de desarrollo y mantenimiento y complica la gestión de la configuración. La proliferación de versiones diluye los recursos destinados al tratamiento de la usabilidad y requiere invertir en el mantenimiento de la consitencia de la *interfaz de usuario* entre las distintas plataformas.

El *framework* de referencia *Cameleon* fue un resultado del *EU-funded CAMELEON Project* [1]. El objetivo principal de este proyecto [32] era dar soporte al diseño y desarrollo de software interactivo altamente usable y sensible al contexto basándose en dos principios clave: el desarrollo basado en modelos y la cobertura de las fases de diseño y ejecución de *interfaces de usuario* multidispositivo. Desde su definición en el 2003, el *framework* de referencia *Cameleon* ha sido ampliamente aceptado por la comunidad de la *Interacción Persona-Computador* como una referencia para clasificar sistemas que soportan múltiples plataformas o múltiples contextos de uso basándose en el *MB-UID* [70] y por lo tanto, es una referencia donde comparar diferentes propuestas [69]. El *framework* describe diferentes capas de abstracción relacionadas con el *MB-UID*:

- El nivel de *Conceptos y Tareas* (*Concepts-and-Tasks*, C&T) especifica las jerarquías de tareas que deben ser realizadas en/con objetos del dominio (o conceptos del dominio) para un sistema interactivo particular [69]. Los tradicionales modelos de *Dominio* y *Tarea* pertenecen a esta capa de abstracción.

- La *IU Abstracta* (*Abstract UI*, AUI) expresa la *interfaz de usuario* en términos de *unidades de interacción* sin hacer cualquier referencia a la implementación ni en

[3]El contexto de uso de un sistema interactivo puede definirse como un espacio de información dinámico y estructurado que incluye las siguientes entidades: el usuario que emplea el sistema, la plataforma empleada y el entorno en que se usa.

términos de la modalidad de la interacción (*i.e.* táctil, gráfica, vocal, etc.) ni en términos del espacio tecnológico (es decir, plataforma, lenguaje de programación o de marcas) [122]. Se corresponde con el *Modelo independiente de la plataforma* de la MDE. El *Modelo de Presentación Abstracta* o el *Modelo de Diálogo* pertenecen a este nivel de abstracción.

- La *IU Concreta* (*Concrete UI*, CUI) describe de forma concreta cómo es percibida la interfaz por los usuarios empleando *Objetos Concretos de Interacción* que son dependientes de la modalidad, pero independientes de la implementación y del lenguaje empleado. Se corresponde con el *Modelo específico de la plataforma* de la MDE. El *Modelo de Presentación Concreta* pertenece a esta capa.

- La *IU Final* (*Final UI*, FUI) expresa la *interfaz de usuario* en código fuente dependiente de la implementación. Puede ser expresada en un lenguaje de programación o utilizando lenguajes de marcas (p.ej. Java, HTML o VoiceXML), por lo tanto, puede ser compilada o interpretada. Además, un mismo trozo de código no tiene por qué ser mostrado siempre de la misma manera. Por esta razón *Cameleon* considera dos subniveles de la *IU Final*: el código fuente y la interfaz en ejecución.

El proceso de desarrollo de la *interfaz de usuario* según *Cameleon* consiste en transformar *Modelos de la IU* desde los niveles más abstractos hasta los más concretos empleando para ello operaciones sucesivas. De este modo se consigue evolucionar desde las especificaciones de los modelos iniciales (los de más alto nivel) hasta la *IU Final* (el nivel más concreto) [32].

La operación que transforma un modelo de un nivel abstracto en otro más concreto se denomina refinamiento (*reification*). Su operación opuesta que parte de un modelo de menos nivel de abstracción hacia otro de mayor nivel se denomina abstracción (*abstraction*). También existen operaciones horizontales, es decir que trabajan en el mismo nivel de abstracción. La Traducción (*translation*) es una operación que transforma una descripción orientada a un contexto de uso determinado en otra al mismo nivel pero para un contexto diferente. Por último, la reflexión (*reflection*) transforma una descripción en otra al mismo nivel de abstracción y para el mismo contexto de uso.

La *IU Final*, expresada en código fuente se genera de forma manual o automática desde la *IU Concreta* [32]. La generación de código es un caso particular de refinamiento que transforma la *IU Concreta* en código fuente.

En la figura 2.3, se puede apreciar un diagrama explicativo del proceso de desarrollo según el *framework* de referencia *Cameleon*. Podemos ver cómo desde el nivel más alto de abstracción (*Conceptos y Tareas*) se puede ir descendiendo empleando operaciones de refinamiento (R). También se puede apreciar que para pasar de niveles de abstracción más bajos a otros más altos se utiliza la operación de abstracción (A). Obsérvese también en la figura que de un modelo de mayor nivel de abstracción se pueden derivar varios modelos de un nivel de abstracción más bajo y que el contexto está presente durante todo el proceso en cualquier nivel de abstracción.

La *IU Final* puede ser interpretada o compilada. Puede tratarse de una *interfaz de usuario* pretratada dirigida a contextos de uso específicos [13] (*i.e.* usuario, plataforma

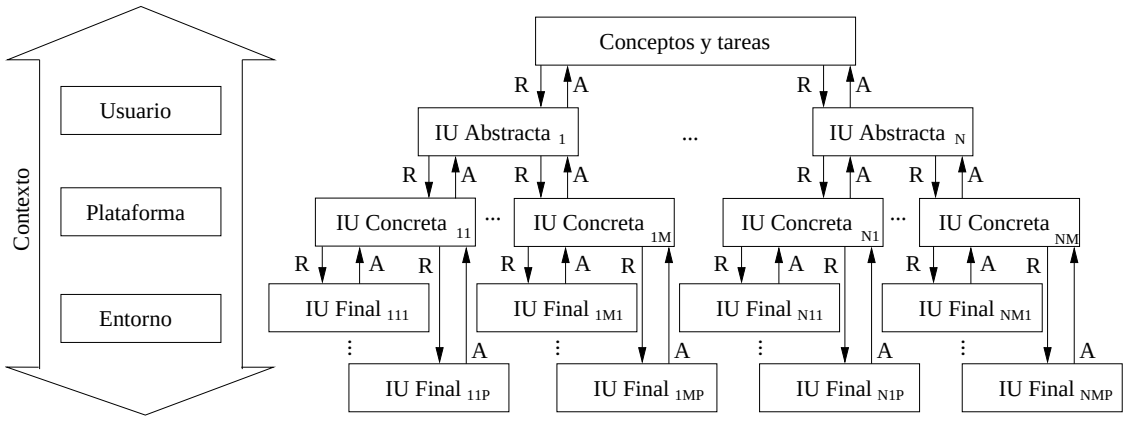


Figura 2.3: Framework de referencia Cameleon

y/o entorno), o por el contrario puede ser insertada en un *run-time* que soporte la adaptación dinámica a múltiples contextos en tiempo de ejecución. Tradicionalmente se han empleado tres arquitecturas para la generación de la *interfaz de usuario* [95, 75]:

- En la primera (figura 2.4a), el código fuente de la *interfaz de usuario* se genera apoyándose en un *toolkit* determinado. En este caso el *MB-UIDE* genera el código fuente de la interfaz y algunas veces el esqueleto de la aplicación. Esta arquitectura es seguida por TEALLACH [27] y TADEUS [25] entre otros.

- En la segunda propuesta (2.4b), una versión pretratada la *interfaz de usuario* es ejecutada por un *Run-time* (normalmente para un UIMS[4] particular) enlazado con la aplicación. Sistemas como TRIDENT [10] y METAGEN [76] poseen un generador de código intermedio. Este código permite más flexibilidad e independencia que los modelos originales.

- En la tercera propuesta (2.4c), la aplicación interpreta los *Modelos de la IU* directamente gracias a un *Run-time* del *MB-UIDE* que está enlazado en la aplicación. Siguen esta arquitectura sistemas como MERLIN [79] o MECANO [102].

Las *interfaces de usuario* generadas siguiendo la arquitectura (a) tienen la ventaja de que están completamente codificados en la aplicación, ofreciendo una integración natural entre la *interfaz de usuario* ésta. Sin embargo, la *interfaz de usuario* producida es más estática que las generadas empleando las arquitecturas (b) y (c), que son más flexibles y adaptables en tiempo de ejecución. Las *interfaces de usuario* generadas mediante la tercera propuesta tienden a tener peores prestaciones que las que emplean las dos primeras, ya que es más costoso interpretar los *Modelos de la IU* en tiempo de ejecución.

[4]Un *User Interface Management System* [70] (UIMS) es una herramienta (o un conjunto de ellas) diseñada para estimular la cooperación interdisciplinar en el desarrollo rápido, construcción y gestión (control) de la interacción en un dominio de aplicación sobre distintos dispositivos, técnicas de interacción y estilos de la *interfaz de usuario*.

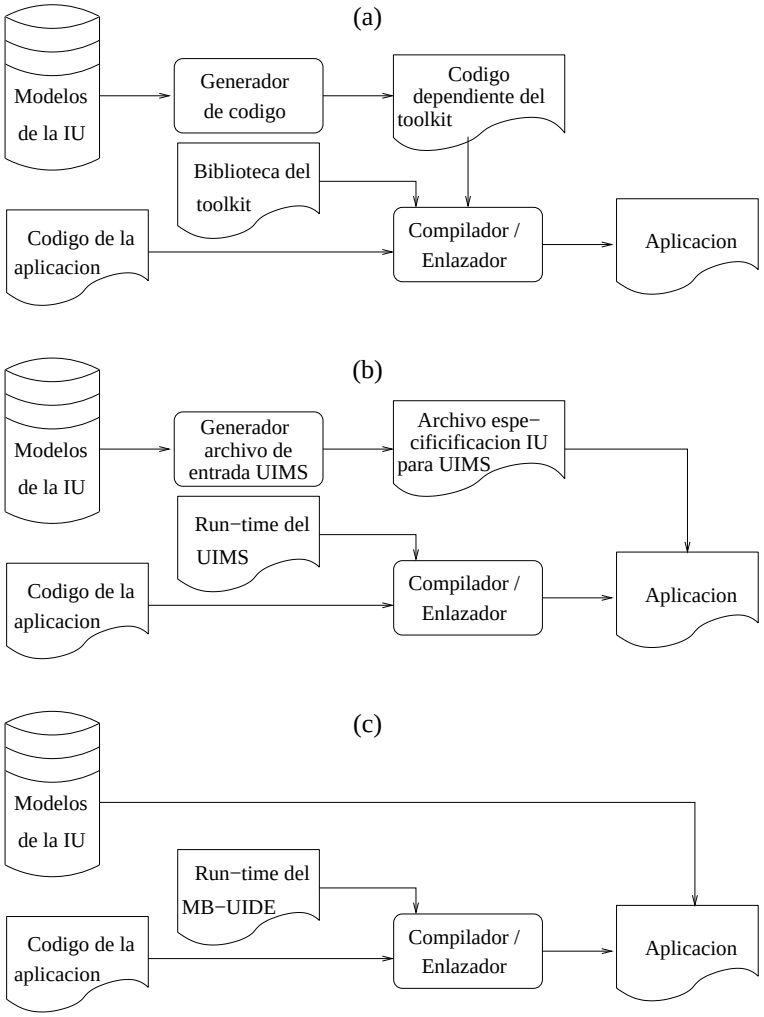


Figura 2.4: Arquitecturas para la generación de la interfaz de usuario

Dado que a lo largo de esta tesis se volverá a hacer referencia a las arquitecturas expuestas en este apartado, para facilitar la lectura del documento, se ha decidido asignarle un nombre a cada una de ellas. Así, a partir de este momento, a las arquitecturas de tipo (a) (b) y (c) que aparecen en la figura 2.4, las denominaremos *Generativas*, *Mixtas* e *Interpretativas* respectivamente.

Las herramientas de desarrollo empleadas en las arquitecturas de la figura 2.4, pueden clasificarse en dos tipos:

- *Generadores de interfaces de usuario*: se trata de herramientas que hacen la aplicación independiente de los *Modelos de la IU*. Los generadores de código fuente (figura 2.4a) y los generadores de código intermedio (figura 2.4b) son generadores de *interfaces de usuario*. Obviamente, los *MB-UIDEs* que puedan ejecutar una *interfaz de usuario* directamente desde los *Modelos de la IU* no poseen generadores de código.

- *Run-times de la interfaz de usuario*: son herramientas que generan la *interfaz de*

usuario cuando la aplicación está ejecutándose. La propia aplicación en la primera propuesta (figura 2.4a) y los *Run-times* de las arquitecturas *Mixta* e *Interpretativa* (figuras 2.4b y 2.4c) son *Run-times de la IU*.

Como colofón de este apartado, en la figura 2.5, extraída en parte de [70], se muestra una comparativa de varias propuestas actuales que siguen el Modelo de Referencia *Cameleon*, entre ellas la del *MB-UIDE* empleado en esta tesis.

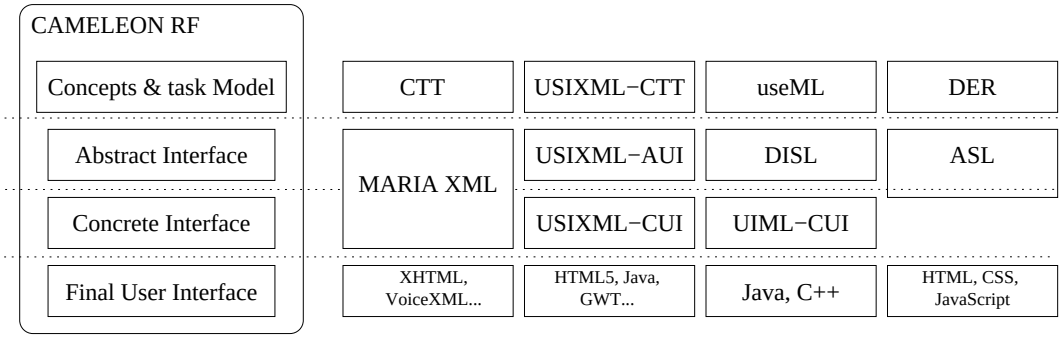


Figura 2.5: Comparativa de propuestas en el framework de referencia Cameleon

- La primera propuesta utiliza CTT y MARIA. MARIA cubre dos niveles de abstracción el nivel de la *IU Abstracta* y el de la *IU Concreta* y dispone de una herramienta propia para realizar especificaciones: MARIAE [93].
- En segundo lugar UsiXML se estructura de acuerdo a los cuatro niveles de abstracción definidos por el *framework Cameleon*, incluyendo además modelos de contexto y de calidad.
- La tercera propuesta (de Meixner *et al.* [71]), está basada en tres lenguajes a diferentes niveles de abstracción: useML, DISL y UIML. useML [80, 105] fue diseñado para soportar el proceso de desarrollo centrado en el usuario (ISO 9241-210) y permite representar los resultados del análisis inicial de tareas. DISL se emplea para especificar el *Modelo de Diálogo* derivado del *Modelo de Tareas* expresado en useML. Por último, UIML define el *Modelo de Presentación*.
- En último lugar aparecen los lenguajes utilizados por WAINE. El *Modelo de Dominio*, expresado mediante un *Diagrama Entidad-Relación*, comprende el nivel de *Conceptos y Tareas*. Del *Modelo de Dominio* se derivan el *Modelo de Diálogo* y el *Modelo de Presentación* que son descritos en un lenguaje denominado ASL [20, 21]. A partir del documento ASL y mediante diversas transformaciones se genera la *IU Final* compuesta por HTML, CSS y JavaScript.

2.2. Reutilización

En esta segunda sección se trata la reutilización software como disciplina científica y posteriormente, se profundiza en aquellos trabajos que tienen una relación más estrecha

con el estudio presentado en esta tesis: la reutilización de la *interfaz de usuario* en el *MB-UID*.

Una de las bases del proceso de diseño en la mayoría de las disciplinas de ingeniería es la reutilización de sistemas o componentes existentes [114]. En la ingeniería del software, las propuestas para la reutilización fomentan el desarrollo de sistemas usando elementos preexistentes en lugar de crearlos de nuevo [55, 49]. Desde sus orígenes, la reutilización ha sido valorada como una vía para superar la crisis del software y se ha propugnado como mecanismo para reducir el tiempo de desarrollo, incrementar la productividad de los desarrolladores y reducir la densidad de errores [49, 11, 73].

La reutilización del software se puede aplicar a cualquier producto del ciclo de vida del desarrollo, no sólo al código fuente. Esto significa que los desarrolladores pueden reutilizar requisitos, especificaciones, diseños y cualquier otro elemento del ciclo de desarrollo. Se han identificado diez aspectos potencialmente reutilizables de los proyectos software [34] (ver tabla 2.2). En otras fuentes [55, 84, 59, 86] también se hace referencia a la reutilización de otros elementos como diseños, módulos, especificaciones, etc.

1. Arquitecturas	6. *Templates*
2. Código fuente	7. Interfaces de usuario
3. Datos	8. Planes
4. Diseños	9. Requisistos
5. Documentación	10. Pruebas

Tabla 2.2: Elementos potencialmente reutilizables

Una ventaja obvia de la reutilización de software es que los costes totales de desarrollo deberían reducirse. Se necesitan especificar, diseñar, implementar y validar menos componentes software [114]. Asimismo, la reutilización hace los desarrollos más confiables ya que los posibles fallos que existieran en las implementaciones y/o diseños han debido ser detectados y reparados con anterioridad. También, en algunas ocasiones sacar al mercado un producto tan pronto como sea posible es más importante que los costes totales de desarrollo. En este caso, la reutilización es una herramienta útil ya que puede reducir los tiempos de desarrollo y validación del software.

Por otra parte, la reutilización presenta costes y problemas asociados que deben ser señalados [114]. Los ingenieros deben localizar los componentes software dentro de una colección de elementos reutilizables, comprender su funcionamiento, analizar si el componente es adecuado para la situación concreta y en algunos casos adaptarlos para su uso en un entorno nuevo. También ocurre que cuando el código fuente de un elemento reutilizado no está disponible los costes de mantenimiento pueden incrementarse debido a que los componentes reutilizados son cada vez más incompatibles con las versiones sucesivas del sistema. Por último, aunque la mayoría de los desarrolladores (un 72 % según Frakes [35]) prefiere reutilizar componentes antes que construir su software por completo, algunos ingenieros siguen prefiriendo reescribir componentes por que opinan que así pueden mejorarlos versión tras versión.

Existen al menos seis dimensiones (ver tabla 2.3) desde las que analizar la reutilización de software [98, 34]:

- El ámbito de la reutilización puede ser interno (también privado) o externo (también público) en función del origen de los elementos reutilizados. Así, cuando los elementos no provienen de un repositorio externo al sistema en construcción se habla de reutilización interna y cuando la situación es la contraria la llamamos reutilización externa.

- El dominio indica la forma y alcance de la reutilización. Puede ser horizontal cuando el objetivo es reutilizar elementos entre diferentes aplicaciones o vertical cuando se realiza dentro del mismo dominio/aplicación.

- La intención expresa cómo serán reutilizados los recursos. La reutilización de caja negra (*black-box reuse* o *verbatim reuse*) es la reutilización de recursos software sin modificación. Por contra la reutilización de caja blanca (*white-box reuse*) requiere la modificación/adaptación de los elementos que se desea reutilizar. Por último, la reutilización adaptativa (*adaptative reuse*), está identificada como una técnica que mantiene grandes estructuras de software invariantes y restringe las variaciones a localizaciones aisladas de bajo nivel.

- La técnica define los métodos empleados para reutilizar. Además de técnicas concretas que trataremos con mayor profundidad más adelante, en esta categoría se han analizado otros aspectos que citamos a continuación. Se entiende por reutilización indirecta aquella que emplea entidades intermedias entre el elemento que reutiliza y el reutilizado, por contra, la reutilización directa no emplea elementos intermedios. También se emplean los términos *carried-over reuse* cuando un componente software empleado en una versión determinada del proyecto es empleado en las siguientes versiones y *leveraged reuse* cuando esa reutilización requiere modificaciones. Por último, la reutilización *in-the-large* emplea grandes paquetes autocontenidos (p.ej. hojas de cálculo), mientras la reutilización *in-the-small* usa componentes dependientes del entorno de la aplicación.

- La gestión indica cómo se dirige la reutilización. Puede ser sistemática o pragmática. La reutilización sistemática o planificada [33] utiliza guías, procedimientos formales y métricas para evaluar el rendimiento de la reutilización, mientras que

Ámbito	**Dominio**	**Intención**	**Técnica**	**Gestión**	**Entidad**
Interna	Vertical	Caja blanca	Composicional	Sistemática	código
Externo	Horizontal	Caja negra	Generativa	Pragmática	arquitectura
		Adaptativa	Directa		diseños
			Indirecta		pruebas
			carried-over		etc.
			leveraged		
			in-the-large		
			in-the-small		

Tabla 2.3: Dimensiones de la reutilización

la reutilización pragmática [49] (también oportunística o ad-hoc) emplea componentes que no han sido diseñados para ser reutilizados. La mayor parte de la investigación en reutilización software se ha enfocado en propuestas de reutilización planificadas [56, 107], sin embargo, la situación muchas veces es la opuesta, donde los desarrolladores deben crear algún elemento nuevo similar a alguno creado con antelación pero que no ha sido diseñado para ser reutilizado [49].

- La entidad reutilizada indica el tipo de producto reutilizado: codigo fuente, diseños, especificaciones, objetos, arquitecturas, etc.

Debido al especial interés que tienen en esta tesis las técnicas para la implementación de la reutilización, se ha considerado interesante profundizar un poco más en las propuestas existentes en este sentido. Mohagheghi *et al.* [73] identifican las siguientes clases de técnicas:

- La reutilización por composición o composicional (*Compositional Reuse*) utiliza componentes (de grano fino) existentes en colecciones o bibliotecas como bloques constructivos de nuevos sistemas. Implica que el sistema software (sistema de mayor entidad) es construido por un desarrollador a partir de elementos o componentes (elementos de menor entidad). Por ejemplo, el empleo de funciones o subrutinas se considera reutilización por composición.

- Reutilización de módulos o componentes software. Los módulos o componentes son elementos de mayor entidad que las funciones o subrutinas. Proveen servicios a otros módulos o componentes del sistema, pero si se les analiza por separado, no pueden llegar a considerarse un sistema completo. Algunas de las aproximaciones para la modularización incluyen la Programación Orientada Objetos o los paquetes de Ada.

- Reutilización de *templates* o meta-componentes. Un *template* es un esqueleto de un artefacto, descrito a diferentes niveles de abstracción. Normalmente se almacenan en repositorios y están compuestos de una parte fija, independiente de la aplicación y una parte que debe ser definida por el desarrollador y especializada para el producto software final en el que se desea incluir. Así, el esfuerzo requerido para la reutilización consiste en especializar el contenido del *template* para el contexto concreto.

  En la literatura se distinguen varios tipos de *templates*: de arquitectura: que especifican un artefacto desde el punto de vista de la arquitectura; de documentación: que definen la documentación relacionada con el artefacto; y de pruebas: que definen el conjunto de casos de pruebas para el artefacto. Los *templates* son útiles para la adopción de estándares en la codificación y documentación de proyectos software.

- La ingeniería de dominio (*domain engineering*) para familias o líneas de productos, consiste en agrupar, organizar y almacenar la experiencia obtenida en la construcción de sistemas (o partes de los mismos) sobre un dominio particular

en forma de recursos reutilizables, ofreciendo métodos adecuados para reutilizar esos recursos en la construcción de nuevos sistemas. Es un concepto clave en la reutilización sistemática.

- *Frameworks* orientados a objetos. Los *frameworks* son aplicaciones semicompletas, generalmente implementadas como una jerarquía de clases. Una vez que el *framework* ha sido desarrollado pueden derivarse de él múltiples aplicaciones. Los *frameworks* son una técnica de reutilización que soporta las líneas de productos.

- La reutilización generativa es una propuesta para la reutilización asociada a la ingeniería del dominio. El proceso consiste en la codificación de conocimiento sobre el dominio y sobre la construcción de sistemas en ese dominio en un generador de aplicaciones específico. Los nuevos sistemas son creados escribiendo especificaciones en un lenguaje propio del dominio. El generador traduce las especificaciones en código para el nuevo sistema en un lenguaje objetivo. Este proceso puede ser completamente automático o requerir la intervención manual de los desarrolladores. Por ejemplo, los tradicionales analizadores lex y yacc son generadores de código.

- Reutilización basada en componentes ligada a modelos como CORBA/CCM/EJB. El término componente aparece en muchas propuestas de reutilización cuando los sistemas son ensamblados a partir de unidades independientes de producción. Los componentes reutilizables pueden ser recuperados de un repositorio, compartidos entre productos en una familia de productos o pueden ser obtenidos como componentes OSS (*Open Source Software*) o COSTS (*Commodity off-the-shelf*) [23]. Sin embargo, en esta categoría, debemos entender la reutilización de componentes adherida a un modelo de componentes particular y dirigida a una plataforma de componentes concreta en oposición a otros modelos sin restricciones sobre la conformidad con una arquitectura.

- Los repositorios o bibliotecas de reutilización (*reuse repository or library*), constan de almacenes de recursos reutilizables, herramientas para la búsqueda de recursos en el almacén, un método para la representación de los recursos y utilidades para la gestión del cambio y el aseguramiento de la calidad. Los repositorios de reutilización, pueden ser genéricos o específicos de un dominio y pueden ser combinados con otras propuestas para la reutilización.

El tratamiento científico de la reutilización del software exige que se le considere una disciplina empírica [33]. La profundización en conceptos como reutilización y reusabilidad[5] nos llevan a cuestiones sobre cómo medirlos adecuadamente [30] y cómo realizar experimentos que permitan establecer su impacto sobre la calidad y la productividad. Para ello se han definido métricas en varias áreas de la reutilización software [34]. Las métricas son indicadores cuantitativos de un atributo de un producto o proceso software. La experimentación es el proceso de establecer el efecto de unas variables sobre otras. Los modelos y métricas sobre reutilización y reusabilidad se clasifican en seis categorías [34]:

[5]La reusabilidad es la probabilidad de que un elemento sea reutilizado.

- Los modelos coste-beneficio incluyen el análisis económico coste-beneficio así como el análisis de la mejora de la calidad y de la productividad. En las organizaciones que contemplan la reutilización de software sistemática seguramente las primeras preguntas que aflorarán estarán relacionadas con los costes y los beneficios. La justificación de la inversión en estos casos se convierte en una necesidad.

- Los modelos de estimación de la madurez clasifican los programas de reutilización según su nivel de progresión en la implementación de la reutilización sistemática. Los modelos de madurez son el centro de la reutilización planificada, ayudando a las organizaciones a entender sus pasados, presentes y futuros objetivos para las actividades de reutilización.

- La cantidad de reutilización es usada para evaluar y monitorizar la reutilización de los elementos de desarrollo a lo largo del ciclo vida de un proyecto. Este tipo de métricas suele incluir factores como el nivel de abstracción del objeto reutilizado, así como definiciones formales sobre reutilización interna y externa.

- El análisis modal de fallos se emplea para identificar y clasificar los obstáculos para la reutilización en un determinado organismo. Ofrece una aproximación para medir y mejorar un proceso de reutilización basándose en un modelo de las formas en las que dicho proceso puede fallar.

- Las métricas de reusabilidad indican la probabilidad de que un elemento sea reutilizado. Estas métricas son potencialmente útiles en dos áreas clave de la reutilización: el diseño de la reutilización y la reingeniería para la reutilización.

- Las métricas para bibliotecas de reutilización [6] se usan para gestionar y seguir el uso de un repositorio de reutilización. Los recursos de las bibliotecas pueden ser obtenidos de sistemas existentes a través de la reingeniería, diseñados y construidos desde cero o pueden ser adquiridos.

Silva [95] indica que el uso de *Modelos de la IU* en el *MB-UID* ofrece la capacidad de reutilizar especificaciones de la *interfaz de usuario*. De hecho, el *MB-UID* en si puede ser considerado una propuesta para la reutilización generativa ya que reutiliza las especificaciones de la *interfaz de usuario* mediante generadores de código [121]. A continuación nos centraremos en las propuestas que tienen una relación más estrecha con el campo de trabajo de esta tesis: la reutilización de la *interfaz de usuario* en el *MB-UID*. Las propuestas existentes se pueden clasificar en los siguientes grupos:

1. Soporte para la reutilización en los *Lenguajes de Descripción de la IU*.

2. Desarrollo de *interfaces de usuario* multi-dispositivo/multi-contexto basado en modelos.

3. Métodos de reutilización basados en patrones.

[6] Una biblioteca de reutilización es un repositorio para almacenar recursos reutilizables junto con una interfaz que permite realizar búsquedas en el repositorio.

2.2.1. Soporte para la reutilización en los UIDLs

Los *Modelos de la IU* usados en el *MB-UID* se expresan a través de lenguajes gráficos o textuales. Hoy existen un gran número de *Lenguajes de Descripción de la IU* [115, 45] que son usados con este propósito. Los *Lenguajes de Descripción de la IU* son lenguajes de alto nivel que se emplean para describir las características de la *interfaz de usuario* que son de interés en las aplicaciones interactivas. Al tratarse en la mayoría de los casos de lenguajes textuales, una tentación habitual es copiar y pegar las porciones de código que se necesita reutilizar. Esta acción está identificada como una mala práctica en el desarrollo (*cut&paste* programming [14]) ya que deriva en un código engrosado e inmantenible. Una opción mucho más correcta sería encapsular el fragmento reutilizable de una forma apropiada, sin embargo, pocos *Lenguajes de Descripción de la IU* han tenido en cuenta esta necesidad:

- XUL [50] es el *Lenguaje de Descripción de la IU* desarrollado por el *Mozilla project* para sus desarrollos multi-plataforma como por ejemplo el navegador web *Firefox*. XUL dispone de *XUL Ovelays*, un mecanismo que permite: añadir componentes adicionales a una *interfaz de usuario* existente, sustituir pequeños trozos de código en un fichero XUL sin necesidad de suministrar la *interfaz de usuario* completa y reutilizar trozos específicos de la *interfaz de usuario*. En este último caso, el mecanismo permite definir elementos de la *interfaz de usuario* en archivos de *overlay* e incluirlos en documentos que necesiten utilizarlos. Luego, empleando el atributo *id* en un nodo vacío del documento que realiza la inclusión, se referencia al subárbol definido en el *overlay* allí donde sea necesario. Por ejemplo, los botones *Aceptar* y *Cancelar* que aparecen en la parte más baja de multitud de diálogos pueden definirse en un archivo y ser empleados en todos los diálogos que los necesiten. Para ello, los diálogos que quieran reutilizar esa definición deben primero "declarar" (incluir) el *overlay*:

```
<?xul-overlay href="chrome://global/content/dialogOverlay.xul"?>
```

  Y posteriormente incluir una "caja" vacía referenciando al elemento reutilizado (en este caso con identificador *okCancelButtons*) en la definición de la *interfaz de usuario*:

```
<box align="horizontal" id="bx1" flex="100%" style="margin-bottom:
    1em; width: 100%;">
<html:input type="checkbox" id="dialog.newWindow" />
<html:label for="dialog.newWindow">&openWin.label;</html:label>
<spring flex="100%" />
</box>

<box id="okCancelButtons" />
```

  Los *overlays* de XUL se pueden cargar de forma explícita usando la instrucción especial de procesamiento XML (<?xul-overlay?>). No hay límite al número de *overlays* que se pueden cargar desde un documento XUL. A su vez un *overlay* puede cargar otros *overlays*.

- XICL [18, 19] es un lenguaje extensible basado en XML para definir *interfaces de usuario* y componentes de la *interfaz de usuario*. Está orientado especialmente a aplicaciones web. Los componentes de XICL se desarrollan empleando elementos de HTML y otros elementos (etiquetas) propios del lenguaje XICL:

```
<COMPONENT name="SUBMIT">
 <STRUCTURE>
  <span>\$label</span><input type="submit" maxlength="\$length"
      name="subBnt">
 </STRUCTURE>
 <EVENTS>
  <EVENT name="onclick" trigger="subBtn.onclick"
      function="\$onclick" >
 </EVENTS>
</COMPONENT>
```

  Los componentes quedan descritos a través de su estructura, propiedades, eventos, métodos y su modelo de interacción (ver figura 2.6).

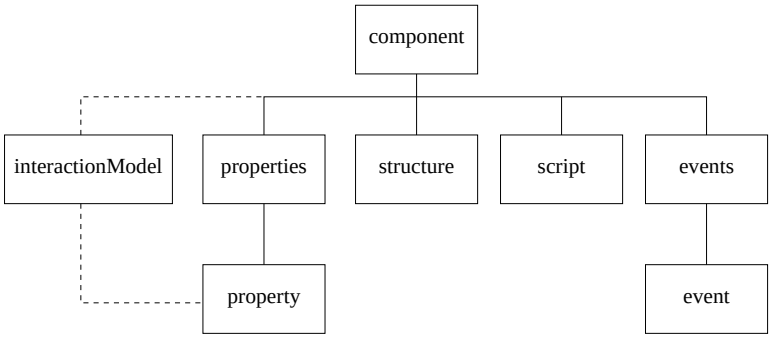


Figura 2.6: Componente XICL

  Una *interfaz de usuario* en XICL puede construirse empleando componentes reutilizables almacenados en una biblioteca (mediante la instrucción *import*). Los componentes se pueden definir extendiendo un componente existente (usando el atributo *extends* de la etiqueta <component>) y modificando alguna de sus propiedades. Una vez que el desarrollador ha especificado la *interfaz de usuario* y/o sus componentes el intérprete de XICL enlaza el código fuente de los componentes con la *interfaz de usuario* y lo traduce en código HTML dinámico.

- UIML permite empaquetar partes de la definición de una *interfaz de usuario* en *templates* [2]. Los *templates* pueden ser reutilizados posteriormente en nuevos diseños de *interfaces de usuario* de una manera flexible. Para su funcionamiento se emplean dos piezas fundamentales en UIML: el *placeholder* y el propio *template*. El *placeholder* indica qué *template* se debe aplicar y cómo. El atributo *source* identifica el *template* que se desea incluir. Su valor se corresponde con una URI (*Unified Resource Identifier*) que referencia la localización del *template* seguida del caracter *'#'* y el identificador del mismo. El atributo *how* indica la estrategia a seguir para el fusionado del *template* en el documento. En su valor pueden aparecer los métodos *replace*, *union* y *cascade*.

Por ejemplo, muchas aplicaciones de comercio electrónico incluyen un diálogo para el pago mediante tarjeta de crédito. Ese diálogo puede describirse como un *template* UIML y ser incluido varias veces en la misma *interfaz de usuario* o entre distintas *interfaces de usuario*.

```
<template id="DialogBox">
 <part id="TopLevel">
  <part id="CompanyLogo" class="ImageContainer"/>
  <part id="Message" class="Text"/>
  <part id="Accept" class="Button"/>
 </part>
</template>

...

<interface>
 <structure>
  <part id="FileNotFoundBox" class="TopContainer"
      source="#DialogBox" how="replace">
  <!-- Default implementation -->
  </part>
 </structure>

</interface>
```

UIML también admite la parametrización de los *templates*. La parametrización permite pasar datos a un *template*. Para su uso se emplean los elementos: *d-template-param* (un parámetro que será recibido por el *template*), *d-template-parameters* (un contenedor de *d-template-param*), *template-param* (un parámetro que se pasa a un *template*), *template-parameters* (un contenedor de *template-param*). Para obtener el valor de un parámetro se antepone el caracter *'$'* a su identificador.

```
<uiml>
 <interface>
  <structure>
   <part id="id1" source="#tpl" how="replace">
 <template-parameters>
       <template-param name="button_id">btn_copy</template-param>
       <template-param name="button_label">Click to
           copy</template-param>
       <template-param name="entry_id">entry_copy</template-param>
    </template-parameters>
   </part>
  </structure>
 </interface>

 <template id="tpl">
  <d-template-parameters>
   <d-template-param name="button_id"/>
   <d-template-param name="button_label"/>
   <d-template-param name="entry_id"/>
  </d-template-parameters>
```

```
  <part>
   <part id="$entry_id" class="Entry"/>
   <part id="$button_id" class="Button">
    <style>
     <property name="label">
      <template-param id="button_label"/>
     </property>
    </style>
   </part>
  </part>
 </template>
</uiml>
```

Hemos visto que distintos *Lenguajes de Descripción de la IU* han desarrollado sus propios mecanismos para la reutilización. Sería deseable que un mecanismo de reutilización pudiera ser empleado en cualquier *Lenguaje de Descripción de la IU*, siendo lo más estándar posible y minimizando las modificaciones a realizar en la sintaxis del lenguaje. UIML, XICL y XUL usan diferentes métodos para reutilizar fragmentos de especificaciones de *Modelos de la IU* y éstos no podrían ser empleados directamente en otros *Lenguajes de Descripción de la IU*, ya que tienen comportamientos muy característicos, definen sus propias etiquetas XML o usan instrucciones especiales de procesamiento de XML.

2.2.2. Desarrollo de interfaces de usuario multi-dispositivo basado en modelos

Como se ha comentado con anterioridad todo un grupo de propuestas (tercera y cuarta generación de *MB-UIDEs*, sección 2.1.3) se han orientado a tratar la problemática del gran número de plataformas de interacción disponibles y del desarrollo sensible al contexto. El propio *Framework* de Referencia *Cameleon* [13] representa un marco en el que clasificar sistemas de *MB-UID* que soportan múltiples plataformas o múltiples contextos de uso. Existen por lo tanto varios *MB-UIDEs* (TERESA [77], Dygimes [124], XMobile [125], etc.) y *Lenguajes de Descripción de la IU* (e.g. AUIML [7], UIML [3, 6], XIML [100] , etc.) cuyo objetivo es la generación de *interfaces de usuario* para diferentes plataformas o dispositivos partiendo de una especificación única de *Modelos de la IU*:

- TERESA (*Transformation Environment for inteRactivE Systems representAtions*) [89, 78] es una herramienta que soporta el *MB-UID* para múltiples plataformas. Algunas de las plataformas para las que genera *interfaces de usuario* son HTML, WML y VoiceXML. La herramienta parte de un *Modelo de Tareas* (en notación CTT) que es transformado de forma semi-automática para producir las *interfaces de usuario* multiplataforma. Mediante el filtrado de este modelo inicial, se crean *Modelos de Tareas* específicos para cada plataforma particular. En este proceso de filtrado se pueden añadir nuevas tareas para ciertas plataformas así como reestructurar los árboles de tareas. Luego se generan Conjuntos de Tareas Habilitadas (*Enabled Task Sets*) partiendo del modelo de tareas del sistema. Estos

conjuntos representan tareas habilitadas simultáneamente que combinadas con algunas heurísticas producen conjuntos de tareas de presentación. Los conjuntos de tareas de presentación son la base para la generación de la *IU Abstracta* y de ésta se derivan sucesivamente la *IU Concreta* y la *IU Final*.

- El sistema Dygimes (*Dynamically Generating Interfaces for Mobile and Embedded Systems*) [61, 15] es un sistema usado para generar *interfaces de usuario* para sistemas empotrados y dispositivos en movilidad. Este sistema genera la *interfaz de usuario* en tiempo de ejecución. También parte de un *Modelo de Tareas* del que se deriva un *Modelo de Diálogo* que dirige el desarrollo de la *interfaz de usuario*. El primer paso para obtener el *Modelo de Diálogo* consiste en identificar Conjuntos de Tareas Habilitadas en el *Modelo de Tareas*. Esos conjuntos son usados para crear Redes de Transición de Estado, que indican las transiciones entre los distintos conjuntos de tareas. Las transiciones son calculadas automáticamente utilizando operadores temporales del *Modelo de Tareas*. Finalmente las redes de transición de estado ayudan en el diálogo entre los elementos de la *interfaz de usuario* que representan las tareas.

- XMobile [125] genera de una forma semiautomática aplicaciones adaptativas para dispositivos en movilidad. El sistema está compuesto por el *framework* XFormUI y el generador de código UIG (*Use Interface Generator*). XFormUI es un *toolkit* para la creación de *interfaces de usuario* gráficas independientes del dispositivo y de la plataforma de programación. Permite a los ingenieros describir las *interfaces de usuario* en Xhtml, XForms y CSS. El desarrollador también debe especificar la navegación de la aplicación. Partiendo de esta información, el generador de código (UIG) crea de forma automática la *IU Final* para diferentes plataformas. El generador utiliza para ello una serie de reglas de mapeo que describen cómo debe adaptarse la *interfaz de usuario* a cada tipo de dispositivo y cada plataforma de desarrollo.

Las aportaciones de este grupo permiten emplear una misma especificación para la generación de *IUs Finales* para distintas plataformas/contextos. Representan desde luego un importante avance para la reutilización de las *interfaces de usuario*. Sin embargo la reutilización de partes del modelado en proyectos futuros, sólo sería posible si los *Lenguajes de Descripción de la IU* utilizados en el desarrollo de la *interfaz de usuario* soportan la encapsulación y distribución de los elementos que conforman los modelos y como hemos visto en el punto anterior, sólo algunos *Lenguajes de Descripción de la IU* ofrecen mecanismos para la reutilización de especificaciones de *Modelos de la IU*.

2.2.3. Métodos de reutilización basados en patrones

El *MB-UID* permite crear *interfaces de usuario* partiendo de modelos conceptuales abstractos sin tratar aspectos técnicos de bajo nivel, sin embargo la especificación de los modelos es una tarea compleja y pesada.

Las propuestas basadas en patrones encapsulan soluciones frecuentes formando bloques constructivos que los desarrolladores combinan para crear los *Modelos de la IU*.

Mediante este sistema se mejoran la reutilización y la legibilidad y se reduce la complejidad [104] en el *MB-UID*. Las propuestas más relevantes en este ámbito son:

- PIM (*Patterns In Modelling*) *tool* [104] trata de ayudar a los desarrolladores de *interfaces de usuario* en la creación de *Modelos de la IU* a través de la aplicación de patrones. Como los patrones son genéricos es necesario instanciarlos. Generalmente la aplicación de un patrón consta de cuatro pasos:

  1. Identificación: se identifica un subconjunto M' del modelo objetivo M.
  2. Selección: se selecciona un patrón apropiado para aplicarlo a M'.
  3. Instanciación: se instancia el patrón para aplicarlo al modelo. En este paso se define la estructura concreta del patrón y se asignan valores a las partes variables del patrón para adaptarlo al caso concreto.
  4. Integración: la instancia es integrada en el modelo objetivo M. Se conecta al resto de partes del modelo para crear un diseño apropiado.

  PIM *tool* ayuda al desarrollador en el proceso de aplicar los patrones a los modelos siguiendo la secuencia anterior.

- El Proceso para el Desarrollo Sistemático de la *interfaz de usuario* Basado en Patrones y Componentes (*Pattern and Component-Based Process for Systematic UI Development*, PCB) [113] unifica las metodologías de patrones y componentes en un proceso de desarrollo:

  - En primer lugar se seleccionan los patrones necesarios para abordar diferentes aspectos de diseño de la *interfaz de usuario*. Existen diferentes tipos de patrones (*Tarea*, *Diálogo*, *Presentación* y *Layout*). Luego se ajustan estos patrones al contexto de uso, obteniéndose un conjunto de patrones instanciados. Esta fase de adaptación puede ser vista como un afinamiento básico de la futura aplicación sin ser específico de la implementación y/o lenguaje de programación.
  - A continuación, las instancias de los patrones se usan como bloques constructivos de los distintos *Modelos de la IU*. A partir de esta etapa se comienza a implementar la *interfaz de usuario*. Primero se seleccionan los componentes necesarios para la implementación. Dado que los modelos están formados por patrones se deben seleccionar los componentes necesarios para implementar esos patrones. Finalmente, los componentes deben ser adaptados al contexto de uso.

  Tanto los patrones como los componentes, se emplean como vehículos para la reutilización: los patrones se usan como un mecanismo para la reutilización del diseño y como bloques de construcción que permiten establecer modelos conceptuales de la *interfaz de usuario*; los componentes se utilizan como una herramienta orientada a la implementación de patrones.

- PD-MBUI (*Pattern-Driven and Model-Based UI*) [4] trata de reconciliar y unificar en un único *framework* las propuestas del *MB-UID* y del desarrollo dirigido por patrones. Los patrones son usados para superar las carencias de reutilización en la construcción y transformación de modelos. Se utilizan como bloques de construcción que pueden ser empleados para definir diferentes modelos que posteriormente son instanciados en elementos concretos de la *interfaz de usuario*.

- La Arquitectura Uniforme de Patrones de la *interfaz de usuario* (*Uniform UI Patterns Architecture*, UUIPA) [29] está basada en el patrón Modelo-Vista-Controlador. A través de una personalización visual, los patrones son instanciados como varias unidades de la *interfaz de usuario* que satisfacen las necesidades de los usuarios. Además, la propuesta, también se emplea para el desarrollo de *interfaces de usuario* multi-dispositivo extendiendo estilos que son ajustados a diferentes dispositivos gradualmente mientras se mantienen fijos los objetos de interacción.

Como hemos visto en estas propuestas, los patrones son empleados como medio para reutilizar diseños y/o construir *Modelos de la IU*. Algunas suelen estar complementadas por herramientas (p.ej. PIM *tool* [104]) u otros artefactos (p.ej. componentes en PCB [113]) para conseguir implementar la *interfaz de usuario*. Con estos métodos, reutilizar diseños no contemplados por el conjunto de patrones de partida puede ser difícil. Es más, los patrones, por supuesto, deben ser instanciados durante el proceso de desarrollo de la *interfaz de usuario* y una vez hecho esto se está trabajando en una solución particular más difícil de reutilizar.

2.3. Conclusiones

En este capítulo se ha expuesto el estado del arte sobre el *MB-UID* y sobre las propuestas de reutilización en este ámbito. En la primera sección se han presentado los avances en la investigación del *MB-UID*: las bases fundamentales de esta tecnología, su evolución histórica y el *framework* de referencia *Cameleon*. En la segunda sección del capítulo se ha tratado la problemática de la reutilización de forma general para, posteriormente, centrarnos en las propuestas de reutilización dentro en el *MB-UID*. Éstas han sido clasificadas en tres categorías: soporte para la reutilización en *Lenguajes de Descripción de la IU*, desarrollo de *interfaces de usuario* multi-contexto basado en modelos y métodos basados en patrones. En la tabla 2.4 se presenta una clasificación de estas propuestas indicando el tipo de técnica soportada (según [73]), el ámbito y la granularidad de la reutilización.

En el análisis que se ha realizado de las técnicas de reutilización existentes en el ámbito del *MB-UID*, ha quedado patente que su empleo presenta algunas limitaciones. Pero la problemática de la reutilización en el *MB-UID* es más profunda. Desde el punto de vista de algunos autores [104] muchos sistemas carecen de un concepto avanzado de la reutilización (más allá del *copy-and-paste-reuse*). Silva y Meixner [95, 70] argumentan que la falta de estandarización en la estructura y notación de los *Modelos de la IU* han dificultado la reutilización en el *MB-UID*. Sin embargo, aunque la reutilización ha sido

Categoría	Propuesta	Método principal	Ámbito y granularidad
Basados en el lenguaje	XUL	Composicional	interno o externo (de grano fino)
	XICL	Composicional, O.O.	
	UIML	Composicional, *templates*	
Multi-dispositivo	TERESA , XMobile, MARIA, GUMMY	Programación generativa	interno (grano grueso)
Basadas en patrones	PD-MBUID, PCB,	patrones	interno o externo (grano grueso)
	PIM tool, UUIPA	patrones, componentes	

Tabla 2.4: Propuestas para la reutilización en el MB-UID

identificada como un factor clave en el éxito y aceptación del *MB-UID* [4], no hay estudios que afronten específicamente esta temática.

Pensamos que se debe profundizar en la investigación de la reutilización en el ámbito del *MB-UID*. Es necesario proponer técnicas aplicables, identificar qué elementos de los *Modelos de la IU* y en qué circunstancias pueden ser reutilizados y analizar las consecuencias de la reutilización en el *MB-UID*.

Por último, es necesario recordar que son escasas las referencias a casos de aplicación con éxito del *MB-UID* al mundo industrial y que son insuficientes los casos de estudio que aparecen en la literatura [70]. Si añadimos la dimensión de la reutilización a este campo la ausencia de referencias es total: no existen casos de estudio ni experiencias prácticas publicadas. En esta situación es importante aportar casos de estudio, pero son aún más necesarias las experiencias que tratan la problemática y el análisis de la reutilización en el *MB-UID*.

Capítulo 3

Técnicas de reutilización en el *MB-UID*

Este capítulo presenta la primera contribución de esta tesis: un conjunto de técnicas genéricas aplicables a la reutilización de los objetos de la *interfaz de usuario* en el ámbito del *MB-UID*. Las técnicas provienen de diversos ámbitos de la ingeniería del software, pero nunca se ha realizado un compendio de estas técnicas ni se ha analizado su aplicación al *MB-UID*. Algunas de las técnicas pueden ser compatibles con las mencionadas en el estado del arte y podrían ser implementadas en *MB-UIDEs* presentes o futuros con algunas limitaciones que se discutirán a lo largo del capítulo.

Si observamos las distintas arquitecturas de la figura 2.4, los elementos que componen el modelado de la *interfaz de usuario* están presentes (de distinta forma) en las siguientes localizaciones: en las especificaciones de los *Modelos de la IU* (en todas las arquitecturas de la figura 2.4), en el código fuente de la *interfaz de usuario* (arquitectura *Generativa*, figura 2.4a), en los archivos (u otros soportes) utilizados por los *Run-times* (arquitectura *Mixta*, figura 2.4b) y por supuesto, en la memoria de la aplicación durante su ejecución (*Running User Interface*, en todas las arquitecturas). En cualquiera de estas localizaciones los objetos de la *interfaz de usuario* son potencialmente reutilizables.

Por la orientación de este trabajo no se tratará la reutilización de los elementos de la *interfaz de usuario* en el código fuente. Nos centraremos en la reutilización en las especificaciones y en lo que hemos denominado repositorios de *Modelos de la IU*, es decir, aquellos soportes distintos de un documento codificado en un *Lenguaje de Descripción de la IU* (como por ejemplo los archivos específicos para un UIMS que aparecen en las arquitecturas *Mixtas* de la figura 2.4) que son susceptibles de ser interpretados por un *Run-time de la IU*.

En las siguientes secciones se describirá cada una de las técnicas propuestas, clasificándolas de acuerdo con la localización en la que tiene lugar la reutilización: la especificación o el repositorio. Posteriormente, se realizará un análisis de los factores influyentes en la selección y aplicación de cada una de ellas.

3.1. Técnicas de reutilización en especificaciones

A lo largo de esta sección se describen métodos que permiten reutilizar especificaciones completas o fragmentos de las mismas. Algunos de los métodos propuestos tratarán de solventar algunas de las limitaciones comentadas en la sección 2.2.

3.1.1. Técnicas soportadas por el lenguaje de especificación

Los lenguajes de programación han evolucionado para permitir a los desarrolladores emplear herramientas de reutilización cada vez más refinadas [33]; desde las subrutinas en lenguajes ensambladores, hasta módulos, clases, *frameworks*, etc. Herramientas similares deben estar presentes en los *Lenguajes de Descripción de la IU*. Su uso permitiría reducir el tamaño y la complejidad de las especificaciones, redundando en soluciones más compactas y menos susceptibles de provocar errores (*i.e.* de más calidad). Se proponen a continuación técnicas que pueden ser aplicadas en estos lenguajes en el momento de construir las especificaciones:

3.1.1.1. Subespecificación (S)

Al igual que en los lenguajes procedurales ocurre con los procedimientos o funciones, fragmentos que aparezcan repetidamente en una especificación deben ser definidos de forma "aislada" e invocados o referenciados posteriormente en otras partes de la especificación. A este método que permitiría crear nuevos elementos por composición de fragmentos de la especificación lo denominamos "subespecificación". Por ejemplo, si en el *Modelo de Tareas*, una tarea se repite varias veces, se podría definir como una subespecificación y ser referenciada cuantas veces sea necesario. Recordemos que algunos de los lenguajes presentados en la sección 2.2.1 (como XUL y UIML) permiten definir fragmentos de una especificación (p.ej. *templates*) y referenciarlos posteriormente cuando sea necesario.

3.1.1.2. Templates (T)

Un *template* (o plantilla) se puede definir como una subespecificación en la que además, ciertas partes de la misma pueden ser sustituidas por ciertos valores mediante el empleo de parámetros. Hemos visto en el estado del arte (sección 2.2.1) que UIML permite la definición de *templates* y que admite que determinados valores sean trasladados al *template* empleando parámetros. Recordemos que la reutilización basada en *templates* está identificada como uno de los métodos de reutilización para diversos elementos [73] (código, documentación, etc.) y que ha sido empleada en otros ámbitos y lenguajes [46, 16].

3.1.1.3. Adaptación (A)

La adaptación permite transformar un elemento contenido en una subespecificación en otro distinto tras aplicarle una configuración determinada. La diferencia respecto al método anterior consiste en que mientras que el código de los *templates* es estático

y los valores de los parámetros, podríamos decir que son simplemente sustituidos en ese código, la adaptación pemitiría, mediante la parametrización del elemento origen, una reutilización más flexible del elemento reutilizado (más adaptativa) modificando su composición, estructura y/o comportamiento. La figura 3.1 muestra cómo una ventana (*IU Concreta*) se obtiene por adaptación de una existente (*sample*). Observe que la ventana original ha sido adaptada habilitando o deshabilitando los elementos que la componen utilizando los parámetros $field_i$ y $action_i$.

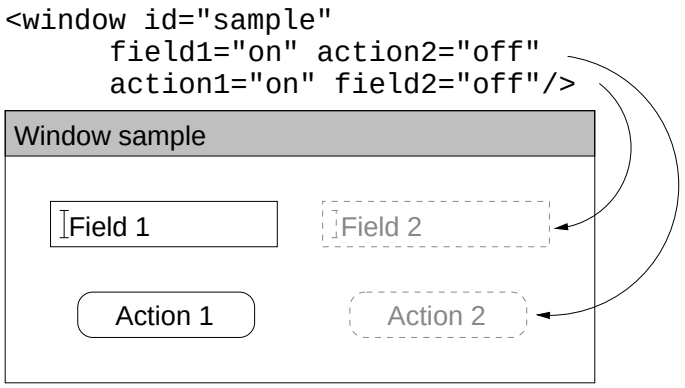


Figura 3.1: Nueva ventana obtenida por adaptación

3.1.1.4. Herencia (H)

Algunos *Modelos de la IU* se definen empleando clases y objetos. Parece lógico en estos casos, que los mecanismos de reutilización básicos de la *Programación Orientada a Objetos*, como la herencia [40], puedan ser empleados en la definición de los modelos de forma natural. Por ejemplo, en la sección 2.2.1, hemos visto, que XICL permite emplear el mecanismo de herencia en la definición de nuevos componentes de la *interfaz de usuario* (atributo *extends* en la etiqueta <component>). También es habitual que los *widgets* sean definidos como clases en *toolkits* o bibliotecas gráficas, permitiendo definir nuevos *widgets* de manera sencilla empleando el mecanismo de herencia. Por último, se ha de mencionar que existen lenguajes como CSS que definen relaciones "no ortodoxas" de herencia en este caso en combinación con HTML (*CSS Inheritance* [37]).

Naturalmente, la aplicabilidad de los métodos anteriores (S, T, A y H) dependerá del soporte de los *Lenguajes de Descripción de la IU* empleados por el *MB-UID*. Aquellos lenguajes que ofrezcan estas técnicas, obtendrán de forma inmediata ventajas como la reducción de tamaño de las especificaciones, el decremento de la complejidad de las mismas y la simplificación de su mantenimiento.

Hay que tener en cuenta que añadir estos mecanismos a un lenguaje existente puede ser enormemente costoso, por no decir imposible de llevar a cabo sin realizar una reestructuración profunda del lenguaje, por lo tanto, recomendamos que estos métodos sean tenidos en cuenta en el diseño de los futuros *Lenguajes de Descripción de la IU*. Por último, hay que destacar que estos métodos por sí mismos no permiten la reutilización horizontal (*Inter-proyecto*) ya que en principio, referencian o emplean elementos que forman parte de la misma especificación.

3.1.2. Inclusión (I)

Muchos lenguajes de programación poseen una directiva, denominada de forma habitual *include* [116] (también *copy*, *require*, etc.) que causa que el contenido de un segundo archivo sea insertado en el archivo original. Este mecanismo en algunos casos se apoya en elementos externos (p.ej. un preprocesador) y se ha usado para definir estructuras de datos comunes, realizar declaraciones adelantadas, encapsular y reutilizar código, etc. En lenguajes más modernos la inclusión ha evolucionado hacia sistemas más avanzados (p.ej. paquetes en Ada [118] o Java [43]) que resuelven el problema de la compilación separada y manejan conceptos avanzados como los espacios de nombres o la visibilidad/ocultación de definiciones.

En la sección 2.2.1, hemos visto que algunos *Lenguajes de Descripción de la IU* como XUL y XICL poseen directivas (*xul-overlay* e *import* respectivamente) con un uso similar. Por su parte, UIML permite referenciar *templates* tanto internos como externos a través de la URI indicada en el atributo *source*, pero en cualquier caso, si el recurso a reutilizar es externo, éste debe ser recuperado y añadido a la especificación actual de alguna forma.

El mecanismo de inclusión permite en primer lugar que especificaciones de gran tamaño puedan ser modularizadas en fragmentos más sencillos de manejar, mantener y comprender. También posibilita que cualquier componente reutilizable de una especificación pueda ser definido en su propio fichero y que éste sea incluido en cualquier otra especificación que requiera usarlo, facilitando la reutilización horizontal. Recordemos que las técnicas de la sección 3.1.1 (S, T, A y H), no permiten por sí mismas la reutilización *Inter-proyecto*, por lo tanto, pueden ser combinadas con alguna forma de inclusión para resolver esta limitación.

Es necesario tener en cuenta que la inclusión presenta un problema que podemos denominar "fusión". Cada vez que un elemento reutilizable sea incluido en un nuevo proyecto, ese elemento será "fusionado" en la nueva especificación, dificultando las posteriores tareas de mantenimiento y actualización del componente que deben ser realizadas en cada proyecto donde el componente ha sido incluido. De hecho, en la mayoría de los casos será necesario volver a regenerar modelos, refinarlos o volver a generar el código de la *interfaz de usuario*.

3.1.3. Paquetes (P)

De forma general, podemos decir que un *Sistema de Gestión de Paquetes* (*Package Management System*) es un método de distribuir, instalar y gestionar componentes en un sistema informático. Los paquetes son utilizados hoy día con múltiples propósitos como por ejemplo el archivado de ficheros con o sin compresión (p.ej. TAR, ZIP, CAB), la distribución de software ejecutable (p.ej. MSI, RPM, DEB), o la distribución de clases y unidades en diversos lenguajes de programación (p.ej. JAR, WAR, EAR, PPM).

En la sección 2.1 hemos visto que en el *MB-UID* se pueden emplear varios *Lenguajes de Descripción de la IU* tanto en el mismo nivel de abstracción como a distintos niveles. Es más, cuando se trabaja con varios lenguajes podría ocurrir que la definición de un componente complejo esté compuesta por fragmentos pertenecientes a distintos modelos

que a su vez han sido definidos en distintos lenguajes. Los paquetes permitirían unificar todos los fragmentos necesarios en una única entidad. También ayudarían a encapsular y distribuir el código de los componentes reutilizables en escenarios *Inter-proyecto* de forma cómoda y segura (*i.e.* cifrados y/o firmados). Además, un paquete puede contener cualquier información adicional sobre el componente reutilizable de la *interfaz de usuario*. Se puede incluir en el paquete documentación, diagramas explicativos, ejemplos de uso o cualquier otra información relevante para la explotación del componente. Esto permitiría realizar *asset-based development*[1] [57] en la *interfaz de usuario*.

Hay que destacar que para poder emplear paquetes es necesario implementar un *Sistema de Gestión de Paquetes*. En absoluto es un asunto trivial, ya que se deben abordar problemas complejos como las dependencias de paquetes [47], conflictos de versiones entre los mismos o aspectos relativos a la seguridad [63]. Además, al igual que en la inclusión, los recursos reutilizados serán "fusionados" en las nuevas especificaciones y los cambios realizados en los recursos originales no se trasladarán a los sistemas que hacen uso de ellos.

Finalmente, se ha de tener en cuenta que las potenciales implementaciones que se pudieran realizar de un *Sistema de Gestión de Paquetes*, muy probablemente, serían dependientes de la plataforma/*MB-UIDE* y por lo tanto difícilmente aplicables a entornos diferentes.

3.1.4. Bibliotecas de reutilización (B)

Las bibliotecas de reutilización están formadas por un almacén de recursos y algún método para la búsqueda en el almacén junto a otras utilidades y herramientas. En el ámbito de este trabajo, los recursos reutilizables serían especificaciones de *Modelos de la IU*.

En los apartados anteriores hemos visto dos técnicas que pueden servir de base para la creación de bibliotecas de recursos reutilizables de la *interfaz de usuario*: la inclusión y los paquetes. Ambos métodos permitirían emplear elementos de la *interfaz de usuario* desarrollados con anterioridad en nuevos proyectos. Las bibliotecas de reutilización mantendrían un repositorio organizado (directorio, servidor web, etc.) de elementos reutilizables de la *interfaz de usuario* (contenidos en archivos o paquetes).

El empleo de bibliotecas de reutilización implica un compromiso (al menos una concienciación) de la organización en la gestión de la reutilización. Se incluye la reutilización como un proceso básico dentro del desarrollo, indagando sobre qué elementos pueden ser reutilizados en el futuro y manteniéndolos organizados en un repositorio.

[1] El desarrollo basado en recursos (*asset-based development*) organiza las "inversiones" (*investments*) relativas al software, requisitos, modelos, código, pruebas y *scripts* de despliegue para ser usadas en futuros proyectos software.

3.2. Técnicas de reutilización en repositorios de modelos

En los *MB-UIDEs* con arquitectura de tipo *Run-time* (arquitecturas *Mixta* e *Interpretativa*, ver figuras 2.4b y 2.4c) los modelos involucrados en la generación de la *interfaz de usuario* (o al menos algunos de ellos) pueden residir en ficheros o algún tipo de base de datos (para nosotros a partir de ahora repositorios de *Modelos de la IU*). Estos repositorios pueden convertirse en un espacio en el que compartir/reutilizar elementos de la *interfaz de usuario* en escenarios *Inter-proyecto*, utilizando técnicas habitualmente empleadas en otros tipos de almacenes como bases de datos, directorios de usuarios, etc. Para que esta propuesta sea factible, los repositorios deben permitir el acceso simultáneo de varios *Run-times de la IU* y estos últimos deben ser capaces de acceder a varios repositorios. Evidentemente como cada objeto residirá en un único repositorio, se elimina el problema de la fusión presente en la inclusión y en los paquetes (ver secciones 3.1.2 y 3.1.3). Una restricción para la posible aplicación de los siguientes métodos es que el acceso a los repositorios compartidos debe ser suficientemente rápido para no afectar al tiempo de respuesta de la *interfaz de usuario*.

3.2.1. Paquetes (P)

En la sección anterior se ha propuesto el uso de paquetes para reutilizar fragmentos de especificaciones (subsección 3.1.3). Sin embargo, los paquetes también podrían funcionar sobre repositorios. Recordemos que un *Sistema de Gestión de Paquetes* es un software complejo capaz de realizar muchas funciones. En este caso particular, el *Sistema de Gestión de Paquetes* podría encargarse de realizar las transformaciones necesarias sobre los fragmentos de las especificaciones contenidos en el paquete para prepararlos y dejarlos operativos en los repositorios empleados por el *MB-UIDE*.

En las arquitecturas *Interpretativas* las especificaciones contenidas en el paquete prácticamente sólo necesitarían ser "vertidas" al repositorio de especificaciones. En las arquitecturas *Mixtas* las especificaciones serían trasformadas al formato empleado por el *Run-time* y añadidas al repositorio.

3.2.2. Repositorios centralizados (C)

Centralizar puede definirse como el acto de consolidar recursos bajo un control central. Los sistemas centralizados han sido aplicados en la informática en multitud de ámbitos como las comunicaciones, las bases de datos o los servicios en general. En el ámbito del *MB-UID* se pueden consolidar en un repositorio central los elementos de los *Modelos de la IU* comunes a un conjunto de sistemas. Éste podría ser un caso frecuente en las aplicaciones corporativas de grandes organizaciones donde el aspecto de las aplicaciones y otros elementos como tareas, *unidades de interacción*, etc. pueden ser comunes a varios sistemas.

Como se puede apreciar en la figura 3.2 un repositorio puede almacenar los elementos comunes a varias aplicaciones y además, cada sistema puede emplear su repositorio local para almacenar los elementos particulares de su *interfaz de usuario*.

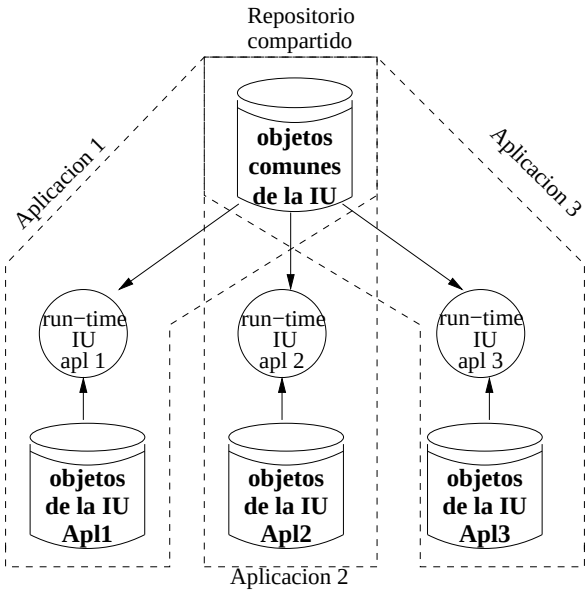


Figura 3.2: Arquitectura centralizada para compartir objetos de la interfaz de usuario

Las ventajas que presenta la centralización son muy interesantes. Se simplifica el mantenimiento de las *interfaces de usuario* siendo únicamente necesario realizar modificaciones en el repositorio común para mantener los recursos actualizados. Asimismo, la centralización de los *Modelos de la IU*, puede permitir implementar ciertos mecanismos de seguridad para el control de acceso y de cambios.

Por contra, el tiempo de respuesta de este tipo de sistemas podría verse sustancialmente degradado y de hecho, será totalmente dependiente de la comunicación con el repositorio central. Es más, los fallos o errores en los repositorios centralizados afectarán a cada sistema que haga uso de los mismos. Además, los desarrolladores que realicen actualizaciones sobre modelos compartidos podrían encontrar dificultades debidas a bloqueos de acceso sobre componentes que se encuentren en uso por otros usuarios o desarrolladores.

3.2.3. Repositorios federados (F)

Se entiende por federación un método que permite enlazar objetos de dos o más localizaciones diferentes, haciendo que el acceso/enlace sea transparente, como si los objetos estuvieran localizados en el mismo lugar. La federación se ha aplicado con éxito en campos como los directorios de usuarios [96] o las bases de datos [28]. Así, las bases de datos federadas son un tipo de "meta-basededatos" que integra múltiples bases de datos autónomas en una única base de datos, permaneciendo las bases de datos que constituyen la federación independientes y autónomas. De manera análoga, un nuevo *Modelo de la IU* puede ser construido por federación de elementos de modelos existentes, dejando el control y la gestión de los elementos que forman la federación en manos de sus respectivos administradores.

La figura 3.3 muestra cómo el *Run-time de la IU* de la aplicación 4 toma objetos

de los repositorios de varias aplicaciones.

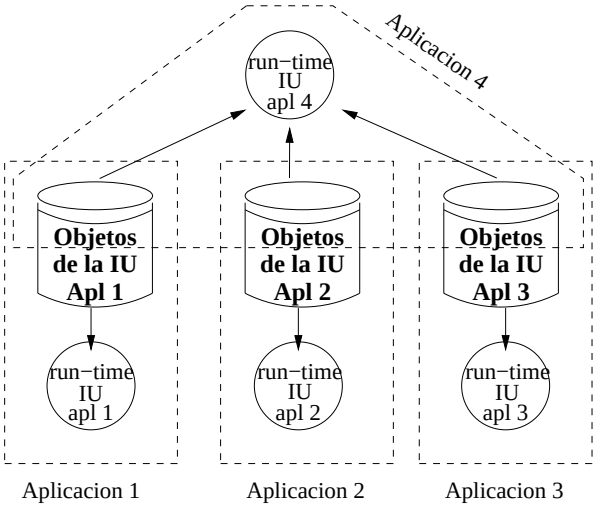


Figura 3.3: Arquitectura federada para compartir objetos de la interfaz de usuario

En los sistemas federados también se elimina el problema de la "fusión". Las actualizaciones sobre componentes compartidos sólo se realizarán sobre el repositorio que contiene el elemento y los cambios se trasladarán a todos los sistemas que hacen uso de él.

La federación será aconsejable cuando los *Modelos de la IU* o sus componentes reutilizables están distribuidos en diferentes dominios de responsabilidad. Los objetos federados de la *interfaz de usuario* permanecerán en su localización original, manteniéndose una vista uniforme de acceso para los usuarios, que no tienen consciencia de la existencia de múltiples sistemas independientes.

Por último, destaquemos que las arquitecturas federadas suelen ser más escalables que las centralizadas y distribuyen la carga de trabajo de manera eficiente.

Los sistemas federados también presentan limitaciones. La detección de cambios en los sistemas origen puede ser costosa, además, *a priori* no existe consistencia ni integridad referencial entre los objetos de los repositorios. Al igual que en sistemas centralizados, otro aspecto a tener en cuenta es la seguridad; para cada sistema será necesario determinar qué elementos son accesibles para el modelo federado y en qué condiciones.

La implementación de sistemas centralizados y federados puede llegar a ser extremadamente compleja. De forma general, los *MB-UIDEs* que formen parte de un sistema centralizado o federado pueden ser de distinto tipo, es decir, sistemas con estructuras, lenguajes y elementos constituyentes totalmente distintos. En este caso habría que tratar problemas como la homogeneización de los distintos *Modelos de la IU*, la realización de capas intermedias de adaptación/conversión de modelos, etc.

3.3. Factores a considerar en la selección y aplicación de las técnicas propuestas

En esta sección se analizan algunos factores que pueden influir a la hora de seleccionar o aplicar las técnicas de reutilización propuestas.

3.3.1. Arquitectura del MB-UIDE

La arquitectura empleada por el *MB-UIDE* para la generación de la *IU Final*, determinará cuál de las técnicas es aplicable.

Los *MB-UIDEs* que siguen la arquitectura *Generativa* de la figura 2.4a, producen el código fuente de la *interfaz de usuario* utilizando un *generador de código*. No poseen un *Run-time de la IU* ni emplean ningún tipo de almacén para los *Modelos de la IU* en tiempo de ejecución. El resultado del trabajo realizado sobre los modelos durante el desarrollo se ha enlazado con el resto de la aplicación y forma parte de un archivo ejecutable. En este tipo de arquitectura, las técnicas de reutilización para repositorios de *Modelos de la IU* (C y F) no son aplicables.

Por el contrario, las arquitecturas *Mixtas* e *Interpretativas* de la figura 2.4, emplean repositorios: el archivo o base de datos que soporta los *Modelos de la IU* pretratados de la figura 2.4b o directamente el conjunto de especificaciones de la figura 2.4c. En ambos casos el *Run-time de la IU* transforma en tiempo de ejecución las especificaciones (pretratadas o no) en la *IU Final* y por lo tanto se pueden aplicar las técnicas C y F.

Un aspecto a analizar es en qué momento se lleva a cabo la reutilización de los elementos de los *Modelos de la IU* en función de la arquitectura del *MB-UIDE*. En la tabla 3.1 se muestra un resumen indicando para cada aquitectura y cada técnica propuesta cuándo se puede llevar a cabo la reutilización del recurso: en tiempo de desarrollo (D) o en tiempo de ejecución (E).

Técnica	*Generativa* (a)	*Mixta* (b)	*Interpretativa* (c)
Subespecificación	D/E	D/E	E
Templates	D/E	D/E	E
Adaptación	D/E	D/E	E
Herencia	D/E	D/E	E
Inclusión	D	D	E
Paquetes	D	D	D
Bibliotecas	D	D	D
Centralización	-	E	E
Federación	-	E	E

Tabla 3.1: Resolución de la reutilización por arquitectura y técnica

Según la arquitectura del *MB-UIDE* existirán las siguientes opciones:

- En las arquitecturas de tipo *Interpretativa* (figura 2.4c), las especificaciones, definidas en sus lenguajes correspondientes, se interpretan en tiempo de ejecución.

- Los métodos de reutilización soportados por los lenguajes de especificación (S, T, A y H, sección 3.1) deben resolverse por lo tanto en tiempo de ejecución. Por ejemplo, los *templates* deben expandirse (sustitución de parámetros) y la herencia debe materializarse en tiempo de ejecución. Como no existe ninguna herramienta que realice un tratamiento previo de las especificaciones antes de que éstas sean utilizadas por el *Run-time de la IU*, las técnicas de reutilización para las especificaciones deben ser resueltas por el intérprete. Igualmente ocurre con la inclusión (I), debe ser resuelta en tiempo de ejecución dado que no existe otra forma de que sea tratada. Recordemos que XICL [18, 19] sigue una arquitectura de este tipo, un *Run-time* interpreta el lenguaje en tiempo de ejecución resolviendo la herencia y la inclusión.
- El despliegue de un paquete (P) o la utilización de bibliotecas de especificaciones (B) sólo tiene sentido en tiempo de desarrollo. Obviamente los desarrolladores seleccionarán qué componentes de la biblioteca o qué paquetes van a emplear durante el desarrollo de la *interfaz de usuario*, ya que en tiempo de ejecución esos elementos tienen que formar parte del repositorio de *Modelos de la IU*.
- Por último, los elementos que residan en repositorios compartidos (técnicas C y F) deben ser recuperados en tiempo de ejecución. Todas las transformaciones posteriores que se deban realizar sobre estos objetos hasta generar la *IU Final* serán desempeñadas por el *Run-time de la IU* en tiempo de ejecución.

- En arquitecturas *Mixtas* (figura 2.4b), el *Run-time* del sistema utiliza una forma pretratada de las especificaciones de los *Modelos de la IU*.

  - El nivel de las transformaciones realizadas en el pretratamiento de los modelos, hará que la reutilización en las técnicas S, T, A y H (sección 3.1) se resuelva en tiempo de desarrollo o en tiempo de ejecución. Un proceso pesado, puede resolver cuestiones como la expansión de *templates* dejando los elementos preparados en el repositorio. En el extremo opuesto, un proceso más ligero, puede trasladar imágenes levemente transformadas de los objetos originales al repositorio, dejando para el *Run-time* la resolución de todos los aspectos relacionados con la reutilización.
  - Sin embargo, la inclusión debe ser resuelta en tiempo de desarrollo, antes de que los *Modelos de la IU* sean pretratados. Igualmente ocurrirá con el empleo de paquetes y bibliotecas de reutilización. Los componentes deben ser seleccionados por los desarrolladores y añadidos a las especificaciones en tiempo de desarrollo, antes de que se realice el pretratamiento de las especificaciones.
  - Por último, en estos sistemas, los archivos que contienen las especificaciones transformadas pueden ser considerados un repositorio de *Modelos de la IU*. En estos repositorios sería posible reutilizar elementos de *interfaz de usuario* mediante las ténicas C y F en tiempo de ejecución. El *Run-time de la IU* sería

el responsable de localizar elementos en los repositorios remotos, recuperarlos y transformarlos.

- Para finalizar, en las arquitecturas *Generativas* (figura 2.4a), un generador de código crea el código fuente de la *interfaz de usuario* partiendo de las especificaciones. El código de la *interfaz de usuario* es combinado posteriormente con el código principal de la aplicación y es compilado añadiendo entre otras la biblioteca del *toolkit* para el que se ha generado la *interfaz de usuario*.

  - Al igual que en el caso anterior, la reutilización en las técnicas S, T, A y H puede resolverse durante el proceso de generación de código o puede dejarse esta tarea para que se realice en tiempo de ejecución con ayuda del *toolkit* gráfico.
  - También los procesos relacionados con la reutilización mediante las técnicas I, P y B deben ser aplicados en tiempo de desarrollo.

Antes de concluir este apartado, hemos de tener en cuenta que las decisiones que se tomen sobre dónde realizar la reutilización pueden tener impacto en la flexibilidad y prestaciones del *MB-UIDE*:

- En las arquitecturas *Generativas*, resolver ciertas técnicas de reutilización en tiempo de desarrollo o de ejecución puede estar determinado por el *toolkit* seleccionado para el desarrollo de la *interfaz de usuario*.

- En las *Mixtas*, aplicar la reutilización en tiempo de desarrollo en ciertas técnicas permitirá simplificar la construcción del *Run-time* del *MB-UIDE* y seguramente se obtendrán mejores rendimientos en la generación de la *IU Final*. Sin embargo, esta decisión puede perjudicar al empleo de las técnicas C y F en estos sistemas, ya que los elementos en los repositorios estarán pretratados para su uso en una situación específica, menos útiles para un uso general y menos reutilizables por lo tanto.

3.3.2. Estructura de los recursos reutilizables

Un recurso reutilizable de la *interfaz de usuario* puede estar definido en el dominio de un único modelo (p.ej. una *unidad de interacción* de la *IU Abstracta*) o puede estar compuesto por un conjunto de elementos de varios modelos (p.ej. la *interfaz de usuario* de un pequeño sistema de mensajería formado por elementos del *Modelo de Diálogo*, del *Modelo de Presentación* y del *Modelo de Dominio*). Según su composición, podemos denominar a estos componentes *Recursos Mono-modelo* y *Recursos Multi-modelo* respectivamente (ver figura 3.4).

La aplicabilidad de las técnicas de reutilización propuestas en este documento estará limitada por la estructura y composición del recurso reutilizable (ver tabla 3.2):

- Las Técnicas de reutilización en especificaciones (S, T, A y H, sección 3.1), por definición funcionan referenciando a un fragmento de especificación o a un *template* o extendiendo una clase padre, es decir sólo son aplicables sobre *Recursos Mono-modelo*.

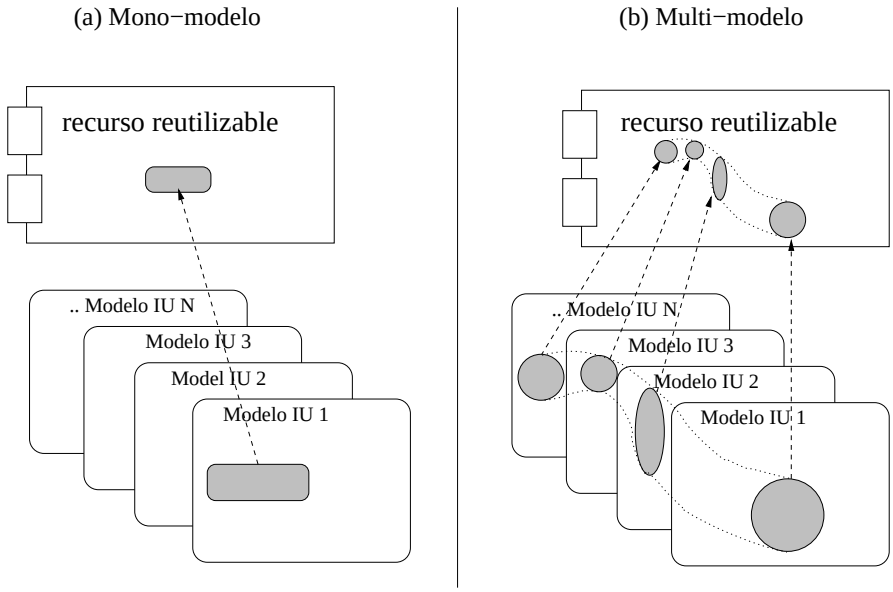


Figura 3.4: Recursos reutilizables de la interfaz de usuario

- La inclusión (I) puede emplearse para reutilizar *Recursos Mono-modelo* o *Recursos Multi-modelo* con limitaciones. Su aplicación sobre este último tipo de recurso sólo será posible siempre que todos los componentes del recurso reutilizable sean definidos en el mismo lenguaje y documento XML.

- Los paquetes (P) están dirigidos a la reutilización de elementos heterogéneos, por lo tanto son apropiados para reutilizar ambos tipos de recursos.

- Las bibliotecas de reutilización (B) estarán limitadas por el soporte empleado para almacenar los recursos reutilizables (I o P).

- Los métodos sobre repositorios compartidos (C y F). No están limitados por la estructura del recurso reutilizable.

También es interesante analizar qué técnicas pueden ser implementadas y en qué momento pueden resolverse, tiempo de desarrollo (D) o tiempo de ejecución (E), en función de la arquitectura del *MB-UID* y la estructura del recurso reutilizable. En la tabla 3.3 se presenta un resumen de las conclusiones de este análisis:

- Todas las técnicas propuestas para la reutilización sobre especificaciones serían aplicables en tiempo de desarrollo en las arquitecturas *Generativas* y *Mixtas* sobre *Recursos Mono-modelo*. Esto es debido a que ambas arquitecturas disponen de generadores de código que pueden encargarse de resolver estos tipos de reutilización. Sin embargo, en las arquitecturas *Interpretativas* que no disponen de generador de código y en el que las especificaciones son interpretadas directamente, las únicas técnicas que se pueden (y deben) aplicar en tiempo de desarrollo son las técnicas P y B.

- Para la reutilización de *Recursos Mono-modelo* en tiempo de ejecución, podrán emplearse en cualquier caso las técnicas S, T, A y H; en las arquitecturas *Interpretativas* y *Mixtas* tratadas por el *Run-time*; en las *Generativas* tratadas por la biblioteca del *toolkit*. Las arquitecturas *Interpretativas* y *Mixtas* al disponer de *Run-time* y repositorios, también podrán emplear las técnicas C y F. Únicamente los *MB-UIDEs* con arquitectura *Interpretativa* tendrán obligatoriamente que resolver la técnica I en tiempo de ejecución.

- La reutilización de *Recursos Multi-modelo* en tiempo de desarrollo, podrá realizarse con las técnicas P y B sobre cualquier arquitectura. En aquellas que empleen generadores de código, arquitecturas *Generativas* y *Mixtas*, la inclusión también podrá ser utilizada. Recordemos que la aplicación de las técnicas I y P sobre *Recursos Multi-modelo* está limitada en el primer caso por el lenguaje (debe soportar los modelos necesarios) y en el segundo por el artefacto empleado para almacenar el recurso reutilizable en la biblioteca (ficheros o paquetes).

- En último lugar, la reutilización de *Recursos Multi-modelo* en tiempo de ejecución sólo será posible en las arquitecturas *Interpretativas* y *Mixtas* que son aquellas que emplean repositorios de modelos. Las técnicas C y F podrán emplearse sobre ambas arquitecturas y la técnica I sólo en el caso de la arquitectura *Interpretativa* (con las limitaciones conocidas).

Antes de finalizar este apartado, es necesario mencionar, que en la reutilización de *Recursos Multi-modelo*, la relación entre los elementos de cada modelo y el grado de acoplamiento y dependencia entre los *Modelos de la IU* involucrados tendrá también impacto sobre el potencial de reutilización.

3.3.3. Dominio de reutilización y otros aspectos

El dominio indica la forma y el alcance de la reutilización. En el *MB-UID* podemos hablar de un dominio horizontal cuando pretendemos reutilizar elementos entre distintas

Técnica	Mono-modelo	Multi-modelo
Subespecificación	✓	
Templates	✓	
Adaptación	✓	
Herencia	✓	
Inclusión	✓	(1)
Paquetes	✓	✓
Bibliotecas	✓	(2)
Centralización	✓	✓
Federación	✓	✓

(1) Limitado por el lenguaje XML
(2) Limitado por soporte del elemento reutilizable

Tabla 3.2: Aplicación de las técnicas según el tipo de recurso

		Arquitecturas		
		Generativa (a)	*Mixta* (b)	*Interpretativa* (c)
Mono-modelo	D	S, T, A, H, I, P, B		P, B
	E	S, T, A, H	S, T, A, H, C, F	S, T, A, H, I, C, F
Multi-modelo	D	I*, P, B**		P, B**
	E	—	C, F	I*,C, F

(*) Limitado por el lenguaje XML
(**) Limitado por soporte del elemento reutilizable

Tabla 3.3: Técnicas por arquitectura y composición del recurso

especificaciones de *interfaces de usuario* y vertical cuando la reutilización se realiza en la misma especificación.

En la tabla 3.4 se muestra un resumen en el que se indica para cada técnica el ámbito de reutilización soportado. Recordemos que las técnicas S, T, A y H por sí mismas no permiten la reutilización *Inter-proyecto*, es decir, están más orientadas a su empleo en dominios verticales (dentro de la misma especificación). La inclusión (I), permite que elementos de la *interfaz de usuario* sean reutilizados tanto dentro de la misma especificación como entre distintas especificaciones. Sin embargo, las técnicas P y B por definición tienen el objetivo de permitir reutilizar recursos entre distintos proyectos, esto es, en dominios horizontales. En último lugar las técnicas sobre repositorios posibilitan la compartición de objetos entre distintos sistemas, es decir su objetivo es la reutilización horizontal.

Técnica	Ámbito
Subespecificación	Vertical
Templates	Vertical
Adaptación	Vertical
Herencia	Vertical
Inclusión	Vertical/Horizontal
Paquetes	Horizontal
Bibliotecas	Horizontal
Centralización	Horizontal
Federación	Horizontal

Tabla 3.4: Ámbito de aplicación más apropiado para cada técnica

Para finalizar este análisis, hemos de tener en cuenta que además de los factores expuestos anteriormente, las características de los lenguajes empleados por el *MB-UIDE* determinarán qué técnicas de reutilización pueden ser implementadas sobre especificaciones.

3.4. Conclusiones

En este capítulo se han propuesto una serie de técnicas para la reutilización en el *MB-UID*: algunas aplicables sobre especificaciones y otras sobre repositorios de *Modelos de la IU*. Las técnicas presentadas han sido aplicadas en otros ámbitos de la informática, sin embargo, nunca se había realizado un compendio de técnicas aplicables al *MB-UID*, ni se habían analizado las consecuencias de su implantación en este ámbito.

Respecto al análisis realizado podemos destacar que:

- La arquitectura del *MB-UIDE* tiene una influencia directa en las técnicas de reutilización aplicables y en las posibilidades para su implementación.
- La aplicación de determinadas técnicas está limitada por estructura y composición del recurso reutilizable.
- La expresividad y sintaxis de cada lenguaje así como la semántica de cada *Modelo de la IU*, tendrá un impacto decisivo sobre cuáles serán los componentes reutilizables en cada entorno.

Capítulo 4

WAINE

En este capítulo se presenta WAINE, el *MB-UIDE* empleado para experimentar con las técnicas de reutilización propuestas en esta tesis. En esta introducción a WAINE conoceremos sus características principales: los modelos que utiliza, los lenguajes empleados, sus componentes y herramientas, etc. Posteriormente, veremos cómo algunas de las técnicas propuestas en la sección anterior han sido implementadas en este *MB-UIDE*.

4.1. Introducción a WAINE

WAINE (*Web Application INterface Engine*) [20, 21] es un *MB-UIDE* que genera aplicaciones web basadas en el paradigma del formulario. WAINE fue diseñado como un proyecto académico orientado a facilitar el desarrollo sistemático de la *interfaz de usuario* a estudiantes de ingeniería que no estuvieran familiarizados con la plétora de tecnologías web actuales. Uno de los objetivos de diseño fue simplificar y acelerar el proceso de desarrollo eliminando la necesidad de definir aspectos concretos de la *interfaz de usuario* y soportando de forma proactiva la reutilización de las especificaciones.

Entre los criterios iniciales de diseño se encuentran los siguientes:

- *Interfaz Web*. Actualmente existen navegadores web prácticamente en cada dispositivo y plataforma. Las interfaces web permiten el acceso a los sistemas desde cualquier localización en la red y reducen casi a cero los costes de instalación y distribución de la aplicación.

- *Mínima programación posible*. La programación debería reducirse al mínimo. De forma ideal, se debería carecer de cualquier tipo de código, excepto claro está, el propio de los lenguajes declarativos empleados por el *MB-UIDE*.

- *Independencia*. Es un factor clave que engloba a otros como: independencia del sistema operativo, del origen de los datos, del servidor web, de los navegadores y de las herramientas empleadas para la especificación de los *Modelos de la IU*.

- *Seguridad*. Es otro factor de gran importancia, pero que creemos que no se ha tratado con la profundidad necesaria en el ámbito del *MB-UID*. Un sistema actual

debe contemplar distintos métodos de autenticación de usuarios y garantizar que éstos acceden únicamente a los objetos a los cuales se les ha dado acceso y en el modo asignado.

- *Flexibilidad*. El sistema debe ser fácilmente extensible y adaptable a modificaciones del entorno: nuevos orígenes de datos, *widgets*, sistemas de autenticación, etc.

- *Eficiencia*. La eficiencia es una característica siempre deseable, pero en este caso particular el sistema debe ser capaz de ejecutarse en equipos de prestaciones reducidas o en sistemas empotrados.

En el resto de la sección se presentan los componentes del *MB-UIDE*: el conjunto de *Modelos de la IU* empleados, los lenguajes en los que se define cada modelo, los repositorios de modelos, el *Run-time de la IU* y finalmente el proceso y las herramientas de desarrollo.

4.1.1. Modelos de la interfaz de usuario

WAINE trata de simplificar al máximo el modelo de una aplicación de gestión. En la figura 4.1 se puede observar un esquema del tipo aplicación soportada por el *MB-UIDE*. En este diagrama informal, se puede apreciar que existen usuarios organizados en grupos. Cada uno de ellos accede a las funcionalidades que ofrece la aplicación a través de las opciones de un menú. Las opciones del menú pueden lanzar acciones o mostrar un espacio de trabajo (construido con formularios) que también puede invocar a acciones a través de botones o eventos.

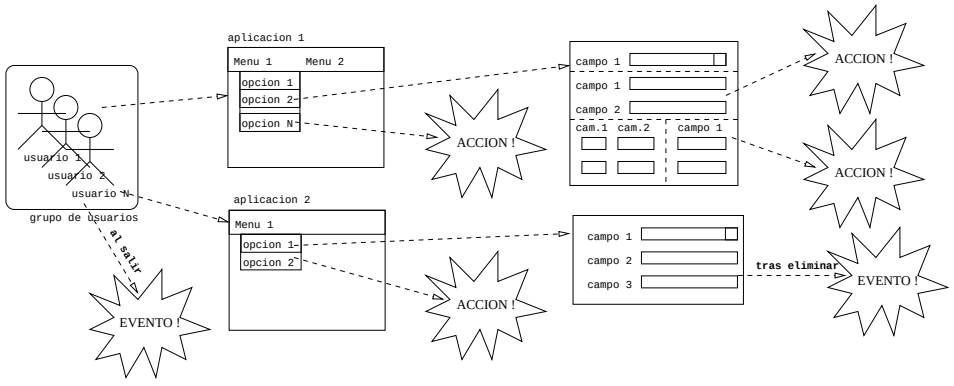


Figura 4.1: Diagrama informal del tipo de aplicación soportada por WAINE

La *interfaz de usuario* de una aplicación es definida por un conjunto reducido de modelos. La figura 4.2(b), muestra los modelos empleados por WAINE y su relación con el *framework* de referencia *Cameleon* (a).

El *Modelo de Dominio* de WAINE describe los datos que los usuarios manejan empleando la *interfaz de usuario* y es especificado a través de un *Diagrama Entidad-Relación* (ver la figura 4.2c). Sus componentes principales son entidades y relaciones.

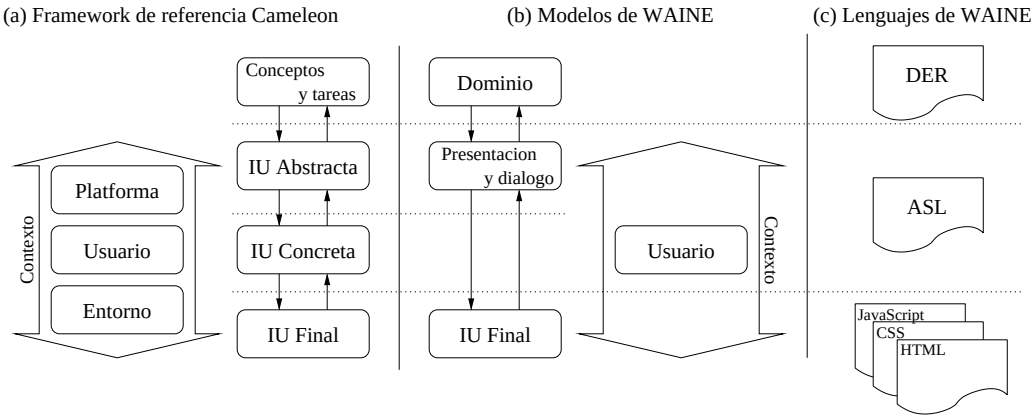


Figura 4.2: Modelos y lenguajes de WAINE relacionados con el framework Cameleon

A un nivel de abstracción inferior de acuerdo con el CRF, los siguientes modelos (cuyos componentes y estructuras aparecen ilustrados en la figura 4.3) determinan más directamente el contenido y apariencia de la *interfaz de usuario*.

- El *Modelo de Presentación*. Describe principalmente la *IU Abstracta* a través de la definición de espacios de trabajo mediante *Objetos Abstractos de Interacción*. El *Modelo de Presentación* posee dos constructores principales: los formularios y los contenedores.

    1. Un formulario con sus campos, muestra y manipula una entidad del *Modelo de Dominio* (o parte de ella) y permite la ejecución de acciones [1] a través de eventos (disparadas por el sistema ante una interacción del usuario) o de lanzadores de acciones (disparadas de forma directa por los usuarios). Los campos de un formulario (*Objetos Abstractos de Interacción*) se caracterizan por su tipo y por su origen de datos. El tipo define el dominio del campo, es decir el conjunto de valores que el campo admite. El origen del campo es el atributo de la entidad vinculado al campo. Algunas veces el origen de un campo puede ser una expresión en la que intervienen valores de otros campos, en este caso les llamamos "campos calculados".
    2. Los contenedores agregan formularios y/o contenedores para definir la estructura, contenido y comportamiento básico de una unidad de interacción [122] (*i.e.* un espacio de trabajo tal como se muestra en la figura 4.11). Así, una *unidad de interacción* es definida como una estructura jerárquica de contenedores y formularios y es identificada por el conenedor de más alto nivel (contenedor raíz). Los contenedores pueden ser de tipo *form* (para definir una unidad de interacción compuesta por un único formulario), *split* (para crear una división del espacio en varias areas), o *relation* (para crear una división en dos áreas que manejan datos interrelacionados). Los contenedores usan determinados parámetros para especificar cómo se realiza la división

[1] El *MB-UIDE* soporta por defecto el conjunto de acciones del modelo CRUD (*i.e. Create*, *Retrieve*, *Update* y *Delete*) para manipular los datos asociados a un formulario.

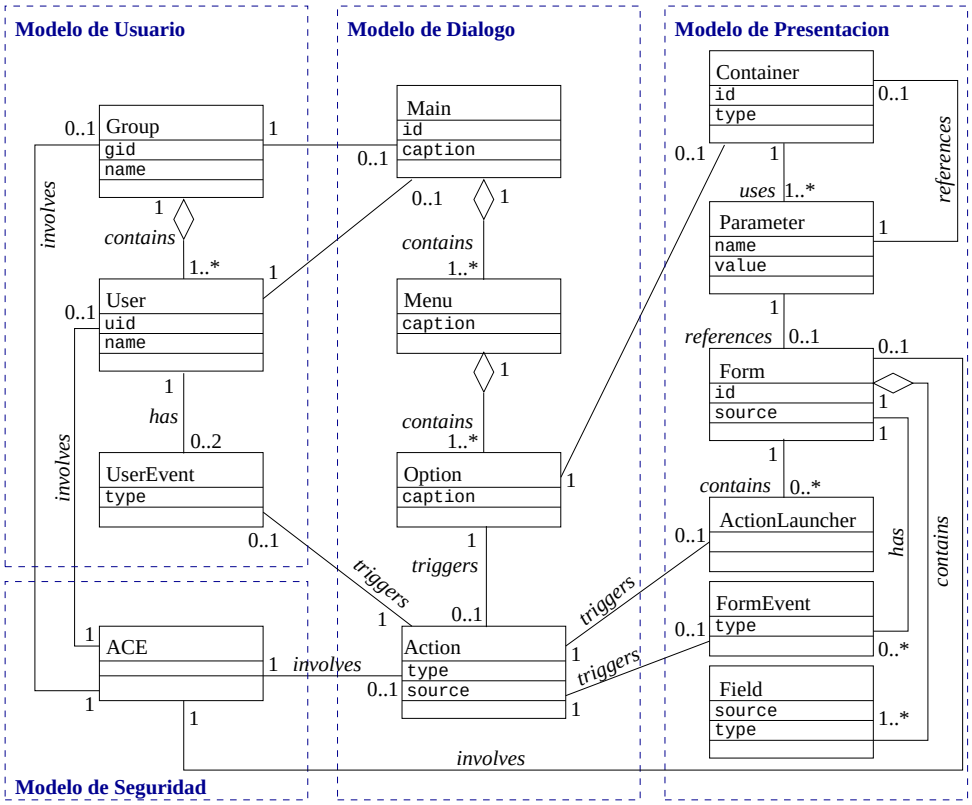


Figura 4.3: Modelos de la interfaz de usuario en WAINE

espacial (p.ej. filas o columnas) y el contenido de cada zona (p.ej. un formulario u otro contenedor). Se tratará a los formularios y contenedores con mayor profundidad en la sección 4.1.4.

- El *Modelo de Usuario*. Los usuarios se organizan en grupos que tienen roles comunes para la aplicación. Son componentes del modelo grupos, usuarios y sus eventos asociados:

  1. Los grupos tienen como atributos principales un *gid* (identificador de grupo), un nombre y una descripción. También pueden incluir información relativa al envejecimiento de las cuentas de usuarios del grupo y datos relativos al perfil (que permiten personalizar la *interfaz de usuario* a las necesidades de los integrantes del grupo). Estas propiedades son comunes a todos los usuarios que pertenecen al grupo.
  2. Los usuarios pertenecen a un único grupo. Además de los mismos atributos del grupo (que en caso de ser definidos "sobrescriben" los valores definidos para el grupo) disponen de un *uid* (identificador de usuario), de algún tipo de información para su autenticación y de campos de datos personalizables por los desarrolladores.

  Tanto para grupos como para usuarios es posible definir vistas específicas del *Modelo de Diálogo* (lo que se traducirá en la *IU Final* como el menú principal de

su aplicación como ya veremos) así como eventos que son disparados cuando el usuario accede a la aplicación o la abandona.

- El *Modelo de Diálogo*. Simplemente incluye un conjunto de opciones de menú accesibles a los usuarios de acuerdo con su grupo. Cada opción está asociada a una funcionalidad: una *unidad de interacción* a mostrar (identificada por su contenedor de más alto nivel), una acción a ser ejecutada (disparada por el usuario) o simplemente un enlace con otro recurso de internet a través de una URL. Son elementos del *Modelo de Diálogo*: menús, opciones y acciones. Además de las acciones predefinidas para los formularios (aquellas del modelo CRUD), WAINE permite a los desarrolladores definir sus propias acciones.

- *Modelo de Seguridad*: Aunque no nos consta la existencia de trabajos en los que se defina un modelo de seguridad en el *MB-UID*, creemos que es necesario dotar a estos sistemas de mecanismos que garanticen su seguridad. El principal elemento de este modelo es la *Entrada de Control de Acceso* que define el acceso de los usuarios o grupos a las funcionalides del sistema: acciones o formularios. También forman parte de este modelo los métodos de autenticación de usuarios.

4.1.2. Arquitectura

WAINE tiene una arquitectura de *Run-time* (figura 4.4) que es la base para generar la *IU Final* en tiempo de ejecución. Los dos pilares básicos de la arquitectura de WAINE son los repositorios y el motor.

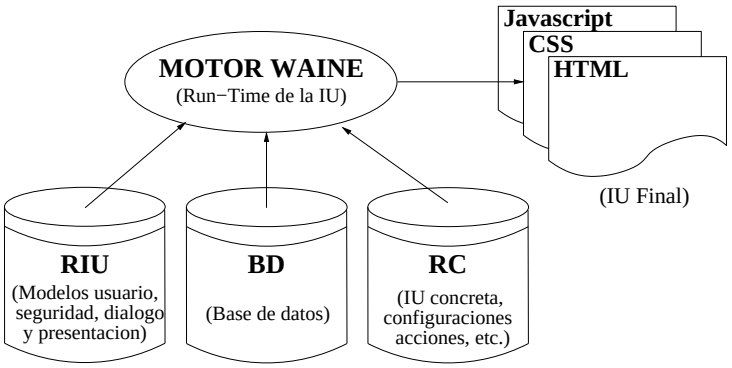


Figura 4.4: Arquitectura de WAINE

Las aplicaciones desarrolladas con WAINE emplean tres repositorios:

- *Repositorio de la IU* (RIU), almacena las instancias de los modelos de la *interfaz de usuario* de la figura 4.3.

- *Base de Datos* (BD), soporta los elementos manejados por la *interfaz de usuario* (*i.e.* los objetos del *Modelo de Dominio*). El motor de WAINE usa una capa de abstracción de acceso a datos (ver figura 4.6) que permite la conexión a distintos orígenes de datos, proporcionando independencia en este sentido (se detalla en la siguiente subsección).

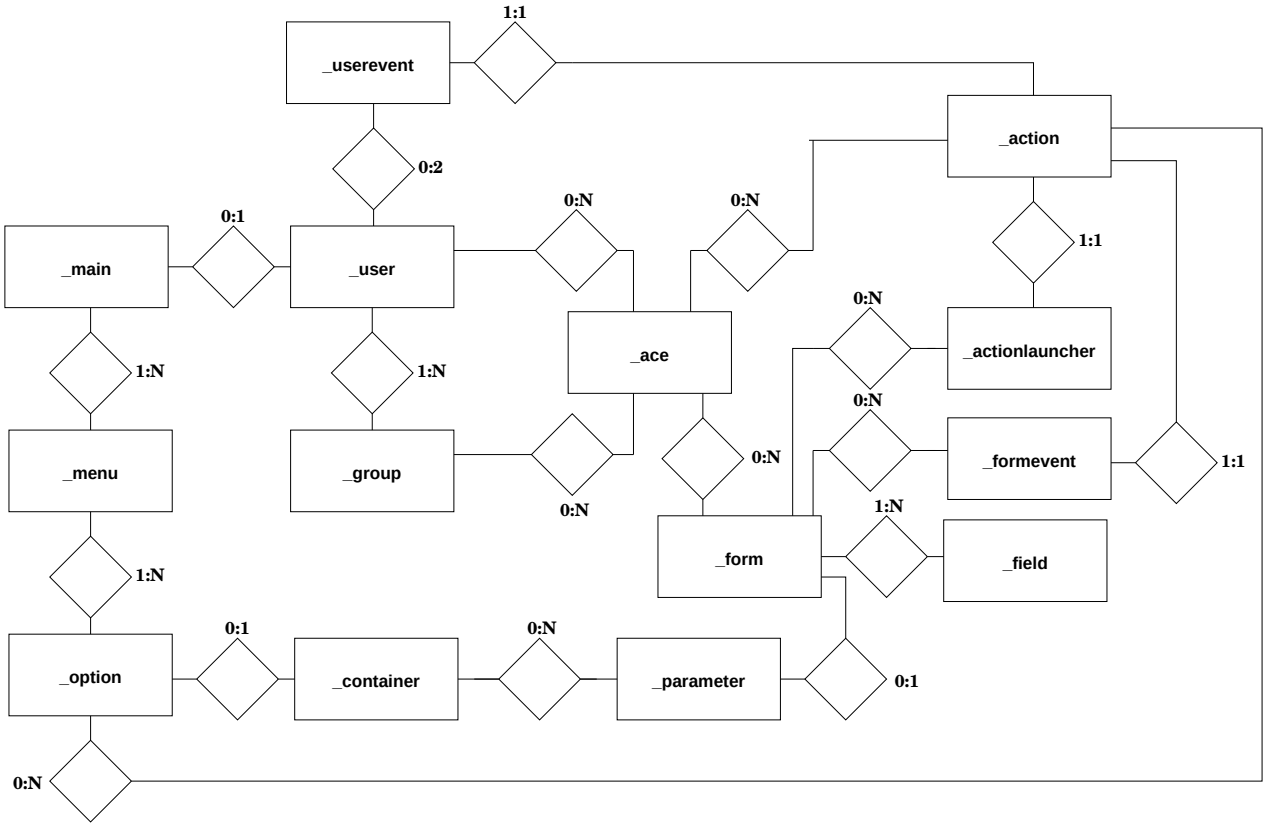


Figura 4.5: Modelo conceptual del repositorio de la interfaz de usuario

- *Repositorio de Configuraciones* (RC), almacena elementos de distinta naturaleza:

  1. Configuraciones de diversos aspectos de la aplicación: método de autenticación de usuarios a emplear, accesos a bases de datos, idioma por defecto para la *interfaz de usuario*, etc.
  2. Definiciones de la presentación concreta de la *interfaz de usuario*: estilos, fuentes, colores, etc. Estas definiciones pueden ser asignadas a la aplicación completa o un determinado formulario, menú, usuario o grupo.
  3. Acciones definidas por los desarrolladores: WAINE permite a los desarrolladores definir sus propias acciones en cualquier lenguaje de programación (aunque normalmente se emplea para ello PHP). Las acciones son almacenadas en un directorio específico del *Repositorio de Configuraciones*.
  4. Clases/objetos definidos por los desarrolladores: uno de los objetivos de WAINE es ser flexible y extensible. Para ello consta de varios *pools* de objetos que permiten a los desarrolladores extender las capacidades del motor, añadiendo nuevos métodos de autenticación, drivers de acceso a datos, *widgets* [2] y

[2]Un *widget* (*Objeto Concreto de Interacción* en WAINE) define el aspecto y comportamiento concreto de un campo. El dominio de un campo asigna un *widget* por defecto al mismo, pero los desarrolladores pueden asignar uno de los widgets soportados por WAINE o definir sus propios *widgets*.

layouts[3].

Al crear una instancia para una nueva aplicación, WAINE genera un *Repositorio de Configuraciones* con valores por defecto. Esto dota a la aplicación de una configuración y presentación concreta (colores, fuentes, etc.) que los desarrolladores pueden modificar. Para poner la aplicación en funcionamiento sólo será necesario configurar en el *Repositorio de Configuraciones* el método de autenticación de los usuarios y el acceso a la *Base de Datos*.

El motor de WAINE o *Run-time de la IU*, lee los repositorios anteriores para generar en tiempo de ejecución la *IU Final*, formada por HTML, CSS y Javascript. Teniendo en cuenta los tipos de arquitecturas de *MB-UIDEs* identificadas por Silva [95] (ver sección 2.1) podemos apreciar que WAINE sigue una arquitectura *Mixta*, en la que las especificaciones son pretratadas y almacenadas en un soporte (*Repositorio de la IU*) que posteriormente es empleado por el *Run-time* del sistema.

En la figura 4.6, se muestra la arquitectura interna del *Run-time*. Podemos observar que el motor está compuesto por un conjunto de módulos o subsistemas (Menús, Forms, Exports, etc.) que emplean los servicios ofrecidos por dos capas de abstracción: la de acceso a datos y la de renderizado. Estas capas de software permiten acceder a distintos orígenes de datos y generar todos los objetos de la *interfaz de usuario*. De forma adicional, el módulo de usuarios y seguridad ofrece control de acceso y servicios de autenticación al resto de módulos. Las capas de acceso a datos, renderizado y seguridad pueden ser extendidas por los desarrolladores para dotar al motor de nuevas capacidades.

4.1.3. Lenguaje de descripción de la interfaz de usuario

WAINE emplea varios lenguajes durante el proceso de desarrollo de la *interfaz de usuario*. En la figura 4.2, se pueden apreciar los principales lenguajes utilizados (c), su relación con los modelos de WAINE (b) y con el *framework* de referencia *Cameleon* (a). Los *Diagramas Entidad-Relación* son empleados como lenguaje gráfico para expresar el *Modelo de Dominio* (nivel de *Conceptos y Tareas*). Los modelos de *Presentación* y *Diálogo* (*IU Abstracta* en WAINE), el *Modelo de Usuario* (*Contexto*), así como el *Modelo de Seguridad* se definen en ASL [20]. ASL (*Application Specification Language*) es un *Lenguaje de Descripción de la IU* propio de WAINE basado en XML. La *IU Final* es generada de forma automática y se define empleando HTML, CSS y JavaScript. Se ha de tener en cuenta que en el proceso de desarrollo también se emplea PHP para definir *widgets*, *layouts*, etc. y para implementar acciones definidas por los desarrolladores (para esto último también es posible emplear otros lenguajes de programación).

Emplear varios lenguajes en WAINE fue una decisión de diseño para intentar facilitar el aprendizaje del sistema: se decidió utilizar, siempre que fuera posible, un lenguaje bien conocido (CSS, PHP, etc.) y si este requisito no se pudiera cumplir, definir un

[3]Las plantillas y los *layouts* se emplean para describir cómo deben distribuirse los campos dentro de un formulario, pero mientras que la plantilla es una definición estática para un formulario particular, los *layouts* se ajustan dinámicamente a las necesidades de cualquier formulario. Existen varios *layouts* predefinidos (p.ej. tabla, ficha, rejilla, etc.).

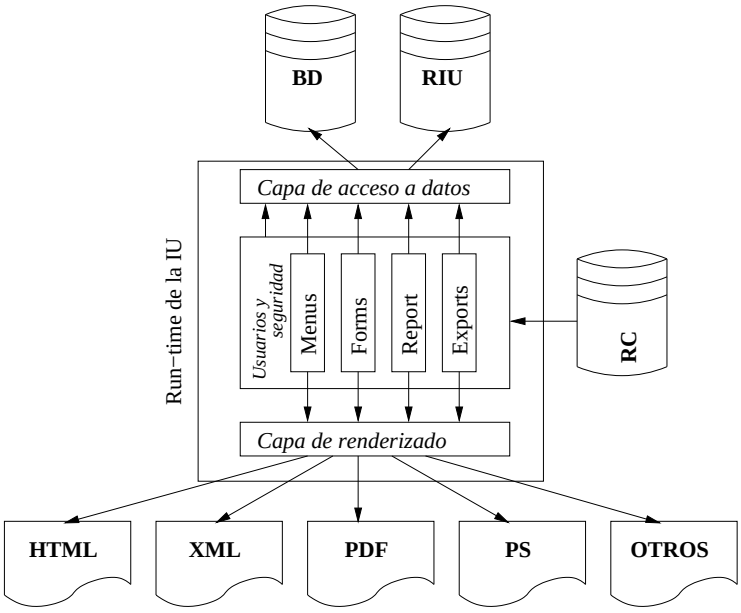


Figura 4.6: Arquitectura del run-time

lenguaje de especificación sencillo (ASL). Se realizó un análisis simple de los *Lenguajes de Descripción de la IU* candidatos. La tabla 4.1 muestra un resumen de los lenguajes estudiados. Para cada lenguaje se muestran los modelos que soporta y el número de etiquetas que utiliza (como indicador básico de su complejidad). Todos los lenguajes de la relación ofrecen características interesantes [45, 115], pero finalmente se decidió definir un *Lenguaje de Descripción de la IU* propio para WAINE por las siguientes razones:

- No todos los lenguajes están disponibles de forma libre (p.ej. XIML).
- Algunos lenguajes presentan demasiada complejidad (p.ej. UsiXML, ver columna *tags* de la tabla 4.1).
- Algunos lenguajes están orientados a la solución de problemas específicos (p.ej. XUL).
- Otros no soportaban el conjunto de modelos necesario para cumplir con los requisitos de diseño (p.ej. AUIML, UIML, XUL, XICL).
- En general, los *Lenguajes de Descripción de la IU* presentan una larga curva de aprendizaje, ya que manejan conceptos propios de la *Interfaz Hombre-Máquina*, a los que nuestros "potenciales clientes" (estudiantes y programadores noveles) no están habituados.

En ASL se ha intentado obtener un equilibrio entre el número de modelos soportados y la complejidad del lenguaje (número de etiquetas), así como utilizar un "vocabulario" conocido por los desarrolladores (*i.e.* usuario, grupo, menú, formulario, etc.). Obsérvese, que disponer de un *Lenguaje de Descripción de la IU*, independiza al sistema de las herramientas de especificación (uno de los criterios iniciales de diseño, sección 4.1), ya

Lenguaje	MDO	MU	MT	MP	MD	tags
UIML	✓			✓	✓	50
AUIML			✓	✓		más de 55
XUL			✓	✓		más de 60
XIML	✓		✓	✓	✓	32
Teresa XML			✓	✓	✓	19
UsiXML	✓	✓	✓	✓		más de 100
XICL				✓	✓	13
IMML	✓		✓	✓		52
ASL		✓		✓	✓	**50**

Tabla 4.1: Comparativa de UIDLs: modelos soportados y etiquetas empleadas

que se pueden emplear simples editores de texto, editores especializados, herramientas gráficas, etc. para generar las especificaciones.

Un documento ASL tiene las siguientes secciones:

- La cabecera: suele contener *meta-información* del documento (p.ej. autor, fecha de creación, etc.).
- Grupos y usuarios: descripción de usuarios, grupos y sus propiedades (*Modelo de Usuario*).
- Menús: conjunto de menús a los que acceden los usuarios (*Modelo de Diálogo*).
- Formularios y contenedores: formularios con sus campos y contenedores de formularios (*Modelo de Presentación*).
- Entradas de control de acceso implicando a usuarios/grupos con formularios/acciones (*Modelo de Seguridad*).

La sintaxis completa del lenguaje ASL en formato DTD se muestra en el apéndice A.

4.1.4. Proceso de desarrollo y generación de la interfaz de usuario

En este apartado se presenta el proceso de desarrollo de aplicaciones con WAINE y su relación con la generación automática de la *interfaz de usuario*. Analizaremos cómo los anteriores elementos: modelos, lenguajes y repositorios están relacionados con ambos procesos (desarrollo y generación de la *interfaz de usuario*).

WAINE define un proceso sistemático de desarrollo de aplicaciones. Esta característica del *MB-UIDE*, permite guiar el desarrollo de la *interfaz de usuario* ofreciendo a los desarrolladores un método para la construcción de aplicaciones. Este proceso tiene cinco fases. La figura 4.7 muestra las fases de desarrollo y los productos resultantes de cada una de ellas.

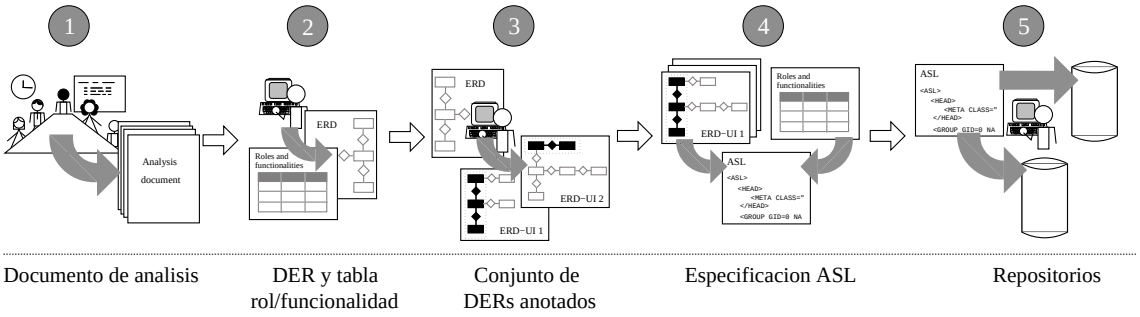


Figura 4.7: Proceso de desarrollo de una aplicación con WAINE

1. La redacción de un breve documento de análisis.

2. El diseño conceptual de la BD y la especificación de funcionalidades del sistema.

3. La definición del *Modelo de Presentación* a través de la anotación del modelo conceptual.

4. La descripción de los modelos de *Usuario*, *Diálogo*, *Presentación* y *Seguridad* en el lenguaje ASL.

5. La generación de los repositorios: *Repositorio de la IU*, *Base de Datos* y *Repositorio de Configuraciones*.

En la figura 4.8 se muestran las cinco fases del proceso de desarrollo de la *interfaz de usuario*, los productos resultantes de cada una de ellas y todo ello se relaciona con los niveles del *framework* de referencia *Cameleon*. Pasamos a describir en profundidad cada una de las etapas.

1. Análisis. El proceso de desarrollo comienza realizando entrevistas a usuarios expertos. En estas entrevistas se capturan los requisitos de la aplicación a desarrollar y con la información obtenida se redacta un pequeño documento de análisis en el que se incluyen las funcionalidades, los roles de los usuarios y bocetos de las principales *interfaces de usuario* e informes empleados por cada rol.

2. Modelado de conceptos. El documento de análisis es la base para la elaboración de dos documentos: (a) la tabla de roles/funcionalidades (ver tabla 4.2) que indica las acciones e *interfaces de usuario* necesarias para cada rol y (b) el modelo conceptual de la base de datos que se plasma en un *Diagrama Entidad-Relación*. Este diagrama que representa el *Modelo de Dominio* de la aplicación, recoge los conceptos que son manejados por la misma y es el modelo de partida para el desarrollo de la *interfaz de usuario*.

3. Anotación del *Diagrama Entidad-Relación*. En la tercera fase se especifica cada espacio de trabajo (ver figura 4.11) de la *interfaz de usuario* mediante un proceso de refinamiento que parte del *Diagrama Entidad-Relación* obtenido en el paso 2. Para ello se toma la porción de *Diagrama Entidad-Relación* que afecta a la *unidad*

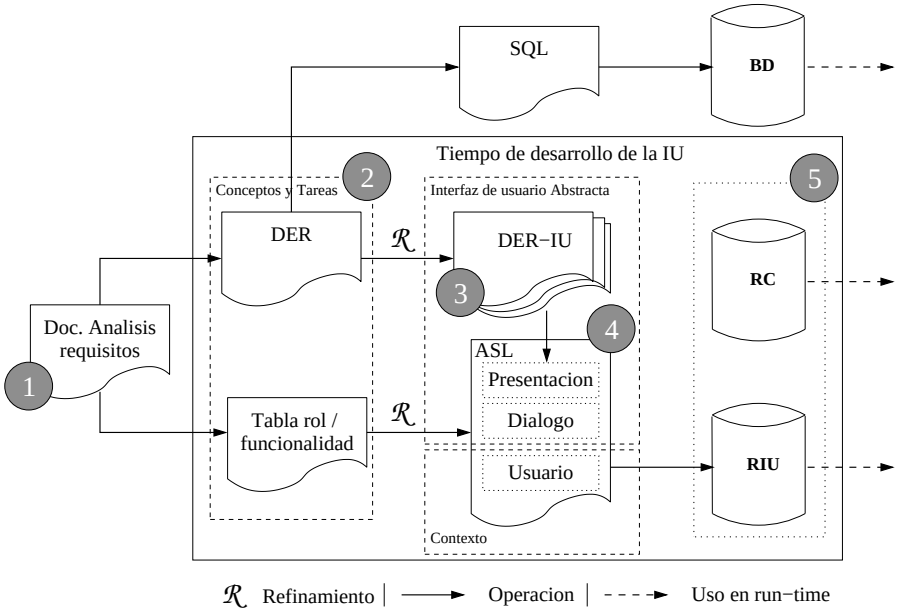


Figura 4.8: Proceso de desarrollo de la interfaz de usuario

de interacción que se está definiendo y se enriquece anotando sobre él las partes que lo forman: formularios y contendores.

Vamos a ilustrar y profundizar en la especificación de la *unidad de interacción* a través de un ejemplo. Desarrollaremos una aplicación que administre citas de pacientes a través de un único espacio de trabajo. La estructura visual de la unidad de interacción de nuestro ejemplo está compuesta de dos zonas horizontalmente divididas que sostiene una relación uno-muchos. Esto es definido a través de un contenedor de tipo relación identificado como *csample*. La figura 4.9(a) muestra cómo el contendor se anota sobre el *Diagrama Entidad-Relación* a través de una línea punteada alrededor de las entidades implicadas y etiquetada con el tipo del contenedor y su identificador. El área superior contendrá un formulario que maneja las tuplas de la entidad *Patient*. El identificador del formulario (*fpatient*) se anota en la esquina superior izquierda de la propia entidad. Para cada formulario también se anota su *layout* en la esquina inferior derecha. Recordemos que el *layout* expresa la forma en la que los campos se distribuyen dentro del formulario (p.ej. *table*, *grid*, *list*, etc.). En la zona inferior del contenedor *csample* se divide

	IU1	IU2	..	IUN	A1	A2	..	AN
ROL1	✓			✓	✓	✓		✓
ROL2		✓		✓				
:				✓				
ROLN	✓			✓	✓			✓

Tabla 4.2: Tabla Rol/funcionalidad

el espacio disponible verticalmente usando un nuevo contenedor: *csample_n* de tipo *split*. El contenido de cada zona de *csample_n* será un formulario que manejará datos sobre citas (parte izquierda, formulario *fappointment*) y un formulario que gestionará notas sobre cada paciente (parte derecha, formulario *fremark*).

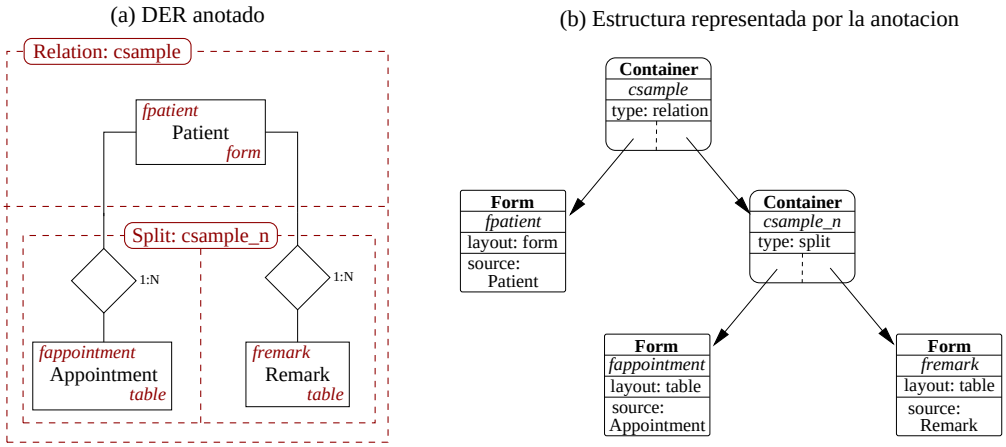


Figura 4.9: Ejemplo de anotación del diagrama entidad-relación

El comportamiento de la unidad de interacción está definido por el contendor *csample*, que define dos zonas espaciales y una relación uno-muchos entre ambas. Esto significa que si un elemento en la zona "uno" cambia (una ocurrencia de *Patient* en nuestro ejemplo) los elementos de la zona "muchos" (ocurrencias de las entidades *Appointment* y *Remark*) deben ser automáticamente recalculados. Este proceso de sincronización es facilitado por la relación jerárquica entre los elementos de la *interfaz de usuario* ilustrados en la figura 4.9(b).

El producto resultante de esta fase es una colección de *Diagramas Entidad-Relación* anotados, uno por cada espacio de trabajo.

4. Especificación en ASL. En esta fase, el desarrollador toma la tabla rol/funcionalidad y los *Diagramas Entidad-Relación* anotados de los pasos 2 y 3 respectivamente y crea un documento codificado en lenguaje ASL [20], donde se especifican instancias de las clases de la figura 4.3 (*Modelos de la IU* de WAINE). El listado 4.1 muestra la definición de un grupo con un usuario y los elementos del *Modelo de Diálogo* generados desde la tabla rol/funcionalidad (etiquetas <group>, <user>, <main>, <menu> y <option>). Este listado también incluye la especificación de la *unidad de interacción* previamente definida por el *Diagrama Entidad-Relación* anotado de la figura 4.9. Como podemos apreciar, la definición del *Modelo de Presentación* comprende la mayor parte del documento ASL.

```
<!-- User model example -->
<group gid="1" name="rol1" mainid="rol1_taks">
    <user uid="1" name="user" descr="A sample user"/>
</group>

<!-- Dialog model example -->
<main id="rol1_taks" caption="Select an option">
```

```
    <menu caption="Data managing">
        <option caption="A sample UI" call="csample"/>
    </menu>
    <menu caption="Misc">
        <option caption="Waine web page " url="http://waine.us.es"/>
        <option caption="About" call="cabout"/>
        <option caption="Logout" action="LOGOUT"/>
    </menu>
</main>

<!-- Presentation model from Figure 4 -->
<form id="fremark" source="Remark" caption="Remarks">
   <fields>
      <key source="pk"/>
      <string source="descr" caption="Description" len="30"/>
      <fkey source="fkpatient"/>
   </fields>
</form>

<form id="fappointment" source="Appointment" caption="Appointments">
   <fields>
      <key source="pk"/>
      <date source="adate" caption="Date"/>
      <string source="descr" caption="Description" len="30"/>
      <fkey source="fkpatient"/>
   </fields>
</form>

<form id="fpatient" source="patient" caption="Patient">
   <orderby>name</orderby>
   <fields>
      <key source="pk"/>
      <string source="name" caption="Name" len="40"/>
      <date   source="bdate" caption="Birth date"/>
      <string source="address" caption="Address" len="40"
          canbenull="Y"/>
      <int source="phone" caption="Phone" len="13" canbenull="Y"/>
   </fields>
</form>

<container id="csample" type="relation">
   <param name="form_split" value="rows=20%,*"/>
   <param ord="1" name="formid" value="fpatient"/>
   <param ord="2" name="containerid" value="csample_n"/>
</container>

<container id="csample_n" type="split">
   <param name="form_split" value="cols=50%,*"/>
   <param ord="1" name="formid" value="fappointment"/>
   <param ord="1" name="form_layout" value="table"/>
   <param ord="2" name="formid" value="fremark"/>
   <param ord="2" name="form_layout" value="table"/>
</container>
```

Listado 4.1: Extracto de un documento ASL

Observe en el listado 4.1 que:

a) Una definición de formulario (*form*) requiere la asignación de algunos atributos como identificador (*id*), origen (*source*) o título (*caption*) e incluye la definición de la clave primaria (etiqueta *key*) y opcionalmenter de claves ajenas (*fkey*), así como de los campos que lo componen que incluyen la definición de su origen y su tipo (p.ej. *string*, *date*, etc.).

b) La etiqueta contenedor (*container*) tiene dos atributos: identificador (*id*) y tipo (*type*). Cada contenedor incluye un número variable de parámetros. Los parámetros pueden ser clasificados en tres categorías:

- Estructurales, usados para definir una estructura espacial (p.ej. el parámetro *form_split* se usa en el contenedor *csample* para crear una división en dos filas, asignando el 20 % del espacio disponible al área superior).
- De contenido, que describen el contenido de cada zona (p.ej. los parámetros *containerid* y *formid* referencian un contenido para una zona identificada por el atributo *ord*) y sus propiedades básicas (p.ej. *form_layout*)
- De personalización, orientados a sobrescribir valores definidos en el formulario referenciado. La sección 4.2 mostrará un ejemplo de uso de este tipo de parámetros.

5. Generación de repositorios y personalización. El proceso de desarrollo finaliza con la generación de los tres repositorios usados por el *Run-time* (RIU, BD y RC; figura 4.8). EL *Repositorio de la IU* se genera automáticamente empleando una herramienta que convierte el documento ASL en un repositorio. Esta trasformación es muy ligera, así que los elementos que se encuentran en el repositorio son los mismos que los especificados en ASL con modificaciones mínimas. También, partiendo del *Diagrama Entidad-Relación* se genera el modelo físico de la *Base de Datos* y el código SQL necesario su la creación. Esto sería suficiente para obtener una aplicación operativa con el aspecto por defecto suministrado por el motor (como se ha mencionado en la sección 4.1.2).

   Si los desarrolladores desean especificar aspectos concretos de la *interfaz de usuario* (colores, bordes, fuentes, etc.) distintos de los asignados por defecto deben modificar determinados ficheros del *Repositorio de Configuraciones*. En los listados 4.2 y 4.3 podemos ver cómo se definen los *widgets* por defecto para cada tipo de campo y algunos estilos de la *interfaz de usuario*. Las acciones que sea necesario implementar también deben añadirse al *Repositorio de Configuraciones*. Por último, los desarrolladores pueden extender el *Run-time* añadiendo nuevos *widgets*, *layouts* o métodos de autenticación. Estos elementos se codifican en PHP y se añaden a los *pools* correspondientes del motor.

```
integer.widget='editbox';
float.widget ='editbox';
string.widget ='editbox';
time.widget ='timebox';
date.widget ='datebox';
text.widget ='textbox';
blob.widget ='filesel';
image.widget ='imagebox';
```

Listado 4.2: Conf. widgets

```
$BODYATTR=['bgcolor','background','text'];
$BODYVAL =['#00659c','bg.png','#005989'];
$MENUATTR=['class','bgcolor'];
$MENUVAL =['clmenu','#e7efff'];
$OPTATTR=['class','bgcolor'];
$OPTVAL =['clopt','#efeff7'];
$FRMTITATTR=['style'];
$FRMTITVAL =['border: 1px solid black;'];
```

Listado 4.3: Conf. de colores y estilos

Generación de la *IU Final*

Una vez que el desarrollo ha finalizado la aplicación está lista para usarse. En las aplicaciones WAINE la *IU Final* es generada automáticamente por el *Run-time* (figura 4.10). A la recepción de una acción de un usuario sobre la *interfaz* (1- petición de usuario), el *Run-time* genera la *IU Final* (2b) que se envia (3) al navegador web del usuario. En el proceso de generación automática de la *interfaz de usuario* se emplean los tres repositorios: RIU, BD y RC.

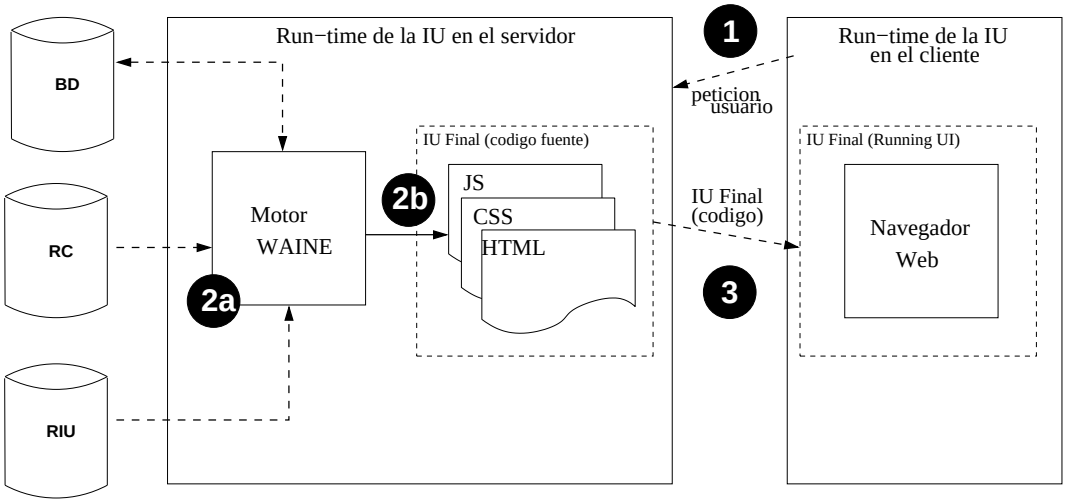


Figura 4.10: Generación de la interfaz de usuario final

El proceso de generación de la *IU Final* consta de dos fases:

1. El run-time extrae el *Modelo de Presentación* del *Repositorio de la IU* y lo completa estableciendo colores, fuentes y estilos y asignando *widgets* a los campos de los formularios (figura 4.10, 2a). Este proceso requiere información de configuración que reside en el *Repositorio de Configuraciones* (como la que aparece en los listados 4.2 y 4.3). Si un campo de un formulario necesita un *widget* determinado o un formulario necesita una configuración particular de su aspecto el nombre del *widget* o el del estilo específico para el formulario se debieron especificar en el documento ASL durante la fase 4 del proceso de desarrollo (ver listados 4.4 y 4.5).

```
<int source="age"
    caption="Age">
   <widget>spinbutton</widget>
</int>
```

Listado 4.4: Widget específico

```
<form id="f_st1" source="staff" >
  <theme>high-contrast.sty</theme>
  <fields>
     <key source="pk"/>
     <string source="name" ...
```

Listado 4.5: Estilo para un formulario

Como vemos, ASL permite por lo tanto definir algunos aspectos concretos de la *IU Concreta*. En la tabla 4.3 se presentan los elementos abstractos del *Modelo de Presentación* junto a los aspectos de presentación concreta que pueden definir.

Objeto Abstracto de Interacción	Aspectos opcionales de la IU concreta
Contenedor	Division espacial: parámetro *form_split*
	Layout: parámetro *form_layout*
	Tema: parámetro *form_theme*
Formulario	Tema: etiqueta *theme*
Campo	Estilo: propiedad *style*
	Widget: etiqueta *widget*

Tabla 4.3: Aspectos de presentación concreta tratados en ASL

2. Una vez completado el *Modelo de Presentación* se construye la *IU Final* compuesta por HTML, CSS y JavaScript (figura 4.10, 2b). El código es construido por una serie de clases existentes en el *Run-time*. Existe un *pool* de clases de *layout* que generan el código para distribuir los *widgets* dentro de un formulario. También para cada *widget* existe una clase encargada de generar su código final correspondiente. En este proceso el *Run-time* recupera los componentes que los desarrolladores hayan podido definir (clases *layout*, *widgets*, etc.) del *pool* correspondiente. Por último, las clases internas *render* hacen uso de las clases *layout* y *widget* para generar el código de la *IU Final*.

Obviamente estos procesos son transparentes al usuario. Desde su punto de vista, cuando accede a la aplicación, se le presenta automáticamente un formulario de acceso (generado por el *Run-time*, no es necesario especificarlo en ASL). Una vez que el usuario se ha autenticado, se muestran las opciones de su menú. Para ello, el motor obtiene el *Modelo de Diálogo* que corresponde al usuario (o su grupo) del *Repositorio de la IU*. Cuando el usuario selecciona una opción del menú el *Run-time* ejecutará una acción o generará el código de una *unidad de interacción*. En este último caso, los elementos involucrados (contenedores y formularios) se obtienen del *Repositorio de la IU* y la *Base de Datos* es accedida para obtener los datos que deben ser mostrados. Este proceso se repite hasta que el usuario abandona la aplicación. En la figura 4.11 podemos ver la apariencia final de la *interfaz de usuario* correspondiente al ejemplo presentado en esta sección.

Para profundizar en la construcción de aplicaciones con WAINE, se presenta en el apéndice B el proceso completo de desarrollo para crear una aplicación simple.

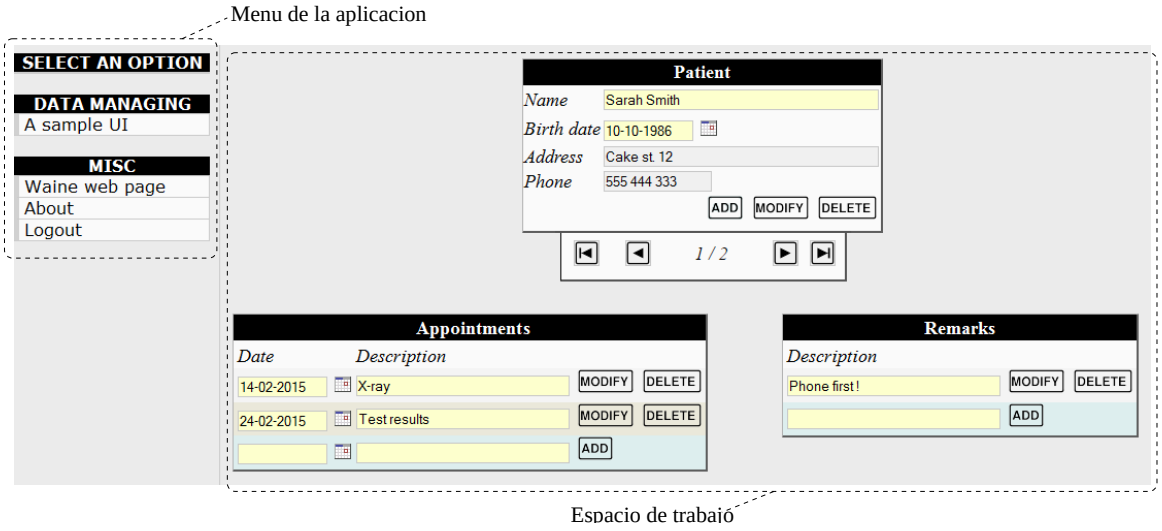


Figura 4.11: Interfaz de usuario de la aplicación de ejemplo

4.1.5. Herramientas de desarrollo

En este último apartado sobre WAINE enumeramos las herramientas de desarrollo que forman parte del *MB-UIDE*. Estas herramientas permiten realizar, entre otras tareas, las transformaciones que aparecen en el proceso de desarrollo:

- **mkapp**: Crea una nueva instancia de aplicación enlazada a una determinada versión del *Run-time de la IU*. Este proceso crea un *Repositorio de Configuraciones* por defecto para la aplicación.

- **asl2mdb**: Esta herramienta traduce especificaciones ASL a distintos formatos de base de datos (sql, csv, etc.). Esto permite almacenar el modelado especificado en el documento ASL en un soporte apropiado para ser empleado como *Repositorio de la IU* por el *Run-time* de WAINE. Se pueden emplear distintos tipos de soporte (BD sqlite, archivos csv, etc.) por razones de eficiencia o por limitaciones de la plataforma en la que se ejecuta el *Run-time*.

- **mdb2asl**: Esta herramienta genera especificaciones ASL partiendo de un *Repositorio de la IU*. Es una herramienta de ingeniería inversa que realiza la trasformación opuesta a la de la herramienta *asl2mdb*.

- **db2asl**: Esta herramienta construye especificaciones ASL de formularios y contenedores partiendo de tablas y vistas existentes en una *Base de Datos*. Con ello se intenta acelerar el desarrollo de aplicaciones de las que ya existe una *Base de Datos* previa (p.ej. Microsoft Access) o las migraciones de entornos existentes a WAINE.

- **asldoc**: Genera documentación para una aplicación partiendo del documento ASL. El formato de salida es DocBook [127], que posteriormente puede traducirse a latex, HTML, rtf, etc.

- **aslmod**: Valida un fragmento de especificación ASL para que pueda ser incluida posteriormente desde otra especificación. Se trata con mayor profundidad en la sección 4.2.3.
- **wpkg**: Es la herramienta para gestión de paquetes de WAINE. Hablaremos de ella con más profundidad en la sección 4.2.4.

Las dos últimas herramientas, están relacionadas con técnicas de reutilización implementadas en WAINE y se tratarán con mayor profundidad en las siguientes secciones.

4.2. Técnicas de reutilización soportadas por WAINE

Uno de los objetivos principales de WAINE es el desarrollo rápido de aplicaciones. Ello nos ha conducido a soportar de forma proactiva la reutilización. En esta sección se presentan las técnicas de reutilización presentes en WAINE; implementaciones de algunos de los métodos que de forma más general hemos presentado en las secciones 3.1 y 3.2.

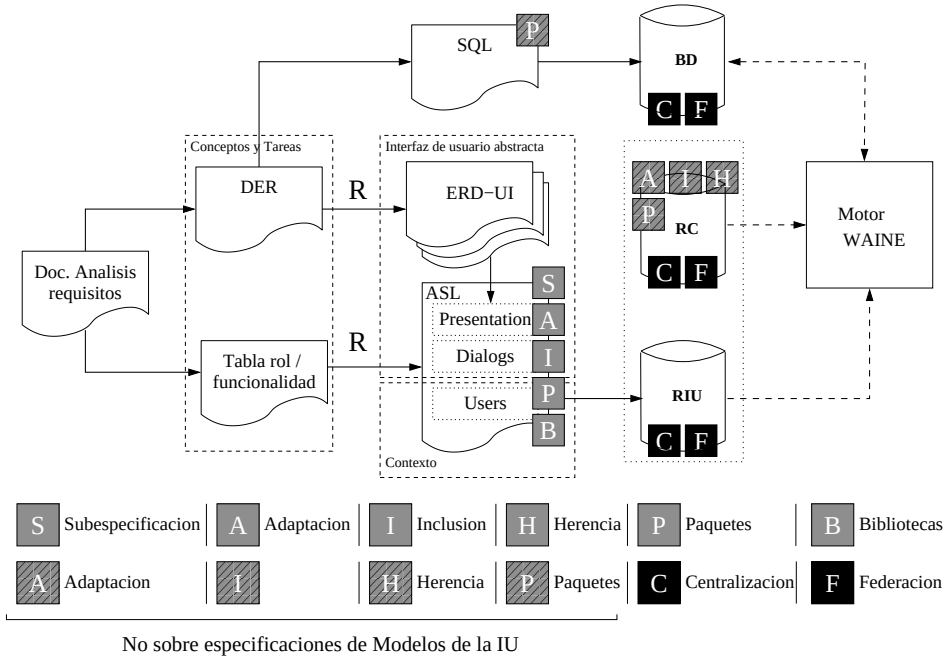


Figura 4.12: Contexto de uso de las técnicas de reutilización implementadas

4.2.1. Subespecificación y adaptación (S,A)

WAINE utiliza varios lenguajes durante el proceso de desarrollo de la *interfaz de usuario*. De todos ellos ASL es el lenguaje principal y ha sido desarrollado para permitir el uso de la subespecificación y de la adaptación.

Respecto a la subespecificación, hay varios elementos que pueden ser "subespecificados" (fragmentos de código referenciados desde otros puntos de la especificación). En la figura 4.13, se muestra la jerarquía de las etiquetas principales de ASL y sus relaciones. Las flechas punteadas entre etiquetas indican subespecificación. Podemos ver que desde la definición de usuarios o grupos se puede referenciar a un mismo menú. También que desde varias opciones de distintos menús se puede referenciar a la misma *unidad de interacción* (identificada por el contenedor de más alto nivel). Observe también en la figura 4.13 que objetos de otros modelos (que residen en el *Repositorio de Configuraciones* o en la *Base de Datos*) también pueden ser "subespecificados" y por lo tanto reutilizados, como por ejemplo tablas, acciones o estilos.

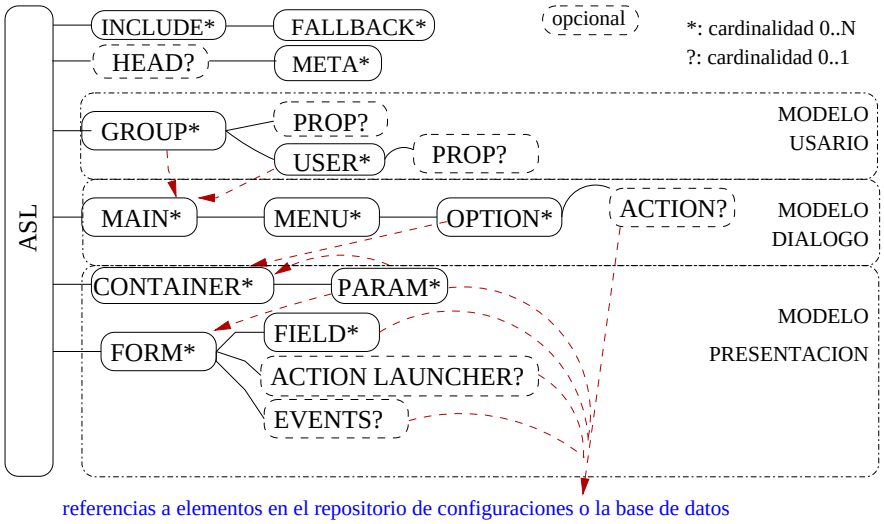


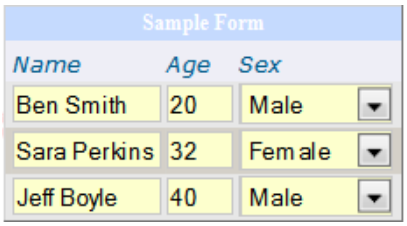
Figura 4.13: Subespecificación y adaptación en ASL.

La adaptación en ASL se emplea sobre todo en el *Modelo de Presentación*. Los formularios pueden ser adaptados en muchos aspectos a través de la etiqueta del lenguaje <param> (ver figura 4.13). Prácticamente cualquier atributo o componente de un formulario puede ser modificado, lo que dota de una enorme flexibilidad al *Modelo de Presentación* en cuanto a reutilización se refiere.

A modo ilustrativo veamos un par de ejemplos de adaptación de un formulario. En la figura 4.14(a) vemos el formulario *frm_sample* utilizando un *layout* de tipo tabla, mostrando su apariencia por defecto. En la figura 4.14(b), podemos observar, cómo por medio de la adaptación, el formulario ha perdido el campo *Sex* (tercer campo) y cómo los campos *Name* y *Age* (primer y segundo campo) pasan a estar en modo lectura. Por último, en la figura 4.14(c), se modifica el tercer campo (*Sex*) para que emplee el *widget radio* en lugar del que se estaba usando por defecto. La adaptación en el *Modelo de Presentación* es una herramienta potente, que permite que un mismo formulario pueda reutilizarse para circunstancias y usos distintos.

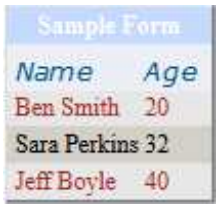
Como ya se ha comentado en la secciones anteriores, la arquitectura de WAINE es de tipo *Mixta*, empleando un *Repositorio de la IU* que contiene versiones pretratadas de las especificaciones. En este tipo de sistemas, la técnicas de reutilización en especificaciones (S, T, A, H) pueden ser resueltas en tiempo de desarrollo o en tiempo de ejecución

(a)



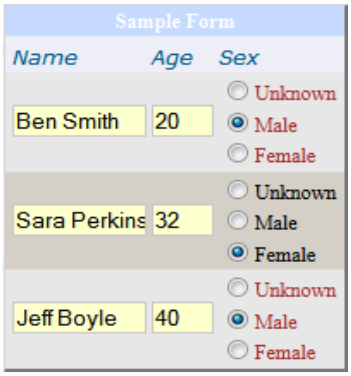

```
<container id="sample" type="form">
<param name="formid" value="frm_sample"/>
<param name="form_layout" value="table"/>
</container>
```

(b)




```
<container id="sample_p1" type="form">
<param name="formid" value="frm_sample"/>
<param name="form_layout" value="table"/>
<param name="fields_readonly" value="1-2"/>
<param name="fields_remove" value="3"/>
</container>
```

(c)




```
<container id="sample_p2" type="form">
<param name="formid" value="frm_sample"/>
<param name="form_layout" value="table"/>
<param name="fields_modify[0]"
       value="widget#radio#3"/>
</container>
```

Figura 4.14: Ejemplo de adaptación en ASL

(sección 3.3.1). El pretratamiento ligero de las especificaciones realizado en WAINE (herramienta **asl2mdb**) hace que las técnicas de subespecificación y adaptación se resuelvan en tiempo de ejecución por el motor.

ASL no es el único lenguaje implicado en el desarrollo de la *interfaz de usuario* con

WAINE. La adaptación también es aplicable en la implementación de *widgets* en PHP. El comportamiento y código final de cada campo de un formulario se corresponde en WAINE con un objeto de las clases *widget* o *mwidget*[4] codificados en *PHP* (ver figura 4.15). Existen multitud de *widgets* predefinidos, pero los desarrolladores pueden crear nuevos *widgets* adaptando las clases existentes (con el método *setCode*) o extendiendo las clases básicas (se verá con más detalle en el siguiente apartado).

En el listado 4.6, se muestra un nuevo *widget* (*passwordbox*) que es creado adaptando la clase base *widget*. En el ejemplo vemos cómo se redefine el código HTML que debe generarse cuando el objeto se muestra en modo escritura (W).

Obsérvese que en este caso, la adaptación tambien es resuelta en tiempo de ejecución (aunque propiamente no se trate de una especificación de la *interfaz de usuario*).

```
// Se crea un nuevo widget
$newwdg=new widget();

// El widget es adaptado indicando el codigo final a
// generar cuando es presentado en modo escritura 'W'

$newwdg->setCode('W',
                 "<INPUT type='password' name='%name%'
                         value='%value%' size='%len%'
                         maxlength='%maxlen%' style='%style%'/>");

// El nuevo objeto se introduce en el pool de widgets
$WDGPOOL->add('passwordbox',$newwdg);
```

Listado 4.6: Ejemplo de nuevo widget creado por adaptación

4.2.2. Herencia (H)

ASL no define ningún mecanismo de herencia. Sin embargo, como se ha comentado en el punto anterior, la herencia es empleada como mecanismo para permitir la implementación de nuevos *widgets*. A modo de ejemplo, podemos ver en la figura 4.15 la relación de herencia de algunos de los *widgets* de WAINE.

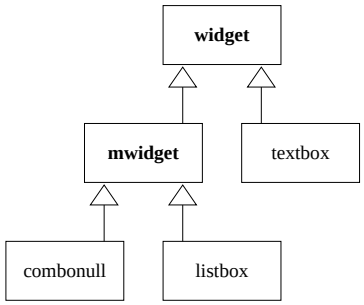


Figura 4.15: Relación de herencia de algunos widgets

[4]La clase *widget* es la clase de la que heredan todos los *widgets* del sistema. La clase *mwidget* también deriva de la clase *widget*, pero es la clase base para aquellos *widgets* cuyo valor proviene de la selección de un valor dentro de una lista de valores posibles (p.ej. un *combobox*).

En el código que se presenta en el listado 4.7 se puede apreciar cómo la clase *combonull*[5] hereda de la clase *mwidget* y cómo la nueva clase redefine el método *RenderW*[6] reutilizando el resto del código de la clase padre.

```
class combonull extends mwidget
{
   function RenderW($IN_name, $IN_value, $IN_fld, $IN_lstval)
   {

      // se incluye el valor nulo en la lista de valores seleccionables
      $lstval=array(array('',$IN_fld->wdgparam));

      // se invoca al metodo RenderW de la clase padre con $lstval
      return(parent::RenderW($IN_name, $IN_value, $IN_fld,
                             array_merge($lstval,$IN_lstval))
                            );
   }
}

$newwdg=new combonull();

// El nuevo objeto se introduce en el pool de widgets
$WDGPOOL->add('combonull',$newwdg);
```

Listado 4.7: Ejemplo de nuevo widget definido por herencia

La herencia de las clases *widget* es resuelta por el intérprete de PHP en tiempo de ejecución.

Por último, entendiendo el término de forma flexible, podríamos decir que la relación existente entre usuarios y grupos en ASL es también una relación de herencia. En el apéndice A.1, podemos observar cómo los atributos de las etiquetas <user> y <group> son comunes en un alto porcentaje. En la sección 4.1.1, se ha comentado que los atributos del grupo son aplicados a todos los usuarios que forman parte del mismo, pero que cuando los usuarios definen alguno de los atributos comunes con los del grupo, los valores del atributo del usuario ocultan o sobrescriben los del atributo del grupo. Este funcionamiento del *Modelo de Usuario* es análogo a una relación de herencia como la mostrada en la figura 4.16. El *Run-time* de WAINE es el encargado de recrear esta relación en tiempo de ejecución.

4.2.3. Inclusión y Bibliotecas de reutilización (I,B)

ASL es el lenguaje principal de modelado en WAINE. Es el lenguaje que define los modelos de *Usuario*, *Diálogo*, *Presentación* y *Seguridad*. También es el lenguaje que suele emplear mayor número de líneas de código en el desarrollo de la *interfaz de usuario* (como se verá más adelante). Debido al tamaño que llegan a alcanzar algunas especificaciones se hace necesario disponer de un mecanismo para la modularización

[5]El widget *combonull* es un elemento de interacción que permite la selección de una única opción de una lista de valores dentro de la que se admite el valor nulo.

[6]Método encargado de generar el código del *widget* cuando se encuentra en modo escritura (W).

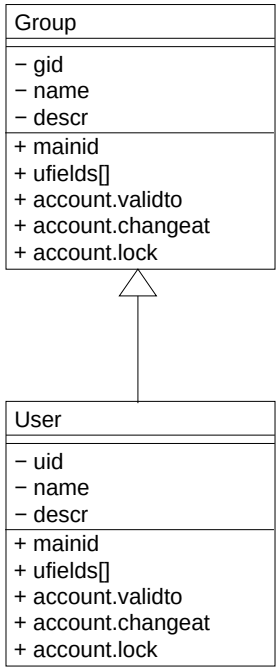


Figura 4.16: Relación de herencia entre grupos y usuarios

de las mismas. Al tratarse de un lenguaje XML, existe una solución inmediata para el mecanismo de inclusión: XInclude.

XInclude [65] es un estándar muy extendido del W3C que define un modelo de procesamiento y una sintaxis de propósito general para la inclusión o fusionado de documentos XML. Es ampliamente aceptado por la comunidad de desarrolladores, se emplea en multitud de ámbitos, entornos y plataformas y existen gran número de herramientas y utilidades que lo soportan. Sin embargo, XInclude, además de los problemas generales de los métodos de inclusión (fusión), presenta algunas limitaciones. El estándar no define ninguna relación con los procesos de validación y transformación del documento. XInclude describe una transformación *infoset-to-infoset* [65] pero no especifica el comportamiento respecto al análisis del documento XML. Así, un documento puede ser validado antes o después de realizar la inclusión, en ambos casos o en ninguno. Queda en manos del desarrollador decidir cuándo los documentos XML deben ser validados y/o transformados en este proceso.

Respecto a su implementación en ASL, el lenguaje en sus primeras versiones no soportaba XInclude, así que fue necesario modificar el DTD original (añadiendo el código mostrado en el listado 4.8) para soportar las etiquetas definidas por el estándar [65].

```
<!ELEMENT xi:include (xi:fallback?)>
<!ATTLIST xi:include
    xmlns:xi        CDATA       #FIXED "http://www.w3.org/2001/XInclude"
    href            CDATA       #IMPLIED
    parse           (xml|text)  "xml"
    xpointer        CDATA       #IMPLIED
    encoding        CDATA       #IMPLIED
    accept          CDATA       #IMPLIED
```

```
    accept-language CDATA          #IMPLIED
>

<!ELEMENT xi:fallback ANY>
<!ATTLIST xi:fallback
    xmlns:xi    CDATA    #FIXED "http://www.w3.org/2001/XInclude"
>
```

Listado 4.8: DTD para las etiquetas definidas por el estándar XInclude

Sólo con realizar esta modificación al lenguaje, es posible modularizar las especificaciones de gran tamaño en fragmentos más manejables que posteriormente son fusionados en una única especificación mediante la inclusión (ver listado 4.9).

```
<ASL>
    <XI:INCLUDE HREF="cabecera.asl" />

    <XI:INCLUDE HREF="usuario.asl" />
    <XI:INCLUDE HREF="dialogo.asl" />
    <XI:INCLUDE HREF="presentacion.asl" />
</ASL>
```

Listado 4.9: Modularizando una especificación con XInclude

Sin embargo, esta solución no es suficiente para la distribución de componentes reutilizables. Recordemos que XInclude presenta una limitación importante: el estándar no define ninguna relación con la validación ni con la transformación del documento. Obviamente un recurso reutilizable que puede ser distribuido para su uso en distintos proyectos (o incluso ser incluido en una biblioteca de reutilización) debe ser validado antes de su distribución, por lo tanto, es necesario disponer de herramientas para validar estos recursos. Esto que en principio puede parecer trivial no lo es, ya que los fragmentos reutilizables no son documentos ASL "bien formados" (son trozos de una especificación ASL) y por lo tanto requieren de una validación propia. Para resolver este problema se han introducido dos nuevos elementos en ASL:

- La etiqueta <aslmod> es empleada para definir un componente reutilizable (un fragmento de código ASL) que debe ser especificado en su propio documento y encerrado entre las etiquetas <aslmod> y </aslmod>.
- Por otra parte, la herramienta *aslmod* que valida los componentes reutilizables y los deja preparados para ser incluidos en futuras especificaciones (incluyendo valores por defecto para atributos, etc.).

En el apéndice C se muestra el código correspondiente a un componente ASL: la *unidad de interacción meta.container.appinfo* empleada de forma frecuente en las aplicaciones desarrolladas con WAINE para mostrar información sobre la propia aplicación (al estilo del típico diálogo "Acerca de" o "Sobre este programa").

Recordemos (ver tabla 3.1), que la técnica de inclusión en arquitecturas de tipo *Mixta* como la de WAINE debe ser resuelta en tiempo de desarrollo.

En la figura 4.17, se muestra el proceso completo de validación → inclusión → transformación de una especificación ASL. Como se ha comentado en este apartado, los

componentes ASL deben ser previamente validados y preparados con la herramienta *aslmod* (1). Las especificaciones son validadas antes de la inclusión de los componentes (2) y posteriormente se incluyen éstos para obtener la especificación completa (3). Finalmente se efectúa el proceso de transformación de la especificación final al formato que soporta el *Repositorio de la IU* con la herramienta *asl2mdb* (4).

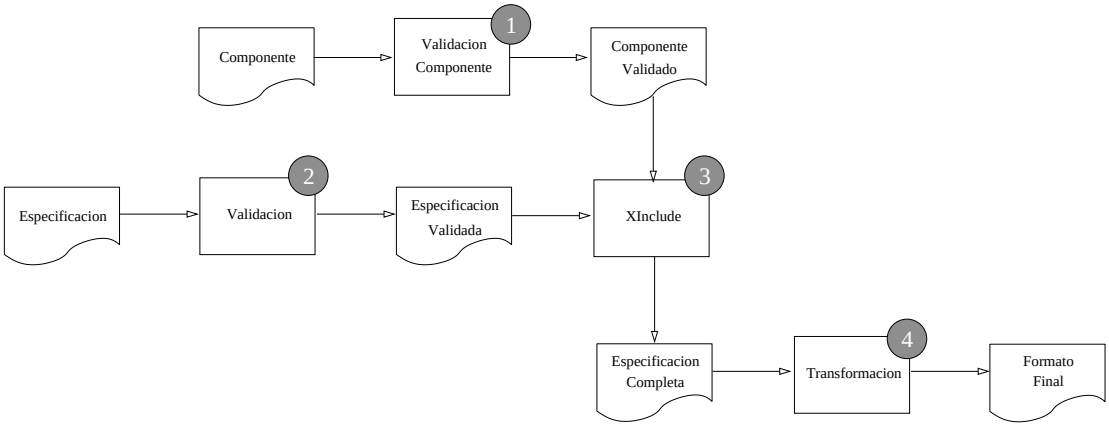


Figura 4.17: Validación, inclusión y transformación de una especificación ASL

Respecto a la inclusión, también es necesario mencionar que fuera de las especificaciones, en el *Repositorio de Configuraciones*, también es utilizada. En este caso, suele emplearse cuando se están definiendo en lenguaje PHP nuevas acciones, *widgets*, *layouts*, etc. Esta inclusión que afecta a la implementación de ciertos artefactos, es resuelta en tiempo de desarrollo al contrario que inclusión en especificaciones (sección 3.3) que es resuelta en tiempo de desarrollo.

Para finalizar, se ha de destacar que en cada distribución de WAINE existe un directorio denominado *include* que contiene la "biblioteca estándar" de especificaciones ASL. Como se comentó en la sección 3.1.4, agrupar diversos componentes en un directorio junto con la técnica de inclusión permite crear bibliotecas de componentes de especificaciones. La biliblioteca estándar de WAINE, aunque aún muy reducida y en estado incipiente, contiene una serie de *unidades de interacción* de uso común:

- chpasswd.asl: Formularios para el cambio de claves de usuario.
- motd.asl: Visualización/edición del mensaje del día de la aplicación (mensaje al estilo motd, *Message of the Day* de los sistemas Unix).
- todo.asl: Lista de tareas personales para cada usuario.
- useradm.asl: Espacios de trabajo para la administración de usuarios.
- meta.asl: Visualización de metainformación de la aplicación.

4.2.4. Paquetes (P)

WAINE no sólo utiliza ASL para crear una *interfaz de usuario* funcional, sino que se emplean otros ficheros y lenguajes (SQL, PHP, ficheros de configuración, etc.). Esto

dificulta la reutilización de una *interfaz de usuario* con todos los elementos que pueden llegar a constituirla (formularios, colores, widgets, acciones, etc.). La idea de emplear paquetes surgió para resolver el problema que representa la distribución y reutilización de este tipo de componentes.

Para WAINE se ha desarrollado un pequeño *Sistema de Gestión de Paquetes* con el objetivo de reutilizar recursos de cualquier complejidad y composición [22]. Sus principales componentes son los archivos *wpk* y la herramienta *wpkg*:

- Los archivos *wpk* (ver figura 4.18), contienen fragmentos en cualquiera de los lenguajes que emplea el *MB-UIDE* (ver sección 4.1.3), un archivo *meta.xml* conteniendo metainformación sobre el paquete y un par de scripts *preins.sh* y *postins.sh* que son ejecutados antes y después de la instalación del componente respectivamente. Todo el contenido se comprime en un archivo con formato *tgz* (tar+gzip) al que se le da extensión *.wpk*.

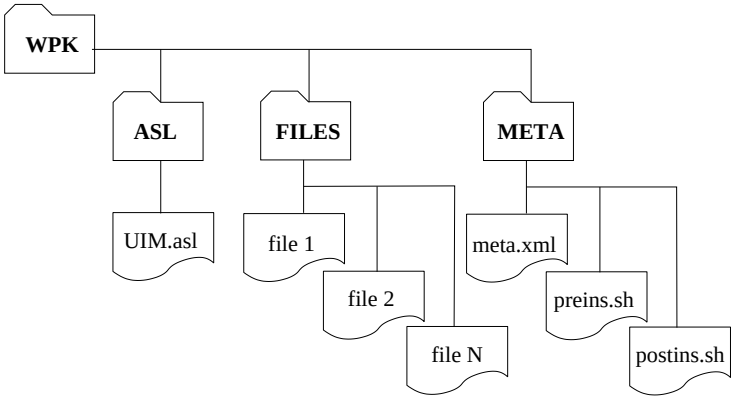


Figura 4.18: Contenido de un fichero wpk

El archivo *meta.xml* contiene información sobre el propio paquete: nombre del paquete, versión, autor, etc. Esta información es utilizada para alimentar la base de datos de paquetes instalados, resolver dependencias y detectar posibles conflictos.

En el listado 4.10 se muestra el contenido del archivo *meta.xml* para el paquete *Percon*. Este paquete contiene una *unidad de interacción* para gestionar datos de contacto de personas. Como podemos ver en el archivo aparece información como el nombre y versión del paquete, el nombre del autor y la fecha de creación, así como una descripción del mismo. También podemos ver que aparece una relación de funcionalidades ofrecidas por el paquete (etiqueta <provides>), entendiendo por funcionalidad el nombre de los elementos contenidos en el paquete (formularios, widgets, acciones, etc.). Asimismo, aparece una relación de paquetes de los que depende la instalación del paquete actual (etiqueta <depends>).

```
<wpkg>
 <package>
  <name>Percon</name>
  <ver>0.2</ver>
  <author>A.L. Delgado</author>
  <date>27-11-2006</date>
```

```
  <description>
   A component to manage personal contact
   information: addresses, telephons, emails, ...
  </description>
 </package>

 <provides>
  <func>percon.container.main</func>
  <func>percon.form.people</func>
  <func>percon.form.address</func>
  <func>percon.form.telephon</func>
  <func>percon.form.email</func>
  <func>percon.form.notes</func>
 </provides>

 <depends>
 </depends>
</wpkg>
```

Listado 4.10: Archivo meta.xml del paquete Percon

- La herramienta *wpkg* se emplea para gestionar los paquetes. Permite listar los componentes instalados, añadir, modificar y eliminar paquetes a una instancia de aplicación. Los paquetes pueden ser aplicados tanto en sistemas en desarrollo como en sistemas en producción (en este caso sobre un reposiorio). En caso de ser aplicados en sistemas en desarrollo, el paquete contendrá diversos fragmentos de código que el desarrollador añadirá a su proyecto. Si por el contrario el paquete es aplicado sobre un sistema en producción, el script *postins.sh* se encargará de procesar y añadir cada uno de los elementos a su repositorio correspondiente.

Los paquetes son utilizados en tiempo de desarrollo. Su empleo sobre sistemas en producción resulta de utilidad en las fases de mantenimiento.

4.2.5. Repositorios compartidos: centralización y federación (C,F)

La arquitectura *Mixta* de WAINE posibilita que la centralización y la federación de repositorios sean implementadas en este sistema. Para que estas técnicas sean aplicables, los repositorios compartidos deben soportar el acceso concurrente de varios de *Run-times* y estos últimos deben ser capaces de acceder a varios repositorios [7].

También hay otros asuntos que es necesario tratar. En primer lugar es necesario resolver el problema de la localización de objetos, es decir, hay que determinar en qué repositorio se encuentran los elementos que deseamos utilizar. En segundo lugar, es necesario controlar el acceso a los *Modelos de la IU* centralizados, teniendo en cuenta qué *Run-times* pueden hacer uso de los modelos centralizados y cuáles no. En realidad el control de acceso en estos sistemas debe ser más exhaustivo, siendo necesario tomar en consideración a los usuarios, determinando cuáles son los objetos del modelado a

[7] Al menos dos para el caso de la centralización: el local y el centralizado.

los que puede tener acceso un determinado usuario de forma remota. A continuación veremos cómo se han resuelto estas cuestiones en WAINE:

- **Acceso a múltiples respositorios**: Una aplicación WAINE utiliza tres repositorios: *Repositorio de la IU*, *Repositorio de Configuraciones* y *Base de Datos* (ver sección 4.1.2).

  Desde sus primeras versiones WAINE permitía emplear varias *Bases de Datos*, pero sólo se consideraba necesario un *Repositorio de la IU* y un *Repositorio de Configuraciones*. Para soportar tanto la centralización como la federación de todos los repositorios fue necesario modificar el motor de WAINE:

  1. El *run-time* ha necesitado un par de cambios para poder soportar varios *Repositorios de la IU*. Primero el motor debe admitir como orígenes para el *Repositorio de la IU* una lista de repositorios (anteriormente sólo se definía uno). En segundo lugar, el *Run-time* debe ser capaz de acceder a todos esos repositorios. Para ello, el *Run-time* emplea la misma capa de abstracción que se emplea para acceder a las *Bases de Datos*, con lo que se garantiza el acceso a distintos tipos de soporte para el *Repositorio de la IU*. En el listado 4.11 se muestra el archivo de configuración *dsource.cfg* de una aplicación. En él, podemos ver que se indica al *Run-time de la IU* que los ficheros mdb.cfg y reservas.mdb.cfg contienen la configuración necesaria para acceder a dos *Repositorios de la IU*.

```
<?php

  $DSRCMDBCFG="mdb.cfg";
  $DSRCMDBCFG1="../common/_CONF/reservas.mdb.cfg";

?>
```

Listado 4.11: Contenido del archivo dsource.cfg

  2. El *Repositorio de Configuraciones* está formado por un conjunto de carpetas y ficheros y soporta, entre otros, elementos de la presentación concreta y la configuración de la instancia de aplicación. Para compartir el *Repositorio de Configuraciones* se ha optado por compartir los ficheros que lo componen utilizando varios métodos (sistemas de ficheros en red, HTTP, etc.).

- **Localización de objetos**: En realidad en WAINE el problema de la localización está restringido al *Repositorio de la IU*, ya que los elementos de los *Repositorios de Configuraciones* y de las *Bases de Datos* son referenciados de forma "directa" desde los elementos del *Repositorio de la IU*, es decir, se indica cuál es el objeto al que hay que acceder y en qué repositorio se encuentra (p.ej. *source* para referenciar tablas y vistas). Para la localización de los distintos objetos que conforman el *Repositorio de la IU*, se ha optado por una solución sencilla. El proceso de localización de objetos recorre la lista de *Repositorios de la IU* de forma secuencial buscando el identificador del elemento requerido. Si el objeto es encontrado en el primer repositorio se recupera del mismo, si no es así, se continúa con el siguiente

repositorio de la lista. Si finalmente el objeto no se puede localizar en ninguno de los repositorios se emite un mensaje de error y el *Run-time* detiene la ejecución. Este método de localización de objetos es válido tanto para sistemas centralizados como federados.

- **Control de acceso**: Dada la variedad de soportes para los repositorios, se ha resuelto confiar en cada uno de ellos (bases de datos, sistemas de ficheros, etc.) el control de acceso a los repositorios. En las *Bases de Datos* se ha utilizado el método propio del SGBD (dirección IP, usuarios, etc.). Sobre sistemas de ficheros la seguridad basada en usuarios y grupos de usuarios habitual en este tipo de sistemas. Es una solución trivial que debe ser mejorada en el futuro.

Obviamente todos los procesos mencionados anteriormente y que afectan tanto a la centralización como a la federación en WAINE son tratados en tiempo de ejecución por el *Run-time*. Además debemos señalar que ambos métodos sólo se han implementado para funcionar con otros sistemas WAINE (ya se ha comentado la complejidad que presentaría la implementación de estas técnicas sobre sistemas heterogéneos, sección 3.2).

Para finalizar este apartado, en la tabla 4.4 se muestra un resumen de las técnicas de reutilización soportadas por WAINE. En la tabla aparece para cada técnica el momento en el que se resuelve la reutilización (columna *Resolución*), el elemento *software* que trata la técnica (columna *Herramienta*), los principales recursos reutilizados con la técnica (columna *Elementos reutilizados*) y el soporte sobre el que se lleva a cabo la reutilización (columna *Localización*). Las filas resaltadas son aquellas relacionadas con técnicas aplicadas sobre especificaciones o repositorios de *Modelos de la IU*, mientras que aquellas con un color de tinta más claro son tecnicas aplicadas en implementaciones o lenguajes que no podemos considerar parte de la especificación de la *interfaz de usuario*.

4.3. Conclusiones

En el capítulo anterior se propusieron un conjunto de técnicas para la reutilización en el *MB-UID*. Para su validación es necesario implementarlas sobre un sistema concreto. Por ello, se ha presentado un *MB-UIDE*, WAINE [20, 21], sobre el que se han implementado varias de las propuestas. Este sistema tiene entre sus objetivos principales acelerar y simplificar el desarrollo de la *interfaz de usuario* a estudiantes y programadores sin experiencia. Dispone de *Lenguaje de Descripción de la IU* propio y propone un proceso sistemático para el desarrollo de la *interfaz de usuario* basado en el modelo conceptual de la *Base de Datos* de la aplicación a construir. La arquitectura de WAINE permite la implementación de técnicas de reutilización tanto para especificaciones como para repositorios de *Modelos de la IU*.

Durante las últimas secciones se ha mostrado cómo la mayoría de las técnicas propuestas (concretamente subespecificación, adaptación, herencia, inclusión, paquetes [22], centralización y federación) han sido implementadas sobre WAINE. Esta implementación, ha puesto de manifiesto los principales problemas a tratar en la puesta en práctica de cada técnica y algunos métodos para resolverlos de forma sencilla.

Técnica	Resolución	Herramienta	Elementos reutilizados	Localización
Subespecificación	Ejecución	motor WAINE	main, containers, forms	ASL
Adaptación	Ejecución	motor WAINE	forms	ASL
Adaptación	Ejecución	intérprete PHP	widgets	RC
Herencia	Ejecución	motor WAINE	user	ASL
Herencia	Ejecución	intérprete PHP	widgets	RC
Herencia	Ejecución	web browser	estilos	RC
Inclusión	Desarrollo	asl2mdb	Fragmentos de código ASL	ASL
Inclusión	Ejecución	intérprete PHP	varios	RC
Paquetes	Desarrollo	wpkg	Fragmentos ASL, SQL, etc.	varios
Bibliotecas	Desarrollo	*dir. include*/asl2mdb	Fragmentos de ASL	ASL
Centralización	Ejecución	motor WAINE	containers, forms, users	RIU
Centralización	Ejecución	motor WAINE	widgets, presentación concreta	RC
Centralización	Ejecución	motor WAINE	tablas y vistas	BD
Federación	Ejecución	motor WAINE	containers, forms, users	RIU
Federación	Ejecución	motor WAINE	widgets, presentación concreta	RC
Federación	Ejecución	motor WAINE	tablas y vistas	BD

Tabla 4.4: Técnicas de reutilización implementadas en WAINE

Podemos concluir que las técnicas de reutilización propuestas son aplicables a la reutilización de *interfaces de usuario* en el ámbito del *MB-UID* y que los problemas que plantea su adaptación a este entorno son fácilmente resolubles.

Capítulo 5

Experiencia de reutilización en el MB-UID

En este capítulo se describe el estudio de un caso real en el que se aplican algunas de las técnicas de reutilización propuestas durante el desarrollo de una serie de proyectos con WAINE.

Se comenzará presentando el escenario en el que se ha llevado a cabo la experiencia. Con ello se pretende dar información suficiente al lector para que pueda hacerse una idea de la complejidad del entorno y de la dimensión de las aplicaciones desarrolladas. A continuación se presentan resultados cualitativos, es decir, resultados que no son obtenidos mediante métricas, pero que pensamos que pueden ser interesantes para el lector y de los que extraeremos algunas ideas que pueden ser trasladadas a otros *MB-UIDEs*. Finalmente se exponen los resultados cuantitativos obtenidos mediante diferentes métricas aplicadas sobre los proyectos desarrollados. Para ello se enumerarán las métricas empleadas, se evaluarán cuántos elementos han sido reutilizados y cómo, se caracterizarán estos elementos y se medirán los beneficios obtenidos por la aplicación de las técnicas de reutilización.

Aunque las características de los proyectos y las particularidades de WAINE limitan la generalidad de los resultados, esta experiencia ofrece pistas interesantes sobre qué técnicas son más beneficiosas para la reutilización de componentes en el *MB-UID*, así como sobre cuáles son los factores que pueden tener impacto en la reutilización de los recursos de la *interfaz de usuario*.

5.1. Caso de estudio

En la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla era necesario un entorno de desarrollo común para un conjunto de proyectos interrelacionados.

La escuela cuenta con 500 profesores y 5.400 estudiantes de 12 titulaciones[1] que hacen uso de diversos recursos: 112 salas, 856 asignaturas, 16 pantallas de notificaciones y que comparten una serie de actividades como exámenes, seminarios, reuniones, clases, etc. Las aplicaciones desarrolladas (seis en total) cubren la problemática que representa

[1]Datos académicos del año 2.013.

la administración de estos recursos comunes, así como la gestión de algunas de las actividades organizadas por la escuela. Las aplicaciones que forman parte de este proyecto y que han sido utilizadas en la experiencia presentada en esta tesis son las siguientes:

A1. Reservas: Gestiona las reservas para las distintas ubicaciones físicas de los edificios de la escuela. Las reservas pueden ser para distintos actos (exámenes, clases, reuniones, etc.) y pueden realizarse para un día, periodo o curso académico. El sistema detecta colisiones entre reservas y realiza diversas notificaciones a los interesados. La información manejada por el sistema es usada también por varias de las aplicaciones que se detallan a continuación e incluso por sistemas externos como la web de la escuela.

En la figura 5.1, se puede apreciar la *interfaz de usuario* de la aplicación, una consulta de la ocupación de un aula por medio de un código QR y cómo la web de la escuela muestra información de los próximos exámenes de una asignatura.

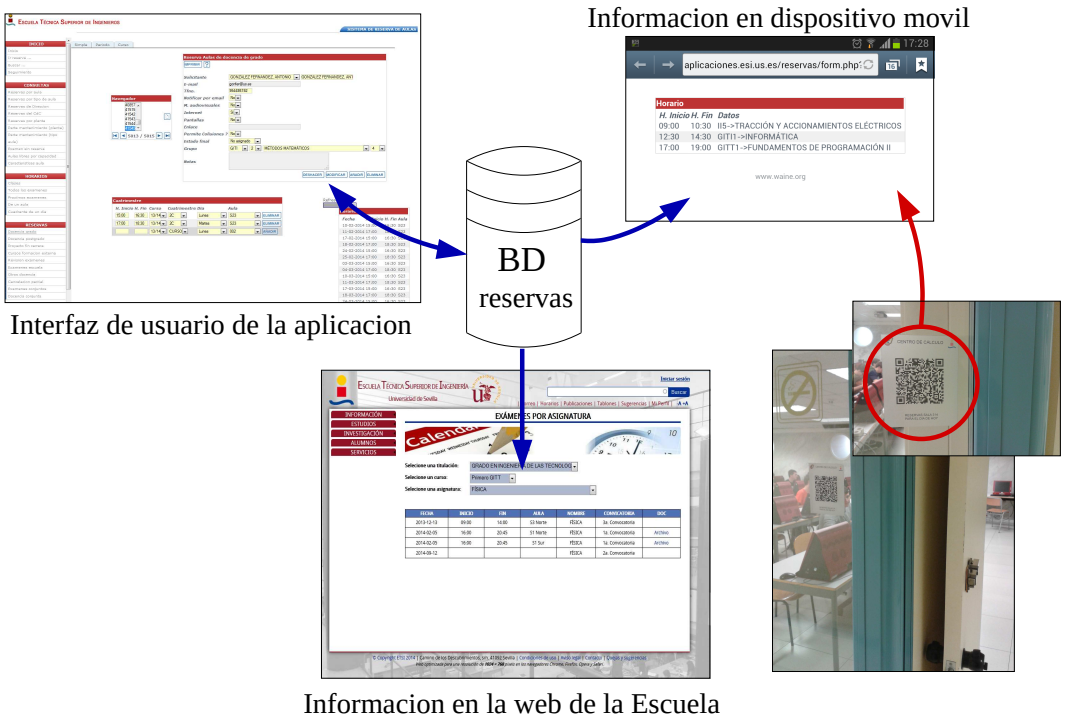


Figura 5.1: Aplicación de gestión de reservas

A2. Inventario: Esta aplicación mantiene el inventario manual del equipamiento TIC de la escuela. Maneja conceptos como familias de equipos, expedientes de compra, fechas de garantía o características de los equipos.

A3. Tablón: Es un tablón de anuncios virtual que permite a distintos tipos de usuarios (profesores, personal de administración, etc.) publicar información de interés para el alumnado: notas de exámenes, publicación de apuntes de asignaturas, etc.

A4. Prácticas: La aplicación *Prácticas* ayuda al personal que administra las prácticas en empresa a gestionar las ofertas de prácticas realizadas por las empresas, ponerlas en conocimiento de los alumnos y tratar las solicitudes de éstos.

A5. RelExteriores: Este sistema ayuda a la Subdirección de Relaciones Exteriores en la gestión de acuerdos de estudios con otras universidades, empresas y administraciones.

A6. InfoPanel: Es empleada para definir la planificación de los paneles de información distribuidos por los edificios de la Escuela Superior de Ingeniería. El sistema permite definir los contenidos y los periodos de tiempo en los que serán mostrados para cada pantalla, ya sea de forma individual o bien por zonas.

En la figura 5.2 se muestra la *interfaz de usuario* de la aplicación y algunas pantallas de información mostrando contenidos.

Figura 5.2: Aplicación de planificación de contenidos para los paneles de información

La tabla 5.1 (ordenada por fecha de desarrollo) pretende dar una idea de la complejidad de cada proyecto.

Las aplicaciones fueron desarrolladas secuencialmente, es decir, el desarrollo de la aplicación A_n no comenzó hasta que finalizó el de la A_{n-1}. Las columnas 2^a a 5^a ofrecen

para cada aplicación los datos relativos a la segunda etapa del proceso de desarrollo: número de grupos de usuarios (roles como profesor, estudiante, etc.), número de funcionalidades, número de *unidades de interacción* y número de tablas y vistas de la base de datos. Las columnas 6^a y 7^a muestran la duración del proyecto en semanas (una semana es equivalente a 40 horas de trabajo) y el tamaño de las especificaciones ASL en líneas de código (etapa 4 en el proceso de desarrollo, sección 4.1.4). La última columna resume el objeto de cada aplicación. En el anexo D se detallan las líneas de código empleadas en cada modelo y lenguaje así como los métodos empleados para su obtencion.

Aplicacion	Roles	Funcionalidades	UIs	Tablas Vistas	Duracion (semanas)	ASL (LDC)	Uso
(A1) Reservas	11	204	188	98	15	5.696	reserva de salas
(A2) Inventario	2	24	23	40	4	578	inventario TIC
(A3) Tablón	5	26	22	10	5	768	tablón de anuncios
(A4) Prácticas	1	20	18	20	4	521	prácticas en empresa
(A5) RelExteriores	1	11	10	16	4	239	relaciones exteriores
(A6) InfoPanel	2	13	11	24	6	904	información en pantallas

Tabla 5.1: Lista de aplicaciones desarrolladas

Las aplicaciones listadas en la tabla 5.1 muestran características comunes como roles, aspecto, formularios y datos. Nuestra motivación inicial no era realizar un experimento exhaustivo sobre reutilización, sino realizar el desarrollo de la forma más rápida posible. Ello nos llevó a explotar los aspectos comunes entre las aplicaciones para reducir el esfuerzo de desarrollo a través de la reutilización de componentes de la *interfaz de usuario* con las técnicas ofrecidas por WAINE.

Los nuevos recursos de la *interfaz de usuario* fueron generalmente desarrollados cuando eran necesarios en cada proyecto, sin un diseño específico orientado a la reutilización, o sea, no se desarrollaron recursos para ser reutilizados [73]. Se ha realizado una reutilización *ad-hoc* sobre recursos creados previamente. Así, el repertorio de recursos disponibles para reutilizar fue mayormente construido con la primera aplicación (obsérvese que A1 es sustancialmente mayor que A2-A6) y creció ligeramente con cada aplicación desarrollada.

El primer proyecto (A1) fue desarrollado por un equipo de gran experiencia compuesto por el autor y un profesor ayudante (con una experiencia media de 5 años). El resto de proyectos (A2-A6) fueron desarrollados por equipos compuestos por un único estudiante de nivel de grado (de baja experiencia) tutorizado por el autor. Cada estudiante desarrolló una aplicación. Todos los estudiantes mostraron habilidades similares y recibieron una formación de tres días sobre WAINE para familiarizarse con el proceso de desarrollo, la codificación ASL y las técnicas de reutilización a través del desarrollo de una aplicación de ejemplo. Todas las técnicas para reutilización soportadas por WAINE fueron practicadas en las sesiones de entrenamiento. El tutor, guiaba los pasos de desarrollo del 1 al 3 (sección 4.1.4: requisitos, *Diagrama Entidad-Relación*, tabla rol/funcionalidades). Los estudiantes colaboraban en estas etapas, pero su principal responsabilidad era escribir las especificaciones ASL (sección 4.1.4: paso 4) para

su proyecto. Los recursos desarrollados en proyectos anteriores eran conocidos por el tutor que realizaba la búsqueda, selección y evaluación *ad-hoc* de los recursos a reutilizar. Los estudiantes reutilizaban estos componentes siguiendo indicaciones del tutor y realizaban reutilización interna sobre recursos que ellos mismos habían desarrollado previamente. Cada estudiante era supervisado por el tutor un par de veces por semana.

Las aplicaciones se alojan en servidores del Centro de Proceso de Datos de la Escuela Técnica Superior de Ingeniería, cuyo personal se encarga de su administración y mantenimiento. La infraestructura sobre la que se ejecutan las aplicaciones se detalla en el Diagrama de Despliegue que aparece en la figura 5.3. Todas las aplicaciones están instaladas sobre dos servidores: un servidor de aplicaciones y uno de base de datos.

- El servidor de aplicaciones ejecuta Apache 2.2.3, PHP 5.2.0 y el run-time de WAINE versión 0.2.2. Las instancias de cada aplicación están vinculadas a este *Run-time* y el *Repositorio de Configuraciones* de cada una de ellas reside en el sistema de ficheros de este servidor.
- Por su parte, el servidor de base de datos ejecuta PostgreSQL 8.1.11. Este Sistema de Gestión de Base de Datos, soporta la *Base de Datos* y el *Repositorio de la IU* de cada aplicación.

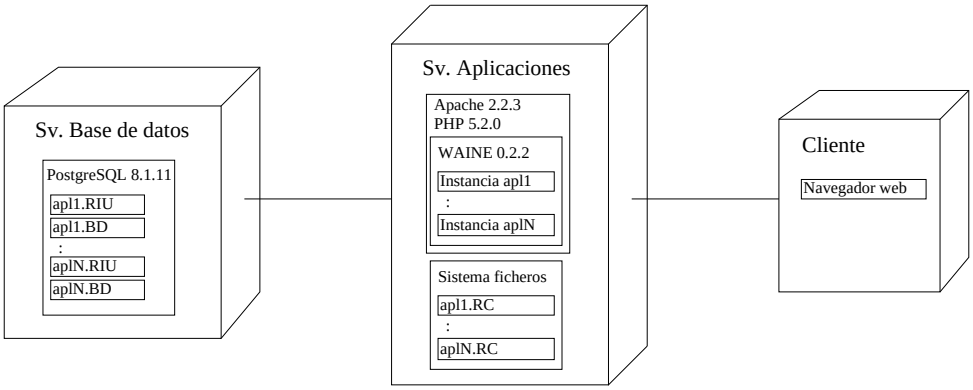


Figura 5.3: Infraestructura para las aplicaciones desarrolladas

5.2. Experiencia cualitativa en la reutilización

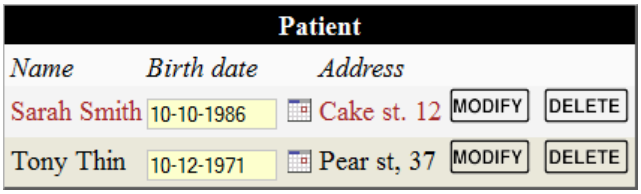
En la sección 4.2 se han descrito las técnicas de reutilización implementadas en WAINE tanto para especificaciones como para repositorios. En esta sección se describe nuestra experiencia no cuantitativa con cada técnica en este caso de estudio particular y de la que extraeremos algunas ideas que pueden ser aplicables a otros *MB-UIDEs*.

5.2.1. Subespecificación y adaptación (S,A)

Estas técnicas han sido claves en la reutilización de contenedores y formularios, además, el proyecto A1 las ha usado de forma intensiva debido a su gran tamaño y al hecho de que ha sido el primero en ser desarrollado.

La reusabilidad de los recursos con estas técnicas composicionales depende de su estructura y complejidad. Por ejemplo, no se han podido subespecificar fragmentos del *Modelo de Diálogo* debido a su estructura sintáctica jerárquica y monolítica (ver figura 4.13). Ésta sólo permite la reutilización del conjunto completo de menús asociados a un rol específico y por su definición en WAINE, roles distintos no pueden tener el mismo conjunto de funcionalidades. Por el contrario, recursos de grano fino como formularios y contenedores (etiquetas *form* y *container*) han sido frecuentemente reutilizados en el *Modelo de Presentación* (p.ej. un contenedor o formulario referenciado desde otros contenedores) así como desde el *Modelo de Diálogo* (p.ej. una *unidad de interacción* identificada por su contenedor de más alto nivel referenciada desde varias opciones de distintos menús).

La adaptación ha jugado un papel fundamental para incrementar la reusabilidad de los formularios definidos en el *Modelo de Presentación*, haciendo la reutilización más atractiva. Un conjunto de formularios similares pueden ser derivados de uno original sobreescribiendo/modificando sus propiedades. Por ejemplo, en Reservas (A1), una definición de formulario (*form_asignatura* con 10 campos para manejar la información de las asignaturas) fue referenciada 12 veces desde diferentes contenedores usando parámetros para obtener distintas variantes. Hay que tener en cuenta que el número de parámetros usados para la adaptación de un formulario puede ser considerado como un factor de coste cuando se utiliza esta técnica. En nuestro caso de estudio nunca se ha dado el caso en el que reutilizar y adaptar un formulario tuviera un coste de implementación mayor que el de escribir un nuevo formulario.




```
<container id="cparam_sample" type="form">
    <param name="formid" value="fpeople"/>
    <param name="form_type" value="table"/>
    <param name="fields_readonly" value="1:3"/>
    <param name="fields_remove" value="4"/>
</container>
```

Figura 5.4: Ejemplo de reutilización por adaptación

En la implementación de nuevos *widgets* también se ha empleado la adaptación (ver sección 4.2). Del total de cinco *widgets* definidos en esta experiencia, dos han empleado la adaptación.

WAINE es un *MB-UIDE*, centrado en los modelos de *Dominio*, *Diálogo* y *Presentación*, pero otros sistemas que usen modelos distintos también podrían beneficiarse de estas técnicas para flexibilizar la reutilización de elementos de las especificaciones.

Las características de los modelos y lenguajes empleados determinarán la aplicabilidad de las técnicas propuestas, así como su potencial para reutilizar. La expresividad y sintaxis de cada lenguaje, así como la semántica de cada *Modelo de la IU*, puede tener

un impacto decisivo para determinar qué recursos de la *interfaz de usuario* pueden ser reutilizados y en qué grado. Por lo tanto, a la hora de diseñar un *Lenguaje de Descripción de la IU* es importante tener en cuenta estos aspectos, analizando qué elementos se desea reutilizar y dándoles las características necesarias para que su reutilización sea posible.

5.2.2. Herencia (H)

Como se comentó en la sección 4.2.2, entendiendo el término de forma flexible, entre usuarios y grupos en ASL existe una relación de herencia. Los atributos del usuario (clase hija) ocultan o sobrescriben los atributos del grupo (clase padre) y los atributos del grupo quedan definidos para el usuario si éste no los han definido (ver figura 4.16). En esta relación característica se reutilizan no ya las clases propias del modelo[2], sino los atributos definidos por cada grupo, que no es necesario volver a asignar para cada usuario que pertenece al mismo. Por esta razón, la reutilización que haya existido por esta relación particular no ha sido tenida en cuenta en el resto del documento.

Sin embargo, en la implementación de algunos *widgets* empleados por el conjunto de aplicaciones se ha empleado la herencia. De los cinco *widgets* definidos en el caso de estudio, tres han utilizado la herencia como mecanismo de reutilización.

Aunque ASL no utiliza la herencia en la definición de elementos de los *Modelos de la IU*, en la sección 2.2.1 hemos visto que otros lenguajes si han empleado este método con éxito (p.ej XICL [18, 19]). La herencia debe ser considerada como método de reutilización en el diseño de nuevos *Lenguajes de Descripción de la IU*.

5.2.3. XInclude y Bibliotecas (I,B)

En nuestro caso de estudio, este método estándar de inclusión ha sido empleado para modularizar documentos ASL de gran tamaño mejorando su claridad y mantenimiento. Sin embargo, si excluimos los elementos tomados de la "biblioteca estándar", XInclude no se ha usado para reutilizar componentes de la *interfaz de usuario* entre las distintas aplicaciones desarrolladas. Esto se debe a que la infraestructura común (ver figura 5.3) sobre la que se han implantado los distintos proyectos hacía más interesantes los métodos de centralización y federación para la reutilización horizontal.

Como ASL, la mayoría de *Lenguajes de Descripción de la IU* están basados en XML [45] independientemente de su nivel de abstracción, por lo tanto son apropiados para usar este mecanismo estándar con cambios mínimos. Sin embargo, en la sección 2.2.1 hemos visto que varios lenguajes como XUL [50] o XICL [18, 19] definen sus propias etiquetas para la inclusión de documentos externos (*xul-overlay* e *import* respectivamente). Desde nuestro punto de vista, no está justificado el empleo de etiquetas específicas, al existir un estándar asentado del W3C como es XInclude. Por lo tanto se recomienda la aplicación de XInclude en los *Lenguajes de Descripción de la IU* basados en XML como mecanismo de inclusión.

[2]En la sección 3.1.1.4 la herencia se presenta como mecanismo de reutilización en la definición de nuevos elementos de los *Modelos de la IU*.

Por otra parte, es necesario tener en cuenta que muchos *Lenguajes de Descripción de la IU*, sólo soportan la definicion de algunos modelos [45, 115] (UIML, XUL, etc.). El desarrollo de un componente de la *interfaz de usuario* puede conllevar el uso de modelos no incluidos en el lenguaje de partida, lo que implicaría el empleo de varios *Lenguajes de Descripción de la IU* para realizar el modelado del componente. En este caso, en el que el recurso reutilizable está constituido por elementos de varios modelos y especificados en distintos lenguajes (que incluso pueden no seguir el estándar XML) XInclude no es una buena opción para la reutilización y distribución del componente de la *interfaz de usuario*. Los paquetes pueden ser una mejor elección en este caso.

Recordemos que XInclude presenta ciertas limitaciones: (a) la técnica está restringida a especificaciones basadas en XML, (b) es aconsejable el uso de herramientas de validación de fragmentos reutilizables ya que XInclude no trata el proceso de validación y (c) las modificaciones en los componentes de la *interfaz de usuario* reutilizados no se harán presentes en aquellas especificaciones que han hecho uso de ellos. Estas restricciones deben ser tenidas en cuenta a la hora de implementar este método en nuestros sistemas.

Por último, se ha de mencionar que algunos de los elementos de la 'biblioteca estándar' de WAINE sí han sido reutilizados en nuestro escenario. Por ejemplo, *interfaces de usuario* presentes en cada aplicación (como el usado para cambio de contraseña) han sido incluidas en todos los proyectos. Esto conlleva que el recurso ha sido fusionado en cada proyecto y por lo tanto aparece repetido en el *Repositorio de la IU* de cada aplicación. Aunque no ha sido necesario realizar un mantenimiento de estos componentes, pensamos que no ha sido la mejor forma de reutilizarlos en el escenario planteado.

5.2.4. Paquetes (P)

En el *MB-UID* se pueden emplear varios *Lenguajes de Descripción de la IU* al mismo nivel de abstracción o en diferentes niveles:

- El empleo de paquetes con lenguajes que pertenecen al mismo nivel de abstracción permite la reutilización y distribución de *Recursos Multi-modelo* como se ha comentado con anterioridad. En WAINE, los paquetes se emplean con los productos finales del proceso de desarrollo: fragmentos de ASL, de SQL o de otros elementos residentes en el *Repositorio de Configuraciones*.

- Pero los paquetes también podrían emplearse durante el proceso de desarrollo con lenguajes en diferentes niveles de abstracción, así un paquete permitiría realizar un *snapshot* de un instante determinado en el proceso de desarrollo.

Los paquetes no han sido empleados en nuestro caso de estudio. Los repositorios compartidos han resultado más cómodos para la reutilización *Inter-proyecto* en este entorno. Sin embargo, de algunos de los proyectos desarrollados se han extraído ciertos componentes complejos de la *interfaz de usuario* que han sido empaquetados para su uso futuro.

En los paquetes se ha incluido información adicional sobre el componente reutilizable: documentación, diagramas, ejemplos, etc. Esta información permite a los desarro-

lladores reutilizar el recurso de forma óptima y ayuda a realizar *asset-based development* [57] en el desarrollo de la *interfaz de usuario*.

5.2.5. Repositorios compartidos (C,F)

Los repositorios compartidos han sido la técnica más frecuentemente usada para la reutilización horizontal en nuestro caso de estudio. Se ha compartido no sólo el *Modelo de Presentación* (en el *Repositorio de la IU*) y los datos manejados por los formularios (*Base de Datos*), sino también algunos de los elementos definidos en el *Repositorio de Configuraciones* (presentación concreta, *widgets*, etc.) y cierta información relativa a los usuarios.

En el *Modelo de Presentación*, las *unidades de interacción* y los formularios usados frecuentemente (para la selección de asignaturas, salas, departamentos, etc.) se han almacenado en repositorios centrales y se han compartido entre varias aplicaciones como se muestra muestra en la figura 5.5.

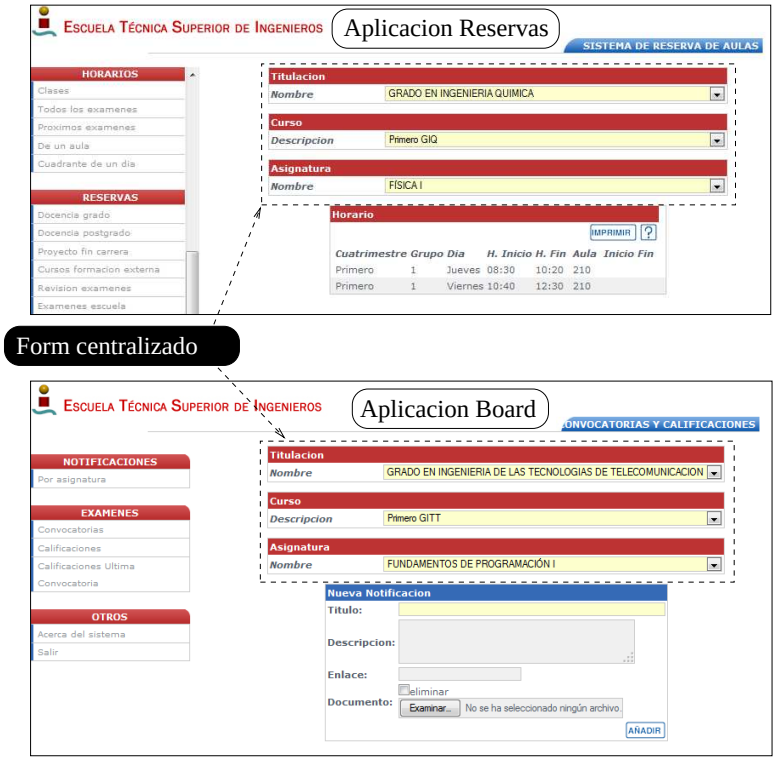


Figura 5.5: Ejemplo de formulario compartido por dos aplicaciones

Del mismo modo que en las *Bases de Datos* se extraen entidades u objetos comunes a distintos sistemas y se centralizan, en nuestro conjunto de aplicaciones se han extraído los componentes del *Modelo de Presentación* empleados para manipular esos objetos comunes del *Modelo de Dominio* y también han sido centralizados. Las ventajas de la centralización (sección 3.1.3) se aplican de este modo tanto al *Modelo de Dominio* como al *Modelo de Presentación*. Las modificaciones en los elementos del *Modelo de Dominio* centralizado que han repercutido en los elementos del *Modelo de Presentación*

que manipulan esos objetos modificados, sólo han requerido la actualización de los componentes del repositorio centralizado encargados de su manipulación en lugar de tener que actualizar cada una de las aplicaciones afectadas. A este modelo en el que tanto los elementos del *Modelo de Dominio* como los elementos del *Modelo de Presentación* que los manipulan están centralizados, lo hemos denominado PC-DOC (*Presentación centralizada - Dominio centralizado*).

En la figura 5.6, tres sistemas distintos, cada uno con sus *Repositorios de la IU*"locales" (RIU AP.1, RIU AP.2, RIU AP.3), utilizan un formulario común almacenado en un repositorio centralizado (RC-MP). Esta *unidad de interacción* presenta para todos los sistemas los mismos elementos del *Modelo de Dominio*, ya que el *Modelo de Dominio* también está centralizado (RC-DOM).

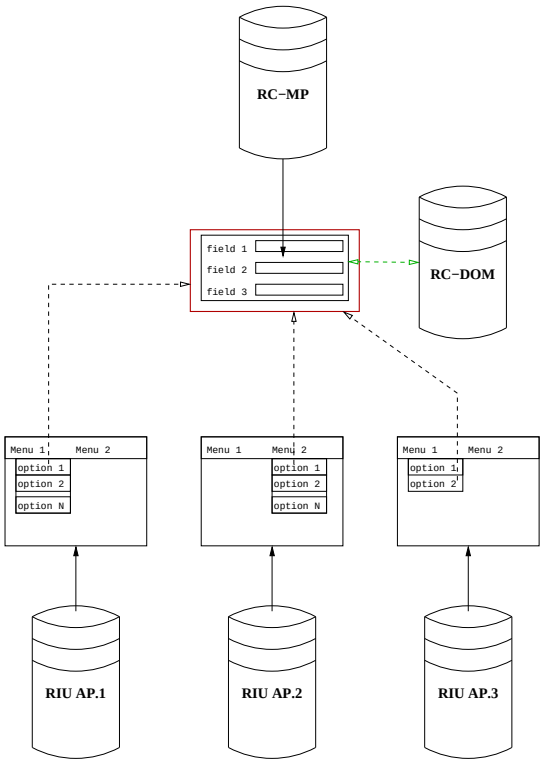


Figura 5.6: Presentación centralizada - Dominio centralizado

También es posible, que un mismo formulario residente en un repositorio centralizado, sea empleado para manejar elementos de distintos *Modelos de Dominio* que tienen una estructura similar pero que pueden residir en repositorios distintos. Así, un mismo formulario puede servir para manipular objetos del tipo "persona" que poseen una misma estructura pero que pertenecen a sistemas distintos. Incluso en el caso en el que los objetos del *Modelo de Dominio* no tuvieran exactamente la misma estructura, combinando la centralización con la adaptación, los formularios pueden ser aplicados fácilmente a esa estructura distinta. A este modelo lo hemos denominado PC-DOD (*Presentación centralizada - Dominio distribuido*).

En la figura 5.7, se puede apreciar cómo dos sistemas independientes, cada uno con su *Repositorio de la IU* particular (RIU AP.1, RIU AP.2), utilizan un mismo formula-

rio almacenado en un repositorio compartido (RC-MP). Sin embargo, en este caso, el formulario accede a *Modelos de Dominio* distintos en función del sistema que lo invoca (el sistema 1 R-DOM1 y sistema 2 R-DOM2), y por lo tanto los elementos del *Modelo de Dominio* manejados desde los dos sistemas no son los mismos.

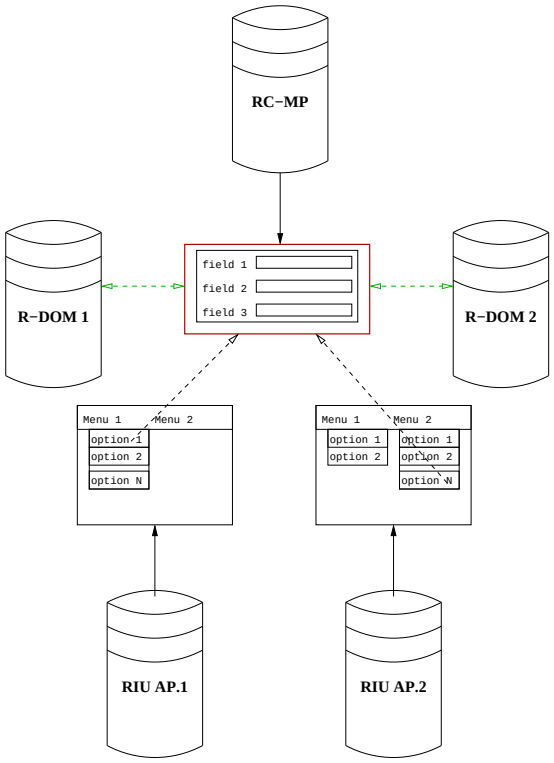


Figura 5.7: Presentación centralizada - Dominio distribuido

Otra posibilidad ha sido la reutilización elementos del *Modelo de Presentación* a través de la federación de repositorios. Algunas *interfaces de usuario* empleadas por una aplicación son mantenidas y residen en repositorios de otras aplicaciones. Por ejemplo, Prácticas (A4), mostraba *unidades de interacción* sobre destinos (p.ej. regiones, provincias, etc.) gestionadas por la aplicación RelExteriores (A5) así como interfaces relativos a profesores y departamentos que eran administrados directamente por Reservas (A1). Así, el *Modelo de Presentación* se ha compuesto desde sistemas externos como se muestra en la figura 5.8. En esta figura podemos apreciar cómo en el *Modelo de Presentación* de un nuevo sistema, cuyo *Repositorio de la IU* principal es RIU AP.1, se están utilizando formularios que residen en otros repositorios (R-MP 1 y R-MP 2).

Sin embargo, esta técnica combinada con la estructura del *Modelo de Presentación* de WAINE se puede llevar hasta el extremo, de manera que una *unidad de interacción* se cree por composición de componentes de sistemas externos, es decir, podemos crear una "*unidad de interacción federada*" compuesta por formularios y/o contenedores de otras *interfaces de usuario*. Naturalmente, para que esto se pueda llevar a la práctica, el sistema empleado debe permitirnos definir una *unidad de interacción* a ese nivel de descomposición jerárquica [20] (ver sección 4.1.1). Esta técnica podría catalogarse,

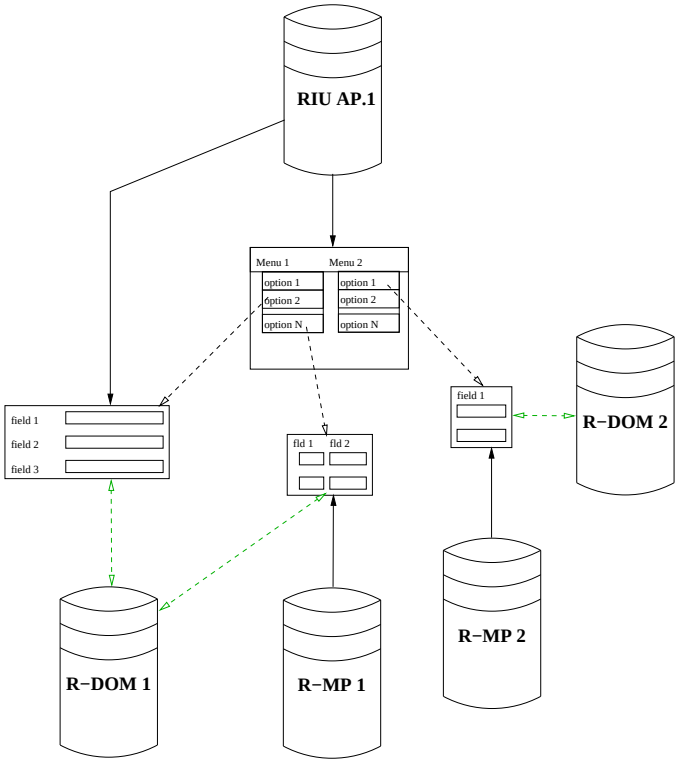


Figura 5.8: Modelo de presentación federado

dentro del ámbito de la *UI Integration* [17] [3] como un tipo o forma de *mash-up*.

En la figura 5.9, se puede apreciar cómo una *unidad de interacción* se construye por integración de formularios que residen en distintos repositorios (RIU AP.1, R-MP 1 y R-MP 2). Cada uno de estos fragmentos, puede acceder a un repositorio del *Modelo de Dominio* distinto (en el ejemplo R-DOM 1 y R-DOM 2). En este caso los elementos del *Modelo de Presentación* y los del *Modelo de Dominio* a los que se accede, provienen de varios repositorios federados, pero no tendría que ser forzosamente así, pudiendo emplear los distintos elementos del *Modelo de Presentación* federado un mismo repositorio de *Modelo de Dominio*.

La centralización de las definiciones de la presentación concreta en el *Repositorio de Configuraciones* nos ha permitido conseguir una presentación consistente de la *interfaz de usuario* tan importante en empresas y organizaciones donde los sistemas deben ofrecer una imagen corporativa. Al residir la presentación concreta en el *Repositorio de Configuraciones* centralizado, las modificaciones realizadas sobre ella han sido inmediatas y transversales a todos los sistemas desarrollados. La federación se ha aplicado también de forma esporádica sobre algunos elementos residentes en el *Repositorio de Configuraciones*.

Por último, la información relativa a la autenticación de los usuarios (contraseña en este caso particular), ha sido almacenada en un repositorio centralizado (directorio

[3] La integración de la *interfaz de usuario* (*UI Integration*) es una disciplina que trata la problemática relativa a la integración de componentes, cuando se combina su presentación más que la lógica de negocio o los datos de los mismos.

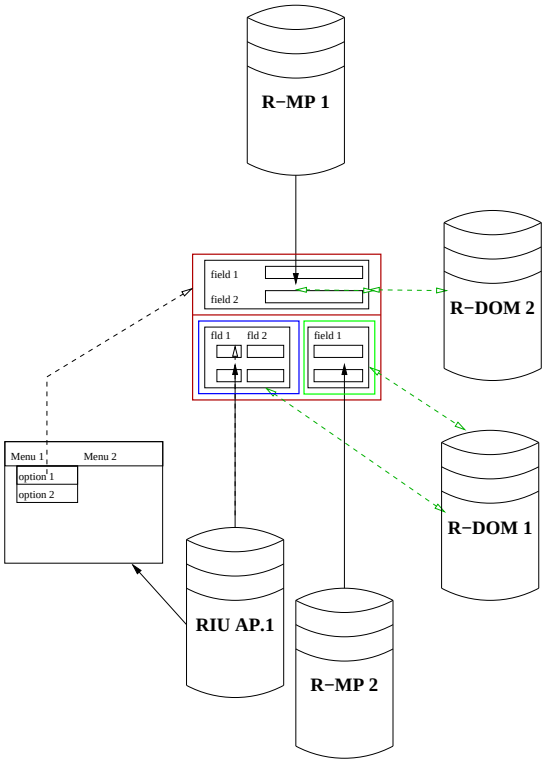


Figura 5.9: Unidad de interacción federada

X.500 accedido mediante LDAP) con el objetivo de implementar un sistema sencillo de *Single Sign-On*. Asimismo, otros datos relativos al usuario como su nombre completo y su información de contacto (teléfono, dirección, etc.) residen en el mencionado directorio. Con esta configuración, podríamos decir que el *Modelo de Usuario* de cada aplicación ha sido construido mediante la unión de su *Repositorio de la IU* local y la información contenida en el directorio X.500. En este caso sería una centralización a nivel de atributos o propiedades.

El uso de directorios es habitual en las organizaciones ya que simplifican la gestión de la información relativa a los usuarios. También permiten emplear una única clave para acceder a los sistemas corporativos (evitando que los usuarios tengan que recordar una clave por cada aplicación) y garantizan que los usuarios no tengan acceso a ninguna aplicación una vez que estos causan baja (bloqueando su información o inutilizando su clave en el directorio). La centralización total del *Modelo de Usuario*, cuando fuera posible, podría tener implicaciones interesantes sobre la administración de identidades y permitiría mantener perfiles homogéneos de los usuarios en todo el parque de aplicaciones.

Hemos de destacar que la centralización del *Modelo de Diálogo* podría permitir que la estructura jerárquica completa de determinadas actividades, o bien determinados subárboles de éstas, fueran compartidos entre un conjunto de aplicaciones (siempre que el lenguaje lo permita). Esto puede resultar interesante en grandes organizaciones, ya que los cambios en los diálogos centralizados (asociados a los *workflows* de la organización) serían inmediatos para los sistemas que hacen uso de ellos, asegurándose de este

modo que todos los sistemas utilizan la última versión de los *workflows* corporativos.

Para finalizar, obsérvese la importancia de la combinación de las distintas técnicas de reutilización. Las técnicas de reutilización en especificaciones (subespecificación, adaptación y herencia) se pueden combinar con la inclusión para permitir su aplicación *Inter-proyecto*. También en nuestro caso práctico, la subespecificación y la adaptación se han aplicado junto a la centralización y la federación.

5.3. Resultados cuantitativos de la reutilización

A continuación se presentan los resultados cuantitativos de nuestra experiencia de reutilización. En primer lugar trataremos de dar una visión general sobre la reutilización en nuestro caso práctico, fijándonos en cada una de las aplicaciones y *Modelos de la IU*, estudiando qué elementos se han reutilizado y en qué grado. Posteriormente pasaremos a realizar un análisis más profundo sobre la reutilización en el *Modelo de Presentación* que es el que ha presentado los mejores resultados en el análisis general.

El análisis de la reutilización de software es una disciplina compleja que comprende todas las fases del ciclo de vida del desarrollo del software [9, 38, 54] y requiere el cuidadoso uso de métricas [67, 8, 73]. En [73] se puede encontrar una excelente revisión sobre reutilización en la industria del software. Esta sección no intenta ofrecer un análisis exhaustivo sobre reutilización de software, pero sí quiere ser un ejemplo ilustrativo de nuestra experiencia particular en la reutilización de componentes de la *interfaz de usuario* con las técnicas implementadas en WAINE.

5.3.1. Recursos de la interfaz de usuario y cantidad de reutilización

Para el resto de este análisis, consideraremos que los componentes reutilizables (recursos de la *interfaz de usuario*) son los siguientes elementos:

- Etiquetas ASL de alto nivel de los modelos de *Usuario* (<group>), *Diálogo* (<main>) o *Presentación* (<container> o <form>). Ver figura 4.13.
- Tablas o vistas de la *Base de Datos* accedidas desde la *interfaz de usuario*.
- Archivos individuales del *Repositorio de Configuraciones* relacionados con aspectos de la presentación concreta: definiciones de estilos, *widgets*, imágenes, etc.

La *cantidad de reutilización* [34] es una métrica que valora la proporción de recursos definidos en un proyecto que han sido usados más de una vez (ya sea dentro del mismo proyecto o reutilizados en otros proyectos). Se define como [34]:

$$\text{Cantidad de reutilización} = \frac{\text{cantidad de objetos reutilizados}}{\text{cantidad total de objetos definidos}} \tag{5.1}$$

Otra forma común de expresar esta métrica es la siguiente:

$$\text{Cantidad de reutilización} = \frac{\text{líneas de código reutilizadas}}{\text{líneas totales de código}} \tag{5.2}$$

La tabla 5.2 muestra, para cada aplicación y repositorio de WAINE, el número de recursos de la *interfaz de usuario* definidos en el proyecto (fila D), el número de recursos reutilizados (fila R) y la cantidad de reutilización (columna %). La última columna y las últimas filas ofrecen las cantidades totales para cada aplicación y fila[4]. Son destacables los siguientes resultados:

			A1	A2	A3	A4	A5	A6	Total
BD		**D**	98	40	10	20	16	24	208
		R	17	2	5	2	3	1	30
		%	17,3	5,0	50,0	10,0	18,8	4,2	14,4
RIU / ASL	(MU)	**D**	11	2	5	1	1	2	22
		R	0	0	0	0	0	0	0
		%	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	(MD)	**D**	11	1	4	1	1	2	20
		R	0	0	1	0	0	0	1
		%	0,0	0,0	25,0	0,0	0,0	0,0	5,0
	(MP)	**D**	346	52	50	25	19	54	546
		R	197	12	9	6	4	14	242
		%	56,9	23,1	18,0	24,0	21,1	25,9	44,3
RC		**D**	190	3	4	3	3	4	207
		R	188	0	1	0	0	1	190
		%	98,9	0,0	25,0	0,0	0,0	25,0	91,8
Total		**D**	656	98	73	50	40	86	1003
		R	402	14	16	8	7	16	463
		%	**61,3**	**14,3**	**21,9**	**16,0**	**17,5**	**18,6**	**46,2**

Tabla 5.2: Recursos definidos y reutilizados por aplicación y modelo

- La aplicación *Reservas* (A1) es la mayor (define 656 del total de 1.003 recursos de la *interfaz de usuario*, un 65 % en porcentaje) y el 61 % de sus recursos han sido reutilizados. Observe que casi el 100 % de los recursos relacionados con aspectos de la presentación concreta en su *Repositorio de Configuraciones* (RC) han sido reutilizados por el resto (A2-A6), obteniéndose por medio de ello un aspecto homogéneo en el conjunto de aplicaciones corporativas.

- Los elementos definidos en ASL de los modelos de *Diálogo* y *Usuario* han sido raramente reutilizados. Esto puede ser explicado por la propia estructura de estos recursos en WAINE. De cualquier modo, los recursos definidos de estos modelos son una parte mínima de los recursos definidos en ASL (sólo el 7 %).

- En WAINE, el *Modelo de Presentación* comprende la mayor parte de los recursos definidos en ASL (93 %). En media, el 44 % de los recursos del *Modelo de Presentación* han sido reutilizados, alcanzándose en A1 una cantidad de reutilización

[4]En el apéndice E se describe cómo se han obtenido los datos de la tabla 5.2.

del 57 %. Los datos manejados desde la *interfaz de usuario* también han sido reutilizados, pero en un grado menor y en todo caso asociados a los formularios que los referencian.

Los resultados anteriores justifican que centremos nuestro interés en los recursos definidos en el *Modelo de Presentación* para el resto de esta sección. Creemos que un estudio más profundo de la reutilización en el *Repositorio de Configuraciones* o de elementos de la *Base de Datos* no es relevante en nuestro caso particular porque:

1. Una parte muy significativa de la definición de la presentación concreta del conjunto de aplicaciones se realizó en el *Repositorio de Configuraciones* de la primera aplicación A1 (*Reservas*). Este repositorio centralizado es casi totalmente reutilizado por el resto, por lo tanto, el beneficio es fácilmente predecible.

2. La reutilización en *Bases de Datos* ha sido estudiada profusamente en la literatura y en nuestro caso siempre tiene lugar asociada al *Modelo de Presentación*.

5.3.2. Reutilización en el modelo de presentación

En esta sección enfocamos nuestro análisis hacia la reutilización de recursos de la *interfaz de usuario* pertenecientes al *Modelo de Presentación*. En primer lugar estudiaremos la composición de este modelo y las características de sus componentes. Posteriormente se analizará la reutilización de estos elementos, observando la sobrecarga, el ámbito y el beneficio de la reutilización.

5.3.2.1. Composición del modelo de presentación

En este apartado, se describe con mayor profundidad el *Modelo de Presentación* de las aplicaciones desarrolladas. La tabla 5.3 muestra en las columnas segunda y tercera el número de formularios (#form) y contenedores (#cont) para cada aplicación. Las columnas cuarta y quinta ofrecen el número de formularios ($\#form_{reu}$) y contenedores ($\#cont_{reu}$) reutilizados y las dos últimas columnas presentan la cantidad de reutilización de dichos elementos para cada especificación ASL ($\%form_{reu}$ y $\%cont_{reu}$).

Aplicación	#form	#cont	$\#form_{reu}$	$\#cont_{reu}$	$\%form_{reu}$	$\%cont_{reu}$
(A1) Reservas	144	202	83	114	58 %	56 %
(A2) Inventario	25	27	12	0	48 %	0 %
(A3) Tablón	9	41	6	3	67 %	7 %
(A4) Prácticas	11	14	6	0	55 %	0 %
(A5) RelExteriores	9	10	4	0	44 %	0 %
(A6) InfoPanel	24	30	9	5	38 %	17 %
Total	222	324	120	122	54 %	38 %

Tabla 5.3: Composición del modelo de presentación por aplicación

Se puede apreciar que en todos los casos el número de contenedores definidos es superior al de formularios (en conjunto suponen casi un 60 % del total de elementos)

y que la cantidad de reutilización ha sido mayor para los formularios que para los contenedores. Estos resultados pueden justificarse por:

1. Los formularios son reutilizados mediante la adaptación, un método potente y flexible que favorece la reutilización de estos objetos.

2. Como los formularios han sido más reutilizados, es lógico que su proporción sea menor.

3. Los únicos casos en los que la reutilización de contenedores tiene relevancia es en las aplicaciones A1 y A6, las dos de mayor tamaño (ver tabla 5.1). De hecho en A1 se alcanza una alta tasa (un 56 %). Ello parece indicar que para que exista un alto grado de reutilización de contenedores la *interfaz de usuario* debe tener un tamaño medio-grande. Los contenedores son reutilizados mediante subespecificación bien desde el propio *Modelo de Presentación*, bien desde el *Modelo de Diálogo* (cuando identifican a una *unidad de interacción*):

   - Para que la reutilización *Intra-modelo* tenga cierta relevancia, las *unidades de interacción* de la *interfaz de usuario* deben tener cierta complejidad, de manera que contenedores que definen parte de una *unidad de interacción* y que aparezcan repetidamente en la aplicación puedan ser reutilizados por subespecificación.
   - Para la reutilización de espacios de trabajo deben existir distintos roles con espacios de trabajo comunes. Por ello, en aplicaciones con pocos roles la tasa de reutilización de espacios de trabajo es mínima.

Respecto a los valores totales, el porcentaje de formularios reutilizados en nuestro caso de estudio ronda el 54 % mientras que el de contenedores está en torno al 38 %.

5.3.2.2. Caracterización de los recursos reutilizados

El tamaño de los 242 recursos reutilizados en el modelo de presentación puede ser caracterizado a través de las líneas de código empleadas para definirlos en ASL. El tamaño de un formulario está influenciado en gran manera por el número de sus campos (aunque como sabemos en su definición pueden intervenir otros elementos), mientras que el de un contenedor está determinado por los parámetros que emplea. En la figura 5.10, se puede apreciar la frecuencia de los elementos reutilizados de formularios (a) y contenedores (b) según su tamaño.

La tabla 5.4 agrupa los resultados obtenidos. Esta tabla presenta la frecuencia tanto de formularios como de contenedores de acuerdo a su tamaño expresado en líneas de código (LDC).

El tamaño medio de un recurso reutilizado ha sido de 11,1 LDC y no hay una gran diferencia entre formularios (con media de 10,8 LDC) y contenedores (media de 11,3 LDC). El 58,3 % de los formularios reutilizados tienen un tamaño muy pequeño (*i.e.* 4-9 LDC), mientras que el 45,1 % de los contenedores son de tamaño pequeño (*i.e.* 10-15 LDC).

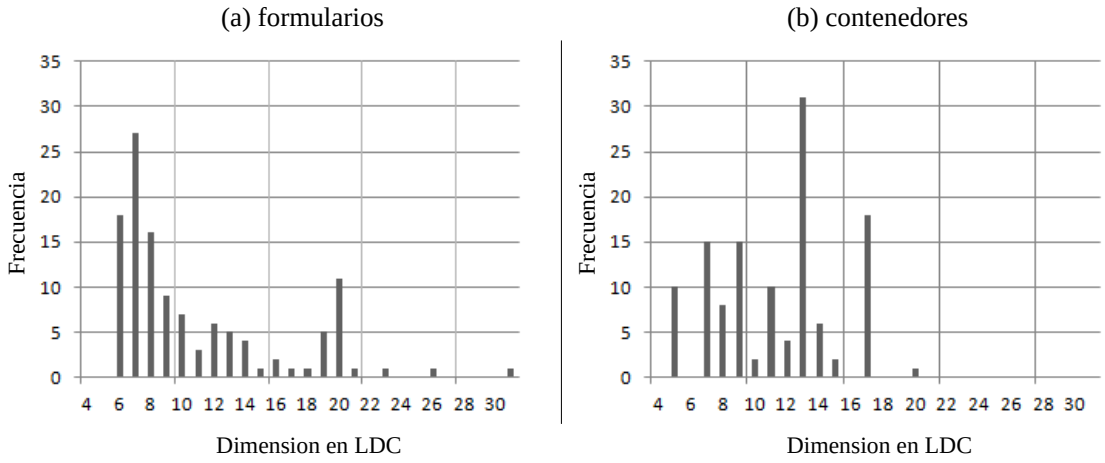


Figura 5.10: Frecuencia por tipo de recurso según su tamaño en líneas de código

LDC(tamaño)	# formularios (%)	# contenedores (%)
4-9 (diminuto)	70 (58,3 %)	48 (39,3 %)
10-15 (pequeño)	26 (21,7 %)	55 (45,1 %)
16-21 (medio)	21 (17,5 %)	19 (15,6 %)
22-27 (grande)	2 (1,7 %)	0 (0 %)
28-33 (enorme)	1 (0,8 %)	0 (0 %)
Total	120	122

Tabla 5.4: Tamaño de los recursos reutilizados en el modelo de presentación

5.3.2.3. Sobrecarga de la reutilización

Un factor importante a valorar son las líneas de código ASL necesarias para reutilizar formularios y/o contenedores (sobrecarga).

En la figura 4.13 podemos apreciar que menús, contenedores y formularios pueden ser reutilizados mediante la subespecificación. La referencia al elemento subespecificado aparecerá en una línea de código determinada, pero esta línea tiene además una función estructural, dado que se usa para enlazar/vincular los elementos que forman parte de los *Modelos de la IU*. Por lo tanto, no podemos contabilizar las líneas de código que contienen referencias a subespecificaciones como líneas dedicadas únicamente a la reutilización.

Por contra, la adaptación sí emplea más de una línea de código. Cada una de estas líneas está asociada a un parámetro y se puede afirmar que son líneas de código dedicadas exclusivamente a reutilizar un elemento de la *interfaz de usuario*. Al número total de líneas necesario para adaptar un formulario lo denominamos sobrecarga de adaptación.

La adaptación se ha empleado en el 98 % de los casos en que un formulario ha sido reutilizado. Identificar el formulario a reutilizar y decidir qué modificaciones deben hacerse sobre él requiere un esfuerzo que desafortunadamente no ha sido medido.

Pero resulta interesante comparar el tamaño medio de los formularios (en LDC) con la sobrecarga de adaptación media necesaria para su reutilización. En la figura 5.11, se presentan ambos parámetros clasificados según el tamaño del formulario. Se puede observar que la sobrecarga de adaptación media es siempre menor que el tamaño medio del elemento adaptado y que crece a un nivel menor que el tamaño del recurso. En otras palabras, cuanto mayor sea el recurso, más productiva es su reutilización en términos de ahorro de líneas de código.

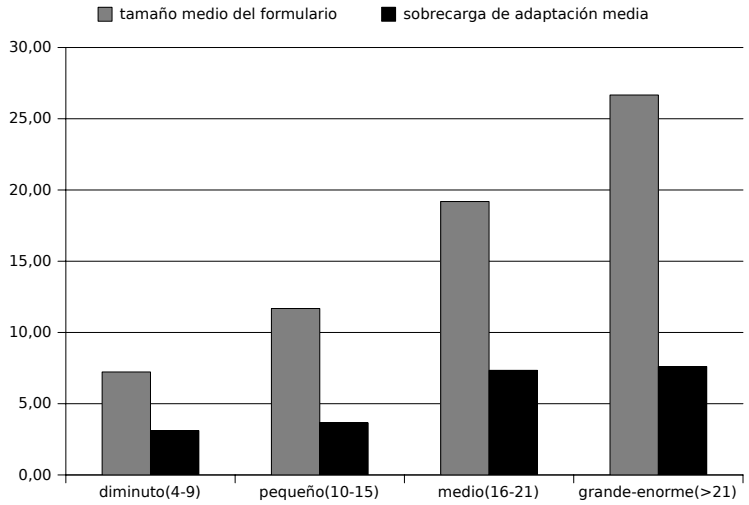


Figura 5.11: Sobrecarga de adaptación por tamaño del formulario

La tabla 5.5 muestra la sobrecarga de adaptación para cada proyecto. La primera columna muestra el tamaño en LDC del *Modelo de Presentación*, la segunda las LDC consumidas por los parámetros y la tercera la relación entre ambas.

Aplicación	tamaño del modelo de presentación (LDC)	líneas de código para adaptación	porcentaje
(A1) Reservas	5.335	1.489	27,9 %
(A2) Inventario	532	158	29,7 %
(A3) Tablón	689	240	34,8 %
(A4) Prácticas	483	168	34,8 %
(A5) RelExteriores	219	94	42,9 %
(A6) InfoPanel	865	245	28,3 %
Total	8.123	2.394	29,5 %

Tabla 5.5: Sobrecarga de adaptación por proyecto

5.3.2.4. Ámbito de la reutilización

La cantidad de reutilización, mostrada en la tabla 5.2, no indica si el ámbito de la reutilización es predominantemente interno (*Intra-proyecto*) o externo (*Inter-proyecto*).

Para observar este aspecto usaremos la métrica *nivel de reutilización*, definida como la relación entre el número de objetos reutilizados en un sistema frente al número total de objetos empleados en el mismo, pero observando el origen de esos objetos. La métrica tiene en cuenta que un sistema está compuesto por elementos a distintos niveles de abstracción. Por ejemplo, un programa escrito en lenguaje C está compuesto por módulos (archivos .c) que contienen funciones y funciones que contienen líneas de código, por lo tanto, el nivel de reutilización de un sistema codificado en C puede ser expresado en términos de módulos, funciones o líneas de código [34]. Sean:

- L = el número total de objetos de bajo nivel en el elemento de alto nivel
- E = el número total de objetos de bajo nivel de un repositorio externo en el elemento de alto nivel
- M = el número total de objetos de bajo nivel no provenientes de un repositorio externo usados más de una vez

Se definen:

$$\text{Nivel de reutilización externo} = \frac{E}{L} \tag{5.3}$$

$$\text{Nivel de reutilización interno} = \frac{M}{L} \tag{5.4}$$

$$\text{Nivel de reutilización total} = \text{Nivel de reutilización externo} + \text{Nivel de reutilización interno} \tag{5.5}$$

Para este análisis se considerarán formularios y contenedores como objetos de bajo nivel y el *Modelo de Presentación* como elemento de alto nivel. Quedarán definidos entonces:

- L = número total de formularios y contenedores en el *Modelo de Presentación* de una aplicación.
- E = número total de formularios y contenedores que provienen de un repositorio externo en el *Modelo de Presentación* de una aplicación.
- M = número total de formularios y contenedores que no provienen de un repositorio externo y son usados más de una vez en el *Modelo de Presentación* de una aplicación.

La tabla 5.6 detalla para cada aplicación: el número de diferentes recursos del *Modelo de Presentación* (contenedores o formularios) usados (L), el número de elementos que se han reutilizado internamente (*i.e.* definidos y reutilizados en el mismo proyecto) (M), el número de elementos no definidos por la aplicación pero importados de otras aplicaciones (E), el nivel de reutilización interna (NR interno), el nivel de reutilización externa (NR externo) y el nivel de reutilización total (NR total).

Aplicación	L	M	E	NR Interno	NR Externo	NR Total
(A1) Reservas	350	197	4	56,3	1,1	57,4
(A2) Inventario	54	12	2	22,2	3,7	25,9
(A3) Tablón	64	9	14	14,1	21,9	35,9
(A4) Prácticas	44	6	19	13,6	43,2	56,8
(A5) RelExteriores	29	4	10	13,8	34,5	48,3
(A6) InfoPanel	58	14	4	24,1	6,9	31,0

Tabla 5.6: Nivel de reutilización del modelo de presentación de cada aplicación

Como podía esperarse, la tabla 5.6 muestra que la reutilización interna es claramente dominante en A1 (*Reservas*) donde se han utilizado de forma intensiva la subespecificación y la adaptación para reutilizar el 56,3 % de sus 350 recursos. Otros proyectos como A4 o A5 se han beneficiado de forma significativa de la reutilización *Inter-proyecto* (principalmente de A1). La reutilización *Intra-proyecto* también es dominante en aquellas aplicaciones que están menos integradas con el resto como la de avisos en pantalla (A6) o la de inventario (A2). Finalmente, la reutilización total ha alcanzado rangos entre el 26 % (A2 es una aplicación pequeña y de reutilización interna) y el 57 % (A1 reutiliza internamente pero es una aplicación grande, A4 es pequeña pero reutiliza externamente). Esto confirma los resultados ya apuntados por Mohagheghi *et al.* [73] respecto a que el ámbito de la reutilización en grandes proyectos software tiende a ser interno, mientras que en proyectos pequeños puede ser interno o externo. Los resultados también sugieren que puede obtenerse un alto nivel de reutilización total a través de la reutilización externa cuando las *interfaces de usuario* de aplicaciones pequeñas (como A4 y A5) tienen un porcentaje elevado de elementos comunes con una aplicación de gran tamaño o por reutilización interna como en el caso de A1.

Es interesante analizar estos resultados junto con los de la sobrecarga de adaptación presentados anteriormente (ver tabla 5.5). Se puede apreciar que aquellos proyectos en los que la reutilización es predominantemente interna (A1, A2, A6) tienen una sobrecarga de adaptación menor que allí donde la reutilización es predominantemente externa (A3, A4, A5). Estos resultados indican que adaptar los recursos propios conlleva una menor sobrecarga que adaptar los recursos definidos por otros proyectos.

La reutilización de un elemento no sólo conlleva el uso de líneas de código para su reutilización. Otros factores como el tiempo empleado en buscar, examinar y evaluar los elementos seleccionados para ser reutilizados [54] deben ser tenidos en cuenta. En nuestro caso particular, hemos asumido que esos factores son despreciables, ya que como se indicó en la sección 5.1, el autor ha desarrollado A1 y diseñado y tutorizado A2-A6, por lo que los recursos han sido identificados sin mucho esfuerzo. Sin embargo, en un entorno industrial, estos factores deben ser considerados.

5.3.2.5. Beneficio de la reutilización

El análisis coste-beneficio afirma que el beneficio derivado de la reutilización puede expresarse por la ecuación [39]:

$$R = [C_{no\_reu} - C_{reu}]/C_{no\_reu} \tag{5.6}$$

Donde C_{no_reu} es el coste de desarrollar una aplicación sin reutilizar recursos y C_{reu} es el coste de desarrollar la aplicación reutilizando recursos.

Esta misma fórmula puede ser aplicada para estimar el ahorro en líneas de código (R_{LDC}) debido a la reutilización. Así:

$$R_{LDC} = [LDC_{no\_reu} - LDC_{reu}]/LDC_{no\_reu} \tag{5.7}$$

Podemos realizar ingeniería inversa y estimar LDC_{no_reu} recalculando cuántas líneas de código deberían haberse escrito si no se hubiera realizado la reutilización. Esto puede hacerse fácilmente examinando los documentos ASL y actualizando el contador de líneas de código cada vez que un contenedor o formulario es reutilizado acumulando su tamaño en líneas de código y restando las líneas codificadas para su reutilización, que asumimos como coste de la reutilización.

La tabla 5.7 muestra para cada aplicación el número de líneas codificadas (LDC_{reuse}), las líneas que habría sido necesario escribir sin reutilizar formularios y contenedores (LDC_{no_reu}) y el beneficio en líneas de código que ha supuesto la reutilización (R_{LDC}) según se define en la ecuación 5.7. Como conocemos el tiempo que se ha empleado en el desarrollo de cada proyecto, se puede estimar cuál hubiera sido su duración si no hubiera existido reutilización.

Aplicación	LDC_{reu}	LDC_{no_reu}	R_{LDC}	T. codificación ASL (semanas)	T. codificación ASL estimado SIN reutilización (semanas)
(A1) Reservas	5.335	22.225	0,76	10,5	43,7
(A2) Inventario	532	812	0,34	2,5	3,8
(A3) Tablón	689	1.636	0,58	2,8	6,6
(A4) Prácticas	483	1250	0,61	2,1	5,4
(A5) RelExteriores	219	525	0,58	2,7	6,5
(A6) InfoPanel	865	1.463	0,41	4,3	7,3
Total	8.123	27.911	0,71	24,9	73,4
Media	1.354	4.652	0,55	4,1	12,2

Tabla 5.7: Beneficio de la reutilización por aplicación

En la tabla 5.7 podemos observar que reutilizar recursos de la *interfaz de usuario* ha supuesto un ahorro de codificación en el *Modelo de Presentación* del 55 % en media, alcanzando un pico del 76 % para la mayor aplicación: *Reservas*. Si se tienen en cuenta los valores agrupados para todas las aplicaciones (fila total), se ha alcanzado un beneficio del 71 % en términos de líneas de código. Podemos estimar el tiempo de desarrollo si

no hubiera existido reutilización (columna 6) en función del tiempo real de codificación de ASL (columna 5). En este caso la reducción hubiera sido casi de un 66 %.

5.3.2.6. Otras consideraciones

Los beneficios de la reutilización del software incluyen mejoras en la productividad, la calidad y el mantenimiento como se ha destacado en numerosos estudios [73]. La tabla 5.7 sólo trata la productividad.

Durante los cuatro últimos años manteniendo las seis aplicaciones, se han realizado cinco modificaciones sobre formularios o contenedores existentes, cuatro de ellas afectaron a recursos que eran reutilizados internamente y que también se beneficiaron de la actualización; doce nuevos recursos fueron añadidos de los cuales un tercio fueron construidos reutilizando recursos existentes.

En el anexo F se muestra la relación de actuaciones realizadas sobre las aplicaciones del caso de estudio.

5.4. Conclusiones

En este capítulo se ha presentado un caso de estudio en el que WAINE se ha empleado para desarrollar varias aplicaciones que tienen ciertos elementos en común. Las técnicas de reutilización soportadas por el *MB-UIDE* han sido puestas en práctica durante la construcción de las aplicaciones para posterioremente valorar las consecuencias cualitativas y cuantitativas de su uso.

Para cada técnica aplicada en nuestro caso de estudio (subespecificación, adaptación, herencia, inclusión, centralización y federación) se ha analizado a qué elementos de los *Modelos de la IU* se han aplicado y con qué objetivo. También se ha puesto de manifiesto cuál ha sido la nuestra experiencia en el empleo de cada técnica de reutilización. Destacamos los siguientes aspectos:

- La adaptación ha sido una técnica muy interesante en el *Modelo de Presentación* para reutilizar formularios. Esta técnica ha permitido que formularios similares hayan sido obtenidos "moldeando" uno original por medio de parámetros.

- Las técnicas de reutilización en repositorios (centralización y federación) han desplazado casi totalmente a otras técnicas de reutilización *Inter-proyecto* (inclusión, bibliotecas y paquetes).

- La centralización y la federación, junto a las posibilidades constructivas que ofrece el *Modelo de Presentación* en WAINE han posibilitado una reutilización rica y flexible de este modelo. Centralización de la *Presentación*, centralización de la *Presentación* y del *Modelo de Dominio*, *Modelo de Presentación* federado y *unidades de interacción* federadas, han sido distitas formas de reutilización del *Modelo de Presentación* en el caso de estudio.

Por último se ha realizado un análisis cuantitativo de la reutilización. Se han obtenido resultados interesantes. Además del alto grado de reutilización general (entorno

al 46,2 % de todos los elementos definidos) y la reutilización casi total de aspectos de presentación concreta (que ha dotado a las aplicaciones de un aspecto corporativo), el *Modelo de Presentación* ha alcanzado una reutilización media del 44,3 %, alcanzando picos del 56,9 % (aplicación A1). Además, queremos destacar los siguientes resultados:

- El nivel de reutilización total ha sido muy elevado en algunos casos (por encima del 45 % en A1, A4 y A5). Estos niveles pueden alcanzarse mediante la reutilización interna en las aplicaciones de gran tamaño (A1) o por reutilización externa en aplicaciones más pequeñas (A4, A5) pero que tienen muchos elementos en común con una gran aplicación (A1).

- Aquellos proyectos en los que la reutilización es predominantemente interna tienen menor sobrecarga de adaptación que aquellos en los que la reutilización es mayoritariamente externa. Adaptar recursos propios es menos costoso que adaptar recursos externos. Además, la sobrecarga de adapatación media siempre ha sido menor que el tamaño medio del recurso reutilizado.

- El beneficio obtenido por el empleo de la reutilización en relación al ahorro en el tamaño de las especificaciones ronda el 71 % y se estima que el tiempo de desarrollo de los proyectos se ha reducido en un 66 %.

Capítulo 6

Conclusiones y trabajos futuros

En esta tesis se han realizado diversas propuestas relacionadas con la reutilización de *Modelos de la IU* en el *MB-UID*.

Para comenzar este trabajo se ha expuesto una panorámica sobre el *MB-UID* profundizando en las distintas propuestas sobre reutilización en este ámbito. A modo de resumen se ha elaborado una clasificación de las propuestas existentes identificando el tipo de técnica de reutilización soportada de acuerdo con los tipos de reutilización identificados por Mohagheghi *et al.* [73].

A continuación, se han presentado una serie de técnicas para la reutilización en el *MB-UID*. Algunas de ellas son generalizaciones de las presentadas en el estado del arte, otras han sido aplicadas en distintos ámbitos de la informática pero nunca en este tipo de entornos [22]. Sin embargo, nunca se había realizado un compendio de técnicas de reutilización aplicables al *MB-UID*, ni un análisis de las consecuencias de su aplicación.

Para validar las técnicas propuestas, se hacía necesario implementarlas sobre un *MB-UID* real y verificar su utilidad. En esta tesis hemos presentado WAINE [20, 21], un *MB-UIDE* sobre el que se han implementado diversas técnicas para la reutilización de elementos de los *Modelos de la IU*. WAINE dispone de su propio *Lenguaje de Descripción de la IU* (ASL) y tiene entre sus objetivos principales simplificar y acelerar el desarrollo de la *interfaz de usuario* a estudiantes y programadores sin experiencia.

Por último, WAINE ha sido empleado en desarrollo de varias aplicaciones que tienen ciertos elementos en común. Las técnicas de reutilización implementadas en el *MB-UIDE* se han puesto en práctica durante la construcción de las aplicaciones y posterioremente se han valorado las consecuencias cualitativas y cuantitativas de su uso. Son destacables los siguientes resultados:

- Las técnicas de reutilización propuestas junto a las posibilidades constructivas del *Modelo de Presentación* de WAINE han aportado una enorme flexibilidad a la implementación de la *interfaz de usuario*.
- La cantidad de reutilización total en este caso de estudio ha alcanzado un 46,2 %. Se ha conseguido un ahorro del 71 % en LDC y se estima que la reducción del tiempo de desarrollo ha alcanzado el 66 %.

Podemos concluir que la reutilización de los objetos del modelado de la *interfaz de usuario* con WAINE ha aportado resultados excelentes en nuestro caso de estudio.

6.1. Lecciones aprendidas

Aunque los resultados obtenidos están influenciados por el contexto del caso de estudio (*i.e.* proyectos, proceso de reutilización y *MB-UIDE*) algunas ideas generales pueden ser empleadas por la comunidad del *MB-UID* a la hora de diseñar nuevos sistemas o modelos con características de reusabilidad reforzadas.

6.1.1. Sobre la reusabilidad de los modelos y la subespecicación

Los *Modelos de la IU* definen elementos (asociados a etiquetas XML de alto nivel en los *Lenguajes de Descripción de la IU*) que pueden ser potencialmente reutilizados a través de la subespecificación. Esos elementos pueden ser referenciados desde otros objetos que pertenecen al mismo modelo (referencia intra-modelo) o desde objetos que pertenecen a otros modelos relacionados (referencia inter-modelo). De acuerdo a nuestra experiencia, la subespecificación intra-modelo mejora significativamente la reusabilidad de un modelo, especialmente si los recursos también pueden ser adaptados. Para profundizar en este asunto, estudiaremos el *Modelo de Diálogo* de WAINE y analizaremos brevemente las consecuencias de modificar la estructura de sus recursos reutilizables.

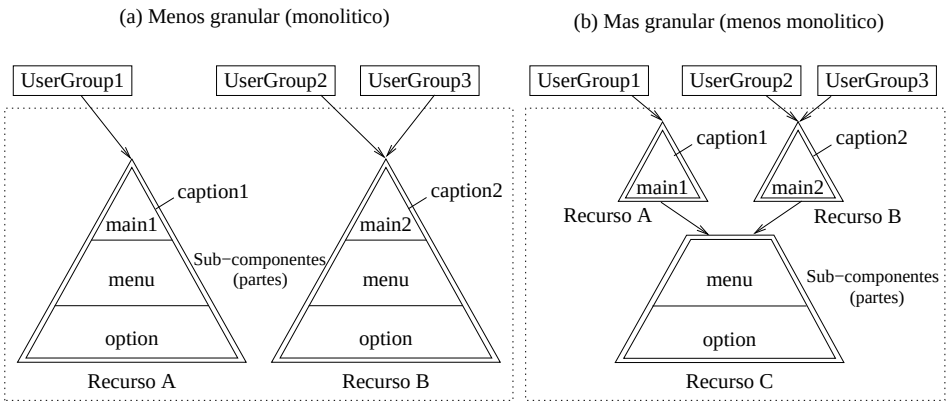


Figura 6.1: Fragmentación en un modelo de la interfaz de usuario

La figura 6.1(a) ilustra la situación actual del *Modelo de Diálogo* en WAINE. Permite definir un único elemento monolítico (*i.e.* una única etiqueta de alto nivel que incluye una definición anidada de sus componentes, ver figura 4.13) que sólo puede ser referenciado desde el *Modelo de Usuario*. En la figura, una diferencia en sólo una característica del elemento <main> (el atributo *caption*) fuerza a especificar dos recursos completos. Sin embargo, si los recursos del modelo no fueran monolíticos, sino compuestos por partes referenciables como aparece en la figura 6.1(b), podría realizarse reutilización intra-modelo. Desde luego, ello requiere cambios en la sintaxis del lenguaje. Por ejemplo, para hacer posible el caso (b), ASL debería definir <main> y <menu> como etiquetas de alto nivel y permitir que estas últimas sean referenciadas por las primeras, de forma análoga a como se hace con contenedores y formularios. La reutilización intra-modelo, junto con la adaptación han sido clave en la alta reusabilidad del

Modelo de Presentación.

Se puede argumentar que una alta fragmentación (p.ej. tres niveles de partes referenciadas en el ejemplo previo, main → menu → option) también complica el proceso de reutilización y añade sobrecarga al método. El coste de reutilizar está también influenciado por factores como el tiempo dedicado a buscar, revisar y evaluar los elementos seleccionados para ser reutilizados [54] no medidos en nuestro caso particular. Sin embargo, en entornos industriales con grandes proyectos, esos factores deben ser tenidos en cuenta. Por lo tanto, es necesario encontrar un balance entre la fragmentación del modelo y los costes asociados al proceso de reutilización.

6.1.2. Sobre el tamaño y adaptabilidad de los recursos

De forma general, podríamos decir que cuanto mayor es un recurso, más beneficiosa es su reutilización. Sin embargo, los grandes recursos de grano grueso son menos reutilizables si no son adaptables. La adaptación incrementa notablemente la reusabilidad de cualquier recurso de acuerdo a nuestra experiencia, pero requiere que los recursos sean parametrizados y conocer su estructura. Los resultados muestran que la mayoría de los formularios reutilizados han sido adaptados en algún grado. Por el contrario, cuanto más pequeño es un recurso menos beneficiosa es su reutilización.

En general, un modelo menos fragmentado tiende a usar recursos mayores y viceversa. Así, podemos concluir que existe una compensación entre el nivel de composición de un *Modelo de la IU*, el tamaño de sus recursos y el beneficio de reutilizarlos.

6.1.3. Sobre las técnicas empleadas

Diferentes técnicas se aplican a recursos de diferente naturaleza: XInclude permite reutilizar fragmentos de XML de cualquier tipo (incluso una línea aislada); la subespecificación requiere reutilizar un recurso de la *interfaz de usuario* bien formado; los repositorios permiten la reutilización de cualquier elemento (p.ej. ficheros, tablas de una *Base de Datos*, etc.). Ello ofrece flexibilidad en cuanto a lo que se puede considerar un recurso reutilizable de la *interfaz de usuario*.

XInclude es un buen mecanismo para modularizar grandes especificaciones. También es recomendable si el proceso está basado en bibliotecas de especificaciones de modelos. Sin embargo, si un recurso es modificado, aquellas especificaciones que lo incluyeron deben ser actualizadas también (problema de la fusión). Esto que es un problema común con otros entornos (p.ej. lenguajes de programación) tiene mayor relevancia en el ámbito del *MB-UID*, ya que si la inclusión se hace en modelos de muy alto nivel (según el *Framework* de referencia *Cameleon*) las distintas transformaciones (*i.e.* refinamiento, reflexión, etc.) deben volver a realizarse en los niveles inferiores.

Las arquitecturas de *MB-UIDE* basadas en *Run-time* facilitan la reutilización externa. En particular, compartir los repositorios e implementar un método de localización de objetos transparente al desarrollador en el *Run-time* nos ha brindado los siguientes beneficios: (a) se elimina el problema de la fusión, (b) ofrece flexibilidad sobre qué reutilizar y cuándo (las especificaciones pueden reutilizarse en tiempo de desarrollo, pero los objetos de la *interfaz de usuario* pueden reutilizarse en tiempo de ejecución), (c) la

localización de los recursos es más sencilla ya que los desarrolladores no necesitan conocer la "ruta" completa al objeto que desean reutilizar. Un buen contexto para utilizar esta técnica podría ser el desarrollo de la *interfaz de usuario* en líneas de productos software.

En relación a la gestión de la reutilización entre aplicaciones, una propuesta centralizada es aconsejable cuando varias aplicaciones tienen muchos elementos en común y existe un único dominio de gestión. Una propuesta federada es aconsejable cuando una aplicación está constituida por elementos distribuidos entre varios dominios de gestión (p.ej. por modelo u otro criterio).

6.2. Líneas de investigación abiertas

Este trabajo ha pretendido ser un primer paso en el campo del análisis de la reutilización en el *MB-UID*. Consecuentemente existen algunos aspectos que pueden ser tratados en investigaciones futuras. Son destacables los siguientes:

- Incrementar el conocimiento sobre reutilización en el contexto del *MB-UID* a través de: (a) explorar la reusabilidad de modelos clave en este ámbito como el *Modelo de Tareas* o el *Modelo de Diálogo*; (b) analizar las repercusiones de otras técnicas de reutilización como las citadas en la tabla 2.4 (O.O., patrones, etc.).

- Analizar casos industriales donde se realice un proceso de reutilización sistemático y en los que los costes deben ser mejor medidos, evaluando no sólo la productividad, sino también la mejora en calidad y mantenimiento.

- Analizar la reusabilidad de los estándares del W3C para la *IU Abstracta* [122] y el *Modelo de Tareas* [91]. Se espera que estos estándares tengan un impacto sustancial en la comunidad del *MB-UID*. Ello afectará de manera positiva a la reutilización, ya que se generarán más recursos comunes a diferentes entornos por: (a) una mayor adopción de modelos estándar en los sistemas futuros; (b) la posibilidad de usar los meta-modelos propuestos y su sintaxis de intercambio para transformar recursos de la *interfaz de usuario* entre diferentes entornos de desarrollo.

6.3. Publicaciones

A continuación se enumeran por orden cronológico las publicaciones del autor que guardan relación con este trabajo:

- **Aportaciones en congresos:**

  1. **WAINE: Automatic generator of web based applications**. *Third international conference on web information systems and technologies*, pág. 226–233, marzo de 2007.

En la conferencia internacional WEBIST 2007, se presentó WAINE como un sistema apropiado para el desarrollo rápido de aplicaciones web. De las características del *MB-UIDE* fueron destacadas el bajo nivel de programación necesario, su sistema de seguridad y sus capacidades de personalización.

2. **Un sistema de desarrollo de interfaces web basado en modelos para aplicaciones de gestión**. IX Congreso Internacional de Interacción Persona-Ordenador, pág. 313–318, junio de 2008.

   Durante el congreso internacional INTERACCIÓN 2008 se presentó WAINE como un *MB-UIDE* orientado al dominio de las aplicaciones de gestión. En la aportación al congreso, se describe el modelado de la *interfaz de usuario*, los lenguajes empleados, su arquitectura y sus posibilidades de reutilización. Se describen también diversas aplicaciones desarrolladas con este sistema (m3m, *conferences*, dbbibtex, etc.).

3. **On the reusability of user interface declarative models**. *Computer-Aided Design of User Interfaces VI*, pág. 313–318, LNCS, Springer Londres, 2009.

   Esta aportación analiza la aplicación de dos técnicas bien conocidas como XInclude y los sistemas de paquetes a la reutilización de especificaciones en el ámbito del *MB-UID*.

- **Artículos en revistas:**

1. **Reusing UI elements with Model-Based User Interface Development**. *International Journal of Human-Computer Studies*. Aceptado en julio de 2015. Pendiente de publicación.

   Este artículo presenta el potencial de la reutilización de elementos de la *interfaz de usuario* en el contexto del *MB-UID* y ofrece algunas indicaciones para crear *MB-UIDEs* con características de reutilización mejoradas. El estudio realizado está basado en el desarrollo de aplicaciones para la Escuela de Ingeniería de Sevilla con WAINE. Se analiza la experiencia y se discuten los beneficios y limitaciones de varias técnicas de reutilización.

ISSN	JCR Data						*Eigenfactor*® Metrics	
	Total Cites	Impact Factor	5-Year Impact Factor	Immediacy Index	Articles	Cited Half-life	*Eigenfactor*® Score	*Article Influence*® Score
1071-5819	2580	1.293	1.830	0.381	63	9.1	0.00314	0.597

Bibliografía

[1] Cameleon (context aware modelling for enabling and leveraging effective interaction) project (fp5-ist4-2000-30104).

[2] Marc Abrams and James Helms. User interface markup language (uiml) specification, 2004.

[3] Marc Abrams, Constantinos Phanouriou, Alan L Batongbacal, Stephen M Williams, and Jonathan E Shuster. Uiml: an appliance-independent xml user interface language. *Computer Networks*, 31:1695 – 1708, 1999.

[4] Seffah Ahmed and Gaffar Ashraf. Model-based user interface engineering with design patterns. *Journal of Systems and Software*, 80(8):1408 – 1422, 2007.

[5] Pierre A Akiki, Arosha K Bandara, and Yijun Yu. Adaptive model-driven user interface development systems. *ACM Computing Surveys*, 47(1):In–press, 2015.

[6] Mir Farooq Ali, Manuel A Pérez-Quinones, and Marc Abrams. Building multiplatform user interfaces with uiml. *Multiple User Interfaces–Cross-Platform Applications and Context-Aware Interfaces*, pages 95–118, 2004.

[7] Azevedo, P., Merrick, R., Roberts, and D. Ovid to auiml - user-oriented interface modelling. Technical report, UML2000, 2000.

[8] Maria Teresa Baldassarre, Alessandro Bianchi, Danilo Caivano, and Giuseppe Visaggio. An industrial case study on reuse oriented development. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 283–292. IEEE, 2005.

[9] Victor R Basili and Gianluigi Caldiera. Improve software quality by reusing knowledge and experience. *Sloan management review*, 37, 2012.

[10] F. Bodart, A.M. Hennebert, J.M. Leheureux, and J. Vanderdonckt. Computer-aided window identification in trident. *5 IFIP TC13 Conf. on Human Computer Interaction*, 1995.

[11] Barry Boehm. Managing software productivity and reuse. *Computer*, 32(9):111–113, 1999.

[12] G Calvary, J Coutaz, L Bouillon, M Florins, Q Limbourg, L Marucci, F Paternò, C Santoro, N Souchon, D Thevenin, et al. The cameleon reference framework, deliverable 1.1, cameleon project, 2002.

[13] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289 – 308, 2003.

[14] Yung-Pin Cheng and Jiue-Ren Liao. An ontology-based taxonomy of bad code smells. In *Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology*, ACST'07, pages 437–442. ACTA Press, 2007.

[15] Tim Clerckx, Kris Luyten, and Karin Coninx. Generating context-sensitive multiple device interfaces from design. In *Computer-Aided Design of User Interfaces IV*, pages 283–296. Springer, 2005.

[16] Weverton Luis da Costa Cordeiro, Guilherme Sperb Machado, Fabio Fabian Daitx, Cristiano Bonato Both, Luciano Paschoal Gaspary, Lisandro Zambenedetti Granville, Akhil Sahai, Claudio Bartolini, David Trastour, and Katia Saikoski. A template-based solution to support knowledge reuse in it change design. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 355–362. IEEE, 2008.

[17] Florian Daniel, Maristella Matera, Jin Yu, Boualem Benatallah, Regis Saint-Paul, and Fabio Casati. Understanding ui integration: A survey of problems, technologies, and opportunities. *Internet Computing, IEEE*, 11(3):59–66, 2007.

[18] Lirisnei Gomes de Sousa and Jair C Leite. Xicl - an extensible mark-up language for developing user interface and components. In *Computer-Aided Design of User Interfaces IV*, pages 247–258. Springer, 2005.

[19] Lirisnei Gomes de Sousa and Jair Cavalcanti Leite. Using imml and xicl components to develop multi-device web-based user interfaces. In *Proceedings of VII Brazilian symposium on Human factors in computing systems*, IHC '06, pages 138–147, New York, NY, USA, 2006. ACM.

[20] Antonio Delgado, Antonio Estepa, Troyano José Antonio, and Rafael Estepa. Waine: Automatic generator of web based applications. *Third international conference on web information systems and technologies*, pages 226–233, March 2007.

[21] Antonio Delgado, Antonio Estepa, Troyano José Antonio, and Rafael Estepa. Un sistema de desarrollo de interfaces web basado en modelos para aplicaciones de gestión. *IX Congreso Internacional de Interacción Persona-Ordenador*, pages 313–318, June 2008.

[22] Antonio Delgado, Antonio Estepa, José A. Troyano, and Rafael Estepa. On the reusability of user interface declarative models. In Victor Lopez Jaquero, Francisco Montero Simarro, José Pascual Molina Masso, and Jean Vanderdonckt, editors,

Computer-Aided Design of User Interfaces VI, pages 313–318. Springer London, 2009.

[23] Piergiorgio Di Giacomo. Cots and open source software components: Are they really different on the battlefield? In Xavier Franch and Daniel Port, editors, *COTS-Based Software Systems*, volume 3412 of *Lecture Notes in Computer Science*, pages 301–310. Springer Berlin Heidelberg, 2005.

[24] Dan Diaper and Neville Stanton. *The handbook of task analysis for human-computer interaction*. CRC Press, 2003.

[25] Thomas Elwert and Egbert Schlungbaum. *Modelling and generation of graphical user interfaces in the TADEUS approach*. Springer, 1995.

[26] Jürgen Engel, Christian Herdin, and Christian Märtin. Evaluation of model-based user interface development approaches. In *Human-Computer Interaction. Theories, Methods, and Tools*, pages 295–307. Springer, 2014.

[27] Tony Griffiths et al. Teallach: a model-based user interface development environment for object databases. *Interacting with Computers*, 14(1):31 – 68, 2001.

[28] Peter Fankhauser, Georges Gardarin, Moricio Lopez, J Munoz, and Anthony Tomasic. Experiences in federated databases: From iro-db to miro-web. In *VLDB*, pages 655–658, 1998.

[29] Shihong Feng and Jiancheng Wan. User interface knowledge reuse and multi-device user interface development. In *Automation and Logistics, 2007 IEEE International Conference on*, pages 1203 –1208, aug. 2007.

[30] Norman Fenton and James Bieman. *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[31] J. Foley, W. Chul, S. Kovacevic, and K. Murray. Uide - an intelligent user interface design environment. *Inteligent User Interfaces*, 1991.

[32] J. M. Cantera Fonseca, J. M. González Calleros, Gerrit Meixner, F. Paternò, J. Pullmann, D. Raggett, D. Schwabe, and J. Vanderdonckt. Model-based ui xg final report. Technical report, W3C, 5 2010.

[33] W.B. Frakes and Kyo Kang. Software reuse research: status and future. *Software Engineering, IEEE Transactions on*, 31(7):529–536, 2005.

[34] William Frakes and Carol Terry. Software reuse: metrics and models. *ACM Comput. Surv.*, 28(2):415–435, June 1996.

[35] William B Frakes and Christopher J Fox. Sixteen questions about software reuse. *Communications of the ACM*, 38(6):75–ff, 1995.

[36] Piero Fraternali, Gustavo Rossi, and Fernando Sánchez-Figueroa. Rich internet applications. *Internet Computing, IEEE*, 14(3):9–12, 2010.

[37] Pierre Geneves, Nabil Layaida, and Vincent Quint. On the analysis of cascading style sheets. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 809–818, New York, NY, USA, 2012. ACM.

[38] Nasib S Gill. Reusability issues in component-based development. *ACM SIGSOFT Software Engineering Notes*, 28(4):4–4, 2003.

[39] Nasib S. Gill. Reusability issues in component-based development. *SIGSOFT Softw. Eng. Notes*, 28(4):4–4, July 2003.

[40] Nasib S Gill and Sunil Sikka. Inheritance hierarchy based reuse & reusability metrics in oosd. *International Journal on Computer Science and Engineering*, 3(6):2300–2309, 2011.

[41] Wolfgang Goebl. A survey and a categorization scheme of automatic programming systems. In *Generative and Component-Based Software Engineering*, pages 1–15. Springer, 2000.

[42] Mohammed Gomaa, Akram Salah, and Syed Rahman. Towards a better model based user interface development environment: A comprehensive survey. *Proceedings of MICS*, 5, 2005.

[43] James Gosling. *The Java language specification*. Addison-Wesley Professional, 2000.

[44] M Grushinskiy. Xmlstarlet command line xml toolkit, 2002. *Retrieved May*, 15, 2012.

[45] J. Guerrero-Garcia, J.M. Gonzalez-Calleros, J. Vanderdonckt, and J. Muoz-Arteaga. A theoretical survey of user interface description languages: Preliminary results. In *Web Congress, 2009. LA-WEB '09. Latin American*, pages 36–43, 2009.

[46] Anja Haake, Stephan Lukosch, and Till Schümmer. Wiki-templates: adding structure support to wikis on demand. In *Proceedings of the 2005 international symposium on Wikis*, pages 41–51. ACM, 2005.

[47] John Hart and Jeffrey D'Amelia. An analysis of rpm validation drift. In *LISA*, volume 2, pages 155–166, 2002.

[48] Dominik Heckmann and Antonio Krueger. A user modeling markup language (userml) for ubiquitous computing. In *User Modeling 2003*, pages 393–397. Springer, 2003.

[49] Reid Holmes and Robert J Walker. Systematizing pragmatic software reuse. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(4):20, 2012.

[50] David Hyatt, Ben Goodger, Ian Hickson, and Chris Waterson. Xml user interface language (xul) 1.0. *Mozilla. org*, 2001.

[51] Christian Janssen, Anette Weisbecker, and Jürgen Ziegler. Generating user interfaces from data models and dialogue net specifications. In *Proceedings of the INTERACT'93 and CHI'93 conference on human factors in computing systems*, pages 418–423. ACM, 1993.

[52] Capers Jones. Software industry goals for the years 2014 through 2018. *Journal of Cost Analysis and Parametrics*, 7(1):41–47, 2014.

[53] John Klein, Harry Levinson, and Jay Marchetti. Model-driven engineering: Automatic code generation and beyond. 2015.

[54] Jyrki Kontio. A case study in applying a systematic method for cots selection. In *Proceedings of the 18th International Conference on Software Engineering*, ICSE '96, pages 201–209, Washington, DC, USA, 1996. IEEE Computer Society.

[55] Charles W Krueger. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183, 1992.

[56] Charles W Krueger. Software product line reuse in practice. In *Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on*, pages 117–118. IEEE, 2000.

[57] G. Larsen. Model-driven development: assets and reuse. *IBM Syst. J.*, 45(3):541–553, July 2006.

[58] Quentin Limbourg and Jean Vanderdonckt. Usixml: A user interface description language supporting multiple levels of independence. In *ICWE Workshops*, pages 325–338, 2004.

[59] Daniel Lucredio, Eduardo Santana de Almeida, and Renata PM Fortes. An investigation on the impact of mde on software reuse. In *Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on*, pages 101–110. IEEE, 2012.

[60] Ping Luo, Pedro Szekely, and Robert Neches. Management of interface design in humanoid. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*, pages 107–114. ACM, 1993.

[61] Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt. Derivation of a dialog model from a task model by activity chain extraction. In *Interactive Systems. Design, Specification, and Verification*, pages 203–217. Springer, 2003.

[62] Kris Luyten, Bert Creemers, Karin Coninx, et al. Multi-device layout management for mobile computing devices. Technical report, Citeseer, 2003.

[63] Fabio Mancinelli, Jaap Boender, Roberto Di Cosmo, Jérôme Vouillon, Berke Durak, Xavier Leroy, and Ralf Treinen. Managing the Complexity of Large Free and Open Source Package-Based Software Distributions. In *Automated Software Engineering*, pages 199–208. ACM, 2006.

[64] Krikor Maroukian, Kevin Lano, and Mohammad Yamin. Towards automatic generation of project-based solutions. In *Information and Knowledge Management in Complex Systems*, pages 123–134. Springer, 2015.

[65] Jonathan Marsh, David Orchard, and Daniel Veillard. Xml inclusions (xinclude). *W3C, version*, 1.1, 2015.

[66] C. Märtin. Software life cycle automation for interactive applications: The ame design environment. *Computer-Aided Design of User Interfaces*, 1996.

[67] Jorge Cláudio Cordeiro Pires Mascena, Eduardo Santana de Almeida, and Sílvio Romero de Lemos Meira. A comparative study on software reuse metrics and economic models from a traceability perspective. In *Information Reuse and Integration, Conf, 2005. IRI-2005 IEEE International Conference on.*, pages 72–77. IEEE, 2005.

[68] Efrem Mbaki, Jean Vanderdonckt, and Marco Winckler. Model-driven engineering of dialogues for multi-platform graphical user interfaces. *UsiXML 2011*, page 169.

[69] Gerrit Meixner, Gaëlle Calvary, and J. Vanderdonckt. Introduction to model-based user interfaces. Technical report, MBUI working gropup, 5 2010.

[70] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. Past, present, and future of model-based user interface development. *i-com*, 10(3):2–11, 2011.

[71] Gerrit Meixner, Marc Seissler, and Kai Breiner. Model-driven useware engineering. In *Model-Driven Development of Advanced User Interfaces*, pages 1–26. Springer, 2011.

[72] Zarko Mijailovic and Dragan Milicev. A retrospective on user interface development technology. *Software, IEEE*, 30(6):76–83, 2013.

[73] Parastoo Mohagheghi and Reidar Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 12(5):471–516, 2007.

[74] Pedro J. Molina. A review to model-based user interface development technology. In *MBUI*, 2004.

[75] Álvaro Freitas Moreira. *Geração Automática e Assistida de Interfaces de Usuário*. PhD thesis, Universidade Federal do Rio Grande do Sul, 2008.

[76] Diego Moreira and Marcelo Mrack. Sistemas dinâmicos baseados em metamodelos. In *II Workshop de Computação e Gestão da Informação (WCOMPI), Lajeado*, 2003.

[77] Giulio Mori, Fabio Paternò, and Carmen Santoro. Tool support for designing nomadic applications. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, pages 141–148, New York, NY, USA, 2003. ACM.

[78] Giulio Mori, Fabio Paterno, and Carmen Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *Software Engineering, IEEE Transactions on*, 30(8):507–520, 2004.

[79] Marcelo Mrack, Álvaro Freitas Moreira, and Marcelo Pimienta. Merlin: Interfaces crud em tempo de execução.

[80] K Mukasa and A Reuther. The useware markup language (useml)–development of user-centered interfaces using xml. In *Proc. of the 9th IFAC Symposium on Analysis, Design and Evaluation of Human-Machine-Systems, Atlanta, USA*, 2004.

[81] Brad Myers, Scott E Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28, 2000.

[82] Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, pages 195–202. ACM, 1992.

[83] Duarte Nuno Jardim Nunes. *Object modeling for user-centered development and user interface design: the wisdom approach*. PhD thesis, Universidade da Madeira, 2001.

[84] E Nwelih and IF Amadin. Modeling software reuse in traditional productivity model. *Asian Journal of Information Technology*, 7(11):484–488, 2008.

[85] Muhittin Oral, Unver Cinar, and Habib Chabchoub. Linking industrial competitiveness and productivity at the firm level. *European Journal of Operational Research*, 118(2):271–277, 1999.

[86] Cristina Palomares, Xavier Franch, and Carme Quer. Requirements reuse and patterns: a survey. In *Requirements Engineering: Foundation for Software Quality*, pages 301–308. Springer, 2014.

[87] Fabio Paterno. *Model-based design and evaluation of interactive applications*. Springer Science & Business Media, 2000.

[88] Fabio Paternò. Concurtasktrees: an engineered approach to model-based design of interactive systems. *The handbook of analysis for human-computer interaction*, pages 483–500, 2002.

[89] Fabio Paterno and Carmen Santoro. One model, many interfaces. In *Computer-Aided Design of User interfaces III*, pages 143–154. Springer, 2002.

[90] Fabio Paternò and Carmen Santoro. A unified method for designing interactive systems adaptable to mobile and stationary platforms. *Interacting with Computers*, 15(3):349–366, 2003.

[91] Fabio Paternò, Carmen Santoro, Lucio Davide Spano, and Dave Raggett. Mbui - task models. Technical report, World Wide Web Consortium (W3C), 2014.

[92] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4):19:1–19:30, November 2009.

[93] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. Engineering the authoring of usable service front ends. *Journal of Systems and Software*, 84(10):1806–1822, 2011.

[94] M Dolores Lozano Pérez. *Entorno metodológico orientado a objetos para la especificación y desarrollo de interfaces de usuario*. PhD thesis, Universitat Politècnica de València, 2001.

[95] Paulo Pinheiro da Silva. User interface declarative models and development environments: A survey. In *Interactive Systems Design, Specification, and Verification*, volume 1946 of *Lecture Notes in Computer Science*, pages 207–226. Springer Berlin / Heidelberg, 2001.

[96] Remco Poortinga-van Wijnen, Martijn Oostdijk, Bob Hulsebosch, Niels van Dijk, Roland van Rijswijk, and Hans Zandbelt. Provisioning scenarios in identity federations. 2010.

[97] Michael E Porter and Rafael Aparicio Martín. *La ventaja competitiva de las naciones*, volume 1025. Vergara, 1991.

[98] Rubén Prieto-Díaz. Status report: Software reusability. *software, IEEE*, 10(3):61–66, 1993.

[99] A. Puerta, E. Cheng, T. Ou, and J. Min. Mobile: User-centered interface building. *SIGCHI conf. on Human factors in computing systems: the CHI is the limit*, May 1999.

[100] Angel Puerta and Jacob Eisenstein. Ximl: A universal language for user interfaces. *White paper*, 2001.

[101] Angel Puerta and Jacob Eisenstein. Ximl: a common representation for interaction data. In *Proceedings of the 7th international conference on Intelligent user interfaces*, IUI '02, pages 214–215, New York, NY, USA, 2002. ACM.

[102] Angel R Puerta. The mecano project: Comprehensive and integrated support for model-based interface development. In *CADUI*, volume 96, pages 19–36, 1996.

[103] Juan C Quiroz, Sushil J Louis, and Sergiu M Dascalu. Interactive evolution of xul user interfaces. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 2151–2158. ACM, 2007.

[104] Frank Radeke, Peter Forbrig, Ahmed Seffah, and Daniel Sinnig. Pim tool: Support for pattern-driven and model-based ui development. In Karin Coninx, Kris

Luyten, and KevinA. Schneider, editors, *Task Models and Diagrams for Users Interface Design*, volume 4385 of *Lecture Notes in Computer Science*, pages 82–96. Springer Berlin Heidelberg, 2007.

[105] David Raneburger, Gerrit Meixner, and Marco Brambilla. Platform-independence in model-driven development of graphical user interfaces for multiple devices. In *Software Technologies*, pages 180–195. Springer, 2014.

[106] David Raneburger, Roman Popp, and Jean Vanderdonckt. An automated layout approach for model-driven wimp-ui generation. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 91–100. ACM, 2012.

[107] Thiagarajan Ravichandran and Marcus A Rothenberger. Software reuse strategies and component markets. *Communications of the ACM*, 46(8):109–114, 2003.

[108] Dave Roberts, Dick Berry, Scott Isensee, and John Mullaly. Designing for the user with ovid: Bridging the gap between software engineering and user interface design. 1998.

[109] Marcus A Rothenberger, Kevin J Dooley, Uday R Kulkarni, and Nader Nada. Strategies for software reuse: A principal component analysis of reuse practices. *Software Engineering, IEEE Transactions on*, 29(9):825–837, 2003.

[110] R Schaefer, B Steffen, and M Wolfgang. Task models and diagrams for user interface design. In *Proceedings of 5th International Workshop, TAMODIA'2006*, pages 39–53, 2006.

[111] Egbert Schlungbaum. *Model-based user interface software tools current state of declarative models*. Graphics, Visualization & Usability Center, Georgia Institute of Technology, 1996.

[112] Douglas C Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):0025–31, 2006.

[113] Daniel Sinnig, Homa Javahery, Peter Forbrig, and Ahmed Seffah. Patterns and components for enhancing reusability and systematic ui development. *Proceedings of HCI International, Las Vegas, USA*, 9, 2005.

[114] Ian Sommerville. *Ingeniería del software*. Pearson Educación, 2005.

[115] Nathalie Souchon and Jean Vanderdonckt. A review of xml-compliant user interface description languages. In *Interactive Systems. Design, Specification, and Verification*, volume 2844 of *Lecture Notes in Computer Science*, pages 391–401. Springer Berlin / Heidelberg, 2003.

[116] Richard M Stallman and Zachary Weinberg. The c preprocessor. *Free Software Foundation*, 1987.

[117] P Szekely, P Sukaviriya, P Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools: the mastermind approach. *Engineering for Human-Computer Interaction*, pages 120–150, 1996.

[118] S Tucker Taft. *Consolidated Ada Reference Manual: Language and Standard Libraries: International Standard ISO/IEC 8652/1995 (E) with Technical Corrigendum 1*, volume 2219. Springer, 2000.

[119] Hallvard Trætteberg. Integrating dialog modeling and domain modeling - the case of diamodl and the eclipse modeling framework. *J. UCS*, 14(19):3265–3278, 2008.

[120] Jean Vanderdonckt, Elizabeth Furtado, V Furtado, Quentin Limbourg, W Silva, D Rodrigues, and L Taddeo. Multi-model and multi-level development of user interfaces. *Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces*, pages 193–216, 2003.

[121] Jean Vanderdonckt and Francisco Montero Simarro. Generative pattern-based design of user interfaces. In *Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems*, PEICS '10, pages 12–19, New York, NY, USA, 2010. ACM.

[122] Jean Vanderdonckt, Ricardo Tesoriero, Nesrine Mezhoudi, Vivian Motti, Fraçois Beuvens, and Jérémie Melchior. Mbui - abstract user interface models. Technical report, World Wide Web Consortium (W3C), 2014.

[123] Jean M Vanderdonckt and François Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*, pages 424–429. ACM, 1993.

[124] Chris Vandervelpen and Karin Coninx. Towards model-based design support for distributed user interfaces. In *Proceedings of the third Nordic conference on Human-computer interaction*, pages 61–70. ACM, 2004.

[125] Windson Viana and Rossana M.C. Andrade. Xmobile: A mb-uid environment for semi-automatic generation of adaptive applications for mobile devices. *Journal of Systems and Software*, 81(3):382 – 394, 2008.

[126] Karel Vredenburg, Ji-Ye Mao, Paul W Smith, and Tom Carey. A survey of user-centered design practice. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 471–478. ACM, 2002.

[127] Norman Walsh and Richard L Hamilton. *DocBook 5: The definitive guide*. .º'Reilly Media, Inc.", 2010.

[128] Stephanie Wilson and Peter Johnson. Empowering users in a task-based approach to design. In *Proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques*, pages 25–31. ACM, 1995.

Apéndice A

DTD del lenguaje ASL

```
<!ELEMENT asl (xi:include*, head?, (group | main | form |container
    |workflow )* )>

<!ELEMENT xi:include (xi:fallback?)>
<!ATTLIST xi:include
    xmlns:xi          CDATA       #FIXED "http://www.w3.org/2001/XInclude"
    href              CDATA       #IMPLIED
    parse             (xml|text)  "xml"
    xpointer          CDATA       #IMPLIED
    encoding          CDATA       #IMPLIED
    accept            CDATA       #IMPLIED
    accept-language CDATA         #IMPLIED
>

<!ELEMENT xi:fallback ANY>
<!ATTLIST xi:fallback
    xmlns:xi    CDATA    #FIXED "http://www.w3.org/2001/XInclude"
>

<!ELEMENT head (meta*)>
                                                                    <!--
    Head tag: Contains meta-data about the -->
<!ELEMENT meta EMPTY>
<!ATTLIST meta   class   CDATA #REQUIRED
                 name    CDATA #REQUIRED
                 value   CDATA #REQUIRED
                 ord CDATA '' >

<!ELEMENT doc (#PCDATA)>              <!-- Used for documentation
    porpouses -->

<!ELEMENT prop (vars?, ufields?, uevents?, account?)>
<!ELEMENT vars (#PCDATA)>             <!-- Profile vars: any variable in
    the system can be redefined here -->
<!ELEMENT ufields (uval+)>
<!ELEMENT uval (#PCDATA)>             <!-- Up to ten user values can be
    defined -->
<!ELEMENT uevents (onlogin?, onlogout?)>
```

```
<!ELEMENT onlogin (action)>              <!-- onlogin event -->
<!ELEMENT onlogout (action)>             <!-- onlogout event -->
<!ELEMENT account EMPTY>


<!-- validto: Date format YYYY-MM-DD -->
<!-- changeat: A number followed by day, week or year -->
<!-- lock: Y|N -->

<!ATTLIST account validto CDATA ''
                  changeat CDATA ''
                  lock CDATA '' >

<!ELEMENT group (doc?, user*, prop?)>

<!-- gid: group identifier, a number -->
<!-- name: group name -->
<!-- groupid: main menu for users in this group -->

<!ATTLIST group gid  CDATA #REQUIRED
                name CDATA #REQUIRED
                descr CDATA  ''
                mainid CDATA ''>

<!ELEMENT user (doc?, prop?)>


<!-- uid: user identifier, a number  -->
<!-- username -->
<!-- plain text passwd -->
<!-- main menu for this user -->
<!-- user description -->
<!ATTLIST user uid CDATA #REQUIRED
               name CDATA #REQUIRED
               passwd CDATA #REQUIRED
               mainid IDREF #IMPLIED
               descr CDATA  ''>


<!ELEMENT main (doc?, menu+)>
<!ATTLIST main id ID #REQUIRED
               caption CDATA #REQUIRED
               tooltip CDATA ''>

<!ELEMENT menu (doc?, option+, acl?)>
<!ATTLIST menu  caption CDATA #REQUIRED
                tooltip CDATA ''
                img CDATA ''>

<!ELEMENT option (doc?, action?, param*, acl?)>
<!ATTLIST option  caption CDATA #REQUIRED
                  call CDATA ''
                  action CDATA ''
```

```
                type (form| container| relation| split| tab|
                   tab-buttons| workflow) 'form'
                url CDATA ''
                tooltip CDATA ''
                img CDATA ''>


<!ELEMENT container (doc?, param*)>
<!ATTLIST container  id ID #REQUIRED
                  type (form| relation| split| tab| tab-buttons|
                     workflow) 'form'  >

<!ELEMENT param EMPTY>
<!ATTLIST param name CDATA #REQUIRED
                ord  CDATA '1'
                value CDATA #REQUIRED>

<!ELEMENT workflow (doc?, msg?, prev?, alt?, next?)>
<!ATTLIST workflow  id ID #REQUIRED
                    containerid IDREF #IMPLIED
                    url CDATA ''>

<!ELEMENT prev EMPTY>
<!ATTLIST prev  id IDREF #REQUIRED
                    msg CDATA ''
                    button CDATA ''>


<!ELEMENT next EMPTY>
<!ATTLIST next  id IDREF #REQUIRED
                    msg CDATA ''
                    button CDATA ''>


<!ELEMENT alt EMPTY>
<!ATTLIST alt  id IDREF #REQUIRED
                    msg CDATA ''
                    button CDATA ''>


<!ELEMENT form (doc?, datasource?, filter?, orderby?, theme?, help?,
   fields,events?, buttons?, acl?)>
<!ATTLIST form id ID #REQUIRED
               source CDATA ''
               caption CDATA #REQUIRED
               tooltip CDATA '' >
<!ELEMENT datasource (#PCDATA)>
<!ELEMENT filter (#PCDATA)>
<!ELEMENT orderby (#PCDATA)>
<!ELEMENT theme (#PCDATA)>
<!ELEMENT help (#PCDATA)>
<!ELEMENT events (onload?, onunload?, beforeretrieve?, afterretrieve?,
   beforeinsert?, afterinsert?, beforeupdate?, afterupdate?,
   beforedelete?, afterdelete?)>
```

```
<!ELEMENT onload (action)>
<!ELEMENT onunload (action)>
<!ELEMENT beforeretrieve (action)>
<!ELEMENT afterretrieve (action)>
<!ELEMENT beforeinsert (action)>
<!ELEMENT afterinsert (action)>
<!ELEMENT beforeupdate (action)>
<!ELEMENT afterupdate (action)>
<!ELEMENT beforedelete (action)>
<!ELEMENT afterdelete (action)>
<!ELEMENT buttons (action+)>

<!ELEMENT action (doc?, tooltip?, code?, actparam?, msg?, acl?)>
<!ATTLIST action type      CDATA #REQUIRED
                 caption   CDATA ''
                 datareq   CDATA ''
                 autoparam CDATA ''
                 btn       CDATA ''>

<!ELEMENT tooltip (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT actparam (#PCDATA)>
<!ELEMENT msg (#PCDATA)>

<!ELEMENT acl (doc?, ace+)>
<!ELEMENT ace EMPTY>
<!ATTLIST ace gid CDATA #REQUIRED
              read CDATA '0'
              insert CDATA '0'
              update CDATA '0'
              delete CDATA '0'
              action CDATA '0'>

<!ELEMENT fields (key?, (fkey| field| justcode| string| text| int|
    float| date| time| timestamp| blob-file| image-file)+)>

<!ELEMENT field (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>

<!ATTLIST field source CDATA ''
                type CDATA 'string'
                caption CDATA ''
                len CDATA '0'
                maxlen CDATA '0'
                attr CDATA '-'
                msg CDATA ''
                canbenull CDATA 'N'
                format CDATA '%s'
                picture CDATA '.'
                tooltip CDATA ''
                >
```

```
<!ELEMENT search (#PCDATA)>
<!ELEMENT searchfield (#PCDATA)>
<!ELEMENT browsearch (#PCDATA)>
<!ELEMENT defvalue (#PCDATA)>
<!ELEMENT chkvalue (#PCDATA)>
<!ELEMENT style (#PCDATA)>
<!ELEMENT widget (#PCDATA)>
<!ELEMENT wdgparam (#PCDATA)>
<!ELEMENT width (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ELEMENT rowfunc (#PCDATA)>
<!ELEMENT colfunc (#PCDATA)>
<!ELEMENT tag (#PCDATA)>


<!ELEMENT key (defvalue?, chkvalue?, search?, searchfield?, browsearch?,
    style?, widget?, wdgparam?, width?, height?, rowfunc?, colfunc?,
    tag?)>
<!ATTLIST key source CDATA #REQUIRED
                type CDATA 'string'
                caption CDATA ''
                len CDATA '0'
                maxlen CDATA '0'
                attr CDATA 'H'
                msg CDATA ''
                canbenull CDATA 'Y'
                format CDATA '%s'
                picture CDATA ''
                tooltip CDATA ''
>

<!ELEMENT fkey (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>
<!ATTLIST fkey source CDATA #REQUIRED
                type CDATA 'string'
                caption CDATA ''
                len CDATA '0'
                maxlen CDATA '0'
                attr CDATA 'H'
                msg CDATA 'Unset value in foreign key'
                canbenull CDATA 'N'
                format CDATA '%s'
                picture CDATA ''
                tooltip CDATA ''
>

<!ELEMENT justcode (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>
<!ATTLIST justcode source CDATA ''
                type CDATA 'code'
                caption CDATA ''
                len CDATA '20'
```

```
                maxlen CDATA  '20'
                attr CDATA  '-'
                msg CDATA  ''
                canbenull CDATA 'Y'
                format CDATA  '%s'
                picture CDATA  '^.*$'
                tooltip CDATA ''
>


<!ELEMENT string (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>
<!ATTLIST string source CDATA ''
                type CDATA 'string'
                caption CDATA ''
                len CDATA '20'
                maxlen CDATA  '20'
                attr CDATA  '-'
                msg CDATA  ''
                canbenull CDATA 'N'
                format CDATA  '%s'
                picture CDATA  '^.*$'
                tooltip CDATA ''
>

<!ELEMENT text (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>
<!ATTLIST text source CDATA ''
                type CDATA 'text'
                caption CDATA ''
                width CDATA '20'
                height CDATA '6'
                attr CDATA  '-'
                msg CDATA  ''
                canbenull CDATA 'Y'
                format CDATA  '%s'
                picture CDATA  ''
                tooltip CDATA ''
>

<!ELEMENT int (defvalue?, chkvalue?, search?, searchfield?, browsearch?,
    style?, widget?, wdgparam?, width?, height?, rowfunc?, colfunc?,
    tag?)>
<!ATTLIST int source CDATA ''
                type CDATA 'integer'
                caption CDATA ''
                len CDATA '6'
                maxlen CDATA  '6'
                attr CDATA  '-'
                msg CDATA  ''
                canbenull CDATA 'N'
                format CDATA  '%d'
```

```
                picture CDATA  '^\d+$'
                tooltip CDATA ''
>

<!ELEMENT float (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>
<!ATTLIST float source CDATA ''
                type CDATA 'float'
                caption CDATA ''
                len CDATA '6'
                maxlen CDATA  '6'
                attr CDATA  '-'
                msg CDATA  ''
                canbenull CDATA 'N'
                format CDATA  '%02.2f'
                picture CDATA  '^(-+)?(\d+)?(\.\d+)?$'
                tooltip CDATA ''
>

<!ELEMENT date (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>
<!ATTLIST date source CDATA ''
                type CDATA 'date'
                caption CDATA ''
                len CDATA '10'
                maxlen CDATA  '10'
                attr CDATA  '-'
                msg CDATA  ''
                canbenull CDATA 'N'
                format CDATA  '%10s'
                picture CDATA '^[0-9]{1,2}\-[0-9]{1,2}\-[0-9]{4}$'
                tooltip CDATA ''
>

<!ELEMENT time (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
    colfunc?, tag?)>
<!ATTLIST time source CDATA ''
                type CDATA 'time'
                caption CDATA ''
                len CDATA '8'
                maxlen CDATA  '8'
                attr CDATA  '-'
                msg CDATA  ''
                canbenull CDATA 'N'
                format CDATA  '%s'
                picture CDATA  '^[0-9]{1,2}:[0-9]{1,2}:[0-9]{2}$'
                tooltip CDATA ''
>

<!ELEMENT timestamp (defvalue?, chkvalue?, search?, searchfield?,
    browsearch?, style?, widget?, wdgparam?, width?, height?, rowfunc?,
```

```
    colfunc?, tag?)>
<!ATTLIST timestamp source CDATA ''
                type CDATA 'time'
                caption CDATA ''
                len CDATA '19'
                maxlen CDATA '19'
                attr CDATA '-'
                msg CDATA ''
                canbenull CDATA 'N'
                format CDATA '%s'
                picture CDATA '.'
                tooltip CDATA ''
>

<!ELEMENT blob-file (style?, widget?, wdgparam?, rowfunc?, colfunc?,
    tag?)>
<!ATTLIST blob-file source CDATA #REQUIRED
                type CDATA 'blob'
                caption CDATA ''
                len CDATA '0'
                maxlen CDATA '0'
                attr CDATA '-'
                msg CDATA ''
                canbenull CDATA 'Y'
                format CDATA '%s'
                picture CDATA '.'
                tooltip CDATA ''
>

<!ELEMENT image-file (style?, widget?, wdgparam?, width?, height?,
    rowfunc?, colfunc?, tag?)>
<!ATTLIST image-file source CDATA #REQUIRED
                type CDATA 'image'
                caption CDATA ''
                len CDATA '0'
                maxlen CDATA '0'
                attr CDATA '-'
                msg CDATA ''
                canbenull CDATA 'Y'
                format CDATA '%s'
                picture CDATA '.'
                tooltip CDATA ''
>
```

Listado A.1: Sintaxis del lenguaje ASL en formato DTD

Apéndice B

Desarrollo de una aplicación

Para ilustrar el proceso de desarrollo de una aplicación, en este anexo se va a detallar la creación de un sistema de gestión sugerencias muy simple (ver figura B.1). En este sistema, un secretario recibe sugerencias por distintos medios: email, correo convencional, llamadas telefónicas o SMS y rellenará un formulario con la información requerida. El supervisor sólo se encarga de generar una serie de categorías para las personas que envían sugerencias y otra para las propias sugerencias. En esta aplicación simple no existen informes o consultas.

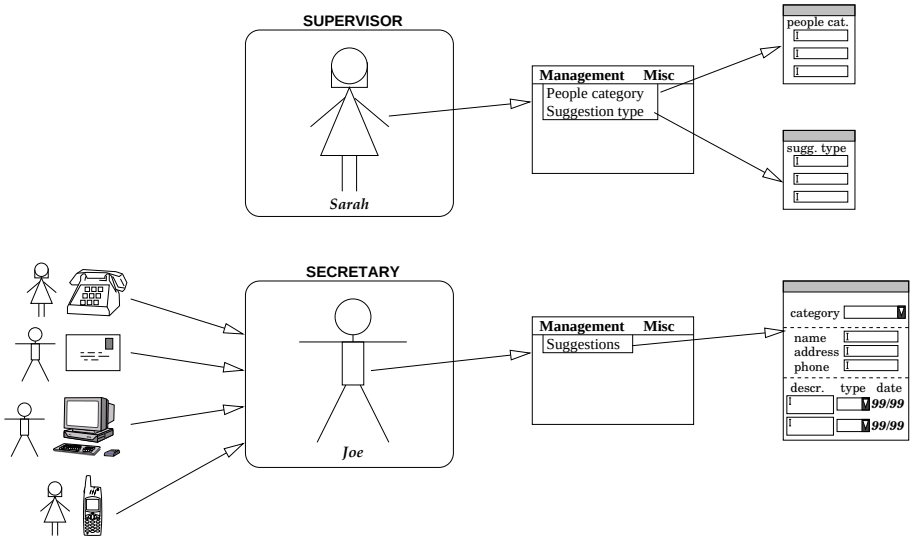


Figura B.1: Diagrama informal de la aplicación

El sistema además debe cumplir las siguientes restricciones:

1. La categoría por defecto para las personas (con código 0) no puede ser eliminada.
2. Una categoría para personas no puede ser eliminada si hay personas que pertenezcan a ella.
3. Cuando una persona es eliminada, todas sus sugerencias deben ser eliminadas.

4. Los tipos de sugerencias no pueden ser eliminados.

Para implementar esta aplicación se ha seleccionado como *Base de Datos* y como *Repositorio de la IU* la base de datos sqlite 2.8.16. Sqlite (http://www.sqlite.org/) es una base de datos relacional pequeña y eficiente que implementa la mayoría de SQL92, pero en la que las claves ajenas no son verificadas. Para solucionar este inconveniente, se pueden emplear *triggers* en la base de datos, pero se ha preferido utilizar algunos eventos de WAINE para ilustrar cómo mediante su uso se pueden implementar las restricciones *1*, *2* y *3*. La última restricción (*4*) se forzará deshabilitando el botón de "eliminar" en el formulario de edición de tipos de sugerencias.

Los pasos necesarios para crear esta aplicación serían los siguientes:

1. Análisis. En este paso se identifican las funcionalidades y roles de la *interfaz de usuario*. También se realiza un esbozo de las *interfaces de usuario* principales.

   - En este sistema sólo aparecen dos roles: *secretario* y *supervisor*. En cada uno de ellos sólo existirá un usuario (*Joe* y *Sarah* respectivamente).
   - Las funcionalidades son acciones o *unidades de interacción*. En la aplicación de ejemplo sólo se identifican tres *unidades de interacción*: una para los tipos de sugerencias (*c_type*), otra para la clasificación de personas (*c_category*) y otra para la gestión de sugerencias (*c_suggestion*). Esta última *unidad de interacción* estará compuesta por varios formularios. En ella se podrá seleccionar una categoría y administrar a las personas de esta categoria y sus sugerencias.
   - Una vez conocemos las *interfaces de usuario* a construir se suele realizar un esbozo de las mismas. En la figura B.2 mostramos un boceto de las *unidades de interacción* de la aplicación.

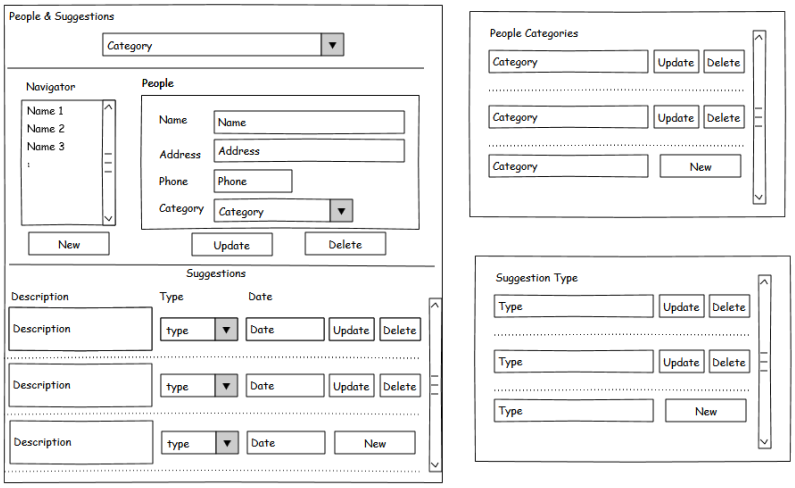
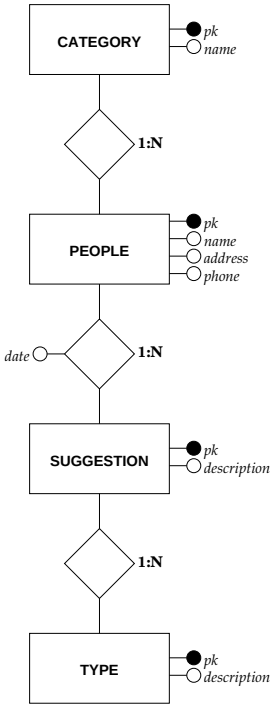


Figura B.2: Bocetos de las interfaces de usuario

2. Modelado de conceptos. Se define el *Modelo de Dominio* y la tabla rol/funcionalidad.

   Para definir el *Modelo de Dominio* se emplea un *Diagrama Entidad-Relación*. Este modelo recoge los conceptos que manejará la aplicación y además de guiar el desarrollo de la *interfaz de usuario* se emplea para generar de forma automática el código SQL correspondiente a la *Base de Datos* de la aplicación. En la figura B.3 podemos ver el *Diagrama Entidad-Relación* y el código SQL para correspondiente al sistema de sugerencias.




```sql
CREATE TABLE category (
   pk   INTEGER PRIMARY KEY,
   name VARCHAR(80) NOT NULL
);

INSERT INTO category VALUES(0,'Default category');

CREATE TABLE people (
   pk          INTEGER PRIMARY KEY,
   name        VARCHAR(80) NOT NULL,
   address     VARCHAR(80),
   phone       VARCHAR(13),
   fkcategory INTEGER NOT NULL REFERENCES category(pk)
);

CREATE TABLE suggestion (
   pk           INTEGER PRIMARY KEY,
   description  TEXT NOT NULL,
   creationdate DATE NOT NULL,
   fkpeople     INTEGER NOT NULL REFERENCES people(pk),
   fktype       INTEGER NOT NULL REFERENCES type(pk)
);

CREATE TABLE type (
   pk    INTEGER PRIMARY KEY,
   descr VARCHAR(80) NOT NULL
);
```

Figura B.3: Diagrama entidad-relación y código SQL de la aplicación

La tabla rol/funcionalidad recoge para cada rol del sistema las acciones que puede lanzar y los espacios de trabajo que puede manejar. En la tabla B.1 se muestra la mencionada tabla para este caso concreto. Además de las *unidades de interacción* mencionadas con anterioridad, se ha incluido *meta.container.appinfo* que es una *unidad de interacción* empleada para mostrar diversa meta-información de la aplicación (en este caso para el formulario "sobre este programa").

Funcionalidad	c_type	c_category	c_suggestion	meta.container.appinfo
supervisor	✓	✓		✓
secretary			✓	✓

Tabla B.1: Tabla rol/funcionalidad de la aplicación de ejemplo

3. Crear los *Diagrama Entidad-Relación* anotados. Cada espacio de trabajo de la *interfaz de usuario* es especificado sobre el fragmento de *Diagrama Entidad-Relación*

que manipula. En este caso será necesario describir tres *unidades de interacción*: c_type, c_category y c_suggestion. No es necesario definir la *unidad de interacción meta.container.appinfo* que será incluida desde la biblioteca de *Modelos de la IU* de WAINE.

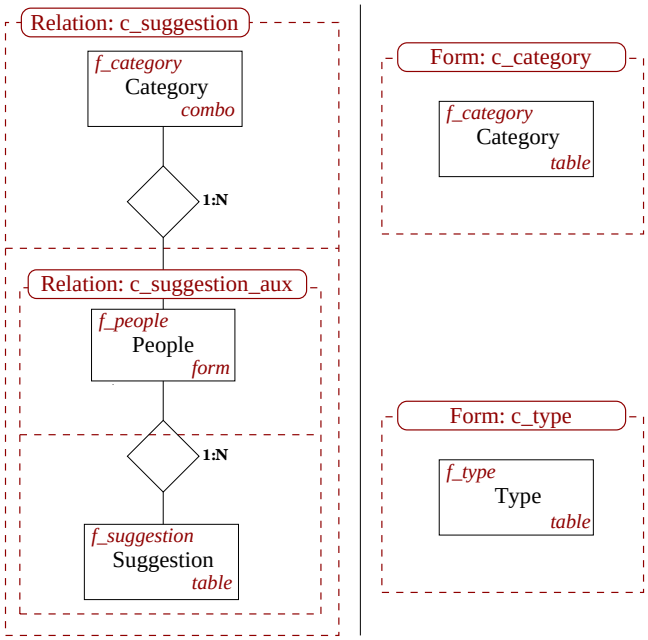


Figura B.4: Diagrama entidad-relación anotado

4. Especificación ASL. Partiendo de los *Diagrama Entidad-Relación* anotados y de la tabla rol/funcionalidad se genera la especificación de los modelos en lenguaje ASL. Un editor específico para documentos XML o un simple editor de textos puede ser usado para generar el archivo con la especificación. A continuación se muestra el código ASL para la aplicación de ejemplo:

```
<?xml version='1.0' ?>
<!DOCTYPE asl PUBLIC "-//ITI//DTD XWF 0.6 //EN"
    "/usr/local/lib/waine-2.2/lib/asl.dtd">
<asl>

    <xi:include href="/usr/local/lib/waine-2.2/include/meta.asl"/>

    <head>
        <meta class="AppInfo" name="appname" value="sample"/>
        <meta class="AppInfo" name="ver" value="0.1"/>
        <meta class="AppInfo" name="date" value="2006-09-21"/>
        <meta class="AppInfo" name="author" value="A.L.Delgado"/>
        <meta class="AppInfo" name="email" value="aldelgado@us.es"/>
    </head>

    <group gid="0" name="supervisor">
        <user uid="0" name="sarah" passwd="sarahkey"
              mainid="main_supervisor" descr="Sarah Smith"/>
```

```
 </group>

 <group gid="1" name="secretary">
     <user  uid="1" name="joe" passwd="joekey"
            mainid="main_secretary" descr="Joe Janus"/>
 </group>

 <main id="main_supervisor"
       caption="en=Supervisor menu|es=Menu del supervisor">
    <menu caption="en=Management|es=Gestion">
        <option caption="en=People Category|es=Categoria para
            personas" call="c_category"/>
        <option caption="en=Suggestion type|es=Tipo de
            sugerencias" call="c_type"/>
    </menu>
    <menu caption="en=Misc|es=Otros">
        <option caption="en=Waine page|es=Web de Waine"
                url="http://waine.us.es"/>
        <option caption="en=Author|es=Autor"
            call="meta.container.appinfo"/>
        <option caption="en=Logout|es=Salir" url="logout.php"/>
    </menu>
</main>

 <main id="main_secretary"
       caption="en=Default user menu|es=Menu del usuario por
           defecto">
    <menu caption="en=Management|es=Gestion">
        <option caption="en=Suggestions|es=Sugerencias"
            call="c_suggestion"/>
    </menu>
    <menu caption="en=Misc|es=Otros">
        <option caption="en=Waine page|es=Web de Waine"
                url="http://waine.us.es"/>
        <option caption="en=Author|es=Autor"
            call="meta.container.appinfo"/>
        <option caption="en=Logout|es=Salir" url="logout.php"/>
    </menu>
</main>


<form id="f_type" source="type" caption="Type">
   <fields>
      <key source="pk"/>
      <string source="descr" caption="Description" len="40"
          maxlen="80"/>
   </fields>
</form>

<container id="c_type" type="form">
   <param name="formid" value="f_type"/>
   <param name="form_type" value="table"/>
   <param name="button_insert" value="1"/>
<param name="button_update" value="1"/>
```

```
 </container>

 <form id="f_category" source="category" caption="Category">
   <orderby>name</orderby>
   <fields>
      <key source="pk" />
      <string source="name" caption="Name" len="40" maxlen="80" />
   </fields>

   <events>
        <beforedelete>
           <action type="php">
              <code>return event_BeforeDelete($pk);</code>

               <msg>You can not delete this category because
                    this is the defatult category or
                    some people belongs to it.
              </msg>
           </action>
        </beforedelete>
   </events>
</form>

<container id="c_category" type="form">
   <param name="formid" value="f_category" />
   <param name="form_type" value="table" />
   <param name="button_data" value="1" />
</container>

<form id="f_people" source="people" caption="People">
   <orderby>name</orderby>
   <fields>
      <key source="pk" />
      <string source="name" caption="Name" len="40" maxlen="80" />
      <string source="address" caption="Address" len="40"
                                        maxlen="80" canbenull="Y" />
      <string source="phone" caption="Phone" len="13"
          canbenull="Y" />
      <int source="fkcategory" caption="Category">
          <search>f_category</search>
          <searchfield>name</searchfield>
      </int>
   </fields>

   <events>
        <beforedelete>
           <action type="source">
               <code>DELETE FROM suggestion WHERE
                   fkpeople=%pk</code>
               <msg>All sugestions from this person deleted</msg>
           </action>
        </beforedelete>
   </events>
</form>
```

```
<form id="f_suggestion" source="suggestion" caption="Suggestion">
   <fields>
      <key source="pk"/>
      <text source="description" caption="Description">
          <width>40</width>
          <height>3</height>
      </text>
      <int source="fktype" caption="Type">
          <search>f_type</search>
          <searchfield>descr</searchfield>
      </int>
      <date source="creationdate" caption="Date" len="10"
          attr="R">
          <defvalue>=date("m-d-Y")</defvalue>
      </date>
      <int source="fkpeople" attr="H" msg="Select person first"/>
   </fields>
</form>


<container id="c_suggestion" type="relation">
   <param name="form_split" value="rows=80,*"/>

   <param name="formid" value="f_category"/>
   <param name="form_type" value="combo"/>

   <param ord="2" name="containerid" value="c_suggestion_aux"/>
</container>

      <container id="c_suggestion_aux" type="relation">
         <param name="form_split" value="rows=240,*"/>

         <param name="formid" value="f_people"/>
         <param name="form_type" value="form"/>
         <param name="button_data" value="1"/>
         <param name="source_filter_field" value="fkcategory"/>
         <param name="navigator_fields" value="name"/>
         <param name="navigator_position" value="W"/>
         <param name="navigator_width" value="20"/>

         <param ord="2" name="formid" value="f_suggestion"/>
         <param ord="2" name="form_type" value="table"/>
         <param ord="2" name="button_data" value="1"/>
         <param ord="2" name="source_filter_field"
             value="fkpeople"/>
      </container>

</asl>
```

Listado B.1: Código ASL de la aplicación

5. Generar y completar los repositorios. El primer paso será crear una nueva instancia de aplicación. Usando la utilidad mkapp se genera una nueva instancia de

aplicación enlazada a una determinada versión del motor. mkapp también genera un *Repositorio de Configuraciones* con valores por defecto para un funcionamiento básico de la aplicación y que posteriormente puede ser modificado para personalizar la instancia de aplicación. Una vez hecho esto podemos generar y completar los repositorios.

- Se genera la *Base de Datos* de la aplicación, utilizando para ello el código SQL obtenido a partir del *Diagrama Entidad-Relación*.
- Se crea el *Repositorio de la IU* a partir de la descripción en ASL. Para ello se emplea la utilidad asl2mdb. asl2mdb es un shellscript que utiliza xmlstarlet un toolkit XML que funciona en la línea de comandos [44]. asl2mdb permite generar el *Repositorio de la IU* sobre distintos tipos de soporte: varias bases de datos relacionales (sqlite, MariaDB, PostgreSQL, etc.), ficheros CSV, ficheros DBF, etc.
- Se completa el *Repositorio de Configuraciones*. La configuración consiste en la edición de diferentes archivos en este repositorio. Al menos dos archivos (db.cfg y mdb.cfg) deberán ser modificados para configurar los orígenes de datos empleados como *Base de Datos* y *Repositorio de la IU*. Por ejemplo, en el listado B.2, podemos ver el código necesario para configurar el acceso al *Repositorio de la IU*.

```
<?php

    $DBDRIVER="dssqlite.inc";

    $DBFILE="./MDB";

?>
```

Listado B.2: Configuración del origen de datos del repositorio de la interfaz de usuario

Otros archivos pueden ser editados para modificar aspectos de la presentación concreta, definir nuestros propios *widgets*, funciones, eventos, etc. En el listado B.3 podemos ver la codificación en PHP del evento *event_BeforeDelete* del formulario *f_category*.

```
function event_BeforeDelete($IN_pk)
{
 if ($IN_pk>0)
   {
   $conn=sqlite_open("./DB");
   $result=sqlite_query($conn,
     "SELECT count(pk) ".
         "FROM".
     "people WHERE ".
      fkcategory=$IN_pk"
     );
   if(sqlite_fetch_single($result)=="0")
     return 1;
   }
```

```
    return 0;
}
```

Listado B.3: Código del evento event_BeforeDelete

Para finalizar este anexo, en la figura B.5 podemos ver la *unidad de interacción* (*c_sugestion*) en su forma final en tiempo de ejecución.

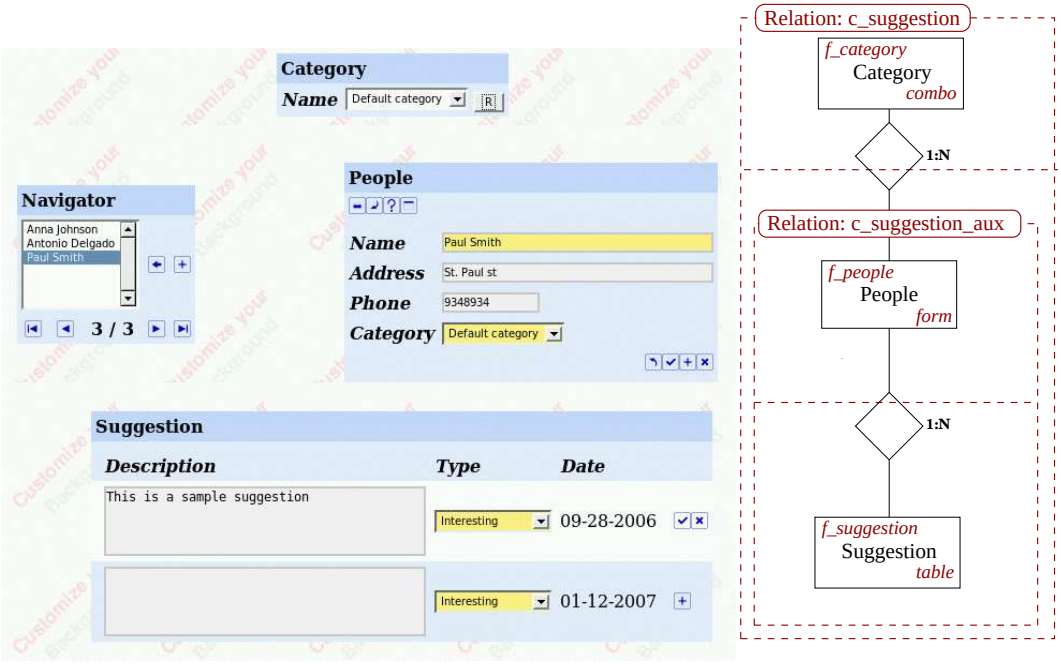


Figura B.5: Formulario para el secretario.

Apéndice C

Componente ASL

```
<!--
        aslmod: meta
        ver:    0.1
        author: Antonio Luis Delgado Gonzalez
        date:   2006-12-28
        objects:
                  meta.container.appinfo: generic 'about' dialog
                  meta.form.meta:         show data stored in _meta (RIU)
-->

<aslmod>
    <form id="meta.form.meta" source="_meta" caption="Info">
        <datasource>mdb.cfg</datasource>
        <fields>
            <key source="metaid"/>
            <string source="name"  caption="Name"  len="36" attr="R"/>
            <string source="value" caption="Value" len="36" attr="R"/>
        </fields>
    </form>

    <container id="meta.container.appinfo" type="form" >
        <param  name="formid" value="meta.form.meta"/>
        <param  name="form_type" value="table"/>
        <param  name="button_data" value="0"/>
        <param  name="button_misc" value="0"/>
        <param  name="source_filter_field" value="class"/>
        <param  name="source_filter_value" value="AppInfo"/>
    </container>
</aslmod>
```

Listado C.1: Código de un componente ASL

Apéndice D

Líneas de código por aplicación

La siguiente tabla detalla las líneas de código de cada modelo y lenguaje empleado en las aplicaciones del caso de estudio.

	DOM (SQL)	%	UM	%	DM	%	PM	%	RC	%	TOTAL
(A1) Reservas	2.019	24,2	33	0,4	328	3,9	5.335	64,0	618	7,4	8.333
(A2) Inventario	137	17,7	6	0,8	40	5,2	532	68,6	60	7,7	775
(A3) Tablón	208	19,8	15	1,4	64	6,1	689	65,7	73	7,0	1049
(A4) Prácticas	199	25,5	3	0,4	35	4,5	483	61,9	60	7,7	780
(A5) RelExteriores	128	30,0	3	0,7	17	4,0	219	51,3	60	14,1	427
(A6) InfoPanel	480	30,7	6	0,4	33	2,1	865	55,4	177	11,3	1.561
Media	529	24,7	11	0,7	86	4,3	1.354	61,2	175	9,2	2.154

Tabla D.1: Líneas de código por aplicación

Para la obtención de los datos de la tabla D.1, es necesario contar líneas de código de diversos documentos.

- *Modelo de Dominio*: En el *Modelo de Dominio* se han contabilizado líneas de código SQL para cada aplicación. Utilizando el *shell* y algunos comandos habituales de UNIX, se han eliminado líneas vacías y se han contabilizado las útiles con wc.

```
$ for i in *.sql; do  echo -n "$i "; egrep -v "^[[:space:]]*$" $i |
    wc -l; done
board.sql 208
inveq.sql 137
pracemp.sql 199
relext.sql 128
reservas.sql 2019
scrplan.sql 480
```

Listado D.1: Cuenta de las líneas de código de archivos SQL

- *Modelos soportados por ASL*: En ASL se definen el *Modelo de Usuario*, el *Modelo de Diálogo* y el *Modelo de Presentación*. Para contar las líneas de código de cada

modelo previamente hemos dividido cada archivo de especificación en archivos que contienen únicamente los elementos propios de cada modelo, se han eliminado de éstos las líneas vacías y se han contado las líneas útiles con el comando wc.

- *Modelos soportados por ASL*: En ASL se definen el *Modelo de Usuario*, el *Modelo de Diálogo* y el *Modelo de Presentación*. Para contar las líneas de código de cada modelo previamente hemos dividido cada archivo de especificación en archivos que contienen únicamente los elementos propios de cada modelo, se han eliminado de éstos las líneas vacías y se han contado las líneas útiles con el comando wc.

```
ls *.*M.asl | xargs wc -l

  689 board.APM.asl
   64 board.DM.asl
   15 board.UM.asl
  532 inveq.APM.asl
   40 inveq.DM.asl
    6 inveq.UM.asl
  483 pracemp.APM.asl
   35 pracemp.DM.asl
    3 pracemp.UM.asl
  219 relext.APM.asl
   17 relext.DM.asl
    3 relext.UM.asl
 5335 reservas.APM.asl
  328 reservas.DM.asl
   33 reservas.UM.asl
  865 scrplan.APM.asl
   33 scrplan.DM.asl
    6 scrplan.UM.asl
```

Listado D.2: Cuenta de las líneas de código de los documentos ASL

- *Repositorio de Configuraciones*: Este repositorio contiene archivos muy diversos (imágenes, archivos PHP, configuraciones, etc.) y además se encuentran distribuidos en diversos directorios dentro de la instancia de aplicación. Nos hemos ayudado de un sencillo *script* cpmstat (ver listado D.3) que realiza la cuenta de las líneas de código de los archivos adecuados.

```
if [ "$1" == "" ]
then
   echo Necesito un nombre de fichero para la salida
else
   for i in `find etc/style -type f`; do  echo -n "$i "; egrep -v
      "^[[:space:]]*$" $i | wc -l; done > $1
   echo ---------------------------------- >> $1
   for i in `find etc/template -type f`; do  echo -n "$i "; egrep -v
      "^[[:space:]]*$" $i | wc -l; done >> $1
   echo ---------------------------------- >> $1
   for i in `find themes -type f`; do  echo -n "$i "; egrep -v
      "^[[:space:]]*$" $i | wc -l; done >> $1
   echo ---------------------------------- >> $1
```

```
    for i in 'find _CONF -type f'; do  echo -n "$i "; egrep -v
        "^[[:space:]]*$" $i | wc -l; done >> $1
    echo ------------------------------ >> $1
    for i in 'find images -type f'; do  echo -n "$i "; egrep -v
        "^[[:space:]]*$" $i | wc -l; done >> $1
fi
```

Listado D.3: Script cpmstat

Apéndice E

Obtención de los datos de la tabla 5.2

Para la obtención de los datos de la tabla 5.2, es necesario contar los objetos definidos de cada modelo y cuántos de ellos han sido reutilizados. A continuación se comenta cuáles han sido los métodos para obtener los datos en cada modelo:

- *Modelo de Dominio*: Son objetos del modelo tablas y vistas. Contar los objetos definidos en este modelo para cada aplicación es trivial. Se ha comentado en la sección 5.1 que el *Modelo de Dominio* se transforma de forma automática para residir en una *Base de Datos*. En nuestro caso la *Base de Datos* empleada es PostgreSQL. Se pueden contar por lo tanto las tablas y vistas definidas para cada base de datos empleando los comandos dt y dv en el intérprete psql. También se pueden obtener esos datos empleando la siguiente consulta:

```
SELECT table_name FROM information_schema.tables WHERE
    table_schema='reservas'
```

Listado E.1: Consulta para obtener tablas y vistas del repositorio de la interfaz de usuario

  Por otra parte, consideramos reutilizadas aquellas tablas y vistas que han sido empleadas en varios proyectos (es decir, han sido compartidas), o aquellas que aún dentro del mismo proyecto han sido referenciadas desde varios formularios (es decir, aparecen en varias *unidades de interacción* de la misma aplicación). Para esto último se consulta al *Repositorio de la IU* (ver modelo conceptual en figura 4.5) de cada aplicación listando los formularios y su origen de datos E.2. En el resultado buscamos sources repetidos:

```
SELECT formid, source FROM _form ORDER BY source;
```

Listado E.2: Consulta para obtener formularios y su origen de datos (tablas o vistas)

- *Modelo de Usuario*: Contar el número de grupos de una aplicación es sencillo. Se puede realizar una sencilla consulta al *Repositorio de la IU* de cada aplicación para obtener el número de grupos de usuarios:

```
SELECT count(*) FROM _group;
```

Listado E.3: Consulta para contabilizar grupos de una aplicación

- *Modelo de Diálogo*: De manera análoga podemos proceder para obtener el número total de menús de la aplicación:

```
SELECT count(*) FROM _main;
```

Listado E.4: Consulta para contabilizar menus de una aplicación

Dado que cada usuario pertenece a un grupo, cada grupo representa un rol frente a la aplicación y el conjunto de funcionalidades a las que un usuario puede acceder está determinado por su menú principal 4.1, no tiene sentido que los menús puedan ser compartidos entre usuarios de distintos grupos, lo que limita totalmente la reutilización de este recurso en el tipo de aplicación soportada por WAINE.

- *Modelo de Presentación*: Son elementos reutilizables del modelo forms y containers. Para contar estos elementos y el número de veces que ha sido reutilizados se ha realizado la siguiente consulta sobre el *Repositorio de la IU* de cada aplicación.

```
SELECT value, sum(ccc) FROM (

          -- formularios y contenedores que aparecen
  -- referenciados como parametros

          SELECT value, count(value) as ccc
               FROM (
                    SELECT value FROM _parameter
                         WHERE (name='formid' OR
                              name='containerid'))
               GROUP BY value

          UNION -- contenedores referenciados desde
                -- el menu de una aplicacion

            SELECT containerid as value, count(containerid) as ccc
                FROM (
                    SELECT containerid FROM _option WHERE
                        containerid IS NOT NULL)
            GROUP BY containerid

          UNION -- formularios referenciados desde
        -- campos de formularios

            SELECT search as value, count(search) as ccc
              FROM (SELECT search  FROM _field WHERE search like
                  'frm_%')
            GROUP BY value

          UNION -- Aquellos formularios que no aparecen
        -- referenciados desde parametros o campos
```

```
            SELECT formid as value, 1 as ccc FROM _form
               WHERE formid NOT IN (SELECT search  FROM _field
                  WHERE search like 'frm_%')
                 and formid NOT IN (SELECT value FROM _parameter
                    WHERE name='formid')
            GROUP BY value

         UNION -- Aquellos contenedores que no aparecen
       -- referenciados desde parametros o desde menus

            SELECT containerid as value, 1 as ccc FROM _container
               WHERE containerid NOT IN (SELECT containerid FROM
                  _option WHERE containerid IS NOT NULL)
                 and containerid NOT IN (SELECT value FROM
                    _parameter WHERE name='containerid')
            GROUP BY value

)
   GROUP BY value ORDER BY value;
```

Listado E.5: Consulta para contar referencias a contenedores y formularios

Se puede apreciar en la consulta que se contabilizan contenedores lanzados desde opciones de un menú (que identifican *unidades de interacción*), formularios y contenedores parametrizados, formularios asociados a campos (empleados en *comobox*, listas de selección, etc.) y también el resto de formularios y contenedores no reutilizados. Obsérvese que la consulta anterior no contabiliza las que denominamos "referencias jerárquicas del *Modelo de Presentación*". Cada *unidad de interacción* del este modelo puede ser vista como un árbol en el que las hojas son formularios y los nodos intermedios son contenedores(ver 4.1.1). De este modo, si el contenedor c_1 referencia a c_2 y a su vez c_2 referencia a c_3, entonces c_1 referencia a c_3, es decir, cuando c_1 es referenciado son referenciados c_2 y c_3.

$$c_1 \mathcal{R} c_2 \wedge c_2 \mathcal{R} c_3 \Rightarrow c_1 \mathcal{R} c_3 \tag{E.1}$$

Dado que PostgreSQL no admite consultas para relaciones jerárquicas, ha sido necesario elaborar un pequeño *script* PHP para contabilizar estas relaciones:

```
<?php

function APM_deep($IN_db,$IN_str)
{
    $q = @$IN_db->query("select * from _parameter where
        containerid='$IN_str' and name='containerid' ORDER BY
        value");
        while ($row = $q->fetch()){
            printf('; '.$row['value']);
            APM_deep($IN_db,$row['value']);
        }
    $q = @$IN_db->query("select * from _parameter where
        containerid='$IN_str' and name='formid' ORDER BY value");
```

```
        while ($row = $q->fetch()){
            printf('; '.$row['value']);
        }
}

// Principal

if ($db = new SQLiteDatabase($argv[1]))
{
    $q = @$db->query('SELECT * FROM _container ORDER BY
        containerid');
        while ($row = $q->fetch()){
            printf($row['containerid']);
            APM_deep($db,$row['containerid']);
            printf("\n");
        }
}

?>
```

Listado E.6: Script para contabilizar referencias jerárquicas

En el *script* presentado en el listado E.6, podemos ver que se realiza una búsqueda en profundidad para cada contenedor definido en la base de datos que soporta el *Repositorio de la IU* cuyo nombre es pasado como parámetro al programa.

- *Repositorio de Configuraciones*: Dado que los objetos que conforman el *Repositorio de Configuraciones* son naturaleza muy variada (estilos, *widgets*, imágenes, etc.),se ha considerado más útil contabilizar como objetos reutilizables ficheros. Por lo tanto se ha tenido en cuenta para cada aplicación cuántos ficheros han sido definidos y cuántos han sido reutilizados.

Apéndice F

Actuaciones de mantenimiento realizadas

En este anexo se enumeran las actuaciones de mantenimento que se han llevado a cabo sobre las aplicaciones desarrolladas en nuestro caso de estudio durante los últimos cuatro años. La tabla F.1 presenta la fecha de cada actuación, la aplicación afectada, el tipo de actuación efectuada, el ámbito y el identificador del elemento modificado y una breve descripción de la actuación efectuada.

Fecha	Apl.	Actuacion	Ámbito	Id. elemento	Reutiliza	Descripcion
19/09/11	A6	Correctiva				Orden en el formulario
20/02/12	A1	Aumentativa		st_cons_prof_CALEND		Calendarios
20/02/12	A1	Aumentativa		frm_cons_prof_CALEND		Calendarios
20/02/12	A1	Aumentativa		CR		calendar.php
24/10/12	A1	Correctiva	Interna	frm_reserva_COEXA		Se añaden los turnos ":No asignado;M:Mañana;D:Mediodia;T:Tarde;N:Noche;"
24/10/12	A3	Correctiva	Interna	frm_examconv		Se añaden los turnos ":No asignado;M:Mañana;D:Mediodia;T:Tarde;N:Noche;"
20/08/13	A1	Aumentativa	Externa	BD		Se añade a la tabla reserva el campo hfexam
20/08/13	A3	Aumentativa	Interna	frm_examconv		Se añade campo hfexam
20/08/13	A3	Correctiva	Interna	frm_examconv		Los campos hexam y hfexam se ponen como obligatorios
17/11/13	A1	Aumentativa	Interna	st_cons_VRD_AREA		Reservas por planta
23/11/13	A1	Aumentativa		st_cons_VRD_AREA_S	2	Reservas por planta
23/11/13	A1	Aumentativa		st_cons_VRD_AREA_P	2	Reservas por planta
23/11/13	A1	Aumentativa		st_cons_VRD_AREA_C	2	Reservas por planta
23/11/13	A1	Aumentativa		option	1	st_cons_VRD_AREA para todos los usuarios de conserjeria
23/11/13	A1	Aumentativa		option	1	st_cons_VRD_AREA para todos los usuarios de conserjeria
23/11/13	A1	Aumentativa		option	1	st_cons_VRD_AREA para todos los usuarios de conserjeria
10/12/13	A1	Aumentativa	Interna	frm_reserva_COEXA		Se añade el campo hfexam con las mismas propiedades que hexam
10/12/13	A1	Correctiva	Interna	frm_reserva_COEXA		Se modifica el caption de los campos hexam y hfexam
09/05/14	A1	Aumentativa		st_examenes_SCR		Imágenes para pantallas
09/05/14	A1	Aumentativa		frm_filtro_fecha		Imágenes para pantallas
09/05/14	A1	Aumentativa		st_examenes_SCR_print	1	Imágenes para pantallas
09/05/14	A1	Aumentativa	Interna	frm_examenes_SCR		Imágenes para pantallas

Tabla F.1: Relación de actuaciones de mantenimiento realizadas