UNIVERSIDAD DE CASTILLA-LA MANCHA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

INTEGRACIÓN DE CALIDAD Y EXPERIENCIA EN EL DESARROLLO DE INTERFACES DE USUARIO DIRIGIDO POR MODELOS

TESIS DOCTORAL

Charles Gilchrist © 2005

Francisco Montero Simarro

Julio, 2005

UNIVERSIDAD DE CASTILLA-LA MANCHA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

Integración de Calidad y Experiencia en el Desarrollo de Interfaces de Usuario Dirigido por Modelos

Tesis doctoral elaborada para optar
al título de Doctor en Informática

Presentada por:
D. Francisco Montero Simarro

Dirigida por:
Dr. D. Pascual González López
Dra. Dña. María Dolores Lozano Pérez

Albacete, 2005

UNIVERSIDAD DE CASTILLA-LA MANCHA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

TESIS DOCTORAL

INTEGRACIÓN DE CALIDAD Y EXPERIENCIA EN EL DESARROLLO DE INTERFACES DE USUARIO DIRIGIDO POR MODELOS

Presentada por:
D. Francisco Montero Simarro

Dirigida por:
Dr. D. Pascual González López
Dra. Dña. María Dolores Lozano Pérez

ALBACETE, 2005

para Yo,
con quien quiero seguir compartiendo mi vida

por y para mis padres y mi hermano,
a quien les debo todo lo que soy

Índice de contenidos

Índice de figuras

Índice de tablas

Agradecimientos

¡Venturoso aquel a quien el cielo dio un pedazo de pan,
sin que le quede obligación de agradecérselo a otro que al mismo cielo!
Miguel de Cervantes Saavedra (Escritor español)

Dice un refrán español que *de bien nacidos es ser agradecido*, y no puedo dejar pasar la oportunidad que ofrece esta sección para agradecer públicamente y dejar constancia de la gratitud que me merecen todos aquellos que, de una u otra manera, han hecho posible la elaboración de esta tesis doctoral y el trabajo de investigación necesario para su logro. Sin ellos, sin duda alguna, no hubiera sido posible.

Mis directores, Pascual y María, me han facilitado, en la medida que sus obligaciones les han permitido, todo aquello que he podido ir necesitando, y si bien determinadas decisiones que he tenido que tomar han supuesto una responsabilidad para ellos, su confianza, esfuerzo y dedicación también la han supuesto para mi en igual o mayor medida.

Todos los compañeros del grupo de investigación al que pertenezco, grupo LoUISE, merecen igualmente reseña y gratitud. De entre ellos, aquellos con los que comparto y he compartido tiempo, estancia, charlas y, en definitiva, buenos y malos momentos, es decir, Víctor, José Pascual y José Eduardo han sido un ejemplo de amistad y espero haberles correspondido y seguir haciéndolo en el futuro de igual manera.

Otros compañeros también han estado ahí en todo momento y así se lo reconozco. Muchas gracias a Gregorio, Julia y Elena.

La elaboración y dedicación que supone realizar un trabajo de investigación, como el que ha resultado necesario para la consecución de esta tesis doctoral, supone que no siempre se disponga de tiempo para estar con mucha gente que te aprecia y que te lo demuestra jugando el papel les dejas en cada momento. A ellos también va dedicado el esfuerzo que he realizado. Me he acordado en muchas ocasiones de vosotros, más de las que pudiera parecer, y he sentido profundamente, en otras, responder *no* a muchas de vuestras propuestas, espero que ahora pueda ser yo quien las haga. Gracias a Carlos, María Dolores, Felipe, Anastasio, Inma y María Salud.

Siete meses de estancia en otra universidad, otra ciudad y otro país, concretamente en la Universidad católica de Louvain en Bélgica, dan la posibilidad de conocer a otras personas y de hacer amigos. El profesor Jean Vanderdonckt se ha comportado como tal pese a su estatus y reconocido prestigio, sin él este trabajo tampoco hubiera sido posible.

Resumen

Dos disciplinas, Ingeniería del Software e Interacción Persona-Ordenador, son las directamente implicadas en el desarrollo de productos software de calidad. Entre ambas se ha identificado un *gap* semántico debido a que trabajan utilizando diferentes conceptos, metodologías, lenguajes y herramientas. Esta tesis doctoral quiere contribuir a disminuir dicho gap semántico, facilitando el diseño de aplicaciones software de calidad desde el punto de vista del usuario final que las use. Para ello utiliza como principales recursos la consideración de la experiencia y la caracterización de la calidad, e incluye ambos recursos en todas las fases de desarrollo de un producto software, especialmente en las iniciales.

La evolución histórica que ha acontecido en la Ingeniería del Software queda descrita por una tendencia cuya dirección última persigue la *industrialización*, en este sentido, *Model Driven Architecture* (MDA) de OMG es uno de los referentes de dicha tendencia. Con ella las labores de implementación pretenden ser definitivamente sustituidas por las de diseño y, a su vez, el concepto de *modelo* se refuerza y cobra su verdadera dimensión: la generación frente a la mera documentación. La calidad en Ingeniería del Software está especialmente centrada en la provisión de facilidades de mantenimiento, y los principales medios para lograrla pasan por potenciar la cohesión y disminuir el acoplamiento.

Paralelamente, la Interacción Persona-Ordenador (IPO) ofrece como principal aportación las técnicas englobadas dentro del término Diseño Centrado en el Usuario (DCU). Además, desde IPO se presta especial atención a las interfaces de usuario y a la calidad. Esta última está centrada en la usabilidad, o calidad de uso que las interfaces pueden ofrecen. Las aportaciones más significativas de la comunidad IPO, en lo que a desarrollo de interfaces de usuario se refiere, pasan por los desarrollos basados en modelos desde comienzos de la década de los noventa. Pero, la carencia de un modelo de calidad centrado en la usabilidad y la disparidad de criterios en cuanto a su consideración y tratamiento sistemático constituyen un reto de investigación importante dentro de la comunidad IPO.

Partiendo de la identificación del contexto en el que nos encontramos, descrito con anterioridad, y teniendo presente el principal objetivo de esta tesis doctoral se han debido acometer los siguientes subobjetivos previos:

- Caracterizar la usabilidad de un producto software. Para lograr este objetivo se han utilizado los estándares internacionales disponibles y aquellos factores y criterios que hemos considerado suficientemente

representativos en función de la calidad que se pretende modelar (usabilidad). Como resultado de ello esta tesis doctoral presenta un modelo de calidad centrado en la usabilidad, basado en estándares, desarrollado utilizando criterios de calidad ergonómicos y asociado a experiencia y a métricas.

- Caracterizar la experiencia. En esta tesis doctoral se apuesta por utilizar el concepto de patrón. Ya que un elemento al que prestamos especial atención son las interfaces de usuario, los patrones de interacción se convirten en el principal referente con el que considerar la calidad en la interacción. La principal limitación de dichos patrones pasaba porque, tradicionalmente, ofrecen un único objetivo, la documentación. En función de las tendencias identificadas esta tesis doctoral ofrece una dimensión adicional y generativa a la experiencia en interacción documentada utilizando patrones. Los patrones de interacción pueden estar asociados a un modelo que se describe utilizando aquellas notaciones más extendidas independientemente de su procedencia (UML, CTT y UIDLs).
- Integración de calidad y experiencia en un único marco de trabajo. Una vez caracterizadas la calidad y la experiencia se establecen relaciones entre las mismas y se presenta una propuesta integradora de teoría y práctica para el desarrollo de interfaces de usuario de calidad centrada en la usabilidad. Dicha propuesta está dirigida por modelos y utiliza el lenguaje usiXML para almacenar, reutilizar y aprender dichos modelos (experiencia).
- Esta tesis doctoral tiene asociada una herramienta, IDEALXML, un entorno de desarrollo de interfaces de usuario basado en modelos y en usiXML. La herramienta ofrece dos facilidades por un lado es un editor/gestor de patrones de cualquier tipo (interacción, colaboración, diseño, etc.) que almacena utilizando el formato PLML y, por otro, es un entorno de desarrollo que facilita al ingeniero experiencia basada en patrones para la especificación de productos software en los que la usabilidad está considerada.

El marco de trabajo presentado y su integración con otras propuestas desarrolladas dentro de proyectos de colaboración europea (red SIMILAR), en los que estamos implicados junto con diferentes grupos de investigación, hace que los trabajos desarrollados en esta tesis doctoral puedan ser integrados con otras propuestas de investigación y aseguran la continuidad futura de labores de I+D+i.

Las principales conclusiones que se derivan de la actividad de investigación desarrollada durante estos años y con la que se ha propiciado la elaboración de esta tesis doctoral pasan por constatar que el principal reto en estos momentos y en un futuro próximo pasará por disminuir el gap semántico existente entre la elicitación de requisitos y las actividades de diseño. Este gap coincide con el identificable entre Ingeniería de Software e Interacción Persona-Ordenador y es el que redunda directamente en la calidad de la que hacen gala los productos software desarrollados. La disponibilidad de experiencia determina la calidad de los productos software diseñados y desarrollados, y mucha de dicha experiencia es modelable y, por ello, puede ser aprendida, diseminada, compartida, reutilizada y enriquecida. A su vez, mucha de la calidad que ofrece cualquier producto software pasa por la necesaria elicitación de requisitos en las fases iniciales del proceso de desarrollo, de lo contrario su posterior logro se incrementa en tiempo y dinero, cuando no resulta inviable.

Albacete, 19 de julio de 2005
Francisco Montero

Abstract

Two disciplines, Software Engineering and Human Computer Interaction, are directly involved in the development software quality products. Between them a semantic gap has been identified because the work by using different concepts, methodologies, languages and tools. This thesis dissertation wants to contribute in reducing that semantic gap, making easier the design of quality software applications from the final user point of view that uses them. To do so, it uses as principal resources the consideration of experience and the characterization of quality, and it includes both resources at every development stage of a software product, especially at the first ones.

The historic evolution of Software Engineering is described by a trend towards industrialization. In this sense, Model Driven Architecture (MDA) from OMG is one of the most relevant examples within that trend. In this approach implementation tasks are supposed to be replaced with design tasks, where models reinforce themselves, not just a simple documentation, but as a generative tool. Quality in Software Engineering is especially interesting in mantainment facilities, and the main techniques to achieve them aim at fostering the cohesion and reducing coupling.

At the same time, Human Computer Interaction (HCI) offers as its main contribution the set of techniques included within the term User Centred Design (UCD). Furthermore, HCI is especially focused in user interfaces and quality. Quality is centred in usability, or quality of use, that user interfaces can offer. Since 90's decade, the contributions more relevant in HCI community, in user interfaces development field, is model-based design. The lack of a quality model centred on usability, and the disparity in the criteria in the way they are systematically treated and considered introduces a great challenge to HCI community.

Starting from identifying the context where we are involved, previously described, and taking into account the main goal of this dissertation: reduce the semantic gap between Software Engineering and Human Computer Interaction, the following subgoals have been pursued:

- Characterize the usability of a software product. To achieve this goal the available international standards, and those criteria we found to be enough representative according to the quality we want to model have been used. As a result, this dissertation introduces a quality model centred in usability, based in standards, developed using ergonomic quality criteria and related to experience and metrics.

- Characterize experience. This work proposes using pattern concept. Because we are focused in user interfaces, interaction patterns became the main reference to consider quality in interaction. The main flaw of interaction patterns is that traditionally they are aimed just at documentation. According to the identified trends, this thesis offers an additional dimension generative to experience in interaction documented by using patterns. Interaction patterns can be related to a model described by using the notations most widely used, regardless of its origin (UML, CTT and UIDLs).
- Quality and Experience Integration within a unique framework. Once quality and experience have been characterized, relationships between them are established and a proposal integrating theory and practice for quality user interface development centred in usability is presented. This is a model-driven environment and uses the UsiXML language to store, reuse and learn the models (experience).
- This PhD thesis implements also a software tool called IDEALXML, a model-based user interface development environment using usiXML. The CASE tool offers two facilities: On the one hand, it is an editor/management of patterns of any type (interaction, collaboration, design, etc.) which stores using the PLML format; and on the other hand it is a development environment that assists software engineers with pattern-based experience in which usability is considered.
- The framework proposed and its integration with other proposals developed within projects of European collaboration (SIMILAR network), in which we are involved together with different European research groups, makes that the outcomes of this PhD thesis can be integrated with other research proposals and assure the future continuity of I+D+i tasks.

The main conclusions derived from the research activity developed during these last years and which has concluded with this PhD thesis are that the main challenge, nowadays and in the near future, is to fulfil the existing semantic gap between requirements elicitation and design activities. This gap coincides with the one existing between Software Engineering and Human-Computer Interaction and it is the one that directly redound in the quality of the software products developed. Experience availability determines the quality of software products, and most of such experience can be modelled and thus, can be learned, disseminated, shared, reused and enriched. At the same time, most of the quality offered by any software product depends on the correct requirements elicitation at the very begin-

ning of the development process, in the earlier phases. On the contrary, its subsequent achievement results much more difficult, or even unviable.

Albacete, 19 de Julio de 2005
Francisco Montero

Capítulo 1 Introducción

Un comienzo no desaparece nunca, ni siquiera con un final.
Harry Mulish (Escritor holandés)

1.1 Motivación

El desarrollo de aplicaciones software ha sido, tradicionalmente, una labor artesanal y como todo proceso de ese tipo dependiente enormemente de la experiencia. Igual que el alfarero (Fig. 1-1) conoce las necesidades del destinatario de sus productos al situar asas o adecuarse a tamaños, formas o colores, del mismo modo, el ingeniero del software desarrolla su labor proporcionando productos de mayor o menor calidad, o más o menos próximos a los deseos del usuario final, dependiendo también de la experiencia que posea en tales desarrollos.

Figura 1-1. Comparaciones odiosas para el desarrollo del software

Los factores a considerar, cuando desarrollamos una aplicación software, son muchos y de muy diversa índole. Así, encontramos factores relacionados con la calidad, tanto en su dimensión interna como externa. Los primeros dependen directamente de las habilidades del ingeniero del software y para su logro se han facilitado diferentes propuestas metodológicas, lenguajes y herramientas. Los segundos, los factores externos, no dependen solamente de la pericia o capacidad que presente el equipo de desarrollo, en ellos el usuario final tiene algo que decir, mucho realmente, si se tienen presentes características cualitativas, subjetivas y de satisfacción o de adecuación entre niveles cognitivos que presenten el usuario por un lado, y la herramienta desarrollada por otro.

Por ello que existe una clara dependencia entre la calidad del desarrollo de aplicaciones software y las habilidades que muestra el equipo que se ocupa de dicho desarrollo. En este sentido, no existen procedimientos sistemáticos para el logro de aplicaciones software y, menos aún, donde la experiencia sea reutilizada de una vez para la siguiente. Esto conlleva que surjan dificultades en el proceso de desarrollo, que serán mayores dependiendo del grado de experiencia de los miembros del equipo de desarrollo y, aunque no se llegue al extremo, la aparición de carencias que han ido asociadas con las diferentes crisis que la Ingeniería del Software ha sufrido en diferentes momentos de su historia, y que padece de forma crónica.

Las facilidades para el uso, para el mantenimiento, para la *trazabilidad* del proceso metodológico y para la posibilidad de integración de los productos desarrollados son los grandes retos que tiene presentes la Ingeniería del Software. Además, debido al constante avance tecnológico, aparecen nuevos desafíos al tener que considerar nuevas formas de interactuar, nuevos dispositivos y existir la exigencia de facilitar a cada vez más colectivos de la población acceso a mayores volúmenes de información y servicios gracias a la Sociedad de la Información en la que vivimos.

1.2 El problema

En cualquier caso, la mera disponibilidad de nuevos dispositivos no supone una dificultad en sí mismo, ya que dicha aparición ha sido una constante a lo largo de la historia. El verdadero problema está en **cómo se aborda el tratamiento que se da a esos avances y a la diversidad que suponen.** Este tratamiento, hasta el momento, ha sido particular y concreto. Ahora, la acumulación de trabajo, que recae sobre diseñadores y desarrolladores, se debe más a una **falta de procedimientos sistemáticos que aseguren el éxito** en el tratamiento de cada uno de ellos, que al hecho de que existan gran cantidad de técnicas y dispositivos debidos a los avances tecnológicos disponibles. **El ingeniero debe saber qué lográr y cómo lograrlo y para ello en esta tesis doctoral se consideran la calidad y la experiencia**.

1.3 Una solución integradora

Con el trabajo de investigación que da lugar a **esta tesis doctoral** se **integran cuatro elementos que influyen decisivamente en la consecución de productos software** en los que la calidad es un factor determinante.

Esos cuatro pilares, que determinarán también la propia estructura de este documento, son (véase Fig. 1-2):

- una **metodología** de desarrollo que contemple factores de calidad,
- un **lenguaje** de especificación donde esos mismos factores tengan cabida,
- una **herramienta** que de soporte a metodología y lenguaje utilizado,
- y, finalmente, la consideración de la **experiencia** asociada al proceso metodológico elegido y donde se garantice la posibilidad de acumular conocimiento resultante de un proyecto de desarrollo para el siguiente.

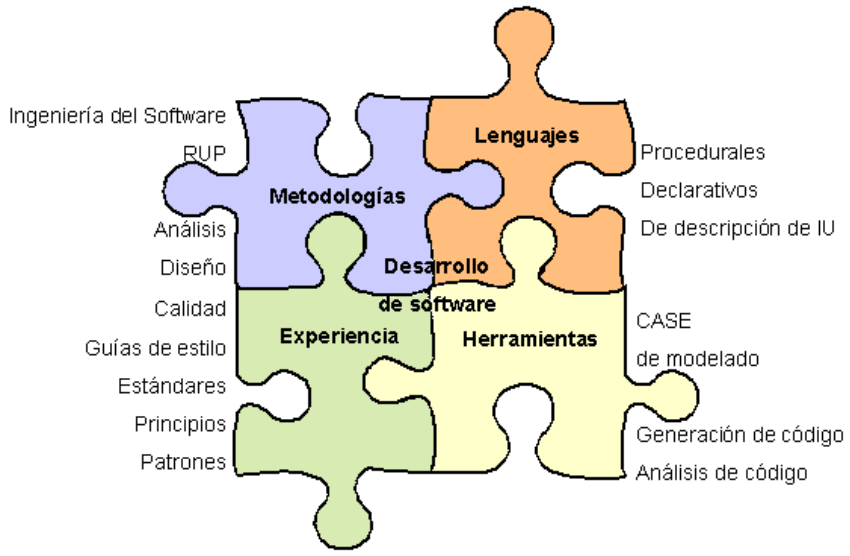


Figura 1-2. Distintas consideraciones al desarrollar software

El principal reto que *a priori* estuvo presente al integrar los elementos antes reseñados pasó por la inclusión en el proceso metodológico elegido de factores de calidad. La inclusión debía hacerse de forma coherente con la experiencia disponible en cualquiera de sus formas y/o estándares, y esa experiencia debía ser trazable en su identificación y utilización. Para ello la experiencia disponible debía presentarse de tal forma que permitiera su manipulación desde el proceso metodológico elegido. Es decir, elegida una metodología e identificadas carencias de logro de calidad en la misma, lo que se persigue es potenciar dicha metodología con la inclusión de conocimiento en cualquiera de sus formas, adecuando, previamente, esas formas para que sean manipulables y útiles. Es por ello que, en primer término debió identificarse qué se entendía por calidad de un producto software,

caracterizando la misma con la búsqueda, y proposición en su caso, de un modelo de calidad asociado al proceso y/o producto.

Dicho proceso (*top-down*) de caracterización de la calidad puede enlazar con conocimiento y experiencia en análisis y diseño a muy diversos niveles de abstracción y estar asociado a diferentes temáticas, como se irá presentado en los diferentes capítulos asociados a este documento. Este conocimiento debe presentarse de forma adecuada para su gestión y utilización y, por lo general, será combinación de las diferentes posibilidades utilizadas a la hora de documentar la experiencia, entiéndase aquí: guías de estilo, principios, reglas, estándares y, sobre todo, patrones como una forma de documentación de experiencia reutilizable.

Estos últimos, los patrones, constituyen en nuestra propuesta el eslabón de enlace necesario entre calidad y desarrollo, y documentan, para ello, la experiencia necesaria. El patrón recoge cómo pasar del *qué* al *cómo*, y presentan dos utilidades significativas. Por un lado, son una *lingua franca* (Erickson, 2001) que puede ser utilizada por todos aquellos que componen el equipo de desarrollo de un producto software, incluyendo al usuario, y, por otro, son una forma de documentación algo más precisa que la asociada a las guías de estilo (Welie, 2001). Los patrones originariamente esbozados en el ámbito de la disciplina arquitectónica (Alexander, 1977, 1979), se han utilizado también en Informática asociados a Ingeniería del Software y a las técnicas Orientadas a Objetos con la aparición de un libro, *Design Patterns,* de *la banda de los cuatro* (Gamma et al., 1995), conjuntamente, en los años que llevamos de este siglo, se están utilizando en Interacción Persona-Ordenador (IPO). En este trabajo de investigación se han identificado deficiencias que impiden la utilización de todo lo que puede aportar el concepto de patrón en el proceso de desarrollo y de reutilización de la propia experiencia.

La utilización de patrones, aunque muy extendida en muchas disciplinas, presenta diferentes cuestiones abiertas relacionadas con los formatos empleados, con la documentación de los mismos, con su organización o con sus facilidades de presentación y utilización soportada por herramientas software. En esta tesis doctoral hacemos importantes contribuciones en este sentido tomando como base de trabajo los consensos alcanzados en diferentes *workshops* internacionales asociados con el *topic* patrón, en este sentido proponemos modificaciones en propuestas como *Pattern Language Makup Language* (PLML) (Fincher et al., 2003), una forma de documentar patrones que será debidamente presentada en el capítulo correspondiente.

El conocimiento o experiencia recopilado en forma de patrones con el que enlazamos calidad y proceso metodológico de desarrollo de software es *generativo*, es decir, en esta tesis doctoral no consideramos al patrón como mera documentación, sino como *experiencia activa* con una parte de

documentación y *algo* adicional que permitirá su uso, previa adecuación al contexto concreto en el que se pretenda utilizar cada patrón. Una parte fundamental la constituyen las relaciones entre patrones que permiten su utilización conjunta y lograr soluciones más elaboradas que las que pueden obtenerse mediante la utilización de las tradicionales guías de estilo.

El proceso metodológico seguido es un proceso *top-down* dirigido por tareas, iterativo, apoyado en patrones y centrado en la calidad. Los patrones aportan experiencia, y lo hacen a través de la documentación de soluciones de diseño que llevan asociadas mejora en la calidad. La calidad y el proceso metodológico de desarrollo tienen como nexo de unión los patrones, como antes se reseñó y, junto con ello, la calidad se descompone utilizando factores, criterios y subcriterios de la misma, que se constituyen en un modelo de calidad.

1.3.1 Los objetivos

Los principales objetivos de la investigación asociada a esta tesis doctoral pasan por :

- Identificar las propuestas metodológicas disponibles para el desarrollo de aplicaciones software que consideran la calidad y la experiencia.
- Establecer, si los hay, nexos de unión entre las propuestas ofrecidas desde la Ingeniería del Software y desde Interacción Persona-Ordenador que resultan de utilidad para el desarrollo de dicho software de calidad.
- Identificar y establecer la dimensión o gap semántico existente entre Ingeniería del Software e Interacción Persona-Ordenador.
- Definir un marco conceptual de desarrollo de aplicaciones donde la calidad esté considerada.
- Determinar qué se entiende por calidad de un producto software y cómo lograrla en función de la disponibilidad de experiencia.
- Identificar cómo se documenta y se utiliza la experiencia en todas y cada una de las fases de desarrollo de aplicaciones software.
- Identificar limitaciones en la forma de documentar de la experiencia, con una intención: la utilización de la misma de forma productiva.
- Utilizar conjuntamente calidad y experiencia en único marco conceptual de desarrollo de productos software.
- Determinar el momento más adecuado de consideración de la calidad.
- Aunar experiencia y calidad e integrarlas en el proceso de desarrollo de productos software.

1.3.2 Los resultados

Presentadas los objetivos, los resultados de esta investigación pasan por:

- Haber definido un proceso metodológico de desarrollo que considera tanto la experiencia como la calidad desde el principio del propio proceso.
- Haber definido un modelo de calidad centrado en la usabilidad, donde se contemplan estándares internacionales y criterios de calidad. Dicho modelo de calidad enlaza, utilizando la experiencia y las métricas disponibles, con marco metodológico anteriormente definido.
- Haber identificado y utilizado un lenguaje de especificación con el que documentar la experiencia y que permite su utilización y reutilización en todas aquellas situaciones donde se identifique la tupla <*problema, contexto, solución*>.
- Haber dotado a la experiencia disponible, especialmente a la relacionada con el terreno de la interacción de una componente generativa compatible con la actual tendencia de desarrollo de software y donde se contemplan características de calidad.
- Haber proporcionado una herramienta CASE que permite abordar los puntos anteriores con éxito, ya que somos de la opinión de que no existen metodologías no factibles sino no respaldadas suficientemente por herramientas.
- Haber colaborado con expertos en los temas tratados a través de la realización de estancias y el establecimiento de contactos con investigadores representativos en los ámbitos tratados en esta tesis doctoral.
- Haber integrado los resultados de esta tesis doctoral dentro de un marco de trabajo que asegura la realización de los trabajos futuros y de investigación que se proponen, así como la diseminación adecuada de los resultados obtenidos con la realización de esta tesis doctoral.
- Haber difundido los resultados obtenidos fruto de la labor de investigación desarrollada en aquellos eventos relacionados que mejor se adaptaban a los temas tratados en esta tesis doctoral.

Considerando estos logros, algunos de los problemas más reseñables encontrados en el camino han tenido que ver con:

- La materialización de las asociaciones, necesarias para la definición del marco conceptual, entre metodología de desarrollo y calidad, a través de la elaboración de un modelo de calidad considerado en la metodología elegida.

- El estudio y la realización de propuestas que abundaran en la potencia que pueden ofrecer los patrones como elemento de documentación activo, paliando los problemas identificados en las tradicionales guías de estilo y que están relacionados con su adecuada puesta en práctica (edición, visualización, selección, aplicación y adaptación en su caso).
- Estudio de viabilidad del concepto de patrón como bisagra entre calidad y metodología, encontrando los necesarios ganchos para dicha bisagra en los criterios de calidad.
- Estudio de diferentes lenguajes de especificación asociados a la interacción y selección de aquel más adecuado en función de las necesidades y pretensiones de esta tesis doctoral.
- Desarrollo de una aplicación CASE donde se considerasen metodologías, lenguajes y asociaciones entre los diferentes modelos de especificación de una aplicación software a diferentes niveles de abstracción. Con ello se debía dar cabida a diferentes modos de interacción y distintos dispositivos con los que el usuario pudiera interactuar.
- La herramienta anterior u otra complementaria debieran dar la posibilidad de gestionar la experiencia (considerando la calidad y su definición) y facilitar al ingeniero un entorno de desarrollo basado en la experiencia y que le permita considerar también la calidad o viceversa.

1.4 Organización del documento

Una descripción somera de este documento de tesis doctoral es la siguiente:

En el **primer capítulo**, el presente, se justifica la realización de la tesis doctoral identificando la motivación que ésta lleva consigo. Ésta no es otra que la necesidad de incorporar en un único marco metodológico la calidad de la aplicación y la experiencia para su logro. Dicho marco debe considerar ambos ingredientes mencionados desde la primeras fases de análisis de requisitos y de diseño de cualquier producto software. Para dar solución a este objetivo se utilizan metodologías, lenguajes y herramientas fundamentadas en la disponibilidad de modelos de calidad y en la reutilización de experiencia cristalizada en forma de patrones.

El **capítulo segundo** ofrece una panorámica sobre tres de los cuatro pilares identificados, y que dictan, a nuestro juicio, el desarrollo de aplicaaplicaciones software de calidad. Nos referimos a las *metodologías* que apuestan por contemplar la calidad en las mismas, a los *lenguajes* de espe-

cificación de interfaces de usuario propuestos y a las *herramientas* disponibles para unas y otros respectivamente. Queda fuera de este capítulo, ya que por su importancia y extensión hemos preferido su tratamiento en capítulo adicional incorporando en ellos la visión propia y el tratamiento que desde esta tesis doctoral se ofrece, el pilar dedicado a la experiencia. También existirá un capítulo específico destinado a la consideración de la calidad, será el tercero.

El **capítulo tercero** contempla la calidad, su definición, la investigación asociada a los modelos de calidad disponibles y a su elaboración y a nuestra propuesta fundamentada en la utilización de criterios de calidad de uso. Estos últimos enlazarían con métricas y con experiencia, y ésta, a su vez, con el proceso de desarrollo.

El **capítulo cuarto** está dedicado a la experiencia: su edición, documentación, formato y organización. Las premisas de las que partiamos han sido las de conseguir *experiencia activa*, es decir, reutilizable. Los patrones no son documentación aislada sino que forman un lenguaje que debe ofrecer diferentes jergas en función de niveles de abstracción y temática. Igual que se utilizan unas palabras u otras cuando hablamos de temas diferentes o que una misma palabra pueda tener diferentes connotaciones en función del contexto en el que la utilicemos, los patrones se utilizan como palabras de un lenguaje más complejo y también ofrecen esa misma característica y su documentación deberá ser distinta, o podrá tener diferentes implicaciones, en función de cuándo y cómo se utilice.

En el **capítulo quinto** se presenta la propuesta integradora asociada a la investigación que conlleva esta tesis doctoral. En este capítulo todos los conceptos presentados en los capítulos anteriores, y las propuestas presentadas de forma aislada relacionadas con ellos, encajan en una única propuesta que presenta implicaciones a nivel metodológico, de selección y utilización de un lenguaje de especificación y de desarrollo de una herramienta que dé soporte al entorno y al lenguaje. En dicho entorno es posible la reutilización de la experiencia almacenada utilizando patrones.

Dicho marco de trabajo sería comentado en mayor detalle, aunque de forma general, en el **capítulo sexto**. En él se presentará la hoja de ruta de la propuesta metodológica asociada a la propuesta. Su presentación se realizará utilizando la herramienta desarrollada, IDEALXML.

El marco de trabajo descrito en profundidad en el capítulo anterior sería sometido a traza con el uso de un caso de estudio concreto en el **capítulo**

séptimo. En él a través de un caso de estudio relacionado con la compra electrónica se presentarían las consideraciones concretas y el alcance de la propuesta realizada en los capítulos anteriores.

El último capítulo, el **capítulo octavo**, está dedicado a recoger las reflexiones, conclusiones, resultados concretos, así como los trabajos futuros relacionados con esta tesis doctoral.

1.5 Consideraciones previas

Antes de comenzar con el desarrollo de este documento debe dejarse patente que el interés de la investigación, que ha conducido a esta tesis doctoral, se centra en potenciar la calidad del software que se produce o especifica. Ésta calidad se entiende como un factor fundamentalmente externo y dependiente del usuario final, pero con ramificaciones e implicaciones que tienen que ver con aspectos internos de especificación del propio producto software. Y es que el software, tanto en su vertiente de producto como de proceso, conlleva trabajar con una serie de descripciones donde debe contemplarse la calidad.

Al hablar de calidad se pueden considerar diferentes dimensiones (ISO 9126, 1991; 2001), que conllevarían aspectos relacionados con la *fiabilidad*, es decir, con la facilidad con la que el producto software funciona y lo hace, además, en todo momento en que se requiera su uso; la *funcionalidad*, garantizándose que el software hace aquello que se espera de él, la *usabilidad*, es decir, el software hace lo que se espera y además permite que lo hagamos de forma natural; la *eficiencia*, facilidad por la que el sistema hace una utilización de recursos adecuada y ajustada a las necesidades; la *mantenibilidad*, característica interna por la que un sistema puede ser analizado, probado o cambiado; y la *portabilidad*, capacidad de un sistema para ser reemplazado, adaptarse o instalarse.

Aún considerando la dimensión externa de la calidad, fruto de un punto de vista determinado por la interacción mantenida entre persona y ordenador, en este documento revisamos metodologías, proponemos métodos, utilizamos lenguajes y facilitamos herramientas que más que centrarse en el producto también consideran el proceso para su logro.

La lectura del presente documento y las aportaciones de esta tesis doctoral irán asociadas con las marcas incluidas en la estructura general de un entorno de desarrollo dirigido por modelos (MD-UIDE, permitase la licencia de utilizar una denominación muy parecida a la empleada para las tradicionales propuestas basadas en modelos, MB-UIDE) que se muestra en

la Fig. 1-3, adaptada de figuras similares disponibles en diferentes documentos (Szekely, 1996) o (Schlungbaum, 1996).

En función de la distribución de contenidos, comentada con anterioridad, se realizará un estudio del estado del arte en las propuestas metodológicas asociadas disponibles (**1**), identificándose ventajas y limitaciones. Se identificarán, también, los diferentes lenguajes de especificación de interfaces de usuario disponibles (**2**), mostrándose las posibles conexiones que pueden establecerse entre modelos y lenguajes. Se revisarán, de igual forma, las herramientas software disponibles (**3**) que dan soporte a las propuestas metodológicas, lenguajes de especificación y todo tipo de técnicas para el logro de calidad en el desarrollo de aplicaciones software. Una calidad que se entiende desde el lado del usuario y que está centrada en el incremento de la usabilidad o calidad en uso.

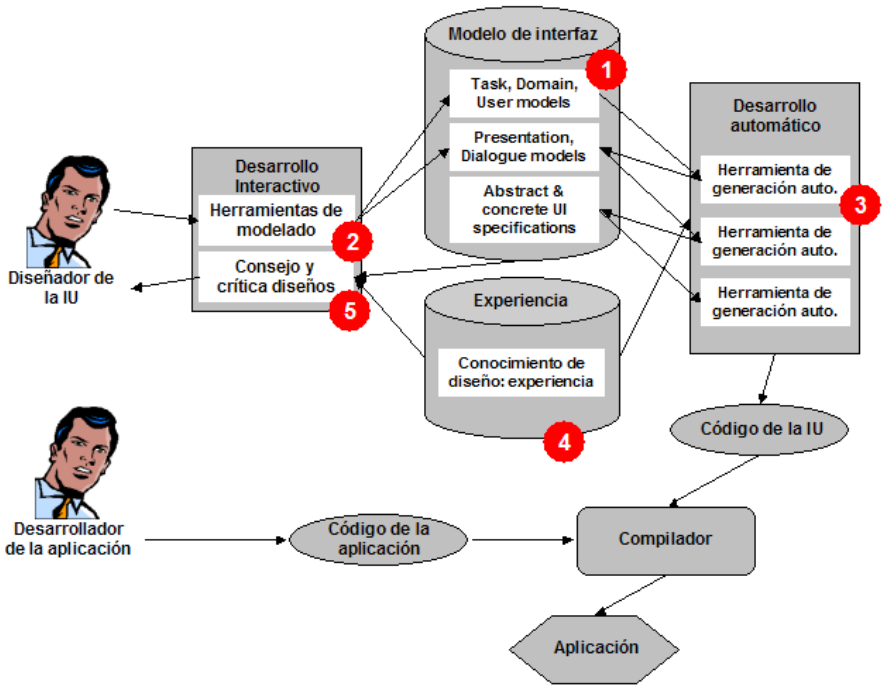


Figura 1-3. Estructura de un entorno de desarrollo de software basado en modelos (Schlungbaum, 1996)

El principal objetivo de esta tesis doctoral pasa por la incorporación de experiencia y de calidad (**4**) en el análisis y en el diseño. Una experiencia documentada en forma de patrones y utilizada mediante la definición de un lenguaje de patrones. Esta experiencia está orientada al logro de la calidad antes mencionada y vendría a complementar, que no a eliminar, la utiliza-

ción de otra experiencia disponible utilizando otros medios, por ejemplo las guías de estilo (**5**).

Antes de comenzar con el desarrollo de cada uno de los capítulos sugeridos es conveniente la definición, aunque sea somera, de una serie de conceptos, que por su relación con el tema que se está tratando aparecerán con asiduidad a lo largo de toda esta tesis doctoral, de hecho algunos términos ya han aparecido en este primer capítulo. Muchos de esos términos serán tratados con mayor detalle en los capítulos específicos donde su aparición sea estelar, sirva por tanto lo siguiente como una mera presentación propia de un capítulo con pretensiones meramente introductorias y descriptivas.

En este documento aparecerá siempre una doble visión y se buscará identificar las relaciones entre cada una de las vistas ofrecidas. Esa doble visión la proporcionan dos disciplinas: Ingeniería del Software (IS) por un lado, e Interacción Persona-Ordenador (IPO) por otro. Ambas disciplinas tienen por objetivo el proporcionar métodos, técnicas, lenguajes y herramientas que permitan obtener aplicaciones software de calidad. Quizá la principal diferencia y lo que establece un hueco, en ningún caso insalvable, entre ambas disciplinas es cómo se considera al usuario final y qué características son realmente relevantes del software obtenido por aplicación de los métodos, lenguajes y herramientas que consideran cada una de ellas.

Si bien ambas disciplinas tienen presente la *calidad*, la primera de ellas, la Ingeniería del Software, se centra especialmente en aspectos internos de calidad, mientras que la segunda busca potenciar aspectos fundamentalmente externos. Esta tesis doctoral busca el logro de factores externos de calidad, más propios de IPO, y propone lograrlos mediante su consideración e incorporación en el proceso de desarrollo de aplicaciones software.

La *interfaz de usuario* es el vínculo entre el usuario y el programa que se ejecuta en la computadora. Una interfaz es un conjunto de comandos, menús o de cualquier otro mecanismo, a través del cual el usuario se comunica con el programa. La elaboración de una interfaz de usuario, bien diseñada, exige una gran dedicación, pues generalmente las interfaces son grandes, complejas y difíciles de implementar, depurar y modificar. Hoy en día las interfaces de manipulación son prácticamente universales. Las interfaces que utilizan ventanas, íconos y menús se han convertido en un estándar, al mismo tiempo han surgido otras formas de interactuar, nuevos dispositivos y cantidades ingentes de información disponibles para cualquier colectivo de usuarios.

Desde hace más de una década, y aunque prácticamente en exclusiva en el ámbito académico y no en el industrial, la propuesta metodológica más extendida para el desarrollo de interfaces de usuario está basada en la utilización de *modelos*. Estos son descripciones del sistema en diferentes direcciones y niveles de abstracción recogiendo características, identificando

tareas, perfiles de usuario y datos que manipular. Después, por compilación de dichas descripciones, se obtiene una aplicación software (Fig. 1-3).

Existen diferentes propuestas metodológicas que pretenden integrar *calidad* y *producción* software, las más representativas son las propuestas de (Nielsen, 1993), (Mayhew, 1999) y (Constantine, 1999), aunque a ellas han seguido otras. En ellas no se incluye expresamente la utilización de modelos ni se apuesta por una metodología concreta de desarrollo que se ponga en práctica paralelamente con ellas, pero si que proponen y dan forma al término *Ingeniería de la Usabilidad*, considerando en él una serie de técnicas cuya utilización contribuye al conocimiento del usuario y del contexto en el que éste desarrolla sus tareas. Todas las propuesta metodológicas se apoyan en la puesta en práctica de un *Diseño Centrado en el Usuario* (DCU) que se considera la clave del éxito para el logro de interfaces de usuario de calidad.

Será la *usabilidad* o *calidad en uso* la característica realmente aludida en este documento cuando se haga uso del término calidad. Con ellos denotaremos a una serie de factores, criterios y subcriterios cuyo logro redundará en la utilización por parte de los usuarios finales de los productos software desarrollados. A ese conjunto de factores, criterios y demás consideraciones con las que puede caracterizarse la calidad es a lo que se conoce como *modelo de calidad* y en el capítulo tres, como se ha comentado, se propondrá un modelo de calidad centrado en la usabilidad donde, además del proceso *top-down* de descomposición típico, se caracterizan los criterios de menor nivel con métricas como proponen los estándares internacionales y con experiencia con la que apostamos por la especificación del *qué* y del *cómo lograr* objetivos de calidad, que consideramos más útiles, en muchos casos, que la propia ponderación *del cuánto* hemos mejorado en según qué criterios.

Cuando nos referimos a *experiencia* lo hacemos desde el punto de vista del desarrollo de interfaces de usuario, y la forma de documentarla por la que se apuesta en este trabajo de investigación es mediante la utilización del concepto de *patrón*. Un concepto cuyo origen no está en el campo de la Informática sino en el de la Arquitectura, pero en una y otra disciplina puede utilizarse de igual forma. Concretamente, en Interacción Persona-Ordenador se usan para buscar el logro de la calidad a través de la identificación y documentación de problemas, del contexto en el que surgen dichos problemas y de las soluciones de análisis y diseño disponibles para esos problemas.

Todos estos conceptos previos relacionados con: metodologías, lenguajes de especificación, herramientas, experiencia y calidad son los que utilizaremos para el logro de interfaces de usuario de calidad. Gráficamente

pueden verse las relaciones entre estos conceptos en la *rosa de los vientos* que se muestra en la Fig. 1-4.

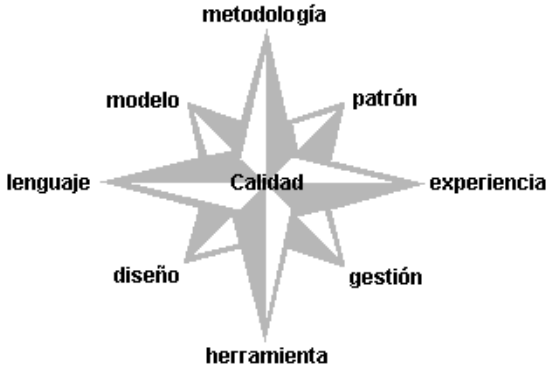


Figura 1-4. Direcciones relevantes en esta tesis doctoral

En esta rosa de los vientos las direcciones asociadas con los cuatro puntos cardinales constituyen los medios para logra fines y las referencias intermedias conforman los fines de esta tesis doctoral. Por ello, el estudio, caracterización y elaboración de *modelos*, *patrones*, *herramientas de diseño* y *herramientas de gestión de conocimiento* son los resultados del trabajo que se presenta en este documento.

1.6 Metodología de trabajo

La metodología de trabajo asociada con el trabajo de investigación que conduce a esta tesis doctoral se ha desarrollado durante una serie de años en los que se han hecho propuestas y participado en diferentes congresos, conferencias y talleres de diferente temática asociada con las palabras clave que describen esta tesis doctoral. Así, se han presentado mejoras metodológicas, modelos de calidad y patrones en el terreno de la interacción y el desarrollo de aplicaciones, aquellas publicaciones más representativas y asociadas con los contenidos presentados en cada capítulo de este documento se recogen en apartados específicos que, para tal propósito, se han dispuesto en cada capítulo justo antes de las referencias bibliográficas utilizadas en la elaboración del propio capítulo.

El lenguaje utilizado para dar soporte al proceso metodológico y a los patrones utilizados fue incorporado en una estancia de siete meses realizada en Bélgica en la Universidad católica de Louvain-La Neuve. En dicha estancia se trabajó bajo la supervisión del profesor Jean Vanderdonckt. El lenguaje elegido es el *USer Interface eXtensible Markup Language* (usiXML, 2004) del que se presentarán, a lo largo de este documento,

aquellos aspectos que consideramos más relevantes desde el punto de vista de nuestra investigación, información adicional sobre este lenguaje puede encontrarse en el Apéndice G.

Otra de los resultados de esta tesis doctoral es la herramienta IDEALXML. La misma ha sido presentada en diferentes congresos y talleres, y está disponible en distintos sitios web relacionados con el ámbito de los patrones de interacción. A la difusión y aceptación de dicha herramienta ha contribuido la confianza que en ella han depositado los creadores del lenguaje usiXML, que la han aceptado dentro de su marco de trabajo en el que está integrada.

Referencias bibliográficas

Alexander, C., Ishikawa, S., Silverstein, M., Jacobsen, M., Fiksdahl-King, I., Angel, S. *A Pattern Language.* New York: Oxford University Press, 1977

Alexander, C. *The Timeless Way of Building.* New York: Oxford University Press, 1979

Borchers, J. hcipatterns.org. http://www.hcipatterns.org/tiki-index.php (act. 2005)

Constantine, L. L., and Lockwood, L. A. D. *Software for Use: A Practical Guide to the Essential Models and Methods of Usage-Centered Design.* Reading, MA: Addison-Wesley, 1999

Erickson, T. The interaction design patterns page. http://www.visi.com/~snowfall. (act. 2005)

Fincher, S.. The Pattern Gallery. http://www.cs.ukc.ac.uk/people/staff/saf/patterns. (act. 2005)

Fincher, S. Workshop Report, Interfaces No. 56, Journal of the BCS HCI Group. 2003

Gabriel, R. P. *Patterns of Software: Tales from the Software Community*. New York: Oxford University Press. 1996

Gamma, E., Helm, R., Johnson, R., Vlissides, J.. *Design Patterns: elements of reusable object-oriented software*, Addison-Wesley, Reading, MA, USA. 1995

ISO 9126. *Software product evaluation. Quality characteristics and guidelines for their use*, ISO, 1991

ISO/IEC9126-1:2001. Software engineering product quality part 1: Quality model. International Standard ISO/IEC 9126-1, International Standard Organization, 2001

Mayhew, D. J. *Usability Engineering Lifecycle, The: A Practitioner's Handbook for User Interface Design*. Morgan Kaufmann, Published 1999

Nielsen, J. and Phillips, V. L. *Estimating the relative usability of two interfaces: heuristic, formal, and empirical methods*, (ed.), Conference on Human Factors and Computing, Amsterdam, The Netherlands, 1993: 214-221.

Schlungbaum, E. *Individual User Interfaces and Model-Based User Interface Software Tools*. Intelligent User Interfaces. 1997: 229-232

Szekely, P. *Retrospective and Challenges for Model-Based Interface Development*. DSV-IS. 1996: 1-27

usiXML. *USer Interface eXtensible Markup Language*. http://www.usixml.org.

van Welie, M., Trtteberg, H. *Interaction patterns in user interfaces*. 7th. Pattern Languages of Programs Conference. 2000

Capítulo 2 Desarrollo de Interfaces de Usuario

El progreso y el desarrollo son imposibles si uno sigue haciendo las cosas tal como siempre las ha hecho.
Wayne W. Dyer (Escritor estadounidense)

La verdadera sabiduría está en reconocer la propia ignorancia.
Sócrates (Filósofo griego)

2.1 Introducción

Ingeniería del software (IS) e Interacción Persona-Ordenador (IPO) son dos disciplinas que buscan aportar métodos y herramientas para potenciar la calidad que ofrecen los productos software. Los puntos de vista que se ofrecen en ellas son diferentes, aunque ambas disciplinas se solapan en parte de sus ámbitos de alcance.

La IS aporta procesos, métodos, notaciones y herramientas que afectan tanto al proceso como al producto, contempla factores internos y externos de un producto software. Mientras, la IPO ofrece otros métodos, en algunos casos complementarios, e intereses que se centran en el proceso, en el producto y en la identificación de dónde, cómo y por quién es utilizado el producto software. La IS está preocupada por ofrecer una visión arquitectónica del sistema, para ello ha propuesto, entre otras facilidades, un Proceso Unificado (UP) y un Lenguaje Unificado de Modelado™ (UML[1]). IPO busca la complicidad del usuario y la determinación de su contexto y para ello propone un Diseño Centrado en el Usuario (DCU).

Será objetivo de este capítulo presentar las aportaciones ofrecidas por una y otra disciplina, aunque sea someramente, y establecer relaciones entre ellas. Junto con dichas aportaciones se hará referencia a lenguajes y herramientas disponibles en la actualidad, que permiten abordar la tarea de desarrollar software y, con ello, el desarrollo de interfaces de usuario.

Como quedó reflejado en el primer capítulo de este documento, nuestro interés es la realización de aportaciones desde el punto de vista de la calidad de un producto software. Esa calidad la contemplamos con dos componentes, una componente externa determinada por la interfaz de usuario que ofrece el producto y una componente interna asociada al proceso de concepción y desarrollo del producto final.

[1] Unified Modeling Language™. http://www.uml.org

A la hora de atacar el problema de desarrollar software apostamos por el uso de metodologías basadas en modelos, porque es la apuesta más extendida y porque es uno de los posos más importantes disponibles en el grupo de investigación al que pertenecemos: el Laboratorio de Interfaces de Usuario e Ingeniería del Software (LoUISE), donde disponemos de una propuesta metodológica, IDEAS (Lozano, 2001), para el desarrollo de interfaces de usuario basada en modelos e influenciada por el proceso RUP® y el lenguaje UML.

Aunque uno de los intereses de esta tesis doctoral es el epígrafe de este capítulo, no podemos dejar de contemplar el desarrollo de interfaces de usuario desde la propia evolución de la IS, de la que la IPO se nutre de metodologías, lenguajes y herramientas. Es por ello que hemos dedicado las primeras secciones de este capítulo a la IS y a sus aportaciones. Al mismo tiempo se presenta un análisis de si dichas aportaciones son suficientes para abordar el desarrollo de interfaces de usuario. El propio documento que contempla el lector está repleto de comparaciones y asociaciones entre las aportaciones de una y otra disciplina con el objetivo de poner de manifiesto sus relaciones. Ésta no es en ningún caso una tarea gratuita, ya que como podrá comprobarse conforme se sucedan los diferentes apartados y entremos en secciones más enfocadas al ámbito IPO, se constatará un trasiego de metodologías y de herramientas propias de IS, a las que se adhieren consideraciones relacionadas con la interacción entre persona y ordenador.

En el presente capítulo, una vez se haya planteado la evolución y las aportaciones que se realizan desde la IS, y habiéndose identificado el alcance de las mismas, se verá cómo desde la IPO se han propuesto otros puntos de vista y se han enriquecido las propuestas elaboradas desde la IS con consideraciones que tienen que ver con el usuario, con la calidad y con las facilidades de interacción entre el producto software y el usuario que finalmente lo utiliza. La principal pretensión de este capítulo es comenzar sentando las bases del desarrollo de software y del papel que en el mismo juega la parte destinada a cubrir la interfaz de usuario, las implicaciones que conlleva su desarrollo y cómo estas se están abordando.

2.2 La dualidad desde el principio

El software desempeña un doble papel. Es un producto y, al mismo tiempo, el vehículo para hacer entrega de dicho producto. Como producto, hace facilita la potencia informática del hardware: produciendo, gestionando, adquiriendo, modificando, mostrando o transmitiendo información. Como

vehículo se utiliza para hacer entrega del producto, el software actúa como base de control del ordenador, la comunicación de información, y la creación y control de otros programas.

En esta tesis doctoral nos interesan ambas vertientes, en tanto en cuanto, la interfaz afecta tanto a una como a otra en el sentido de que el producto debe ser útil para los objetivos del usuario, debiendo proporcionar los mecanismos necesarios para hacer uso de esa utilidad y haciéndolo de forma satisfactoria. Tanto la utilidad como la satisfacción no se logran por accidente y en ello tiene mucho que ver el proceso de desarrollo del propio software.

En el desarrollo de software se han identificado diferentes problemas:

- El avance del software, en su afán de aprovechar el potencial que ofrece el hardware, deja a un lado la habilidad para construir software.
- La demanda de nuevos programas y las exigencias que precisa el mercado no pueden ser cubiertas en la medida en las que éste las solicita.
- La sociedad se ha hecho dependiente de las facilidades ofrecidas por los ordenadores para el tratamiento, acceso y manipulación de la información. Esta dependencia exige, paralelamente, operaciones fiables y seguridad a la hora de trabajar y manipular un ordenador y de las tareas que con ellos pueden llevarse a cabo.
- Se lucha por construir software informático que tenga fiabilidad y alta calidad.
- El mantenimiento de los programas existentes se ve amenazado por diseños pobres, por falta de experiencia y recursos inadecuados.

Fruto del carácter crónico que presentan muchos de los problemas reseñados para el desarrollo de software se acuñó el término *crisis del software*. Pero, los problemas no se limitan a que el software no funcione correctamente, sino que el mal se extiende a los problemas relacionados con cómo desarrollar software, cómo realizar el mantenimiento de un volumen cada vez mayor del software existente, cómo poder atender la demanda creciente del software y cómo asegurar que el software desarrollado sea útil, aceptado y usable. Todas estas cuestiones serán tratadas en esta tesis doctoral.

La IS se centra fundamentalmente en controlar costes, y productividad relacionados con el desarrollo de software y de proporcionar procesos para asegurar de forma sistemática el logro de los mismos. La IPO se centra en la usabilidad y en la necesaria aceptación de un producto en función de las características que como vehículo ofrece y de cómo las ofrece.

Antes de entrar más en materia nos permitimos reseñar una definición previa de *Ingeniería*. Esta definición es la ofrecida por Hardy Cross que entiende Ingeniería como *el arte de tomar una serie de decisiones importantes dado un conjunto de datos inexactos e incompletos con el fin de obtener, para un cierto problema, aquella entre las posibles soluciones, la que funcione de la manera más satisfactoria* (Cross, 1952). Y es, sin duda, la mejor definición posible para describir la actividad seguida en la línea de investigación que ha coducido esta tesis doctoral y con los logros con ella alcanzados.

2.3 Desarrollo de software en IS

Por IS se entiende el establecimiento y uso de principios robustos de la Ingeniería con el fin de obtener económicamente software que sea fiable y que funcione eficientemente sobre máquinas reales (Bauer, 1997).

Otra definición de IS nos la ofrece la IEEE que establece que la misma es la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software.

La IS es una tecnología multicapa (Pressman, 1992). Los cimientos están orientados hacia la calidad. La gestión de la calidad fomenta una cultura continua de mejora del proceso, y es esta filosofía la que conduce últimamente el desarrollo de enfoques cada vez más robustos para la IS.

El fundamento de la IS es la capa de proceso. El *proceso* es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de la IS. Las áreas clave del proceso forman la base del control de gestión de proyectos del software y establecen el contexto en el que se aplican los métodos técnicos, se producen resultados del trabajo, se establecen hitos, se asegura la calidad y se gestiona el cambio de manera adecuada.

Los *métodos* indican cómo construir de manera técnica el software, y abarcan una gama de tareas que incluyen análisis de requisitos, diseño, construcción prueba y mantenimiento. Los métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

Por último, las *herramientas* proporcionan un soporte automático o semiautomático para el proceso y para los métodos. Todos los elementos comentados son importantes e influyen en el desarrollo del software y a todos ellos daremos cumplido tratamiento en este capítulo.

2.3.1 La evolución actual en el desarrollo de software

Estudiar o hacer un repaso por la evolución de la IS pasa por ir haciéndose y respondiendo, de vez en cuando en su evolución, a la pregunta ¿*cuáles son las partes de un programa*?.

Las partes de un programa son *descripciones*, y el desarrollo de software se centra en producir, analizar y ejecutar estas descripciones, que, por otra parte, no necesariamente tienen que estar asociadas directamente con código, como es la tendencia actual.

Posiblemente, uno de los errores más frecuentes en el dominio del desarrollo del software es abordar directamente la solución sin haber dedicado previamente un esfuerzo suficiente para decidir cuál es el problema. El resultado es que la mayoría de las soluciones planteadas para dar valor a los actores de un sistema en discusión, permanecen enterradas en el interior de complejas herramientas de software y los problemas siguen donde siempre.

La evolución en el desarrollo del software ha estado marcada por tres factores clave:

i. la **separación de conceptos**,

ii. el **aumento en el nivel de abstracción** a la hora de abordar el problema de desarrollar software,

iii. y la **necesidad de unificar y reutilizar la experiencia**.

Los tres elementos han sido objetivos, y lo siguen siendo, a la hora de presentarse propuestas para dar soporte al proceso de desarrollo del software.

Seguidamente, se comentará dónde está y de qué dispone la IS en la actualidad para abordar su labor, de entre los distntos recursos se analizará por su relación con el presente trabajo UML, el Proceso Unificado, y las metodologías ágiles y dirigidas por modelos. Posteriormente, se analiza si estos elementos son suficientes para permitir desarrollar interfaces de usuario de forma integrada o paralela con el propio desarrollo del software.

2.3.2 UML: el Lenguaje Unificado de Modelado

Desde los inicios de la Informática se han estado utilizando distintas formas de representar los diseños de manera más bien personal o con algún modelo gráfico. La falta de estandarización en la manera de representar

gráficamente un modelo impedía que los diseños gráficos realizados pudieran compartirse fácilmente entre distintos diseñadores lo que supone un verdadero obstáculo. Se necesitaba un lenguaje no sólo para comunicar ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema. Con este objetivo nació el Lenguaje Unificado de Modelado (UML).

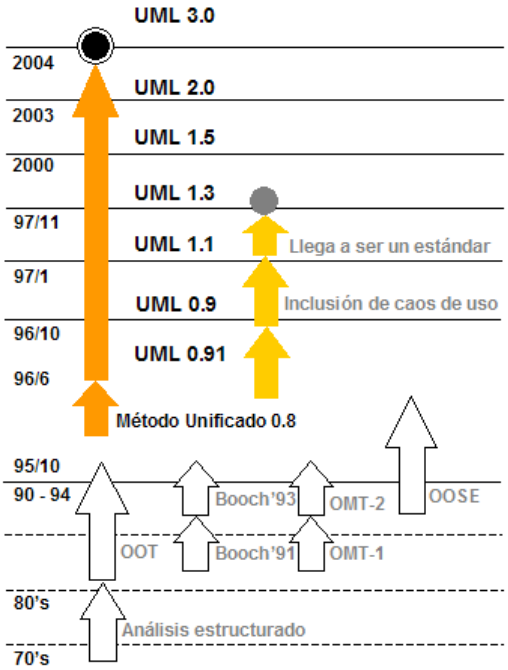


Figura 2-1. Evolución histórica de UML

UML fue adoptado en noviembre de 1997 por OMG (*Object Management Group*) como una de sus especificaciones y desde entonces se ha convertido en un estándar de hecho para visualizar, especificar y documentar los modelos que se crean durante la aplicación de un proceso software.

Desde su concepción UML (Booch et al, 1999) se ha convertido en ese estándar tan ansiado para representar y modelar la información con la que se trabaja en las fases de análisis y, especialmente, de diseño. UML, en definitiva, es un lenguaje que se centra en la representación gráfica de un sistema. Este lenguaje nos indica cómo crear y leer los modelos, pero no dice cómo crearlos. Esto último es el objetivo de las metodologías de desarrollo. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas. Los diagramas más interesantes son los de casos de uso, clases y secuencia. El resto de diagramas muestran distintos aspectos del sistema a

modelar. Para modelar el comportamiento dinámico del sistema están los diagramas de interacción, colaboración, estados y actividades. Los diagramas de componentes y despliegue están enfocados a la implementación del sistema.

UML ha supuesto un gran impacto en la comunidad software, tanto a nivel de desarrollo como de investigación. Sin embargo, como señala Booch (Booch, 2002), todavía sólo un pequeño porcentaje de los desarrolladores utilizan UML y muchos proyectos lo usan tan sólo para documentar sus decisiones de diseño en vez de utilizar los modelos para razonar sobre el sistema, lo que supone su principal beneficio. La notación UML la utiliza el 98% de las herramientas CASE y un 60% de las herramientas de ingeniería de procesos. Ambos usos se espera que lleguen al 99% en los próximos años.

Con la utilización de UML podemos comunicar y compartir el conocimiento de una arquitectura (conceptual, diseño e implementación), donde se entrelazan los procesos de negocio de la organización y las aplicaciones informáticas, mediante la combinación simultánea y sinérgica de las siguientes tareas:

- Definir conceptos a través de sus atributos distintivos y trazar sus límites
- Formalizar unas reglas de relación y actuación
- Hacer visible la granularidad y las relaciones entre elementos
- Mantener la trazabilidad entre la concepción de un problema y su posterior solución y viceversa
- Tomar decisiones y evaluar su impacto dentro de una arquitectura definida
- Organizar la experiencia de los usuarios en patrones de conocimiento
- Comprobar la coherencia, completitud y usabilidad de los entregables
- Alinear la solución tecnológica con la cadena de valor de los actores
- Usar un vocabulario controlado dentro de un espacio de colaboración
- Realizar un plan de producción en base a una factoría de componentes reutilizables

2.3.3 El Proceso Unificado

Aunque UML es bastante independiente del proceso de desarrollo que se siga, los mismos creadores de UML propusieron su propia metodología de desarrollo, denominada Proceso Unificado de Desarrollo (Jacobson et al., 1999; Krutchen et al., 2000)

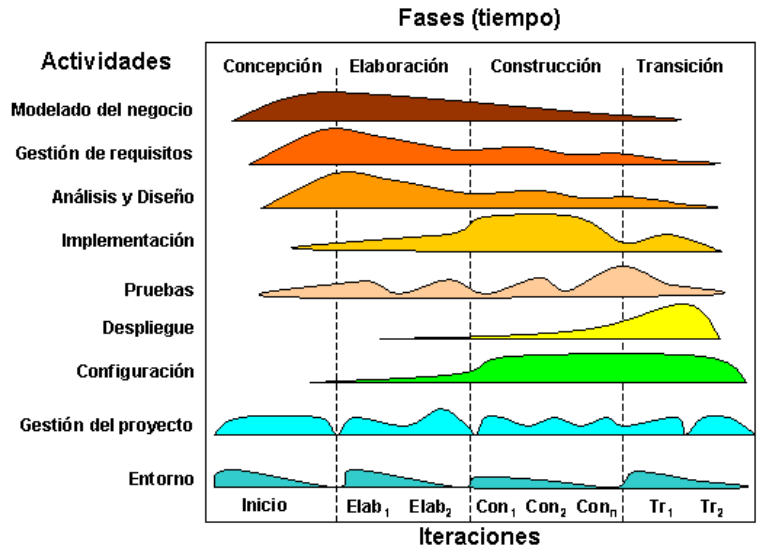


Figura 2-2. RUP: un ejemplo de Proceso Unificado de desarrollo de software

El Proceso Unificado está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. El Proceso Unificado utiliza UML para expresar gráficamente todos los esquemas de un sistema software. Pero, realmente, los aspectos que definen un Proceso Unificado son tres:

i. **Dirigido por casos de uso**: basándose en los casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que los llevan a cabo. Además, estos modelos se validan para que sean conformes a los casos de uso. Finalmente, los casos de uso también sirven para realizar las pruebas sobre los componentes.

ii. **Centrado en la arquitectura**: En la arquitectura relacionada con la construcción de edificios, antes de construir un edificio éste se contempla desde varios puntos de vista: estructura, conducciones eléctricas, fontanería, etc. Cada uno de estos aspectos está representado por un gráfico con su notación correspondiente. Siguiendo este ejemplo, el concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema.

ii. Iterativo e incremental: todo sistema informático complejo supone un gran esfuerzo que puede durar desde varios meses hasta años. Por lo tanto,

lo más práctico es dividir un proyecto en varias fases. Actualmente, se suele hablar de ciclos de vida en los que se realizan varios recorridos por todas las fases. En los proyectos en los que se realizan varios tipos de trabajo (flujos), cada recorrido por las distintas fases se denomina iteración. A su vez, cada iteración parte de la anterior incrementando o revisando la funcionalidad implementada (proceso).

2.3.4 Los casos de uso

En estos años, el área de definición de procesos y técnicas asociadas, como modelado de negocio, ha recibido mucha atención por parte tanto de la industria como del mundo académico. Sin duda, los casos de uso han sido la técnica de modelado que ha gozado de mayor aceptación por su utilidad desde las fases iniciales de cualquier producto software.

En UML, una de las herramientas principales para modelar inicialmente el comportamiento es la construcción de *casos de uso* (Jacobson, 1987). Un caso de uso especifica una manera de usar un sistema sin revelar la estructura interna del mismo. De esta forma el conjunto de casos de uso especifica todas las posibles formas de usar un sistema, sin revelar cómo esto es implementado por el sistema, aunque en este último sentido hay opiniones enfrentadas. (Rumbaugh et al, 1998) y (Constantine et al., 2000) apuestan por la anterior definición y entienden los casos de uso centrados en el usuario, pero Kurchten, en RUP®, recalca el carácter centrado en la arquitectura y dirigido por casos de uso de su propuesta, y define los casos de uso centrados en el sistema más que en el usuario, confirmando, de esta forma, que se trata de un proceso no centrado en el usuario.

En cualquier caso, los casos de uso han sido adoptados casi universalmente para capturar los requisitos de los sistemas software. Sin embargo, los casos de uso son más que una herramienta de especificación ya que tienen una gran influencia sobre todas las fases del proceso de desarrollo.

Los casos de uso, en esta tesis doctoral, especifican un conjunto de secuencias completas de acciones que el sistema puede realizar. Cada secuencia es iniciada por un usuario del sistema. Ésto debe incluir la interacción entre el sistema y su entorno, así como también la respuesta del sistema a estas interacciones, desde este punto de vista, es natural que esta técnica permita la acogida de propuestas IPO como veremos posteriormente.

2.3.5 Desarrollo dirigido por modelos

En la actualidad, la propuesta *Model Driven Architecture* (MDA[1]) (OMG, 2003) del Object Management Group (OMG) está tomando cada vez más fuerza en el ámbito del desarrollo de software. MDA reafirma el concepto de modelo y, a partir de él, asegura mejorar la calidad del software desarrollado, facilitando portabilidad, interoperabilidad y reusabilidad.

MDA promueve el uso intensivo de modelos en el proceso de desarrollo de software. Siguiendo esta aproximación, los desarrolladores construyen modelos de los sistemas software utilizando primitivas de alto nivel de abstracción. Estos modelos son transformados hasta obtener el código fuente del sistema software final. Las partes o descripciones de nuestros sistemas son fundamentalmente *modelos* y por *modelo* se entiende aquella representación declarativa de todos los aspectos relevantes de alguna parte de un sistema donde se incluyen y consideran cada uno de sus componentes, así como aspectos de diseño afines. Por ejemplo, un *modelo de interfaz* sería una descripción declarativa de los aspectos que involucra la interfaz de usuario de un sistema, incluyendo cada componente y el diseño de esa interfaz. Un modelo, por tanto, conlleva la consideración de diferentes elementos a distintos niveles de abstracción, donde cada uno de estos elementos, a su vez, se organiza mediante nuevos modelos.

Todos los autores están de acuerdo en el aspecto novedoso que supone el desarrollo de software guiado por modelos, y por tanto de MDA. La ventaja de este enfoque, respecto a los métodos tradicionales de desarrollo de software que usan modelos, es que los modelos dejan de ser utilizados únicamente para la documentación o como guía para la implementación, convirtiéndose en la herramienta principal del proceso de desarrollo. Una vez especificado el sistema a través de modelos, desde ellos se obtiene el producto final en base a la realización de sucesivas transformaciones.

Al amparo del término modelo, y en este sentido, la *Ingeniería de modelos* es un área que cada vez cobra más importancia tanto en el ámbito académico como de las empresas de desarrollo de software. Con la importancia que cobran los modelos, hay áreas de investigación que han adquirido especial relevancia como son la calidad y evaluación de los modelos, la definición de perfiles, la formalización de las transformaciones de modelos o la creación de herramientas de transformación.

Hoy en día, y debido a la complejidad de los sistemas software desarrollados, cualquier proyecto que pueda aglutinar multitud de subsistemas distintos requiere de modelos para cada subsistema, por lo que definir explícitamente las relaciones entre estos modelos contribuye a aumentar la

[1] Model Driven Architecture (MDA) es una marca registrada de OMG

reutilización del código mejorar el mantenimiento del sistema a lo largo de todo el ciclo de vida del proyecto software, y en consecuencia, a incrementar la productividad del equipo de desarrollo. MDA distingue entre modelos independientes de la plataforma (PIM), que reflejan sólo la lógica de negocio del sistema y modelos dependientes de la plataforma (PSM) a la medida de la plataforma tecnológica empleada. Algo similar, la utilización de modelos y trabajar a diferentes niveles de abstracción, lleva utilizándose en el diseño y desarrollo de interfaces de usuario desde hace más de una década y constituirá la base procedimental de la que partimos en la elaboración de esta tesis doctoral.

2.3.6 Metodologías ágiles

La aparición de UML supuso el reconocimiento de la actividad del modelado (diseño basado en modelos) como una actividad clave para producir software de calidad. Sin embargo, resulta paradójico que al mismo tiempo haya surgido el movimiento del desarrollo ágil de software (Agile, 2001), que considera que el valor está en el código y en las personas más que en los modelos y en los procesos, y la denominada programación extrema (*eXtreme Programing*, XP) (Beck, 1999).

Existen las distintas propuestas metodológicas se incide en diferentes dimensiones del proceso de desarrollo. Por una parte se dispone de aquellas que se centran en el control del proceso, estableciendo: las actividades involucradas, los artefactos a producir, y las herramientas y notaciones que se usarán. Otras propuestas, sin embargo, se centran en otras dimensiones como, por ejemplo, el factor humano o el producto software. Ésta es la filosofía de las metodologías ágiles.

El objetivo de las metodologías ágiles es esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

El punto de partida del desarrollo ágil del software fue el *Manifiesto Ágil* (Agile, 2001), un documento que recoge las principales valoraciones dentro de esta filosofía de desarrollo:

- El individuo y las interacciones del equipo de desarrollo priman sobre el proceso y sobre las herramientas.
- Desarrollar software que funcione mas que conseguir una buena documentación.
- Colaborar con el cliente mas que negociar un contrato.
- Responder a los cambios mas que conseguir estrictamente un plan.

En definitiva la prioridad, en los desarrollos ágiles, es satisfacer al cliente mediante tempranas y continuas entregas de software que le aportan valor.

2.3.7 Conclusiones: IS e Interfaces de usuario

En función del estado actual en el que se encuentra la IS, fruto de las consideraciones presentadas en los apartados anteriores, en lo que respecta al diseño de interfaces de usuario debe tenerse en cuenta que:

En cuanto al Proceso Unificado, una de las versiones más extendidas del Proceso Unificado es la propuesta de Rational (RUP®). Las limitaciones que este proceso puede padecer a la hora de ser utilizado para desarrollar interfaces de usuario están recogidas en (Larman et al, 2001). RUP® contempla el diseño de interfaces de usuario y lo hace en dos fases: modelado de interfaces de usuario y prototipado de interfaces de usuario.

En la fase de modelado de interfaces de usuario, cada caso de uso se describe vía un escenario de caso de uso (*use case storyboard*), que es una descripción contextual de cómo un caso de uso está soportado por la interfaz de usuario. Estos escenarios incluyen la siguiente información (Krutchen et al., 2001):

1. Escenario de flujo de eventos, es decir una descripción textual de la interacción entre el usuario y el sistema

2. Diagramas de interacción: que proporcionan una descripción gráfica de cómo los objetos interactúan en la realización de los casos de uso.

3. Diagramas de clases, que proporcionan una descripción de las entidades y relaciones entre ellas que participan en la realización de cada caso de uso.

4. Requisitos de usabilidad: una descripción textual de los requisitos relacionados con la calidad y la facilidad de uso que ofrece el sistema.

5. Referencias al prototipo de interfaz de usuario, es decir, relaciones entre las entidades manipuladas y los elementos de interfaz de usuario que confeccionan la interfaz.

Los puntos 2 y 3 están soportados directamente por los diagramas de clases y de interacción proporcionados en UML. Para el resto de puntos se

han ido proponiendo diferentes técnicas que contribuyen al desarrollo de interfaces de usuario.

Para el logro del punto 1, por ejemplo, se han propuesto los casos de uso desde UML (Jacobson et al, 1999), con ellos se reconoce la importancia de la caracterización del usuario y de la identificación del role que éste juega. Pero, en los casos de uso, en función de su documentación, se puede introducir información que determina el modo de interacción lo que no resulta siempre deseable. Es por ello por lo que se han propuesto los casos de uso esenciales (Constantine et al., 1999). En ellos se separa la especificación funcional de las consideraciones relacionadas con el diseño de interfaz de usuario. En la misma dirección, también se ha propuesto la recogida tabular de los casos de uso (Phillips et al, 2002). En ella se separan las responsabilidades recogidas en el caso de uso que tienen que ver, por un lado, con el usuario y, por otro, con el sistema.

El punto 4. relacionado con la usabilidad conlleva una especificación textual de requisitos relacionados con los objetivos del usuario en esa dirección. Valga por ahora esta aproximación, pero se volverá a retomar más adelante, ya que resulta insuficiente para lograr evitar problemas posteriores relacionados con la calidad de los productos desarrollados.

El punto 5 pasa por la elaboración de prototipos a diferentes niveles haciendo uso de las herramientas disponibles, y aunque éstas no estén recogidas en forma de diagramas directamente en UML. Por ejemplo, el uso de lápiz y papel, maquetas, o cualquier tipo de herramienta que facilite la elaboración de dichos prototipos debe tenerse en cuenta.

En definitiva, RUP® restringe el diseño de interfaces de usuario a una actividad desarrollada, fundamentalmente, más en la fase de requisitos que en la de diseño o implementación, y siempre manteniendo un punto de vista centrado en la arquitectura y no en el usuario.

En cuanto a UML, como ya se ha comentado algunos de los diagramas presentados con tal iniciativa tienen cabida y utilidad para el posterior desarrollo de interfaces de usuario y, en general, para el modelado. Pero, hay autores que identifican lagunas en UML en diversas direcciones relacionadas, muchas de ellas, con la especificación de interfaces de usuario, por ejemplo:

1. en UML hay una falta de capacidad para modelar, al menos directamente, diagramas de flujo de interfaces de usuario. Distintos autores han utilizado versiones de los diagramas de colaboración y comunicación de UML (Ambler, 2002) y otros han sugerido modificaciones en los diagramas de actividad y de estado (da Silva, 2002) para cubrir dichas carencias.

2. el modelado del comportamiento llevado a cabo en sistemas interactivos utilizando exclusivamente los modelos proporcionados en UML es complejo, es decir, el modelado de acciones que se repiten a lo largo del tiempo, la concurrencia de acciones o la habilitación o deshabilitación de acciones, etc. no está contemplado en los diagramas asociados con UML. Para solucionar este problema se han propuesto nuevas notaciones (Paternò, 1999), modificaciones en los diagramas de comportamiento dinámico propuestos en UML (da Silva, 2000) o definición de clases estereotipadas con UML que consideran esas necesidades (Nunes et al, 2000).

3. UML, por ejemplo, no proporciona mecanismos para denotar elementos de interacción a cualquier nivel, ya sea abstracto o concreto, ni distingue entre las diferentes facetas que con ellas pueden acometerse: introducción de información al sistema, visualización de información al usuario, invocación de comandos o acciones de navegación. Tampoco en UML se ofrecen mecanismos para distinguir entre diferentes tipos de tareas en función de que sean iniciadas por usuarios o por sistemas. En este sentido han aparecido un sinfín de lenguajes de descripción de interfaces de usuario que pueden ser utilizados para este propósito.

4. UML en general es bastante complejo para el diseño de interfaces de usuario, aunque estas sean simples.

Como ha quedado de manifiesto, dar respuesta a estas limitaciones da lugar a notaciones y diagramas adicionales a los propuestos en UML. Puede obtenerse más información en este sentido en los *proceedings* correspondiente al *workshop Towards a UML Profile for Interactive System Development* (TUPIS 2000). En cualquier caso, disponiéndose de un estándar como es UML, la tendencia pasa por contemplar y utilizar dicho estándar, y cualquiera de las notaciones propuestas buscan compatibilidad con él (Paternò, 2001), (da Silva, 2002), (Nunes, 2000), etc.

En cuanto a **los casos de usos** debe reconocerse su potencia para la documentación y el modelado en general de la información asociada a cualquier sistema. Pero se han observado limitaciones en cuanto a su utilidad para recoger información relacionada con el usuario. Los casos de uso sólo facilitan el concepto de *role* de usuario, aunque se han propuesto extensiones en ese sentido. (Halmans et al., 2003; Bühne et al, 2004) utilizan los casos de uso para recoger información con la que representar y analizar la variabilidad, a un nivel de diseño. A otro nivel, a nivel de requisitos, también se han propuesto iniciativas, así (Hui et al., 2003) presentan un marco

de trabajo que considera objetivos, habilidades y preferencias del usuario para hacer al software adaptable. Con estas iniciativas queda patente la preocupación que en IS se tiene del usuario y de como iniciativas en este sentido no están partiendo únicamente del terreno de la IPO.

En cuanto a MDA debe reseñarse que los entornos basado en modelos no son un concepto nuevo en lo que se refiere al desarrollo de interfaces de usuario, pueden encontrarse estudios relacionados con el tema en (Szekely, 1996; Schlumbaugh, 1996; Wilson, 1996; Puerta, 1994; Silva, 2000).

Aunque los entornos de desarrollo basados en modelos de interfaces de usuario no hayan tenido éxito en el terreno comercial y su aplicación se haya circunscrito al ámbito casi enteramente académico, como posteriormente quedará recogido, las propuestas se han sucedido y se suceden desde hace más de una década. En cualquier caso, con la apuesta de OMG por MDA se busca incidir sobre la separación de conceptos y se subraya la idea de trabajar en base a la elaboración de modelos utilizando UML, algo similar a lo que se viene utilizando para el desarrollo de interfaces de usuario en los entornos de desarrollo basados en modelos (MB-UIDEs).

En lo que se refiere a interfaces de usuario, aunque resulta sencillo modelar las componentes habituales de una interfaz de usuario, no es tan directo modelar interfaces de usuario a tal nivel de abstracción que cubran todos los aspectos que deben tenerse en cuenta, como pueden ser: diseño visual, asistencia, manejo de errores, facilidades de ayuda, etc.

Todos los elementos que involucra la interacción entre una persona y un ordenador no pueden ser descritos independientemente unos de otros, en la práctica, todos los aspectos de un sistema tienen un impacto potencial sobre sus usuarios y, debido a ello, el principal reto será el problema del *mapping* o asociación entre los diferentes modelos (Puerta, 1999; Limbourg et al., 2004) que será tratado en el apartado dedicado al desarrollo de interfaces de usuario basado en modelos. Estos *mappings* posibilitan la inferencia y la realización de transformaciones entre unos modelos y otros que es la base de la puesta en práctica de MDA.

Finalmente, **la aplicación de metodologías ágiles** conlleva una serie de beneficios frente a las metodologías pesadas, así como ventajas si se circunscribe la atención exclusivamente al logro final y consideración de la interfaz de usuario. La prioridad por satisfacer al cliente mediante tempranas y continuas entregas de software que le aporten valor, el hecho de que las entregas al cliente sean software y no planificaciones ni documentación de análisis o de diseño, el hecho de que el proceso de desarrollo esté guiado por el cliente, con lo que la interacción con el equipo es muy frecuente, y que la atención a la calidad técnica sea, al igual que al buen diseño y a la

simplicidad a la hora de tomar los caminos más simples, consistente con los objetivos perseguidos, son algunas de las razones por las que este tipo de metodologías redundan de forma positiva en el desarrollo de interfaces de usuario. Muchos de los puntos anteriormente reseñados coinciden, además, con recomendaciones que se han hecho para el desarrollo de interfaces de usuario.

Se han recogido, en este apartado, elementos característicos que la IS ha puesto a disposición del desarrollo de software y que se han utilizado y se están utilizando para el desarrollo de la parte destinada a considerar interfaces de usuario. Se han presentado, conjuntamente, limitaciones que dichos procesos, nomenclaturas y metodologías tienen a la hora de abordar el aspecto específico de desarrollar interfaces de usuario, pero la IS no es la única preocupada del desarrollo de los interfaces de usuario asociados con un producto software y desde la IPO también se han desarrollado metodologías, lenguajes y herramientas que, aunque no han alcanzado el grado de estandarización del que se dispone en IS, si está extendido su uso y gozan de gran popularidad.

Además de los problemas observados en el desarrollo del software, otros problemas específicamente relacionados con el desarrollo de interfaces de usuario, a los que desde la IPO se ha querido dar respuesta, son: el modelado de tareas, la consideración del usuario, y el tratamiento de los problemas relacionados con lo que, por ahora, denominaremos usabilidad.

Antes de pasar a considerar las aportaciones provenientes de IPO, se pueden referenciar iniciativas que se encuentran a medio camino entre ella y la IS. De la primera toman en consideración el desarrollo de la interfaz como un proceso de diseño y con una clara presencia del usuario, de la segunda utilizan metodologías y notaciones específicas. Aportaciones reseñables en este contexto son *Object-View and Interaction-Design* OVID (Roberts et al, 1997), *Whitewater Interactive System Development with Object Models:* Wisdom (Nunes et al, 2000), UML*i* (da Silva, 2002) e IDEAS (Lozano, 2001). Las iniciativas pueden citarse en este momento porque utilizan UML para realizar sus especificaciones, así como las herramientas habituales con las que se desarrolla software para ponerlas en práctica.

OVID fue el primer intento de utilización de UML en el diseño y desarrollo de interfaces de usuario orientados a objetos, acercando el desarrollo orientado objetos a la filosofía de desarrollo centrado en el usuario y viceversa. OVID, frente a otras alternativas de modelado de tareas, utiliza parte de los diagramas de propuestos en UML.

Wisdom (Nunes et al., 2000) pretende ser alternativa a RUP® en el desarrollo de software interactivo y así se presentó. Con Wisdom se enfatiza

la idea del prototipado sobre el que basar un desarrollo incremental de una aplicación, y donde se han revisado y añadido nuevos perfiles a UML para el tratamiento específico de sistemas interactivos (Nunes et al., 2000b). Wisdom hace posible el desarrollo de interfaces de usuario con y desde UML.

UML*i,* de forma similar a Wisdom, presenta perfiles asociados con UML para dar soporte a la elaboración de diagramas relacionados con la interacción llevada a cabo entre persona y ordenador al utilizar una aplicación interactiva. También lleva asociada esta propuesta modificaciones en los diagramas de actividad de UML para dar soporte a la elaboración de diagramas de flujo del proceso de interacción, una de las limitaciones identificadas en UML por algunos autores (Anderson, 2000)(Ambler, 2001).

IDEAS (Lozano, 2001; apéndice F), partiendo de UML y del Proceso Unificado y no perdiendo aportaciones provenientes de IPO, es una propuesta para el desarrollo de interfaces de usuario basado en casos de uso, donde se pone en práctica un proceso iterativo con múltiples realimentaciones desde todas las fases del desarrollo para las que se considera la usabilidad, y el uso de diagramas adicionales a los disponibles en UML. Con ellos se da soporte a la realización de prototipos de interfaz, a partir de elementos de interacción abstractos, y se especifica el modelo de diálogo a través de diagramas de interacción de diálogos donde se recogen los prototipos anteriormente elaborados.

Los principales inconveniente de estas propuestas pasan por no proporcionar mecanismos para generar o inferir a partir de los modelos conceptuales creados directamente interfaz de usuario alguna, los aspectos relacionados con la usabilidad juegan un papel secundario, cuando están presentes, y la reutilización de la experiencia no se explota en ninguna de ellas. Algunos de estos inconvenientes pueden tratarse utilizando técnicas asociadas con la disciplina IPO.

2.4 Desarrollo de interfaces de usuario en IPO

La disciplina IPO se ocupa del estudio y elaboración de productos software que faciliten la realización de tareas a sus usuarios atendiendo a diversos criterios, como pueden ser las facilidades de uso, el tiempo de ejecución, evitar errores y, en consecuencia, aumentar su satisfacción. En (Perlman, 1996) se define IPO como aquella disciplina relacionada con el diseño, evaluación e implementación de sistemas de computación interactivos para uso humano y el estudio de los principales fenómenos asociados con ello.

La IPO se ocupa del análisis y diseño de interfaces entre el hombre y la máquina, estas interfaces se conocen como interfaces de usuario.

De la IPO resalta su carácter interdisciplinar, para cuyo estudio se debe recurrir a otras áreas del conocimiento como son la psicología cognitiva, la informática, la ingeniería del diseño y la lingüística entre otras (Carroll, 2001).

(Booth, 1989) define *interacción* como el intercambio de símbolos entre dos o más partes, asignando los participantes en el proceso comunicativo los significados a esos símbolos. La interacción entre una persona y un ordenador puede ser analizada en función de su estilo, de su estructura, de su contenido y de su comportamiento. El *estilo* se refiere a la forma en que el usuario introduce y recibe la información. La *estructura* tiene que ver con la forma de organizar las componentes. El *contenido* trata el significado semántico y pragmático que se produce durante el diálogo. Por último, el *comportamiento* define las acciones que serán realizadas al interactuar con los objetos de la interfaz.

La investigación en IPO está orientada al desarrollo de nuevos dispositivos e identificación de nuevos estilos de interacción donde se incorporan las capacidades del lenguaje entre personas. Para ello, se basa en el establecimiento de similitudes entre esos dos tipos de diálogo, es decir, ambas partes (emisor y receptor) necesitan compartir unos conceptos y se encuentran en un contexto. El contexto incluye características fundamentales de las personas y de los sistemas. Esas características son tanto las limitaciones físicas (p. e., la capacidad de memoria, los límites de transferencia de información, la habilidad lógica de la máquina) como las conceptuales (p.ej. el conocimiento de las tareas y de los modelos mentales de los objetos y de los procesos) (Marchionini et al., 2003).

2.4.1 Evolución de la Interacción Persona-Ordenador

El interés por el estudio de la IPO está inevitablemente ligado al de la Informática, ya que siempre ha existido la necesidad de estudiar la necesaria comunicación que debe establecerse entre los ordenadores y las personas que los usan. Además, ese mismo estudio es paralelo al de la *ergonomía*, que estudia el manejo de máquinas por parte de personas y cómo éste se produce. A pesar de que podemos situar el inicio de un interes por aspectos relacionados con la IPO en los años 50, hasta la década de los 70 no se aprecia un volumen importante de publicaciones científicas en el ese campo. De la misma forma, los avances de sus aplicaciones no resultan notables sino a partir de los años 80, momento en que comienzan a extenderse el uso de los ordenadores personales.

En esa década de los 80 se produjo también una explosión de publicaciones periódicas, asociaciones y encuentros especializados para el tratamiento de consideraciones relacionadas con la IPO, todo ello se vio favorecido por los grandes avances logrados en las tecnologías informáticas. Estudios más detallados de la evolución de la IPO pueden encontrarse en (Carroll, 2001) y (Karat et al., 2003).

Tabla 2-1. Distribución de artículos en conferencias ACM SIGCHI e IFIP INTERACT atendiendo a diferentes topics

Topics	Porcentaje de artículos en CHI'83, INTERACT'84, y CHI'85 (n = 207)	Porcentaje de artículos en CHI'01, INTERACT'01, y CHI'02 (n = 214)
Sistemas	17	35
Métodos	15	6
Análisis de uso	26	27
Comparación de IU	14	20
Otros	28	12

Sirva la Tabla 2-1 para mostrar el interés reflejado por la comunidad IPO en los artículos enviados a conferencias específicas de dicha disciplina, como pueden ser CHI o INTERACT. De la tabla se desprende que existe un proceso de maduración en disciplina considerada, ya que la publicación de artículos relacionados con métodos ha disminuido, incrementándose paralelamente la publicación sobre herramientas que utilizan dichos métodos. Asociando los datos mostrados en la tabla con el Modelo de Madurez de Usabilidad[1] (Earthy, 1998) se puede identificar que, a nivel de disciplina, en la actualidad la IPO se encuentra a medio camino entre el nivel C y el nivel D que marcan, respectivamente, los niveles de implementación e integración, es decir, los diseñadores y desarrolladores de software son conscientes de que existen problemas relacionados con el uso que los usuarios hacen de productos informáticos, y ante ello la intención es la de intervenir ya que muchos de esos problemas no tienen porque surgir.

2.4.2 Propuestas metodológicas provenientes de IPO

Si el proceso que determinaba la realización de propuestas metodológicas en IS, con el paso del tiempo, ha llegado a ser un Proceso Unificado, en IPO lo que marca la realización de propuestas similares es la puesta en

[1] *Usability Maturity Model* es una escala que presenta diferentes niveles asociados con la consideración y preocupación que una institución tiene de la usabilidad.

práctica de metodologías y técnicas de Diseño Centrado en el Usuario (DCU). En (Gould et al, 1983) se establecieron los principios para el desarrollo de dicho proceso, que pasan por:

i. **conocer al usuario y sus tareas**, según lo cual los diseñadores deben saber quiénes son los usuarios, estudiando sus capacidades cognitivas, comportamientos, y actitudes, y por otro lado la naturaleza de las tareas que llevan a cabo.

ii. **realización de medidas empíricas.** En cuanto sea posible, y cuanto antes en el proceso de desarrollo, a los usuarios se les deben proporcionar prototipos y simulaciones con los que llevar a cabo sus tareas, y con ellos deben observarse, grabarse y analizarse las prestaciones y las reacciones que estos usuarios experimentan.

iii. **aplicar un diseño iterativo**. Fruto de observar problemas con la puesta en práctica de evaluaciones reiteradas, se debe proceder, a través del diseño, a eliminar aquellos problemas surjan en una versión.

Por Diseño Centrado en el Usuario (DCU) se entiende *aquel proceso por el que al usuario y a sus datos se les establece como criterio con el que evaluar los diseños, y como la fuente de ideas de diseño* (Wixon, 1996). Las propuestas de (Gould et al) vistas anteriormente han sido utilizadas por (Vredenburg et al., 2002) para definir el DCU estableciendo que es el proceso por el que se *involucra activamente al usuario, con el fin de entender sus requisitos y tareas, en un desarrollo multidisciplinar basado en el diseño iterativo y en la evaluación*. Por DCU se entiende, por tanto, una filosofía de desarrollo de software, que frente a la tradicional que se centra en la tecnología, pone al usuario como centro y objetivo de todo.

Existe una clara relación entre el DCU y la Ingeniería de la Usabilidad. Los objetivos entre ambas denominaciones son los mismos, lo que varía es el enfoque, siendo el que ofrece la segunda más orientado a lo *ingenieril* que el que está presente en la primera. En la Tabla 2-2 se recogen las características más reseñables de la forma tradicional de desarrollo de software frente a la filosofía propuesta desde el DCU (Seffah, 2004).

Tabla 2-2. Desarrollo de software tradicional *vs* centrado en el usuario

Desarrollo tradicional de software	Desarrollo centrado en el usuario
Dirigido por la tecnología	Dirigido por el usuario
Centrado en el componente	Centrado en la solución
Contribución individual	Contribución multidisciplinar
Centrado en arquitectura interna	Centrado en atributos internos
Calidad medida por defectos del producto y prestaciones	Calidad definida por la satisfacción del usuario y la calidad en uso
Implementación previa a la validación	Implementación de soluciones previamente validadas
Soluciones dirigidas por requisitos funcionales	Consideración del contexto de uso

Partiendo de los principios básicos establecidos en (Gould et al., 1983), hay disponibles diferentes ciclos de vida de desarrollo de software que tratan de ponerlos en práctica. Entre ellos cabe destacar los presentados en (Nielsen, 1993), el ciclo de vida de la Ingeniería de Usabilidad de (Mayhew, 1999), el diseño centrado en el uso de (Constantine et al., 1999) y LUCID de (Cognetics, 2000). Algunas de las principales características de estas propuestas se recogerán seguidamente. Otras iniciativas, como el ciclo de vida en estrella de (Hix et al., 1993), que también podría ser considerada dentro del grupo anterior, puede, igualmente, ser considerada como otra variante del ciclo de vida de desarrollo de software, como son las propuestas en cascada o en espiral (Boehm, 1988). La propuesta en estrella tiene la particularidad de que todas las fases giran entorno a un proceso de evaluación característico, por otra parte, en las técnicas ligadas al DCU. Seguidamente comentaremos algunas de estas propuestas en mayor detalle.

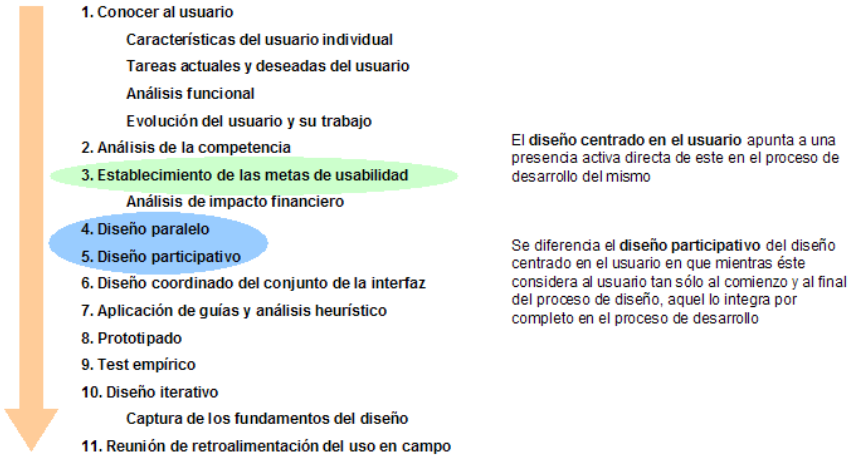


Figura 2-3. Ingeniería de la usabilidad según (Nielsen, 1993)

En (Nielsen, 1993) se propusieron una serie de pasos (véase Fig. 2-3) para poner en práctica un DCU. En él, en las primeras fases se conoce al usuario, se identifican sus tareas y se establecen una serie de objetivos de usabilidad que se perseguirán a lo largo de un diseño iterativo, en el que tienen cabida métodos de análisis y evaluación de la usabilidad así como análisis funcionales, diseño participativo, análisis mediante guías de estilo, utilización de técnicas de prototipado y evaluación heurística, todos estos métodos son algunos de los métodos habitualmente utilizados para llevar a cabo actividades de evaluación de la usabilidad, y serán tratados con más dedicación en el próximo capítulo.

En (Mayhew, 1999) se recoge el *Ciclo de Vida de Ingeniería de la Usabilidad* (Fig. 2-4) destinado al desarrollo de interfaces de usuario usables. Este ciclo de vida estructura las actividades en tres fases: análisis de requisitos, diseño/prueba/desarrollo, e instalación. Este proceso de desarrollo sigue un enfoque de ciclo de vida en cascada entre las etapas reseñadas, por tanto no se trata de un enfoque verdaderamente iterativo. Aunque se afirma que el método está especialmente destinado al tratamiento y consideración de la interfaz de usuario, se incluyen también actividades relacionadas con la ingeniería de requisitos, como por ejemplo, el análisis contextual de tareas. Mayhew precisa cómo se debe integrar su propuesta con la Ingeniería de Software Orientada a Objetos y con los métodos de prototipado rápido, pero admite que la integración debe ser particularizada.

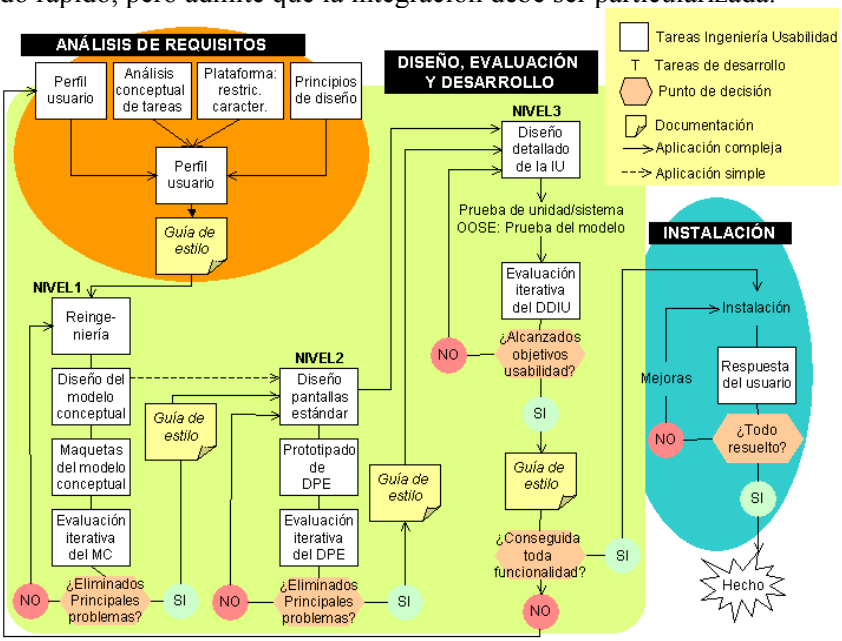


Figura 2-4. Ingeniería de la usabilidad según (Mayhew, 1999)

En (Constantine et al., 1999) se propone el método de Diseño Centrado en el Uso (DC-U) (véase Fig. 2-5). Este método incluye algunas actividades que corresponden al proceso de desarrollo general de software, junto con otras relacionadas con la usabilidad, como el modelado de tareas o el modelado de la interfaz. Los casos de uso esenciales, reseñados en un apartado anterior, son el pilar de este método, y los mismos no son sino una reinterpretación de la técnica de casos de uso que acompaña a UML y es muy popular en IS, no en vano Constantine es una figura reconocida de la IS en el ámbito del desarrollo estructurado.

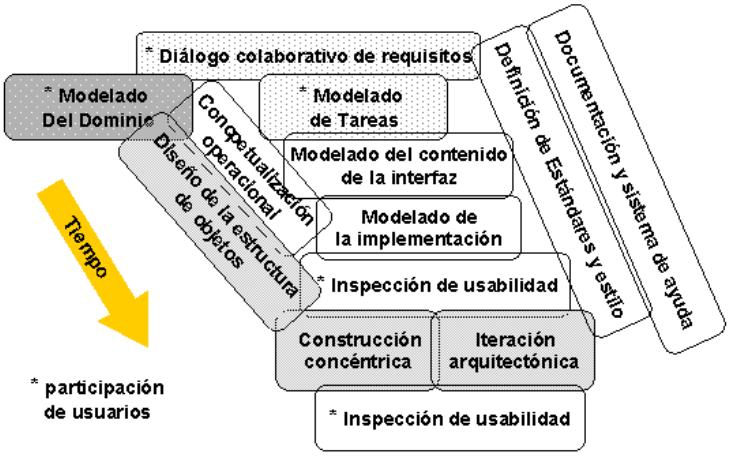


Figura 2-5. Diseño centrado en el uso propuesto en (Constantine et al., 2001)

Aunque incluida dentro de las propuestas centradas en el usuario, el DC-U tiene sus propias características distintivas frente al DCU. Las diferencias más reseñables se recogen en (Constantine et al., 2001) y pasan por que se presta especial atención al uso que del producto software se hará. Así, se desarrollan aplicaciones que dan soporte a la realización de las tareas necesitadas por el usuario, de esta forma, su puesta en práctica está dirigida por modelos en lugar de por las entradas del usuario. Además, el usuario juega un papel dentro de la metodología, aunque en etapas concretas, como son las relacionadas con la inspección de la usabilidad, la validación de modelos o la captura de requisitos. Esta propuesta es sistemática y de inspiración *ingenieril*, no existiendo espacio a la improvisación.

Figura 2-6. Desarrollo LUCID (Cognetics, 2000)

Finalmente, *Logical User Centered Interaction Design* (LUCID) de (Cognetics, 2000) es una propuesta basada en una secuencia de seis etapas

(Fig. 2-6) en las que se lleva al producto desde su concepción inicial a su fase de entrega. Este método fue una propuesta para diseñar interfaces de usuario con consideración de actividades relacionadas con la usabilidad en todas las etapas reflejadas en la Fig. 2-6, tratándose, además, de un proceso altamente iterativo.

En vista de las propuestas recogidas se observan una serie de características comunes que pasan por la necesidad de una fase de conocimiento del usuario y de sus tareas, a modo y manera de ingeniería de requisitos en IS. También se contempla la presencia de la calidad enfocada a la usabilidad, y se hace desde el principio, estableciendo unos objetivos relacionados con ella (propuestas de Nielsen y Mayhew), nos centraremos en este punto en el siguiente capítulo destinado a caracterizar la calidad y la usabilidad. Cabe mencionar también que salvo la propuesta de Mayhew, que establece relaciones entre su propuesta y los desarrollos orientados a objetos, ninguna de las restantes propuestas apuesta por una entorno de desarrollo que las ponga en práctica, aunque desde hace unos años y en el campo del desarrollo de interfaces de usuario los Entornos de Desarrollo basados en Modelos (MB-UIDEs) están logrando buenos resultados, aunque sea sólo en el restringido campo académico. En las propuestas mencionadas, la realización de prototipos y la evaluación de los mismos con usuarios son actividades que siempre aparecen reflejadas.

En secciones anteriores relacionadas con el punto de vista de la IS, se han identificado limitaciones en lo que concierne a actividades destinadas al modelado del usuario, identificación de las tareas que realiza y para abordar la realización de prototipos de interfaz de usuario. También son necesarios métodos para llevar a cabo la elicitación de todas las características mencionadas. Debido a ello, las propuestas provenientes del mundo de la IS pueden verse complementadas con propuestas realizadas desde el ámbito de la IPO y viceversa.

Seguidamente se tratarán los tres primeros elementos reseñados para la puesta en práctica de una metodología de DCU, es decir, **el modelado de usuario**, **el modelado de tareas** y la **presentación de interfaces de usuario**. Más adelante se verá cómo esos elementos de modelado se han utilizado en los MB-UIDEs. Dejaremos para capítulos posteriores, por su extensión y relevancia, la consideraciones que introducen la calidad y la experiencia, así como su inclusión en el ciclo de desarrollo.

2.4.3 La misma arma: el modelado

Si MDA es la tendencia por la que se ha apostado en IS, con la que se aumenta el nivel de abstracción y la especificación de modelos para recoger

los aspectos funcionales y no funcionales de las aplicaciones software, en el desarrollo de interfaces de usuario se lleva utilizando más de una década una idea similar para generar automáticamente o semi-automáticamente la parte destinada a la interfaz. La diferencia fundamental entre una y otra tendencia está en que en IS, tras una larga etapa de unificación y estandarización, se dispone de un lenguaje unificado (UML), y en IPO esto no ocurre. IPO, en este sentido, está diez años atrás y se encuentra en la fase de fragmentación y búsqueda de la unificación en lo referente a disponibilidad de una notación unificada.

Desde hace más de una década, los entornos de desarrollo basados en modelos son habituales para el desarrollo de interfaces de usuario, es lo que se conoce como *Model-Based User Interface Development Environment* (MB-UIDE). La idea que subyace debajo de esa propuesta es la especificación de todos aquellos aspectos relacionados con la interfaz de usuario y que influyen en la misma utilizando modelos. Se pueden revisar (Szekely, 1996; Schlungnaum, 1996; Puerta, 1997; Griffiths et al, 1998; da Silva, 2000; Limbourg et al., 2004) para una discusión de la naturaleza de los modelos que involucra el modelo de interfaz de usuario utilizando una metodología basada en modelos.

Al no existir consenso en la notación o lenguaje utilizado, no todas las propuestas presentan los mismos modelos, ni utilizan las mismas notaciones para su especificación. Prácticamente, la única característica común que en la que se coincide es que son apuestas de carácter académico más que empresarial, y donde el modelado es el mecanismo utilizado. En cualquier caso, aparecen reiteradamente diferentes modelos característicos, algunos de forma constante y otros se han ido añadiendo conforme han ido sucediéndose las diferentes propuestas y los nuevos retos con los que se ha ido encontrando el desarrollo de interfaces de usuario. Los modelos de dominio, de tareas, de usuario, de diálogo, de presentación, de plataforma, de contexto o de comportamiento son los modelos utilizados en las diferentes propuestas, y con ellos se recogen los elementos que tienen influencia para el posterior desarrollo de una aplicación y de su interfaz de usuario. Por ejemplo, a la hora de especificar *el modelo de dominio* se recoge información de aquellos objetos que manipulará el usuario, junto con sus características y el comportamiento que ofrecerán los mismos. Para especificar el modelo de dominio, lo habitual, es utilizar diagramas de clases y la notación propuesta en UML (ej. OVID, Wisdom, IDEAS, etc.), pero eso no quita para que, antes de UML, se utilizaran especificaciones algebraicas (ej. ITS), notaciones propias (ej. Mecano, Mobi-D), modelos de datos OPAC (ej. Diane, Diane+), modelos entidad-relación (ej. Trident o Genius), notaciones previas a UML relacionadas con orientación a objetos

OMT/OOA (ej. Janus, AME) y ODMG (ej. Teallach) con el mismo propósito.

De igual forma sucede con *el modelo de tareas*, donde el desarrollador recoge una especificación de las tareas que el usuario llevará a cabo en el sistema para lograr sus objetivos. Dichas tareas, que pueden ser de diferentes tipos, se representan utilizando una descomposición jerárquica de las acciones involucradas en su desarrollo, haciéndose uso de notaciones que serán presentadas en un apartado posterior. Hay propuestas de MB-UIDEs cuyo modelo esencial es el modelo de tareas, por ejemplo Adept (Wilson et al, 1993) o Tadeus (Schlungbaum, 1996), donde se utiliza la notación *Task Knowledge Structures* (TKS – Johnson et al, 1992); Trident, donde se utiliza *Activity Chainig Graphs* (ACG), pero también se utiliza la notación CTT (Paternò, 1999), UAN (Hartson et al., 1990) y HTA (Annett et al., 1967).

El *modelo de usuario* recoge las características del usuario final. Unas características que deben ser tenidas en cuenta para abordar la personalización final de la interfaz de usuario en sus posibles vertientes; adaptación y/o adaptatividad, así como para permitir proporcionar ayuda al usuario con la que facilitar el uso del sistema software.

Según (Gould et al, 1985) identificar las características del usuario y de las tareas son los primeros aspectos básicos para poner en práctica un desarrollo centrado en el usuario, es por ello, que dedicaremos los próximos apartados a un tratamiento de los diferentes mecanismos que permiten el modelado de los mismos. Junto con las tareas y el usuario, la generación rápida de prototipos y la evaluación de los mismos también dispondrán de consideración especial en éste y posteriores capítulos. Un apartado en este mismo capítulo estará dedicado a los lenguajes de especificación de interfaces de usuario disponibles, y a las diferentes notaciones propuestas que permiten abordar la tarea de modelar la presentación y de, indirectamente, generar prototipos de interfaz. Se deja para el capítulo tercero el aspecto de evaluación de dichas interfaces y de caracterización de su bondad.

En cualquier caso, la verdadera potencia y versatilidad del desarrollo basado en modelos para abordar el diseño de interfaces de usuarios no se encuentra en la propia especificación de modelos utilizando una notación u otra, y recogiendo información más o menos significativa en cada uno de esos modelos. La potencia de los MB-UIDEs está en las diferentes relaciones que se establecen entre los modelos especificados. El establecimiento de estas relaciones da lugar a un problema de *mapping*, identificado por (Puerta et al., 1999; Limbourg et al., 2004) que ha sido abordado a la hora de presentar las diferentes propuestas en mayor o menor nivel de detalle. Consideraciones relacionadas con este problema también serán tratadas en un apartado posterior destinado específicamente a tal fin.

2.4.4 Modelado del usuario y del contexto

Tradicionalmente, el desarrollo del software ha pretendido un *one-size-fits-many*, pero con el aumento de la demanda tecnológica en la sociedad, y la consideración que añade la disponibilidad de diferentes dispositivos y formas de interacción se hace necesario reconocer que diferentes individuos con diferentes niveles de experiencia, preferencias, necesidades y objetivos o simplemente interactuando desde diferentes dispositivos o contextos, precisarán de diferentes clases de servicios.

La IS, la IPO y la Inteligencia Artificial (IA) han aportado diferentes puntos de vista para tratar este problema de demanda y diversidad de individuos. En IS las soluciones pasan, fundamentalmente, por aportaciones en el terreno del diseño y de los requisitos, donde la personalización se considera dentro de la propia funcionalidad. La IPO se ocupa del desarrollo, y puesta en práctica, de mecanismos que permitan manipular la funcionalidad a través de la interfaz de un sistema software, y para ello se ha centrado en la puesta en práctica de técnicas destinadas al análisis y a la evaluación de interfaces de usuario. La IA ha desarrollado técnicas automáticas que permiten registrar las acciones del usuario con el fin de inferir sus objetivos y preferencias traduciéndose en modificaciones en la interfaz de usuario ofrecida.

A la hora de hablar de personalización, y referirnos a la interfaz de usuario, se deben diferenciar dos posibilidades: la *adaptación* y la *adaptatividad* (Fischer, 2001). Un sistema se dice *adaptable* si proporciona mecanismos con los que el usuario puede realizar cambios en la forma de operar que se le ofrece. Un sistema se dice *adaptativo* si es el propio sistema el que dinámicamente se adapta a la tarea que se esté realizando o al usuario que lo esté utilizando. Un análisis del usuario o usuarios y un posterior procesamiento estructurado de los datos, con los que obtener un modelo del mismo, permiten llevar a cabo acciones adaptativas.

Al hablar de modelo de usuario, debe distinguirse, igualmente, entre *modelo de usuario* y *modelo mental* (Fischer, 2001). El primero hace alusión al modelo que el sistema tiene del usuario, el segundo al modelo que el usuario tiene del sistema. En este sentido, y considerando la Fig. 2-7, la funcionalidad conocida por un usuario de un sistema y habitualmente utilizada está representada como D1, D2 representa los elementos que son poco conocidos y que son utilizados ocasionalmente, y D3 son los elementos que el usuario cree que existen en el sistema, son, por tanto, modelos mentales que el usuario tiene del sistema. D4, también en la Fig. 2-7, representa la funcionalidad total ofrecida por el sistema, y la diferencia entre D4 y D3 resulta de especial interés cuando de lo que se trata es de modelar al usuario, ya que representa lo que el usuario ni conoce, ni cree que existe en

la aplicación que está utilizando y, por lo tanto, desaprovecha dicha funcionalidad.

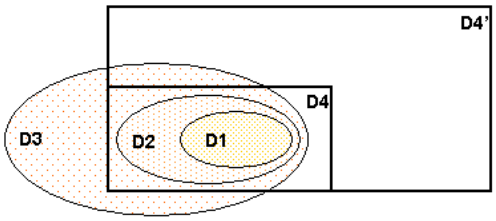


Figura 2-7. Modelos mentales que el usuario puede tener de un sistema

De cualquier forma, y tal y como dijo José Ortega y Gasset: *Yo soy yo y mis circunstancias*, y eso es lo que se termina pensando cuando se considera al usuario y se está desarrollando un producto software. No se puede caer en la tentación de pensar que el usuario es el único centro de atención y debe considerarse también el *contexto de uso* (Jameson, 2001) en el que el usuario se mueve. Jameson ofrece, en uno de los tutoriales que organiza, *Systems that adapt to their users*, información muy completa de lo que debe tenerse en cuenta a la hora de caracterizar el contexto de uso y al usuario y de cómo dicha información influye en la utilidad final que ofrece un sistema (véase Fig. 2-8).

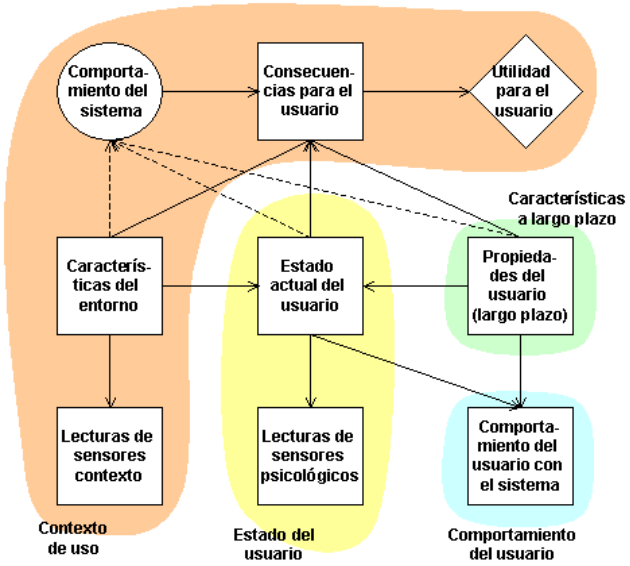


Figura 2-8. Relación entre utilidad de la información ofrecida por un sistema, usuario y contexto

Relacionado con los modelos de usuario, una revisión de las herramientas disponibles para la elaboración de modelos de usuarios puede encontrarse en sendos artículos de Kobsa (Kobsa, 1993; 2001). Este mismo autor

identifica la principal consideración que debe tenerse en cuenta cuando se desea modelar al usuario y su contexto. La consideración referida no es otra que la de asegurar la privacidad de la información recogida del usuario, sobre todo cuando ésta puede ser de carácter sensible. Por ello, en función de la información utilizada para modelar al usuario, se deben tener en cuenta las leyes internacionales vigentes en materia de protección de datos (Kobsa, 2002).

En ocasiones, lo que se desea no es modelar al usuario en el sentido general, sino sólo sus intereses y el conocimiento que necesita en relación con las tareas que realiza en cierto dominio. En esos casos la información a modelar estará más asociada a las tareas que al propio usuario. En este sentido, (Constantine, 2000) propone que cualquier sistema software no deja de ser una herramienta y, como tal, lo realmente importante para el usuario es que le permita hacer su trabajo más fácil, más rápido, de forma más flexible o más agradable y que, por lo tanto, la construcción de software no debe girar alrededor del usuario, sino de su uso abogando, de esta manera, por un modelado de tareas.

En definitiva, las características del usuario y de su contexto influyen de forma importante en el uso que se hace de un producto software, pero no serán tratadas en profundidad en esta tesis doctoral ya que exceden del objetivo principal. En cualquier caso, su consideración quedará recogida como posible trabajo futuro y, en ese sentido, un lenguaje actualmente en desarrollo basado en las propuestas de (Jameson, 2001) es *User Modeling Markup Language* (UserML) (Heckmann et al, 2003) puede ser utilizado para ese propósito, aunque a fecha de realización de esta tesis doctoral está en desarrollo.

2.4.5 Modelado de tareas

Para llevar a cabo el análisis de tareas, podemos utilizar diferentes métodos que se diferencian en el grado de formalismo de su notación, poder de expresividad y finalidad (Paternó, 2000; Limbourg et al., 2001). Si bien todos esos métodos sirven para analizar las tareas de un sistema, la finalidad del estudio puede ser *cognitiva*, *predictiva* o *descriptiva*.

Los métodos cognitivos identifican secuencias de comportamiento correctas, representando el tipo de conocimiento que debe poseer un usuario acerca del uso del sistema para llevarlas a cabo. De esta forma, partiendo de la descripción de tareas generan una especificación del conocimiento del usuario, con lo que pueden ser utilizados para el modelado del usuario indirectamente. Los métodos predictivos describen secuencias de comportamiento y el conocimiento que necesita el usuario para su ejecución, reali-

zando un análisis centrado en rutinas de comportamiento. Finalmente, los métodos descriptivos permiten obtener una descripción más o menos completa del sistema a partir de la información obtenida de las tareas.

Ejemplos de métodos de análisis de tareas son: el *Hierarchical Task Analysis* (HTA), el *Goal-Operations-Methods-Selection* (GOMS), el *Task Action Grammar* (TAG), la *Task Knowledge Structures* (TKS), la *User Action Notation* (UAN) y la, más reciente, notación *ConcurTaskTrees* (CTT).

En HTA (Annett et al., 1967) se realiza una descripción en términos de operaciones y planes. Las operaciones son actividades que realizan las personas para alcanzar un objetivo, y los planes son una descripción de las condiciones que se tienen que dar cuando se realiza cada una de las actividades. Las operaciones se pueden descomponer de forma jerárquica y se asigna un plan a cada una de las subtareas que aparecen. Se define un objetivo como un estado determinado del sistema que quiere alcanzar el usuario. El formato gráfico utilizado en HTA se parece a un árbol con ramas y subramas, en función de las necesidades. HTA es cognitivo y gráfico.

GOMS (Card et al., 1983) comprende a una familia de lenguajes que se basan en la visión del usuario como un sistema procesador de información. El modelo GOMS se basa en el mecanismo de razonamiento humano para la resolución de problemas y realiza la formalización de aquellas actividades que intervienen en esa labor. Para cada tarea se describe el objetivo a satisfacer, el conjunto de operaciones que el sistema pone a disposición del usuario para la interacción, los métodos disponibles para llevar a cabo esas operaciones y, por último, un conjunto de reglas de selección para determinar la alternativa más conveniente en cada caso. Cada tarea se puede descomponer en otras tareas primitivas formando un árbol jerárquico. GOMS es cognitivo y textual.

TAG (Payne et al., 1986) describe el conocimiento del usuario para realizar una determinada tarea utilizando una gramática con características. El conocimiento que posee el usuario se expresa mediante un esquema que engloba a un conjunto de reglas individuales. El esquema estará formado por una estructura sintáctica definida a partir de un conjunto de características que definen la semántica de la tarea. TAG es textual y predictivo, pero al ser una técnica basada en gramáticas no resulta adecuada para la descripción de la interacción llevada a cabo con interfaces gráficas, ya que no permite describir la realimentación visual.

TKS (Jhonson, 1989) en una notación que proporciona facilidades para el análisis y modelado de tareas con el propósito de generar información de diseño (Johnson et al., 1993). TKS asume que la gente crea estructuras de conocimiento en su memoria relacionadas con las tareas que realiza y sobre los objetos que manipula, junto con sus relaciones y comportamientos. TKS es un método cognitivo y gráfico.

UAN (Hartson et al., 1990) es una notación centrada en el usuario cuya principal característica es la descripción física de las acciones a realizar en el proceso de interacción. En una especificación UAN se utiliza una tabla dividida en tres columnas describiendo las acciones de usuario, la realimentación de la interfaz y el estado del sistema tras la acción. Permite la especificación de acciones que conllevan la manipulación de objetos de la interfaz. UAN es un método cognitivo y gráfico.

CTT (Paternò, 1999) es una notación cuyo principal objetivo es el de representar las relaciones temporales existentes entre las actividades y usuarios que son necesarios para llevar a cabo las tareas. Se trata de una notación gráfica, descriptiva y simple en forma de árbol, donde se representa mediante una descomposición jerárquica las tareas existentes en un sistema. CTT permite la utilización de un conjunto de operaciones, tomadas de LOTOS, para describir las relaciones temporales entre tareas y representar operaciones secuenciales, concurrentes, recursivas, iterativas, opcionales, etc., algunas de las que se han identificado delicadas a la hora de ser representadas utilizando los diagramas facilitados con UML. En CTT se pueden distinguir cinco tipos diferentes de tareas en función del actor o actores que las lleven a cabo (Tabla 2-3).

Tabla 2-3. Tipos de tareas en la notación CTT

Tipo de tarea	Descripción
	Abstracta
	De aplicación
	De interacción
	De usuario
	Cooperación

En CTT, junto a los tipos de tareas definidos, se dispone de una serie de operadores para la descripción de relaciones temporales entre las tareas especificadas. Los operadores definidos son los recogidos en la Tabla 2-4.

La notación CTT se ha convertido en una notación muy extendida en IPO ya que permite superar con facilidad las limitaciones de especificación de determinadas relaciones temporales entre tareas que se le han achacado a UML. Aunque, si bien dichas limitaciones pueden superarse también desde el propio UML y, en este sentido, por ejemplo (Nunes et al, 2000) propuso definir una serie de perfiles a UML con los que es posible modelar la misma semántica que se recoge utilizando la notación CTT para el modelado de tareas. También es posible utilizar casos de uso estereotipados y

diagramas de actividad como sugiere (Markopoulus, 2001) que propone una equivalencia entre notaciones propias de esos diagramas y las relaciones propuestas en CTT (véase Fig. 2-9). Con la misma intención (da Silva, 2002) con UML*i* propone una serie de estereotipos con los que extender la potencia ofrecida por los diagramas de actividad de UML, y así posibilitar su uso para modelar el flujo entre objetos (*present*, *interacts*, *cancels* y *activate*). También propone estados estereotipados para superar las dificultades a la hora de representar tareas opcionales o realizadas en múltiples ocasiones (*OrderIndependenceState*, *OptionalState* y *RepeteableState*).

Tabla 2-4. Operadores temporales definidos en la notación CTT

Operador	Descripción
T1 ||| T2	Concurrencia. Las acciones pueden realizarse en cualquier orden sin ningún tipo de restricción
T1 |[ ]| T2	Sincronización. Las acciones pueden realizarse en cualquier orden, pero deben sincronizarse para intercambiar información
T1 >> T2	Activación. Una tarea permite la ejecución de otra cuando termina
T1 [ ]>> T2	Activación con paso de información. T1 proporciona información a T2 y permite su ejecución cuando finaliza.
T1 [ ] T2	Elección. Es posible la elección entre un conjunto de tareas
T1 [> T2	Desactivar. La primera tarea es desactivada cuando comienza la ejecución de la segunda
T1 |> T2	Suspender/Resumir. T2 tiene la posibilidad de interrumpir a T1 que podrá ser retomada cuando aquella finalice
T1 |=| T2	Indepencia en cuanto al orden. Ambas tareas pueden ser realizadas, pero una vez comenzada una debe finalizar antes de que comenzar con la otra
[T]	La realización de la tarea es opcional
T*	Realización reiterada de una tarea

Paralelamente con las notaciones presentadas, los casos de uso son los únicos diagramas de UML que proporcionan una vista del sistema desde el punto de vista del usuario al recoger qué es lo que el sistema hace para cada actor. Los casos de uso son, por tanto, la principal herramienta que se proporciona para llevar a cabo tareas de análisis de tareas. En esta dirección, en (Roberts et al, 1998) hay una comparativa entre HTA y casos de uso, apostándose por el poder de representación de los casos de uso frente a HTA desde el punto de vista de la IS.

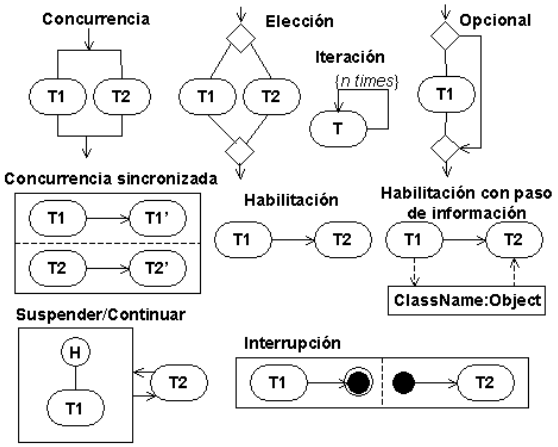


Figura 2-9. Equivalencias entre CTT y los diagramas de actividad de UML

En el libro de (Larman, 2001) se hace una distinción en tipos de los casos de uso según conveniencia y fase del proyecto en que se utilice. De entrada, Larman hace la siguiente diferenciación de casos de uso:

- **Casos de uso en formato de alto nivel** (Jacobson, 1997). Se describe un proceso muy brevemente, en pocas líneas. Conviene utilizar este tipo de formato durante el examen inicial de los requisitos y del proyecto en sí, a fin de entender rápidamente el grado de complejidad y funcionalidad del sistema. Son casos muy vagos en lo que respecta a las decisiones de diseño.
- **Casos de uso en formato expandido.** Se describe un proceso más a fondo; la diferencia básica con el de alto nivel consiste en que cuenta con una sección destinada al detalle, paso a paso, de la evolución de los eventos.
- Los **casos de uso esenciales** (Constantine et al, 1999) son casos expandidos expresados en forma teórica; esto supone que tienen pocos detalles sobre la futura implementación, las decisiones de diseño quedan pospuestas y se abstraen de la realidad (en especial aquellas concernientes a la interfaz de usuario, p.ej. diremos "el usuario valida la introducción de los datos"; en vez de "el usuario pulsa el botón Aceptar", la manera en que se valida no queda detallada en el caso esencial).
- Finalmente, los casos de uso reales, que a diferencia de los anteriores, describen el proceso a partir de su diseño concreto.

De los mencionados casos de uso, los *esenciales* se constituyen en pilar básico en los que se sustenta la propuesta DC-U de (Constantine et al,

1999), donde se utilizan para llevar a cabo labores relacionadas con el modelado de tareas. La información recogida con los casos de uso es, posteriormente, enriquecida con otros diagramas facilitados en UML como son los de actividad.

En definitiva, el modelado de tareas es una de las posibilidades ofrecidas con las que desarrollar productos software altamente interactivos. Para desarrollar dicha labor de modelado se han propuesto numerosas notaciones. Una de las más extendidas es la notación CTT que ha sido comentada en este apartado. En esta tesis doctoral se utilizará la notación CTT junto con los casos de uso con la intención de potenciar la elicitación de tareas y considerar en dicho proceso la calidad de la interfaz y la experiencia en su especificación y elaboración. En cualquier caso, para el usuario, la aplicación es su interfaz y hasta el momento, en este capítulo, no se han tenido en cuenta consideraciones sobre el modelado de la presentación. Se hará seguidamente.

2.4.6 Modelado de la presentación

Uno de los puntos débiles que resaltamos anteriormente que tiene UML es el tratamiento que se da a la interfaz de usuario. En UML no se proporciona ningún diagrama estándar para representar elementos de interacción, ni se ofrece solución a la representación de flujos de navegación, al menos directamente, ya que hay autores que si que identifican la posibilidad de lograr estas necesidades utilizando UML, haciendo uso de diagramas facilitados por dicho lenguaje. Autores como (Anderson, 2000; Lieberman, 2004) utilizan, respectivamente, diagramas de estados y diagramas de actividad y clases coloreadas para modelar la presentación.

Pero no son las únicas aportaciones relevantes en ese sentido, desde que (Vanderdonckt et al, 1993) presentaron la distinción entre *objetos de interacción abstracta* (AIOs) y *objetos de interacción concreta* (CIOs), primero, y con la disponibilidad y extensión de los lenguajes de marcado; HTML o fundamentalmente XML, muchas han sido las notaciones que se han propuesto y que se están proponiendo en la actualidad para modelar la interfaz de usuario.

El lenguaje XML *(Extensible Markup Language*) surgió por la necesidad de crear un lenguaje para la Web compatible con todos los navegadores actuales, ya que el lenguaje HTML con el paso del tiempo, y debido a su continuo parcheo, presentaba problemas de compatibilidad entre navegadores. El problema que existe en la actualidad, relacionado con el desarrollo de interfaces de usuario es similar, son muchos los dispositivos disponibles y cada uno tiene su lenguaje específico. La incompatibilidad o la

no disponibilidad de un lenguaje común que sirva para múltiples dispositivos hace que se hayan sucedido diferentes propuestas para dar solución a este problema. En este sentido, hay propuestas que han apostado por extender UML, otras por utilizar lenguajes de marcado y algunas más que no se incluyen en ninguno de los grupos anteriores, todas ellas, las que se consideran más significativas, serán presentadas seguidamente.

2.4.6.1 Extendiendo UML

Ante la mencionada limitación de UML para el tratamiento de interfaces de usuario, se han desarrollado iniciativas que tratan de cubrir la imposibilidad de representar interfaces de usuario y el comportamiento de los mismos desde los propios mecanismos que UML proporciona para su extensión. Ejemplos de dichas iniciativas son **Wisdom** (Nunes et al., 2000) y **UML*i*** (Silva, 2000).

Wisdom (Nunes et al, 2000) extiende UML mediante el uso de perfiles. Para ello aporta, entre otros, los siguientes estereotipos destinados a tratar el modelo de presentación: *Interaction space*, *navigate*, *contains*, *input element*, *ouput element* y *action* (véase Tabla 2-5), entre otras aportaciones.

Tabla 2-5. Perfiles a UML en Wisdom

Perfiles	Descripción
interaction space	Representa el espacio en el que el usuario interactúa con las funciones, containers e información necesaria para llevar a cabo una tarea o conjunto de tareas.
navigate	Representa una asociación entre espacios de interacción entre los que se mueve el usuario
contains	Representa una asociación entre dos espacios de interacción en el que uno de ellos contiene al otro
input element	Representa un atributo que denota información recibida del usuario
output element	Representa un atributo que denota información puede percibir aunque no manipular
action	Denota algo que el usuario puede hacer sobre el sistema a través de la interfaz y que afecta al estado interno del mismo

El objetivo de UML*i* es tratar de resolver el problema de diseñar e implementar interfaces de usuario sobre dos pilares UML y MB-UIDEs, eliminando el desnivel existente entre funcionalidad y diseño de la interfaz de usuario. UML*i* facilita una propuesta integradora con UML proponiendo un diagrama de interfaz de usuario, que contempla diferentes elementos: *FreeContainers, Containers, Editors, Displayers, Inputters* y *Actioninvokers* (véase Tabla 6), junto con ellos presenta una serie de estereotipos para

extender la interacción de flujo entre objetos (*presents*, *interacts*, *cancels* y *activates*) y estados estereotipados con los que dar respuesta directa a determinados comportamientos (*OrdenIndependeceState*, *OptionalState*, *RepeatableState*) en los diagramas de actividad.

Tabla 2-6. Objetos de interacción abstractos propuestos en UMLi (Silva, 2000)

Símbolo	Definición
	*FreeContainer*. Cada diagrama de interfaz está constituido de un FreeContainer constituido por otros elementos
	*Container*. Puede contener otros Containers, así como Inputters, Displayers, Editors y ActionInvokers
	*Inputter*. Responsables de la recepción de información de los usuarios
	*Displayer*. Responsables del envío de información a los usuarios
	Editor. Actúan simultáneamente como Inputter y Displayer
	*ActionInvoker*. Son responsables de la recepción de información de los usuarios en forma de eventos.

La principal limitación de estas propuestas está en la escalabilidad de su uso, parecen apropiadas para proyectos de complejidad media-baja. Además, y de forma más específica con esta tesis doctoral la experiencia no está contemplada y la calidad sólo lo está en la propuesta Wisdom.

2.4.6.2 Utilizando XML

El Lenguaje Extensible de Marcas, abreviado XML, describe una clase de objetos de datos denominados documentos XML y junto con ella también describe parcialmente el comportamiento de los programas que los procesan. XML es un perfil de aplicación o una forma restringida de SGML, el Lenguaje Estándar Generalizado de Marcado (ISO 8879).

XML fue desarrollado por un Grupo de Trabajo XML (originalmente conocido como *SGML Editorial Review Board*) formado bajo los auspicios del Consorcio World Wide Web (W3C), en 1996.

XML no es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. En este sentido, XML es un *metalenguaje* que establece la sintaxis utilizada para la elaboración de otros lenguajes de etiquetas estructurados. En XML se usan hojas de estilos, como el lenguaje XSL (*Extensible Stylesheet Language*, lenguaje de hojas de estilos extensible) y CSS (*Cascading Style Sheets*, hojas de estilos en cascada), para mostrar los datos en un explorador. En XML se separan los datos de la presentación y el proceso, de forma que es

posible mostrar y procesar los datos según se desee, aplicando distintas aplicaciones y hojas de estilos.

Fruto de las características ofrecidas por XML, y bajo su amparo, han surgido muchas propuestas para su utilización en la descripción y generación de interfaces de usuario. Recogeremos, seguidamente, diferentes propuestas atendiendo a esos dos fines: por un lado aquellas destinadas a cubrir la mera descripción de interfaces de usuario, es el caso de UIML, XUL, XAML, Xforms, o AUIML, y aquellas desarrolladas para dar soporte al desarrollo basado en modelos y, que por lo tanto, no se limitan a describir la interfaz sino que facilitan mecanismos para describir, dependiendo de los casos, al usuario, sus tareas, su dominio, etc, es el caso de XIML, UsiXML o Teresa-XML

UIML (*User Interface Markup Language*) (Abrams, 1999) es un lenguaje XML de interfaz de usuario independiente del dispositivo. Mientras que el lenguaje propuesto es ostensiblemente independiente del dispositivo específico y del medio utilizado en la presentación, no parece tener en cuenta el trabajo de investigación llevado a cabo en la última década sobre enfoques basados en modelos para interfaces de usuario. 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.

El consorcio W3C publicó el estándar (**XForms**) que presenta una descripción de la arquitectura, conceptos, modelo de procesamiento y la terminología subyacente a los formularios Web de la próxima generación. Está basado en la separación del contenido y de la presentación del formulario. Una vez más, esto muestra la importancia de separar el diseño conceptual de la presentación concreta, pero también resalta la necesidad de modelos significativos para soportar ambas aproximaciones.

AUIML (Abstract User Interface Markup Language) (Azevedo et al, 2000) es un lenguaje declarativo basado en HTML donde se separa presentación de semántica de interacción y se centra en ésta última. Ha sido utilizado con OVID y Wisdom para proporcionar la especificación abstracta de una interfaz de usuario, aunque en ambos casos la asociación se ha realizado manualmente.

XUL (Lenguaje Extensible para la Interfaz de Usuario) (Ginda et al, 2000) es la tecnología de interfaz multiplataforma de usuario 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 en 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

GUIs basadas en XML, muchas de ellas basadas en XUL, ejemplos de ellas son: Luxor XUL, XWT, Thinlets o SwingML. XUL se utiliza en IDEAS (Lozano, 2001) para describir el aspecto final de la interfaz que sería creada por aplicación de la metodología.

XAML (*Transaction Authority Markup Language*), propuesto por Microsoft y al igual que XUL, es un lenguaje declarativo que toma la sintaxis de XML y define las interfaces con etiquetas y no con líneas de programación. XAML hace posible crear la interfaz gráfica de nuestra aplicación sin importar si ésta será visualizada dentro de un navegador o de una ventana de escritorio de forma similar a sus predecesores UIML o XUL principalmente.

El segundo grupo de propuestas está integrado por lenguajes de especificación de interfaces de usuario, asociados con entornos de desarrollo basados en modelos. Éstos no se limitan a proporcionar facilidades para describir una interfaz de usuario sino que añaden mecanismos para integrar, junto con ellos, conceptos relacionados con otras consideraciones adicionales, que no se limitan a contemplar la interfaz de usuario: XIML, UsiXML o TeresaXML son algunos ejemplos notables de estos lenguajes de especificación, y serán recogidos seguidamente.

XIML (*eXtensible Interface Markup Language*) (Puerta, 2002) 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 (Puerta, 1996). El objetivo de este lenguaje es describir la interfaz de usuario de forma abstracta y, posteriormente, obtener una especificación concreta a partir de ella. Utilizando un desarrollo basado en modelos, asociado a XIML, se define un modelo de interfaz mediante el uso de cinco modelos adicionales ligados a las tareas, dominio, usuario, diálogo y presentación, junto con el tratamiento de las asociaciones entre modelos, consideradas sólo en sentido descendente.

En (Paternò et al, 2003) se ha propuesto un lenguaje basado en XML, **TeresaXML** (Fig. 2-10), para la especificación de interfaces que, además, enlaza con su propuesta anterior para el modelado de tareas CTT (Paternò, 1999). Este lenguaje especifica cómo se organizan varios AIOs que componen una interfaz de usuario, junto con la especificación del diálogo de la propia interfaz.

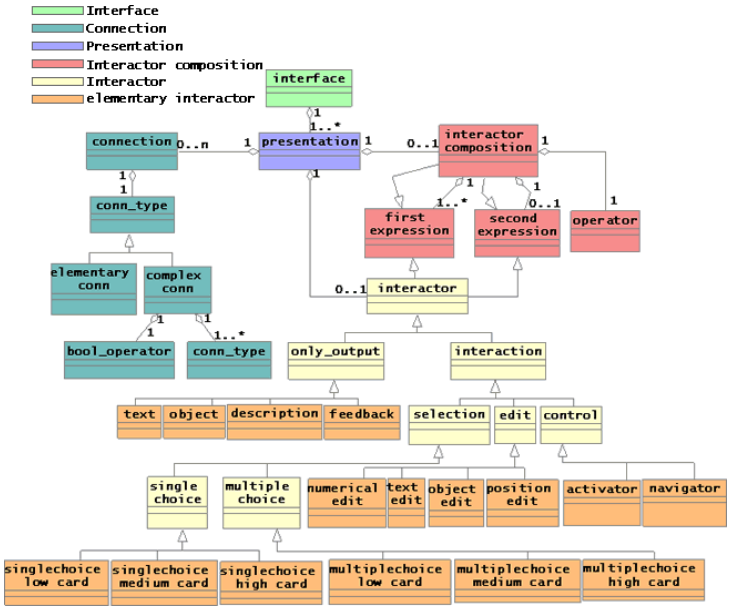


Figura 2-10. Lenguaje de especificación de IU TeresaXML (Paternò et al., 2003)

UsiXML (User Interface eXtensible Markup Language) (Limbourg et al., 2004) 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, y tiene como referente el nivel de abstracción. También se proponen mecanismos para gestionar las asociaciones que se establecen entre los diferentes modelos contemplados.

2.4.6.3 Otras propuestas

No sólo UML o XML han sido utilizados para la realización de labores de especificación de interfaces de usuario, otras propuestas también han sido presentadas, seguidamente se recogen algunas de ellas. Se citará también en este apartado alguna contribución proveniente del ámbito del desarrollo web y, específicamente, destinada a considerar aspectos relacionados con la interfaz de usuario.

En (Hussey et al, 1999) se identifica la necesidad de aumentar el nivel de abstracción utilizado en la especificación de interfaces de usuario, cuando se trabaja bajo cualquiera de los entornos de desarrollo basado en modelos disponibles. El problema que se presenta al trabajar a un nivel de abstracción bajo está relacionado con que de esta forma muchos diseños consideran prematuramente la apariencia visual concreta que ofrecerá la

interfaz. Para resolver este problema se propone utilizar un lenguaje formal de especificación orientado a objetos como es Object-Z (Duke, 1995), pero su propuesta tiene la limitación de la imposibilidad de automatizar el diseño.

Tabla 2-7. Prototipos Canónicos Abstractos (Constantine et al, 2003)

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	action/operation*	Print symbol table, Color selected shape
	start/go/to	Begin consistency check, Confirm purchase
	stop/end/complete	Finish inspection session, Interrupt test
	select	Group member picker, Object selector
	create	New customer, Blank slide
	delete, erase	Break connection line, Clear form
	modify	Change shipping address, Edit client details
	move	Put into address list, Move up/down
	duplicate	Copy address, Duplicate slide
	perform (& return)	Object formatting, Set print layout
	toggle	Bold on/off, Encrypted mode
	view	Show file details, Switch to summary
SYMBOL	**INTERACTIVE FUNCTION**	**EXAMPLES**
	container*	Configuration holder, Employee history
	element	Customer ID, Product thumbnail image
	collection	Personal addresses, Electrical Components
	notification	Email delivery failure, Controller status
SYMBOL	**INTERACTIVE FUNCTION**	**EXAMPLES**
	active material*	Expandable thumbnail, Resizable chart
	input/accepter	Accept search terms, User name entry
	editable element	Patient name, Next appointment date
	editable collection	Patient details, Text object properties
	selectable collection	Performance choices, Font selection
	selectable action set	Go to page, Zoom scale selection
	selectable view set	Choose patient document, Set display mode

En (Constantine et al, 2003) también se ha resaltado la necesidad de disponer de facilidades para el diseño de interfaces de usuario a un nivel de abstracción superior y, en esta dirección, se ha presentado una colección denominada *Canonical Abstract Prototypes* (Tabla 2-7). Con esta propuesta Constantine trata de introducir en la descripción de la presentación lo que propuso años antes en el terreno del modelado de tareas, con los casos de uso esenciales, es decir, permitir una descripción de la interfaz a tal nivel de abstracción que pueda ser más tarde implementada en múltiples entornos, posibilitando la *plasticidad*[1] de las interfaces modeladas (Thevenin et al, 1999). Cada componente abstracto canónico tiene una función interactiva, que viene representada por su símbolo, y una descripción textual. Los símbolos gráficos sirven para facilitar el manejo de dichas funciones. La notación se construye haciendo uso de dos símbolos universales: una

[1] dicese de aquella interfaz de usuario que cambia en función del dispositivo

herramienta genérica o acción y un material o container. Esta propuesta se completa con casos de uso, con la puesta en práctica de un ciclo iterativo y con actividades de evaluación de las interfaces generadas, no en vano está integrada dentro de su metodología de desarrollo centrado en el uso.

En el ámbito Web, y relacionado con el modelado hipermedial y la interfaz de usuario, la propuesta **OO-H** (Cachero, 2003) recoge dos dimensiones relacionadas con la navegación y la presentación de interfaces web mediante la consideración de dos diagramas: el *Diagrama de Acceso Navegacional* (DAN) y el *Diagrama de Presentación Abstracta* (DPA).

El DAN toma como base los requisitos de navegación de cada tipo de usuario del sistema. Para cada usuario, el DAN importa un diagrama de clases (extraído a partir del diagrama de clases de la aplicación) donde se refleja exclusivamente la organización de la información manejada por ese tipo de usuario. El DAN también proporciona los constructores necesarios para enriquecer dicho diagrama con los modos de acceso y navegación necesarios para cubrir sus necesidades de navegación a través de la información. A partir de él, y aplicando una serie de reglas de transformación, es posible generar un DPA por defecto.

El DPA se define como una estructura de plantillas especificadas en XML. Tanto la estructura del diagrama como las plantillas individuales pueden ser refinadas por el diseñador para conseguir los rasgos de interfaz deseados.

2.4.7 La clave: el *mapping*

El problema del *mapping* (Puerta, 1999; Limbourg et al., 2004) consiste en la identificación, establecimiento manual o derivación automática de relaciones entre los modelos recogidos en un entorno de desarrollo basado en modelos. En este sentido, son de singular importancia las relaciones que se establecen entre los modelos independientes de la plataforma (usuario, tarea, dominio y presentación abstracta), y los dependientes de la misma (presentación concreta y diálogo). Según (Olsen, 1993) el conjunto de relaciones de *mapping* da lugar a una serie de transformaciones con las que es posible aspirar al desarrollo automático de interfaces de usuario.

Las relaciones entre modelos han sido consideradas desde el mismo momento en el que los desarrollos basados en modelos aparecieron en escena. Hay modelos que se han demostrado *primarios*, si extrapolamos la denominación utilizada para hacer referencia a determinados colores a partir de cuya mezcla pueden obtenerse otros, de forma similar el modelo de dominio, primero, y, conforme ha ido transcurriendo la década de los noventa, el de tareas, después, tienen un papel predominante en la puesta en

práctica de un MB-UIDE. De esta forma, hay herramientas MB-UIDE que se basan fundamentalmente en la identificación de objetos y en la manipulación que sobre ellos puede realizarse: UIDE (Foley, 1991), Humanoid (Szekely, 1992) o Janus (Balzert, 1996), y otras que parten, casi en exclusiva, de la elaboración de un modelo de tareas, como Trident (Bodart et al., 1995) o Adept (Jhonson et al., 1993) entre otras.

En cualquier caso modelo de dominio y modelo de tareas son modelos que comparten protagonismo a la hora de poner en práctica un desarrollo basado en modelos, y así existen propuestas como ITS (Wiecha et al., 1989), Diane+ (Tarby et al., 1996), Mastermind (Neches et al., 1993), Mobi-D (Puerta et al, 1997), Mecano (Puerta et al., 1996), Fuse (Lonczewski et al., 1996) o Teallach (Griffiths et al, 2001). Y, Junto con ellas, también están aquellas propuestas desarrolladas entorno a UML como OVID (Roberts et al., 1997), Wisdom (Nunes et al., 2000), UMLi (da Silva, 2001) o IDEAS (Lozano, 2001), en las que también se sigue esa misma idea de identificar las tareas mediante diagramas de casos de uso, la parte estática mediante diagramas de clases, la parte dinámica mediante los diagramas de comportamiento facilitados en UML y enriquecer UML con aquello que se considere necesario.

Las relaciones entre modelos facilitan posibles caminos para abordar la generación automática de interfaces de usuario, permitiendo aumentar la productividad en el proceso de desarrollo desde el punto de vista de la IS, y la generación rápida de prototipos de interfaz de usuario desde una visión IPO.

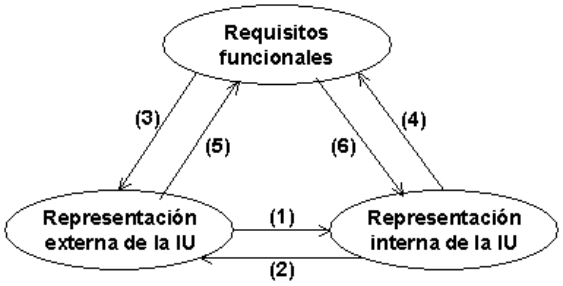


Figura 2-11. Rutas posibles en la generación de interfaces de usuario

En la Fig. 2-11, inspirada en (Vanderdonckt, 1996), se recogen las asociaciones entre diferentes conjuntos de modelos encargados de definir los requisitos funcionales y la presentación interna y externa de la interfaz de usuario. Las trayectorias descendentes se consideran habitualmente en el proceso de desarrollo, así se pasa de los requisitos funcionales a la representación interna y, finalmente, a la representación externa. Por el contrario, el proceso complementario al anterior, que pasa por la representación

externa, la representación interna y los requisitos funcionales, que resulta ascendente en el nivel de abstracción y para el que no hay propuestas significativas de mapeo, caracteriza un proceso de evaluación de la interfaz, de trazabilidad de requisitos y de calidad en uso de un producto software (ISO 9126-4, 2003).

Algunos de los lenguajes de especificación de interfaces de usuario, presentados en el apartado anterior, proporcionan facilidades para definir *mapping*, por ejemplo, en UIML donde a través del uso de un tag *peer* es posible establecer conexiones entre clases de objetos abstractos y su correspondiente componente de interacción específico de la plataforma destino donde será presentado, definiendo así en qué dispositivo será desplegada la interfaz (HTML, Java, VoiceXML, WML, etc.). También, a los componentes de XUL se les puede asociar comportamiento haciendo uso de JavaScript o de aplicaciones más complejas elaboradas con lenguajes de programación de alto nivel. Pero tanto UIML como XUL tienen un ámbito restringido de aplicación ya que no consideran mecanismos con los que tener en cuenta tareas, dominio de aplicación o contexto de uso, limitándose en exclusiva a permitir descripciones de interfaces de usuario. Un estudio más exhaustivo del *mapping* entre modelos se antoja necesario y se considera seguidamente.

Considerando los modelos más típicos que surgen al poner en práctica un entorno de desarrollo basado en modelos para la elaboración de interfaces de usuario, los *mappings* que pueden establecerse entre ellos pasan por:

Entre el modelo de dominio y el de tareas. La propuestas más representativa en este sentido ha pasado por la relación existente entre tareas y dominio reflejada con la notación TKS (Johnson et al, 1989) utilizada, por ejemplo en Adept (Johnson et al, 1993), y ya mencionada con anterioridad. En ella junto con la especificación de cada tarea y de forma textual, se hace referencia a los objetos que, asociados con su realización, se manipulan.

Entre el modelo de tareas o dominio y el de presentación pueden identificarse relaciones y se han tratado de forma diferente, por ejemplo, en Trident (Bodart et al, 1995) se utilizan unidades de presentación definidas a partir de un grafo ACG, utilizado para especificar el modelo de tareas y se proponen estrategias para asociar ventanas a esas unidades de presentación. En (Eisenstein et al., 1998) y (Puerta et al., 1999), trabajos relacionados con Mobi-D, se presentó *Task-Interface Model Mapper* (TIMM), un sistema capaz de asistir al diseñador en la labor de seleccionar un elemento de interacción entre un conjunto posible, a partir de información relaciona-

da con el modelo de dominio. La asistencia se proporciona en base a características y comportamiento que presenta el objeto del dominio. En Genius (Janssen et al., 1993) se asigna automáticamente una ventana a cada vista definida en el modelo de diálogo y a cada atributo de las entidades relacionadas con la vista se les asigna una objeto de interacción abstracto. En Janus (Balzert et al., 1995) y en Mecano (Puerta, 1997) se asigna una ventana a cada clase no abstracta definida en el modelo orientado a objetos elaborado para realizar la especificación conceptual del sistema de la que se parte en esa propuesta. Junto con ello, las estructuras de navegación entre ventanas se generan utilizando las relaciones entre esas ventanas generadas a partir del modelo OOA. En OVID se asocian sus modelos con el lenguaje de especificación de interfaces de usuario AUIML, a través de las facilidades de *script* proporcionadas en Racional Rose (Azevedo et al, 2000). En Teallach (Griffiths et al., 2001) se presentó una idea similar llegando un poco más lejos, ya que la selección se realizaba de forma automática y no manual como era en el caso anterior, por el contrario la solución recogida anteriormente era más versátil. (Stirewalt, 1999) presentó un lenguaje, MDL, asociado con la propuesta Mastermind (Szekely et al, 1996) mediante dicho lenguaje se podían definir asociaciones entre los elementos de tres modelos: tareas, dominio y presentación, permitiendo realizar las asociaciones a diferentes niveles de granularidad. Con la herramienta Vista (Brown et al., 1998) se consideran las relaciones entre modelos y cómo estas relaciones se ven afectadas por actividades de diseño entre grupos de diseñadores de diferentes especialidades. En ella se facilitan mecanismos para lograr la comunicación entre un equipo de diseño para elaborar una propuesta de asociación entre elementos del dominio, tareas y presentación.

En las propuestas recogidas anteriormente los modelos están definidos y las relaciones entre ellos se identifican posteriormente. Existen otros trabajos en los que los *mappings* se derivan o infieren, es decir, a partir de la identificación de relaciones entre modelos, hay modelos que se obtienen total o parcialmente a partir de la elaboración de otros modelos, especialmente desde los modelos de dominio y desde los modelos de tareas del usuario.

En la dirección anterior, y utilizando el concepto de patrón, (Molina, 2003) presentó Just-UI, nombre con el que se denota un lenguaje de patrones que consta de dieciséis patrones y a partir del cual puede inferirse parte o la totalidad del modelo de presentación. Molina definió un *proceso de inferencia* que se realiza sobre el modelo de presentación y que consiste en derivar la información de interfaz de usuario que falte, permitiendo prototipar rápidamente sin la necesidad de especificar por completo el modelo de presentación. En el caso extremo, se posibilita la creación de un proto-

tipo sin modelo de presentación de partida y, por lo tanto, infiriéndolo por completo. Sobre las posibilidades ofrecidas por los patrones, en esta y otras labores relacionadas, se abundará en el capítulo cuarto de esta tesis doctoral, valga por ahora comentar que el modelo propuesto en (Molina, 2003) se estructura en tres niveles:

- *Árboles de jerarquía de acciones*, en los que se recoge cómo la funcionalidad será presentada al usuario que acceda al sistema,
- *Unidades de interacción*, aquellas unidades con las que el usuario llevará a cabo las tareas, pudiendo lanzar servicios (unidad de servicio), visualizar el estado de una instancia (unidad de instancia), visualizar conjuntos de instancias (unidad de población) o modelar unidades de interacción más complejas (unidades de interacción maestro-detalle),
- y *Patrones elementales*, útiles para complementar la realización de las tareas que el usuario puede realizar sobre la interfaz de usuario generada haciendo uso de las unidades de interacción anteriores.

A partir de la experiencia recopilada en los patrones reseñados anteriormente es posible dar soporte a la elaboración de prototipos de interfaces de usuario partiendo de la especificación conceptual del dominio de una aplicación.

Otros trabajos recientes están considerando la posibilidad de derivar elementos de presentación de la información contenida en otros modelos, especialmente del de tareas, así (Limbourg et al., 2004) y (Mori et al., 2004) parten del modelo de tareas, y elaboran una serie de transformaciones basadas en grafos ACG y/o en la notación CTT, para dar lugar a presentaciones que son concretas, en el caso de Limbourg, y abstractas y basadas en el lenguaje de especificación de interfaces de usuario (concretamente en XAUI), las de Mori. Finalmente, (Florins et al., 2004) trabajan en el desarrollo de heurísticas que permitan obtener a partir de los modelos de tareas expresados en CTT agrupaciones de las mismas que traten de dar soporte al diseño de interfaces de usuario multiplataforma.

Entre el modelo de presentación y el de tareas o dominio existen relaciones similares a las que hemos visto en el párrafo anterior en sentido inverso, pero han sido poco explotadas. Teallach (Griffiths et al., 2001) contempla la posibilidad de reunir presentación, tareas y dominio con ello se crea una instancia de presentación con la que es posible realizar una tarea concreta.

2.4.8 Herramientas

En esta sección se recoge información de las herramientas desarrolladas, y en desarrollo, que están relacionadas con los conceptos previos tratados a lo largo de todo este capítulo. Muchas de las herramientas, aquellas asociadas con la puesta en práctica de desarrollos basados en modelos ya, han sido citadas en alguna ocasión a lo largo del capítulo. En este apartado volverán a ser recogidas, al menos las que se consideran más representativas, cuando se comenten las herramientas destinadas a dar soporte a la especificación de modelos, ya que muchas de ellas disponen de editores, interpretes, generadores y otras aplicaciones útiles para llevar a cabo otro tipo de tareas relacionadas con la elaboración, manipulación o generación de modelos.

También tendrán cabida en este apartado herramientas de IS que por afinidad de intereses, y dar soporte a la realización de labores de modelado utilizando los diagramas asociados con UML, pueden ser usadas para abordar temas relacionados con el desarrollo o modelado de interfaces de usuario. A las herramientas de prototipado rápido de interfaces de usuario, a las de mapeo entre modelos o ingeniería inversa también se les prestará atención en estos últimos apartados de este capítulo. En definitiva, en este apartado tendrán cabida las herramientas relacionadas con:

- **Entornos de desarrollo basados en modelos** para la generación de interfaces de usuario,
- **Entornos de desarrollo basados en UML**,
- **Herramientas de prototipado** rápido de interfaces,
- **Otras herramientas** reseñables.

2.4.8.1 Desarrollo de interfaces de usuario basado en modelos

Los MB-UIDEs se han sucedido históricamente desde finales de los ochenta hasta nuestros días. La pervivencia de los mismos ha venido sustentada desde la IS y desde la IPO gracias a que aseguran productividad en el desarrollo de interfaces de usuario y posibilitan la puesta en práctica de metodologías centradas en el usuario.

La idea en todos los entornos propuestos es la misma: elaboración de modelos conceptuales en los que se recogen las características estáticas y dinámicas de una aplicación, incluida la parte de la interfaz de usuario, e interpretación, compilación o transformación de dichos modelos hasta obtener una interfaz de usuario que esté ligada, o que pueda ligarse, con la funcionalidad de la aplicación.

Los MB-UIDEs han pasado por diferentes generaciones de herramientas, pueden encontrarse estudios relacionados con este tipo de entornos de

desarrollo en (Szekely, 1996; Schlungbaum, 1996; da Silva, 2000). Partiendo de dichos estudios se identifican diferentes clasificaciones de los entornos de desarrollo, en los mismos se atiende a los objetivos perseguidos, al número de modelos empleados, a la notación utilizada para crear los diferentes modelos que constituyen el modelo de interfaz, a lo representativo que resulten dichos modelos o al resultado obtenido fruto de su utilización (código fuente en un lenguaje de programación de alto nivel, por ejemplo C++; modelos interpretables en tiempo de ejecución; o descripciones manipulables por un sistema gestor de interfaces de usuario).

Por ejemplo, existe un grupo de entornos de desarrollo basado en modelos que posee herramientas, entre las que se encuentran UIDE (de Baar et al., 1992), Mecano (Puerta, 1996), AME (Martin, 1996) o Janus (Balzert, 1996), en las que el principal interés es obtener automáticamente la interfaz de usuario, la misma se obtiene a partir de un modelo declarativo de interfaz elaborada a partir de muy diversas notaciones. En el caso de las herramientas reseñadas, y respectivamente, se ha utilizado una notación propia, un lenguaje de especificación denominado MIMIC (Puerta, 1997) que posteriormente evolucionó a XIML (Puerta, 2001), una notación orientada a objetos representada a través de la definición de clases ASO (*Application System Object*), o un lenguaje de definición obtenido como extensión de otras notaciones como CORBA IDL (Corba, 1992) y ODMG (Cattell, 1994).

Otro grupo de entornos de desarrollo de interfaces de usuario están interesados más por el proceso de generación que por la mera obtención de una interfaz de usuario: Mastermind (Neches et al, 1993), Mobi-D (Puerta et al., 1997), Fuse (Lozcweski, 1996), Trident (Vanderdonckt et al., 1993) o Tadeus (Schlungbaum, 1996) son algunos ejemplos de ellos. En estos entornos el modelo de interfaz está constituido de un mayor número de modelos declarativos.

Las notaciones utilizadas para especificar los diferentes modelos utilizados para caracterizar los modelos asociados con el modelo de interfaz han sido, en la mayoría de las ocasiones, las proporcionadas por la Ingeniería del Software. Así, a principios de los noventa era común hacer uso de notaciones propias (UIDE, ITS) y modelos entidad-relación (Trident, Genius), pero conforme fueron apareciendo notaciones relacionadas con el análisis y diseño orientado a objetos (OOA/OOD, OSE, OMT, etc.), y extendiéndose su uso, también se utilizaron en el desarrollo de interfaces de usuario, ejemplo de ello lo constituyen herramientas como AME, Humanoid, Tadeus, Janus, Teallach, etc.).

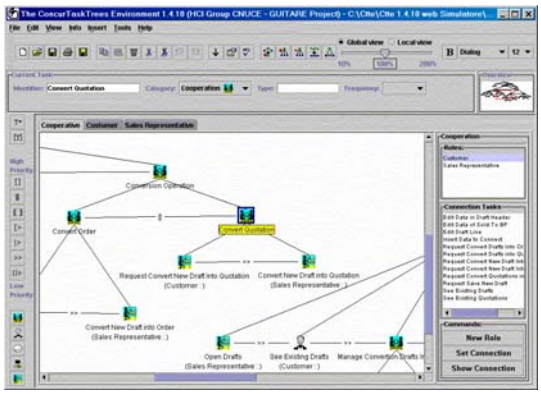


Figura 2-12. Editor ConcurTaskTrees (Paternò, 1999)

La notación ConcurTaskTrees (Paternò, 1999), perfectamente compatible con UML (Paternò, 2001), está muy extendida en IPO y se utiliza para la elaboración de modelos de tareas de usuario. La notación CTT es gráfica y para facilitar su utilización se ha desarrollado un entorno denominado editor ConcurTaskTrees (Mori et al., 2002) (véase Fig. 2-12) a través del cual se tiene la posibilidad de abordar otros aspectos relacionados con la interacción a través de una interfaz de usuario, como son la evaluación o generación automática de interfaces de usuario. La evaluación de interfaces de usuario se aborda con herramientas como RemUsine (Paternò et al., 1999), posteriormente WebRemUsine (Paganelli et al., 2002) donde se extiende la idea original a interfaces desarrolladas para la Web, con la que es posible realizar evaluaciones de usabilidad comparando ficheros de registro de comportamiento de usuarios al interactuar con la interfaz y los modelos de tareas especificados utilizando la notación CTT y almacenados usando XML. La generación de interfaces de usuario se realiza partiendo de la especificación de modelos de tareas con CTT y con la especificación abstracta de la interfaz utilizando la notación de interfaces de usuario, que también han propuesto en el mismo grupo de investigación, todo ello está soportado mediante TERESA (Mori et al., 2004).

La tendencia actual en el desarrollo de software, y las interfaces de usuario no son una excepción, es apostar por un estándar internacional como es el Lenguaje Unificado de Modelado (UML). En este mismo capítulo se han reseñado iniciativas para la especificación y desarrollo de interfaces de usuario en ese sentido: OVID, Wisdom, UML*i*, e IDEAS son ejemplos de ellas, y en todas ellas se cuenta con el soporte software necesario para ponerlas en práctica.

2.4.8.2 Entornos de desarrollo basados en UML

Si la utilización de UML, aún teniendo limitaciones para la especificación de interfaces de usuario, está plenamente extendida en el desarrollo de software, tendrán cabida en este apartado aquellos entornos de desarrollo que permitan realizar especificaciones haciendo uso de dicha notación, utilizando cualquiera de sus versiones disponibles.

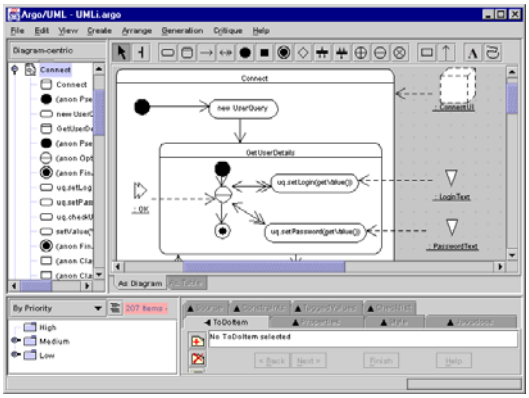


Figura 2-13. Argo*i* un entorno para dar soporte a UML*i* (Silva, 2000)

Podemos destacar como herramientas de soporte al desarrollo basado en UML y que han sido utilizadas para dar soporte a la especificación de modelos: Rational Rose (IBM), ArgoUML (http://argouml.tigris.org/), Poseidon for UML (http://www.gentleware.com/), EclipseUML (http://www.omondo.com/), Together (http://www.borland.com/together/) o System Architect (http://www.popkin.com/). Los diagramas más conocidos y utilizados de UML pueden especificarse haciendo uso de dichos entornos, y, concretamente para UML se ha desarrollado un plugin que permite la utilización de los perfiles y diagramas de actividad modificados especialmente para facilitar el diseño de interfaces de usuario utilizando UML*i* (da Silva, 2002) (véase Fig. 2-13).

En la misma dirección, es decir basándose en la elaboración de modelos, aunque no haciendo uso de UML sino de OASIS (Pastor et al, 1995), OlivaNova (Pastor et al, 2003) es una herramienta concebida para la generación de sistemas software comerciales.

También puede recogerse en este mismo grupo de herramientas aquellas destinadas a dar soporte a MDA como ArcStyler (http://www.io-software.com/) u OptimalJ (http://www.compuware.com/).

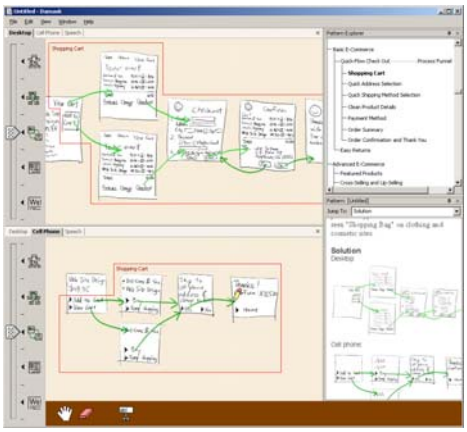


Figura 2-14. Ventana principal de la herramienta Damask (Lin et al, 2002)

2.4.8.3 Herramientas de prototipado de interfaces de usuario

Dentro del grupo de herramientas destinadas al diseño de interfaces que proporcionan facilidades para el prototipado podemos destacar las herramientas Denim (Newman et al, 2003), Suede (Sinha et al, 2002) y Damask (Lin, 2003) (Fig. 2-14) realizadas en el grupo de investigación del profesor James Landay en Berkeley. Son herramientas todas ellas que permiten realizar prototipos informales de interfaces de usuario. La primera de ellas está orientada al desarrollo de bocetos de interfaces de usuario para la Web, la segunda basa el prototipado en sentencias ofrecidas por el diseñador de viva voz, y la última referencia, Suede, introduce y utiliza el concepto de patrón para conseguir dos propósitos, por un lado, para dar soporte a la asistencia al diseñador en labores de prototipado y, por otro, para aprovechar la documentación de la experiencia basada en patrones y conseguir facilidades de prototipado independiente de la plataforma. En todas las herramientas mencionadas se utiliza como notación para especificar la interfaz de usuario una notación propia basada en bocetos que simbolizan los elementos representativos que surgen en la interacción web, acompañados de una descripción que se asemeja a la utilizada habitualmente al describir un patrón. La colección de patrones que considera está recogida en (van Duyne, 2002).

CanonSketch (Campos et al, 2004) es otra herramienta (véase Fig. 2-15) que permite la especificación de interfaces de usuario utilizando la notación canónica abstracta propuesta en (Constantine et al., 2003). Dicha herramienta cuenta con el respaldo de la metodología Wisdom (Nunes et al., 2000), y por lo tanto se basa en UML.

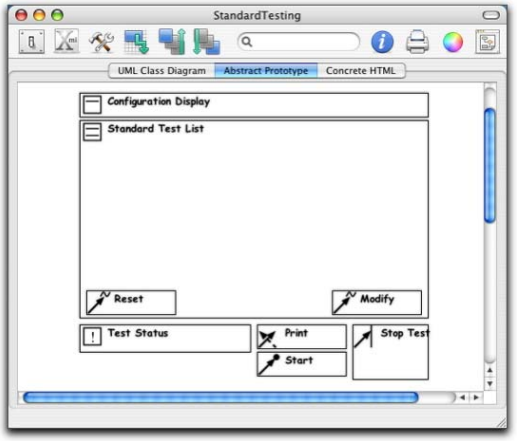


Figura 2-15. Herramienta Canonsketch (Campos et al., 2004)

La utilización de experiencia en forma de guías de estilo para el desarrollo de interfaces de usuario y su protipado o generación automática es una línea de investigación con diferentes trabajos dignos de ser mencionados y considerados. En esta dirección AskJef (Barber et al, 1992) y Guide (Henninger, et al, 1995) utilizan técnicas de IA para la manipulación y gestión de guías de estilo relacionadas con interfaces de usuario, concretamente razonamiento basado en casos (Kolodner, 1991). Otras como Sierra (Vanderdonckt, 1995), HyperSAM (Ianella, 1994) y EXPOSE (Gorny, 1995) proporcionan información de diseño en forma de guías de estilo que se organizan atendiendo a diferentes atributos ergonómicos de calidad.

Algunos de los entornos de desarrollo basados en modelos, tratados en apartados anteriores, se acompañan con conocimiento relacionado con el diseño de interfaces de usuario, este conocimiento llegado el momento facilita la selección de componentes de interacción concretos. La selección se basa en disponer conocimiento experto que ha sido formulado utilizando reglas, por ejemplo Genius (Jannsen et al., 1993), Trident (Vanderdonckt et al., 1993) con Seguia (Vanderdonckt, 1999) o Janus (Balzert, 1994) proporcionan esta facilidad.

Relacionado con la labor de recopilar la experiencia, que puede ser utilizada por diseñadores para la elaboración de sus propuestas de interfaz de usuario, cabe destacar, aunque tenga un ámbito de aplicación específicamente dirigido al diseño para la Web, MetroWeb (Mariage et al., 2000) (véase Fig. 2-16) y Moudil (Seffah, 2003), herramientas que se constituyen en repositorios de guías de estilo y patrones de interacción respectivamente.

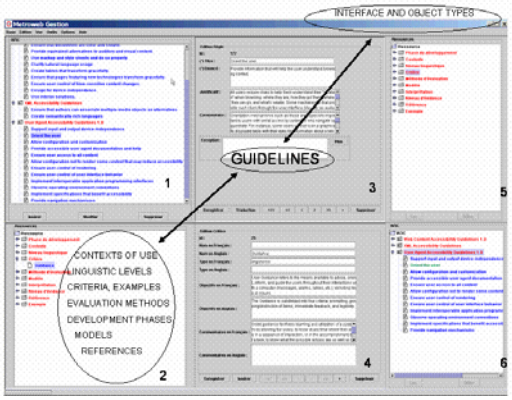


Figura 2-16. Ventana principal de MetroWeb (Mariage et al, 2000)

La limitación de las aportaciones anteriores es que se centran en proporcionar experiencia a nivel de diseño, pero a un nivel de abstracción muy bajo, utilizando guías de estilo que con frecuencia adolecen de períodos de vigencia y que dependen de la plataforma y del modo de interacción (Nielsen, 2004) en la mayoría de los casos. En el capítulo cuarto se retomará está línea de investigación, en la que se conjuga experiencia y diseño de interfaces, ya que estará destinado a considerar la reutilización de la experiencia en el desarrollo de interfaces de usuario de calidad.

2.4.8.4 Otras herramientas reseñables

Algunas herramientas relacionadas con el desarrollo de interfaces de usuario, y que por sus objetivos no encajan dentro de los grupos anteriormente identificados serán recogidas en esta última sección. En este sentido, U-TEL (Chung-Man et al., 1998) es una herramienta para la elicitación de modelos de tareas, construidos a partir de descripciones de actividad expresadas de forma textual, y manipuladas utilizando técnicas de procesamiento de palabras.

Vaquita (Bouillon et al., 2004) es una herramienta que permite desarrollar ingeniería inversa y en la que, a partir de una página web, es posible obtener su modelo de presentación asociado utilizando el lenguaje de descripción de interfaces XIML.

Con SUPPLE (Gajos et al., 2004) se propone una herramienta para la generación de interfaces de usuario de forma automática y adaptada a las características del dispositivo. En ella se parte de tres elementos: una especificación de la interfaz, un modelo del dispositivo donde se visualizará la interfaz y un modelo del usuario basado en trazas de uso, que minimiza una función de costo establecida y relacionada con el esfuerzo destinado por el usuario para hacer uso de las diferentes interfaces generadas. En esta

propuesta sólo se contempla un modelo de dominio, con el que se identifican los elementos que visualizará el usuario y a través de los que se llevará a cabo la interacción.

SUIDT (Baron et al., 2004), a diferencia de la anterior propuesta, es una herramienta ideada originariamente para la elaboración de interfaces de usuario partiendo de especificaciones de tareas utilizando CTT, y que puede utilizarse para identificar relaciones semánticas entre funcionalidad e interfaz permitiendo el prototipado de la interfaz junto con la funcionalidad ligada a la misma.

Otras herramientas recientes, y actualmente en desarrollo, como KnowiXML (Furtado et al., 2004), SketchiXML (Coyette et al., 2004) o UI Pilot (Puerta et al., 2005), desarrolladas sobre lenguajes de descripción de interfaces de usuario como son UsiXML o XIML, tienen como objetivo la elaboración de prototipos abstractos de interfaces de usuario. Otras como GrafiXML, asociado a UsiXML, trabaja exclusivamente describiendo interfaces de usuario a nivel de objetos de interacción concreto y permite su exportación a diferentes lenguajes como Java o XHTML.

2.5 Análisis y conclusiones

Examinando conjuntamente las definiciones correspondientes a Ingeniería del Software y a Interacción Persona-Ordenador:

> *Software engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building software systems that satisfy the requirements of users and customers.*
>
> *Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.*

y teniendo en cuenta las aportaciones realizadas desde uno y otro ámbito, recogidas en este capítulo, queda patente que la diferencia que se observa entre una y otra disciplina es el enfoque y el nivel al que se trabaja. La IS está centrada en el proceso de desarrollo de software y la IPO en que el producto final generado sea aceptado por el usuario de forma satisfactoria. Así, la primera se centra en factores de facilidad de mantenimiento, reutilización y productividad, dichos factores vienen determinados por criterios

internos de calidad del software como son el acoplamiento y la cohesión. La IPO, por el contrario, se centra en lo que hasta ahora hemos denominado, a falta de una discusión posterior realizada en el próximo capítulo, usabilidad. Este otro factor de calidad tiene un transfondo, fundamentalmente, externo y, además, altamente dependiente del usuario. Es por ello que los criterios últimos de los que depende la usabilidad tienden a estar intimamente relacionados con consideraciones ergonómicas y, cabe pensar, aunque de forma no completamente justificada, que no tienen nada que ver con consideraciones internas de desarrollo de software.

La IS en su proceso histórico hacia la industrialización ha pasado por diferentes etapas caracterizadas por una búsqueda de unificación de criterios, primero, y de estandarización después. Así, en los últimos 15 años se han sucedido desde el uso de técnicas estructuradas hasta la estandarización que se ha aportado con UML, adoptado como lenguaje de modelado. El aumento del nivel de abstracción a la hora de considerar una aplicación ha sido cada vez más creciente, primándose el diseño frente a la implementación, y más tarde la elicitación de requisitos frente al diseño. En ese mismo sentido, la propia programación ha cambiado pasando del uso de lenguajes de programación de alto nivel a la elaboración de especificaciones utilizando UML, que tras ser compiladas proporcionan el correspondiente código.

Las carencias más reseñables que se le achacan a UML desde el terreno de IPO, y que impiden su acogida plena por parte de esta otra disciplina, pasan por la carencia en la incorporación de técnicas de Diseño Centrado en el Usuario de forma paralela con la elaboración de un producto software. Debido a ello, la consideración del usuario, la especificación de la interfaz de usuario, la especificación de la navegación entre los elementos que constituyen la interfaz, la caracterización de la calidad desde el punto de vista del usuario de manera que pueda ser evaluada, la consideración de esta última calidad en fases tardías del desarrollo de cualquier producto software son consecuencia de esa carencia. Por otro lado, la falta de madurez, en unos casos, y de aplicación sistemática de las propuestas realizadas desde el terreno de la IPO, en otros, tampoco ayuda a su adopción desde la IS. En definitiva, IS e IPO han trabajado compartiendo lo indispensable y dando la espalda a importantes aportanciones que se han presentado desde una y otra.

En una y otra disciplina aparece la técnica de especificación de modelos para apoyar el desarrollo de productos software. En IS se ha apostado, definitivamente, por un desarrollo dirigido por modelos, bajo una visión arquitectónica y donde priman los factores internos (MDA). En IPO también se están abrazando las mismas técnicas, poniéndose en práctica desarrollos basados en modelos tomándolos de la IS, pero no existe una preocupación

común por integrar las propuestas elaboradas desde una y otra disciplina. Es decir, las técnicas de DCU aportadas desde la IPO, en los últimos 15 años, no se han integrado adecuadamente en las propuestas que se han presentado desde la IS, y en el debe de la IS cabe achacar una aparente *falta de atención* de los aspectos relacionados con el desarrollo y especificación de interfaces de usuario, que en muchos desarrollos se consideran de forma superflua o tardía.

Cabe preguntarse, ¿por qué se ha llegado a esta situación? y ¿qué se puede hacer para remediar este aparente conflicto de intereses cuando el objetivo último es el mismo: desarrollar productos software de calidad?.

Con el fin de dar respuesta a estas preguntas en capítulos posteriores se considerarán detenidamente dos elementos que consideramos esenciales para hacer compatibles las propuestas ofrecidas desde la IS y desde la IPO. Dichos elementos son: la experiencia y la calidad y rara vez se han considerado conjuntamente en las metodologías, lenguajes y herramientas presentados hasta ahora. De esta manera, en la Tabla 2-8 se resumen aquellas aportaciones que, habiendo aparecido a lo largo del presente capítulo, haya sido en los apartados relacionados más directamente con la IS o con los de IPO, consideramos relevantes. En esa misma tabla se presta una atención especial al tratamiento que se hace de los elementos identificados como elementos clave para dar respuesta a las preguntas formuladas con anterioridad. En función de la consideración que reciben tanto la calidad como la experiencia identificamos, seguidamente, las carencias que justifican haber llegado a la situación descrita y que motivaron las preguntas anteriores:

a. **el concepto de calidad es un término confuso**, sobre todo en lo que se refiere a la percepción que del mismo tiene el usuario final (usabilidad). Dicho concepto está poco contemplado, y cuando aparece el proceso de elaboración ya está avanzado y se dispone de una versión preliminar del producto software que se pretende desarrollar. Prototipado y evaluación de prototipos son las principales armas utilizadas, ya se esté en el ámbito de la IS, ya de propuestas presentadas desde el terreno de la IPO.

b. **hay una falta de integración de las técnicas de DCU** en las distintas fases que constituyen el ciclo de vida de un producto software. La IS considera la calidad, pero lo hace bien en la fase de requisitos, bien en la fase de prueba. Una integración de consideraciones de calidad a lo largo de las distintas fases que componen el ciclo de vida redundaría positivamente en la calidad de los productos software finalmente desarrollados,

Tabla 2-8. Tabla comparativa de diferentes propuestas recogidas en este capítulo

	Modelos	Notación	Traza.	Experiencia[1]	Usabilidad	Mod. Exp.	Referencia
Adept	T, U	TKS	×	$\surd_p$	√	×	(Johnson et al, 93)
Trident	T, Do	Enti/Rel.	×	$\surd_p$	×	×	(Vanderdonckt, 95)
FUSE	T, Do, U, Di	Algebraica	×	$\surd_p$	√	×	(Lonczewski , 96)
Janus	Do	OO	×	×	×	×	(Balzert et al., 96)
Mastermind	T, Do, P	Corba IDL	×	$\surd_p$	×	×	(Szekely et a., 96)
Mobi-D	T, Do, U, Di, P	MIMIC	×	$\surd_p$	√	×	(Puerta et al., 97)
Tadeus	T, Do, U, Di	OO	×	$\surd_p$	√	×	(Elwert et al., 95)
Teallach	T, Do, P	ODMG, UML*i*-	×	×	×	×	(Griffiths et al, 96)
OVID	UML	OO	×/√	$\surd_p$	√	×	(Bardon et al., 01)
Wisdom	UML+	OO	√	×	√	×	(Nunes et al., 01)
UML*i*	UML+	OO	√	×	×	×	(da Silva et al., 00)
IDEAS	T, Do, U, Di, P	OO	√	$\surd_p$	√	×	(Lozano, 01)
JustUI	T, Do, U, Di, P	OO	×	√	×	×	(Molina, 03)
OOH	Do, Nav, P	OO	×	√	×	×	(Cachero , 03)
TERESA	T, P	CTT, AUI	√	×	×	×	(Mori et al., 04)
usiXML	T, Do, AUI, CUI, C	usiXML	√	×	×	×	(Limbourg et al, 04)

[1] El símbolo $\surd_p$ denota disponibilidad únicamente a nivel de modelo de presentación, por contra el símbolo × denota falta de disponibilidad.

c. **las *buenas prácticas* deben estar a disposición de los colectivos implicados en la elaboración de software**, ya sean pertenecientes a IS ya a IPO, es decir, ingenieros del software y expertos en usabilidad deben poder compartir su conocimiento. Las técnicas de especificación formal, los lenguajes de modelado orientados a objetos, las técnicas basadas en componentes, la arquitectura software y otros conceptos relacionados, propios de IS, deben considerarse junto con la experiencia de desarrollo centrada en el usuario disponible en IPO,

d. **las herramientas desarrolladas deben considerar los conocimientos citados en el punto anterior** y facilitar su utilización conjunta, modificación y aprendizaje.

e. **el conocimiento o la experiencia** necesaria para la puesta en práctica de productos software de calidad **debe estar disponible y accesible** para permitir su utilización por parte de todos los implicados en el desarrollo de un producto software.

En función de los puntos anteriormente reseñados desde esta tesis doctoral se abogará en los capítulos posteriores por:

1. **caracterizar la calidad** de un producto software, prestando especial atención a aquella vertiente de la misma relacionada con la usabilidad y hacerlo utilizando, en la medida de lo posible, conceptos y notaciones próximas a la IS y utilizando notaciones adicionales cuando esto no sea posible, a la espera de la necesaria y previsible unificación de criterios,

2. **documentar la experiencia** de desarrollo, prestando especial interés a aquella experiencia relacionada con aspectos de puesta en práctica de técnicas de Diseño Centradas en el Usuario, pero sin perder de vista que la actividad que pretendemos realizar es la de desarrollar productos software de calidad.

3. **contribuir a la diseminación de la experiencia.** La misma deberá estar al alcance de todos y cada unos de los miembros involucrados en el proceso de desarrollo, de esta forma se convertirá en verdaderamente útil,

4. **integrar calidad y experiencia** en el propio proceso de desarrollo de cualquier producto software. La consideración de ambas y su incor-

poración metodológica debe realizarse lo antes posible, es decir, desde la fase de análisis. En ese momento el ingeniero ya debe de disponer de experiencia o serle fácil acceder a la misma y poder aprenderla y utilizarla.

5. finalmente, **facilitar herramientas que permitan integrar calidad y experiencia** dentro de las fases características del proceso de desarrollo de un producto software.

Los próximos capítulos estarán destinados a cubrir, con detalle, cada uno de los puntos anteriormente recogidos. Todos ellos, en cualquier caso, quedan englobados en dos conceptos clave como son: la calidad y la experiencia. La consideración de ambos conceptos conjuntamente se influye en que el gap semántico identificable entre IS e IPO no sea tal, y verdaderamente pase a percibirse su verdadera dimensión: la diferencia entre la necesidad de dar respuesta a un requisito de calidad, cualesquiera que este sea, y su especificación en un diseño posterior y, en cualquier caso, previo a una implementación.

2.6 Contribuciones relacionadas con este capítulo

Las principales contribuciones relacionadas con los contenidos recogidos en este capítulo y donde puede encontrarse información adicional a la recogida en el mismo pasan por:

- El trabajo de doctorado con el que se logro la Suficiencia Investigadora en el año 2001 y que estableció las bases de lo que esta tesis doctoral es hoy día. El mismo coincidía prácticamente en título con el que presenta este documento, se denominó **Integrando patrones de Interacción en metodologías de análisis y diseño de interfaces de usuario**, y en él se hacía una primera incursión en el estado del arte relacionado con la usabilidad y su logro, prestándose especial atención al Diseño Centrado en el Usuario, al concepto de usabilidad y al concepto de patrón y a su diferencia con las tradicionales guías de estilo.

Por su carácter introductorio y afín por los contenidos que se han recogido en este capítulo cabe también destacar las publicaciones:

- **Montero, F.**, Lozano, M., González, P. User Interfaces Development by using Patterns. VIIP 2001: 39-43

- **Montero, F.**, López-Jaquero, V., Lozano, M., González, P. Interfaces de usuario. El gran reto de los Sistemas de Información Geográfica. Editado por Fundación Dintel. 2001. (ISBN: 84-931933-1-3)
- **Montero, F**., Lozano, M., González, P. Patrones de Interfaz para entornos de trabajo en grupo. II Jornadas de trabajo Dolmen. Universidad Politécnica de Valencia. 12 y 13 de marzo de 2002.
- Lozano, M., **Montero, F.**, González, P. A usability and accessibility oriented development process. 8th ERCIM Workshop User Interfaces for all. Viena, 28-29 de junio de 2004.
- **Montero, F**., López-Jaquero, V., Ramírez, Y., Lozano, M., González. P. Patrones de Interacción: para usuarios y para diseñadores. VI Congreso de Interacción Persona-Ordenador. 13-16 de septiembre, Granada, España. 2005.

Referencias bibliográficas

Agile. Manifesto for Agile Software Development. http://agilemanifesto.org/. 2001

Ambler, S. W, User Interface Design: Tips and Techniques in The Object Primer: Building Object Applications That Work, and Process Patterns, 2e. Cambridge University Press, 2001.

Anderson, D. Extending UML for UI. SprintPCS.com, Kansas City, MO (2000), UML'2000 Workshop on Towards a UML Profile for Interactive Systems Development (TUPIS'2000). 2000

Annett, J. and Duncan, K.D. "Task Analysis and Training Design," Occupational Psychology, vol. 41, pp. 211-221, 1967.

Anoop K. Sinha, Scott R. Klemmer, and James A. Landay. Embarking on Spoken-Language NL Interface Design. The International Journal of Speech Technology, May 2002, Volume 5, Number 2, pp. 159-169.

Azevedo, P., Merrick, R., Roberts, D. OVID to AUIML - User-Oriented Interface Modelling. UML'2000 Workshop on Towards a UML Profile for Interactive Systems Development (TUPIS'2000). 2000.

BaIzert, H., et al.: The JANUS application Development Environment Generating More than the User Interface. In: Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 183-205.

Balzert, H., Hofmann, F., Kruschinski, V. and Niemann, C. The JANUS Application Development Environment | Generating More than the User Interface. In Computer-Aided Design of User Interfaces, pages 183{206, Namur, Belgium, 1996. Namur University Press.

Barber, J., "AskJef: Integration of Case-Based and Multimedia Technologies for Interface Design Support," Artificial Intelligence in Design '92, J.S. Gero (Ed.), Kluwer Academic, pp. 457-474, 1992

Bardon, D., Berry, D., Bjerke, C., Roberts, D. Crafting the Compelling User Experience Using a methodical software-engineering approach to model users and design. Easy of use, IBM. 2001.

Bauer, F. Software engineering: a practitioner's approach. Mc Graw-Hill. 1997

Booth, P: An Introduction to Human-Computer Interaction. Hove, UK, Lawrence Erlbaum Associates. 1989

Borch et al., 2003, A Model Driven Architecture for REA based systems, In the Proceedings of the Workshop on Model Driven Architecture: Foundations and Application, TR-CTIT-03-27,University of Twente

Bouillon L., Vanderdonckt J., Chieu Chow K., Flexible Re-engineering of Web Sites, in proceedings of the international conference on Intelligent User Interfaces , (IUI'04), January 13-16, Madeira (Portugal), ACM Press, New York, USA, 2004, pp 132-139

Brüne, S., Halmans, G. and Puhl, K. (2003). Modelling Dependencies between Variation Points in Use Case Diagrams. Proceedings of the 9th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ), Klagenfurt, Austria.

Cachero, C. OO-H: Una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales. Tesis Doctoral. Universidad de Alicante. 2003.

Campos, P. and Nunes, N. Canonsketch: a User-Centered Tool for Canonical Abstract Prototyping. In Proceedings of DSV-IS'2004, 11th International Workshop on Design, Specification and Verification of Interactive Systems. To appear. Springer-Verlag, 2004.

Card, S. K., Moran, T. P., & Newell, A., The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates. 1983

Carroll, J. M. (Eds.) (2003): HCI Models, Theories, and Frameworks. San Francisco, Morgan Kaufman Publishers

Carroll, J. M. (Eds.): Human-Computer Interaction in the New Millennium. Addison-Wesley Publishing. 2001

Constantine, L.L. and Lockwood, L.A.D, Software for Use: A Practical Guide to the Models and Methods of Usage Centered design, Addison-Wesley, 1999.

Cross, H. Engineers and Ivory Towers. Ayer Co. Pub. 1952

da Silva, P., Paton, N. W., UMLi: The Unified Modeling Language for Interactive Applications, in Conf. Proc. of UML00, UK, p.117-132, 2000.

da Silva, P.P., UMLi: Integrating User Interface and Application Design TUPIS. 2000.

de Baar, D. J. M. J. Foley J. D.and Mullet, K. E.. Coupling Application Design and User Interface Design Beyond Widgets: Tools for Semantically Driven UI Design. In Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems, 1992, pages 259-266.

Earthy, J. (1998). Usability Maturity Model: Human Centredness Scale. INUSE Project deliverable D5.1.4(s). Version 1.2. London, Lloyd's Register of Shipping.

Elwert, T., Schlungbaum, E. Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In Designing, Specification and Verification of Interactive Systems, pages 193{208, Vienna, 1995. Springer.

Elwert, T., Schlungbaum, E.: Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In: Designing, Specification and Verification of Interactive Systems. Wien: Springer, 1995, 193-208.

Fischer, G. User Modeling: The long and winding road. Procedings of 7th International Conference on User Modeling. Canada. 1999

Gamma, E., Helm, R., Johnson, R., y Vlissides, J. (1995).Design Patterns: Elements of Reusable Object-Oriented Software. AddisonWesley.

Gorny, P. Expose: HCI-counseling for user interface design. Interact'95. pp. 297-304.

Griffiths, T., P. Barclay, J. McKirdy, N. Paton, P. Gray, J. Kennedy, R. Cooper, C. Goble, A. West, and M. Smyth. Teallach: A Model-Based User Interface Development Environment for Object Databases. In Proceedings of UIDIS'99, pages 86{96, Edinburgh, UK, September 1999. IEEE Press.

Günter Halmans, Klaus Pohl: Communicating the Variability of a Software-Product Family to Customers. Software and System Modeling 2(1): 15-36 (2003)

Harkikat, P., A. Seffah and J. Strika. QUIM: A Tool and Knowledge Mp for Usability Measurement 3rd International Workshop on Software Measurement. (IWSM 2003). Sept 23-25, 2003, Montreal, Canada

Hartson, H.R., Siochi, A., Hix, D. The UAN: A user-oriented representation for direct manipulation interface designs. ACM Transactions on Information Systems. Vol 8. No. 3. 1990

Henninger, S., Haynes, K., Reith, M.W., A Framework for Developing Experience-Based Usability Guidelines, Proceeding of the Symposium on Designing Interactive Systems (DIS '95), Ann Arbor MI, August 23-25, 1995 (in press).

Hui Wang, Xueli Yu, Li Wang, Xu Liu: OGSA Based E-learning System: An Approach to Build Next Generation of Online Education. GCC (1) 2003: 217-220

Iannella, R. HyperSAM. A practical User interface Guidelines Management System, in Proc. Of 2nd annual CHISIG. Australia. 1994

INUSE Usability Maturity Model: Human-Centredness Scale, Lloyd's. Register of Shipping project IE2016 INUSE Deliverable D5.1.4s

ISO/IEC 18529 Human-centred Lifecycle Process Descriptions. ISO/IEC TR 18529: 2000

Jameson, A. Modeling Both the Context and the User, Personal Technologies, 5 2001. http://citeseer.ist.psu.edu/jameson01modeling.html

Johnson, P: Human-Computer Interaction. London: McGraw-Hill, 1992.

Johnson, P (1989) Task Knowledge Structures. In Diaper D. (ed.) Task analysis in Human Computer Interaction. Ellis Horwood.

Johnson, P., Wilson, S., Markopoulos, P. and Pycock, J., ADEPT---Advanced Design Environment for Prototyping with Task Models, Demonstration Abstract. In Proceedings, Interchi'93, ACM Press, p. 56, April 1993. http://citeseer.ist.psu.edu/johnson93adept.html

Karat, J. and Karat, C.N.: The Evolution of User-Centered Focus in the Human-Computer Interaction Field, IBM Systems Journal, vol. 42, no. 4, 2003, pp. 532-41.

Kobsa, A., Koenemann, J. and Pohl, W.: 2001, Personalized Hypermedia Presentation techniques for Improving Customer Relationships. The Knowledge Engineering Review, forthcoming.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-Jaquero, V., UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004

Lin, J., "Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces." In Conference Supplement of UIST 2003: ACM Symposium on User Interface Software and Technology, Vancouver, BC, Canada, Nov. 2–5, 2003, pp. 13–16.

Lonczewski, F., S. Schreiber. The FUSE-System: an Integrated User Interface Desgin Environment. In Computer-Aided Design of User Interfaces, pages 37{56, Namur, Belgium, 1996. Namur University Press.

Lozano, M. Entorno metodológico orientado a objetos para la especificación y desarrollo de interfaces de usuario. Tesis doctoral realizada en la Universidad Politécnica de Valencia. 2001

Lozano, M., I. Ramos: Integration of the User Model and Human Factors in an Object Oriented Software Production Environment. ECOOP'99, Lisboa -Portugal-, Junio, 1999.

Marchionini, G., Gregory, C.: Evaluating Hypermedia and Learning: Methods and Results from the Perseus Project. In ACM Transactions on Information Systems (TOIS), 12 (1) p. 5-34. 1994

Marchionini, G., Levi, M. D.: Digital government information services: the Bureau of Labor statistics case. In Interactions, 10 (4) p. 18-27. 2003

Mark W. Newman, James Lin, Jason I. Hong, and James A. Landay, "DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice." In Human-Computer Interaction, 2003. 18(3): pp. 259-324.

Markopoulos, P. Supporting interaction design with UML, task modelling. TUPIS. 2000.

Markopoulos, P., Bekker, M., and Marijnissen, P, Interaction Design and the USDP: Should UML do it all? DSV-IS 2000, Limerick Ireland, June 2000.

Mayhew, D.J., The Usability Engineering Lifecycle, A Handbook for User Interface Design, Morgan Kaufmann, 1999.

Molina, P.J. Especificación de Interfaces de usuario: de la especificación de requisitos a la generación automática. Universidad Politécnica de Valencia. Tesis Doctoral. 2003. http://pjmolina.com/es/investigacion/tesis.php

Mori G., Paternò F., Santoro C. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering (August 2004, pp.507-520).

Nunes, N. J., Cunha, J. F., Towards a UML Profile for Interactive Systems Development: the Wisdom Approach, Proceedings of the 3rd International Conference on the Unified Modeling Language (UML2000), Evans, A. (Ed.), Springer-Verlag LNCS 1939, New York, 2000, pp. 101-116.

Nunes, N. J., Cunha, J. F., Wisdom A UML Based Architecture for Interactive Systems, Proceedings of the 7th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS2000), Paterno, F., Palanque, P. (Eds.), Springer-Verlag LNCS 1946, New Tork, 2000, pp. 191-205.

Nunes, N. J., Cunha, J. F., WISDOM Whitewater Interactive System Development with Object Models, in Mark van Harmelen (Editor.), Object-oriented User Interface Design, Addison-Wesley, Object Technology Series , 2001.

Nunes, N. J., Cunha, J. F., WISDOM: A Software Engineering Method for Small Software Development Companies, IEEE Software, Special Issue on Software Engineering in the Small, Sep./Oct., 2000.

OMG. The Common Object Request Broker: Architecture and Specification, Revision 1.1. Object Management Group, 1992.

Paganelli, L., Paternò, F. Automatic Reconstruction of the Underlying Interaction Design of Web Applications. Proceedings of SEKE 2002 (Ischia, Italy, July 2002).

Pastor, O., Molina, J.C., Iborra, E.: Automated production of fully functional applications with OlivaNova Model Execution, ERCIM News No.57, April 2004. http://www.care-t.com/

Paternò, F., Ballardin, G. Model-aided Remote usability evaluation. Human-Computer Interaction. Interact'99. Published by IOS Press, 1999.

Paternò, F., ConcurTaskTrees and UML: how to marry them? TUPIS. 2000.

Paternò, F., Model Based Design and Evaluation of Interactive Applications, Springer-Verlag, London, 1999.

Pressman, F. Software engineering: a practitioner's approach. McGraw-Hill. 1992

Puerta, A., A model based interface development environment, IEEE Software 14(4) July/August 1997, pp.41-47, 1997.

Puerta, A., Maulsby, D. Management of Interface Design Knowledge with MODI-D. In Proceedings of IUI'97, pages 249{252, Orlando, FL, January 1997.

Puerta, A., Szekely, P.: Model-based Interface Development. CHI'94 Tutorial Notes, 1994.

Puerta, A.R. and Eisenstein, J. Towards a General Computational Framework for Model-Based Interface Development Systems. IUI99: International Conference on Intelligent

User Interfaces, Los Angeles, January 1999, in press. http://citeseer.ist.psu.edu/puerta99towards.html

Puerta, A.R. The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. CADUI'96: Second International Workshop on Computer-Aided Design of User Interfaces, Namur, Belgium, June 1996, pp. 19-25.

R.G.G. Cattell, editor, The Object Database Standard: ODMG-93, Release 1.1, Morgan Kaufmann Publishers, San Francisco, 1994.

Roberts, D., Berry, D., Isensee, S., Mullaly, J.: Developing Software Using OVID. IEEE Software 14(4): 51-57 (1997)

Roberts, D., Berry, D., Isensee, S: Object View and Interaction Design. INTERACT 1997: 663-664

Schlungbaum, E.: Individual User Interfaces and Model-Based User Interface Software Tools. Research Report, Georgia Institute of Technology, Graphics, Visualization & Usability Center, GIT-GVU-96-28, Noviembre 1996.

Seffah, A. Bridging the Educational Gap between SE and HCI: What Software Engineers Should Know?. Workshop on Bridging the Gaps II: Bridging the Gaps Between Software Engineering and Human-Computer Interaction. Internation Conference on Software Engineering (ICSE) 2004. May 24-25, 2004 in Edinburgh, Scotland, UK.

Szekely, P., Retrospective and Challenges for Model Based Interface Development, in Vanderdonckt, J., (Editor), CADUI'96, Presses Universitaires de Namur, Namur, 1996.

Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., Salcher, E. Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach. In Engineering for Human-Computer Interaction, pages 120{150, London, UK, 1996. Chapman & Hall.

Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. In Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces, (Vanderdonckt, J. Ed.). Namur University Press, Namur, 1996. http://citeseer.nj.nec.com/szekely96retrospective

Vanderdonckt, J. "Accessing Guidelines Information with Sierra," Proceedings Fifth IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT '95), Lillehammer, Chapman & Hall, London, 1995, pp. 311-316

Vanderdonckt, J., Advice-giving systems for selecting interaction objects, in Proc. of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edimburg, 5-6 September 1999), N.W. Paton & T. Griffiths (eds.), IEEE Computer Society Press, Los Alamitos, 1999, pp.152-157.

Vanderdonckt, J., Bodart, F., Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection, in Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93.1993

Vanderdonckt, J.: Knowledge-Based Systems for Automated User Interface Generation: the TRIDENT Expierence. Technical Report RP-95-010. Namur: Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique 1995. Available at http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-010.

Wilson, S: Reflections on Model-Based Design: Definitions and Challenges. In: Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 327-333.

Capítulo 3 La calidad y sus modelos

La calidad nunca es un accidente; siempre es el resultado de un esfuerzo de la inteligencia.
John Ruskin (Crítico y escritor británico)

3.1 Introducción

Este capítulo esta dedicado al concepto de calidad de un producto software y a la recopilación e identificación de los mecanismos y recursos disponibles para su logro, teniendo en cuenta que, siendo nuestro principal interés los aspectos relacionados con la interfaz de usuario, la perspectiva ofrecida será una visión desde el punto de vista del usuario.

La **calidad**, según la ISO 8402, es la totalidad de las características de un producto o servicio que le confieren aptitud para satisfacer necesidades establecidas e implícitas. Según Pressman (Pressman, 1995), y desde el punto de vista de la IS, *la **calidad del software** es la concordancia del software producido con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente*.

En esta tesis doctoral, nuestro interés es el desarrollo de interfaces de usuario, pero no podemos caer en la tentación de suponer que la parte, entendiendo como tal la interfaz de usuario de calidad, es el todo o que esa misma parte tiene un protagonismo especial en lo que se refiere al logro de calidad, es cierto que para el usuario la interfaz es su aplicación, pero como iremos viendo la calidad es más que la mera bondad de la interfaz del usuario, que acompaña a la aplicación, y así queda reflejado en la propia evolución de la características distintivas asociadas a la interfaz de usuario.

Los problemas que pretendemos poner de manifiesto, en base a una revisión de los estándares internacionales disponibles y a la consulta de documentación relacionada, serán los siguientes:

i. Estudio y definición de calidad de un producto software desde el punto de vista del usuario, identificando una **falta de consenso** en la definición de los factores de calidad relacionados con el punto de vista seleccionado.

ii. Una vez revisados los estándares relacionados observamos una **carencia en la profundidad de la caracterización de los factores que determinan la calidad** de un producto software.

iii. Se ofrece **poca o nula conexión entre características de calidad y el logro de las mismas** tanto en los estándares, como consultando a los autores representativos en el ámbito de la calidad. Es decir, apenas se dispone de información *know-how* ligada a la caracterización de la calidad, existiendo métricas, en el mejor de los casos. Métricas que, a su vez, no están validadas ni existe información de cómo estimarlas.

iv. Aunque la base para la puesta en práctica de la Ingeniería de la Usabilidad (Nielsen, 1993) es la evaluación, **no existe un procedimiento sistemático para incorporar y llevar a cabo dicha evaluación.** Fundamentalmente, porque desde la IS aún contemplándose la usabilidad, como quedará constatado en este capítulo, no se le ha prestado la debida atención. Así, la misma surge bien en etapas tempranas, bien en finales haciéndose uso de las distintas métricas disponibles (de Marco, 1982). Unas métricas que aunque abundantes, en la mayoría de las ocasiones no están validadas y así queda reflejado en los propios estándares.

En este capítulo trataremos cada uno de los problemas anteriormente mencionados e identificaremos mecanismos que serán utilizados para dar una solución al problema de desarrollar interfaces de usuario de calidad.

3.2 Calidad en IS: Proceso

Desarrollar software es una labor multidisciplinar en la que tiene cabida experiencia de muy diversas fuentes. La IS es la disciplina más directamente involucrada en esta labor de desarrollo, y desde ella se han presentado métodos, lenguajes y herramientas, para abordar la tarea de desarrollar software.

El desarrollo de software está caracterizado por métodos iterativos donde, independientemente de la metodología por la que se apueste, existen unos elementos comunes como son: la iteración, el incremento y el papel destacado que juega el usuario, entre otras características reseñables.

Las primeras fases del ciclo de vida, aquellas asociadas con labores de análisis y diseño, han ido cobrando cada vez más importancia, y la concepción del sistema involucra un nivel de abstracción cada vez mayor. En función de esto, un ciclo de vida simplificado se muestra en la Fig. 3-1. En ella tiene cabida una fase de identificación de requisitos y necesidades de

los mismos, punto 1. El desarrollo continua con un proceso iterativo e incremental donde las iteraciones están determinadas por procesos de evaluación 4. La evaluación pasa por cuantificar y/o constatar una versión interactiva del software 3 que se está produciendo corroborando, así, la presencia de una serie de características en el software inicialmente diseñado 2.

Figura 3-1. Etapas que surgen en el desarrollo de software

Esas características constituirán un **modelo de calidad** (Fenton et al, 1997), por lo que se entiende aquel *conjunto de criterios que son utilizados para determinar si se alcanza determinado nivel de calidad en el desarrollo de un producto software* (Brajnik, 2001).

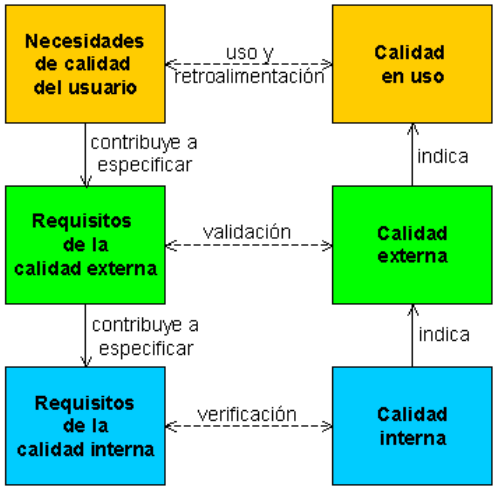


Figura 3-2. Necesidades, requisitos y desarrollo de un producto software

En relación con la calidad de un producto software y su logro, la tendencia actual pasa por la creencia plasmada en la Fig. 3-2, donde se establece que la calidad de un producto software viene determinada por la satisfacción de unas exigencias de usuario. Dichas exigencias se traducen en la existencia de unas necesidades externas, que se logran, a su vez, a través de un proceso de desarrollo en el que se consideran unos requisitos de calidad internos. Posteriormente, el logro de unos y otros requisitos se contrasta a través de sendos procesos de *verificación* y *validación* respectivamente. Con la verificación se comprueba que el producto se está construyendo adecuadamente (punto de vista interno), y con la validación que se está desarrollando un producto software adecuado con las necesidades del usuario (punto de vista externo). Al considerar tanto requisitos internos como externos quedan reflejadas las dos exigencias, igualmente importantes, que se tienen cuando se desarrolla un producto software, como son que (a) el software debe ser conforme con sus especificaciones y, (b) que el software debe hacer lo que el usuario desea.

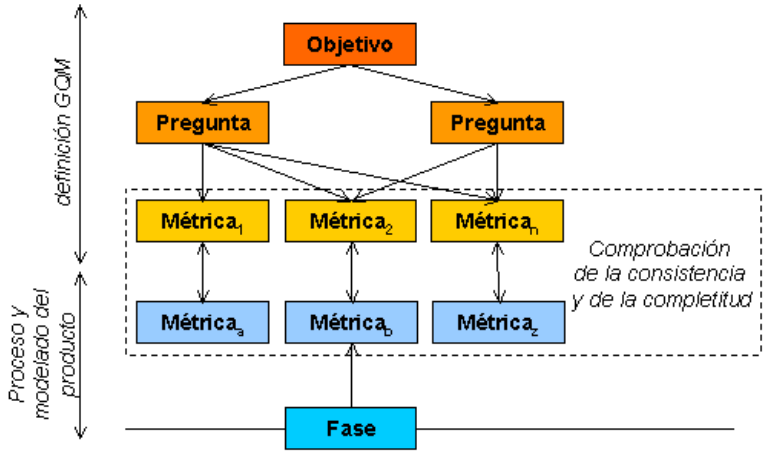


Figura 3-3. Elementos de un modelo de calidad (Basili, 1988)

La elaboración de un modelo de calidad pasa por la identificación de aquellas características, tanto internas como externas, que influyen en la calidad de un producto software. Un modelo de calidad es una descomposición jerárquica, y su elaboración puede abordarse de diferentes formas, por ejemplo, en (Basili et al., 1988) se propone la técnica GQM con la que es posible identificar objetivos y métricas que permitirían la caracterización de la calidad (véase Fig. 3-3).

En IS se han elaborado y están disponibles diferentes modelos de calidad y estándares relacionados con la calidad que ofrece un producto soft-

ware. En los mismos se pueden identificar dos tendencias. Por un lado están los que persiguen identificar características del producto (McCall, Boehm, ISO 9126, etc.) y, por otro, los que tratan de identificar características de calidad en el proceso de desarrollo del producto software, por ejemplo, contemplando los costes de desarrollo, como ejemplo de ello podemos citar algunos modelos y estándares: Cocomo (Boehm, 1981), que atiende a aspectos de organización del proceso de desarrollo; el modelo *Capability Maturity Model* (CMM)(Paulk, 1993) que atiende a aspectos de organización del proceso de desarrollo o aquellos que contemplan el propio proceso de desarrollo de software como es el caso del estándar *Software Process Improvement and Capability dEtermination* (SPICE, también conocido como ISO 15504) o el estándar ISO 9000.

La calidad que se identifica en los modelos de calidad elaborados desde la IS posee múltiples facetas. Los modelos más representativos recogidos cronológicamente son: McCall (McCall et al., 1977), Boehm (Boehm et al., 1978), FURPS (Grady et al., 1987), ISO 9126 (ISO/IEC 9126, 1991), Dromey (Dromey, 1996) y la revisión del estándar ISO 9126.

En el modelo de calidad de McCall se identifican 11 factores de calidad orientados al producto y agrupados entorno a tres factores: *operación*, *revisión* y *reutilización* (Fig. 3-4). La gran aportación de este modelo fue el establecimiento de relaciones entre características de calidad y métricas, aunque haya sido criticado por la objetividad de algunas de las métricas propuestas.

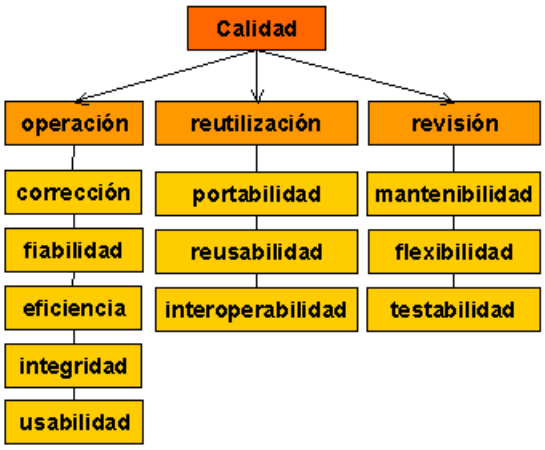


Figura 3-4. Modelo de calidad de (McCall et al., 1977)

El modelo de Boehm (Boehm et al., 1978) es similar al de McCall (Fig. 3-5) al representar la estructura de características de calidad de forma jerárquica, contribuyendo cada una de ellas a la calidad total. Tal como hacía

el modelo de McCall, en el de Boehm también están consideradas las necesidades del usuario, pero también encontramos características de calidad en el terreno del hardware que no están presentes en el otro modelo propuesto.

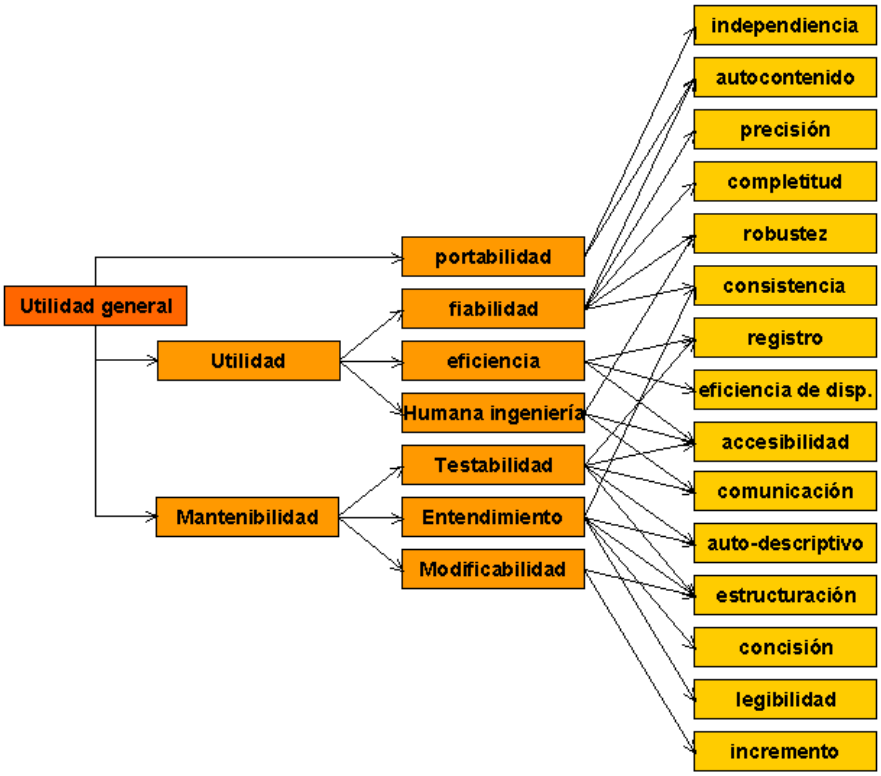


Figura 3-5. Modelo de calidad de (Boehm et al., 1978)

El modelo FURPS (Grady et al., 1987) establece cinco características como factores de calidad que son los que le dan nombre: *Functionality*, *Usability*, *Reliability*, *Perfomance* y *Supportability* (Véase la Fig. 3-6). Una limitación de este modelo de calidad es que no tiene en cuenta la portabilidad de los productos software que se estén considerando, factor digno de consideración en función de las exigencias actuales que recaen sobre el proceso de desarrollo del software.

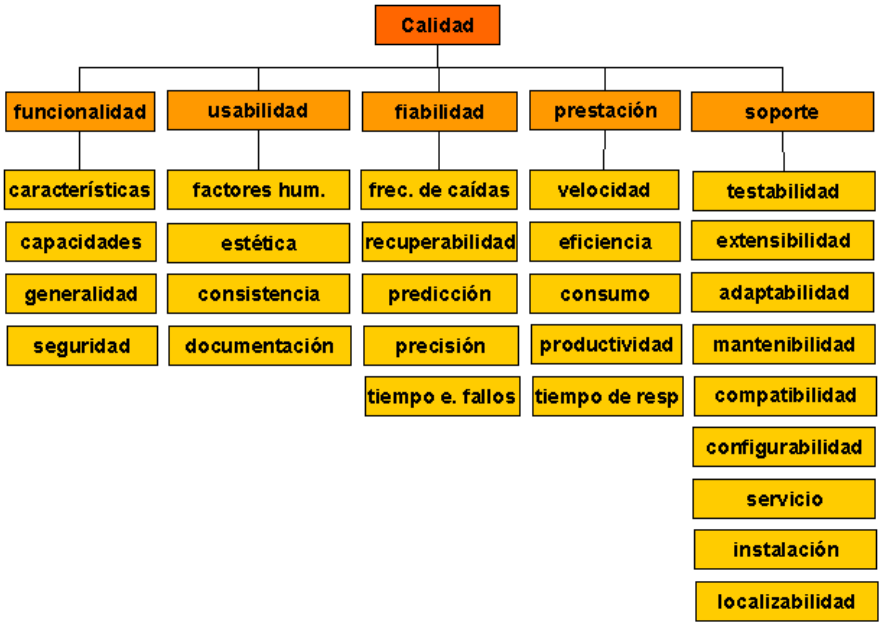


Figura 3-6. Modelo de calidad FURPS (Grady et al., 1987)

Posteriormente, y fruto de las propuestas elaboradas, surgió el estándar ISO 9126 a nivel internacional (ISO 9126, 1991) que, recientemente, ha sido revisado. Este estándar define la calidad de un producto software como un conjunto de características dependientes del producto y cuenta con gran aceptación y extensión, pese a estar limitado a describir la calidad a nivel de factores y, únicamente, introducir una serie de criterios que también pueden considerarse al tratar cada uno de esos factores.

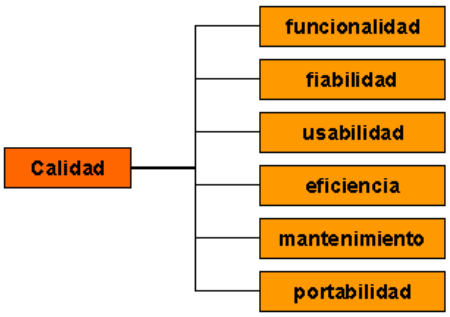


Figura 3-7. ISO/IEC 9126 (ISO, 1991)

Algunas propuestas posteriores al estándar coinciden en la elaboración de modelos de calidad que quedan abiertos y tienen su base en el estándar

9126. En este sentido, Dromey estudió en su propuesta (Dromey, 1996) las relaciones entre las características y las subcaracterísticas asociadas con la calidad de un producto software, concluyendo que existen diferentes características, ya sean éstas factores o criterios, que debido al nivel que ocupan en el modelo de calidad, no pueden ser consideradas en su totalidad dentro del sofware. De esta forma Dromey aboga por la elaboración de modelos de calidad abiertos o mixtos en los que partiendo de atributos identificados en otros estándares, por ejemplo el ISO 9126, se enriquecen con criterios propios del producto software concreto que se pretenda desarrollar. Dromey, también, propone la existencia de diferentes modelos de calidad en función de la fase del ciclo de vida en la que nos encontremos, aspecto también muy a tener en cuenta. En la Fig. 3-7 aparece recogida parte de su propuesta para un modelo de calidad asociado a la fase de implementación.

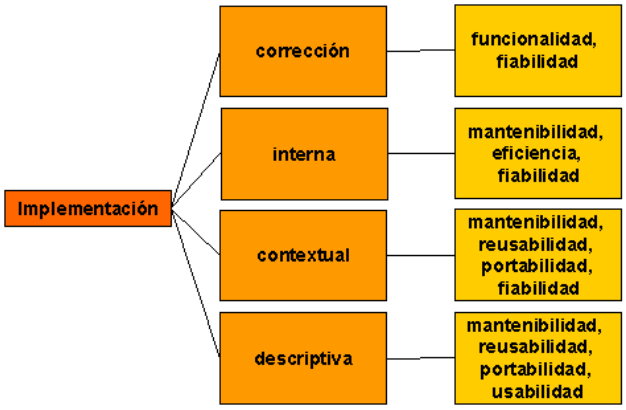


Figura 3-8. Modelo de calidad propuesto en (Dromey, 1996)

Volviendo a retomar el estándar ISO 9126, en él se establece que la calidad de un producto software debe ser descrita en términos de una o varias de las siguientes características (véase Fig. 3-8): *functionality*, *reliability*, *maintainability*, *efficiency* y *portability*. Asociado a las diferentes características mencionadas se dispone, gracias también al estándar, de subcaracterísticas que han ido enriqueciéndose en otras propuestas como QUINT2 (Zeist et al., 1996) (Fig. 3-9), y en posteriores revisiones del mismo estándar como es el estándar ISO/IEC 9126-1 (ISO/IEC 9126-1, 1998).

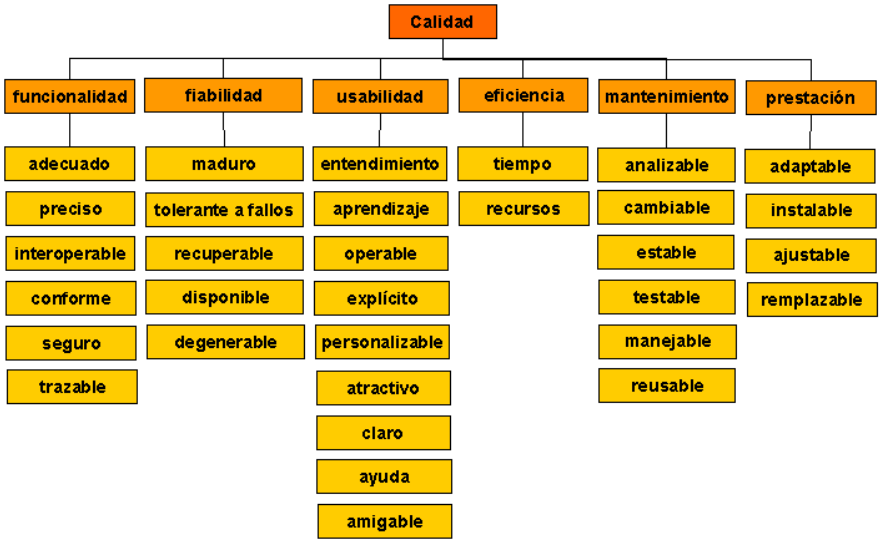


Figura 3-9. Modelo de calidad QUINT2

En esta última versión del estándar 9126, es decir en la ISO 9126-1, se actualiza el modelo de calidad y con ello se introducen criterios que antes aparecían relegados a los distintos apéndices que acompañaban al estándar.

En otro estándar asociado con el estándar que nos ocupa, concretamente el estándar ISO 9126-4, se introduce el concepto de *calidad en uso* (*quality in use*), con el que se pretenden considerar conjuntamente los efectos de los seis factores de calidad propuestos en el estándar cuando el producto está en uso, y junto con ello se presenta un marco de trabajo en el que se recogen las dependencias entre el proceso, el producto y la calidad en uso del mismo, dependiendo de los diferentes contextos en los que éste sea utilizado.

Otros estándares en los que se ha dividido el estándar 9126 proporcionan métricas asociadas con las características internas (ISO 9126-3, 2003) y con las externas (ISO 9126-2, 2003) de un producto software. El problema de las métricas proporcionadas es que no están validadas, ni se datalla cómo obtenerlas.

Una revisión de las propuestas recogidas en este apartado pone de manifiesto la coincidencia en la aparición de diferentes factores de calidad característicos, independientemente del modelo de calidad considerado de entre los presentados. Esos factores que aparecen con reiterada asiduidad son la eficiencia, la fiabilidad, la mantenibilidad, la portabilidad, la funcionalidad y, aunque resulte inesperado, también la usabilidad tiene su hueco entre ellas. Resulta igualmente extraño que a la misma no se le haya prestado una especial atención desde la IS.

Figura 3-10. ISO 9126-4 (ISO, 2004)

El atributo que ha centrado la atención de los investigadores en IPO es, fundamentalmente, la usabilidad, y como se ha recogido en los modelos de calidad reseñados con anterioridad, este factor está entre los factores incluidos en los modelos de calidad propuestos desde la IS, falta saber si el término se utiliza, en una y otra disciplina, con las mismas implicaciones.

En IS se observa una tendencia dirigida a la adopción del término *calidad en uso* (*quality in use*) que puede servir para unificar terminología con IPO, facilitándose así la reducción del *gap* entre ellas. El término de calidad en uso ha sido introducido en el último de los estándares presentado (ISO 9126-4, 2004) (véase Fig. 3-10) y queda definido como: *la capacidad de un producto software para permitir a usuarios específicos el logro de objetivos concretos con efectividad, productividad, seguridad y satisfacción en contextos de uso también específicos.*

En ese mismo estándar se establece una analogía entre los términos calidad en uso y usabilidad, considerándose conceptos idénticos salvo por la consideración adicional de criterios relacionados con la seguridad en el primero de los términos frente al segundo de los reseñados. Pero, hasta este momento, todavía no se ha presentado formalmente el concepto de usabilidad tal y como se contempla en IPO o en IS, esta tarea será abordada seguidamente.

3.3 Calidad en IPO: Usabilidad

La calidad que busca identificar la comunidad IPO está centrada en la que pueden experimentar los usuarios al hacer uso de un producto software, fruto de una interacción con el mismo a través de la interfaz que se le proporciona, ya sea esta interfaz software o hardware.

La comunidad IPO ha abrazado los estándares internacionales relacionados con la definición de calidad que han sido introducidos en el apartado anterior. En cualquier caso su objetivo fundamental es la interacción y la misma se realiza a través de una interfaz de usuario, recordaremos por ello la definición de interfaz de usuario que manejamos, para luego pasar a identificar qué elementos determinaran la interfaz y cómo influirán los mismos en la interacción y a su vez ésta en la calidad final del producto.

Según Bennett (Bennett, 1983) en una interfaz de usuario concurren dos modelos y dos lenguajes. Los modelos tienen que ver con el usuario, en tanto en cuanto éste tiene unos objetivos y unas tareas que realizar y las concibe de cierta forma en función de su nivel cognitivo, y con el sistema que está diseñado para cubrir unos requisitos funcionales. Los lenguajes tienen que ver con la forma en que el diseñador permite al usuario interactuar con el sistema, *lenguaje de acción*, y con la forma que tiene el sistema para presentar la información, *lenguaje de presentación*.

Tabla 3-1. Tabla comparativa de definiciones relacionadas con usabilidad

Shackel	Shneiderman	Preece et al.	Nielsen	Constantine et al.
Efectividad – velocidad	Velocidad de prestaciones	Productividad	Eficiencia de uso	Eficiencia en el uso
Aprendizaje - tiempo	Tiempo para aprender	Facilidad para ser aprendido	Facilidad para ser aprendido	Facilidad para ser aprendido
Aprendizaje – atención	Duración del recuerdo	-	Facilidad para ser recordado	Facilidad para ser recordado
Efectividad – errores	Tasa de errores	Productividad	Seguridad / Errores	Fiabilidad en el uso
Actitud	Satisfacción	Actitud	Satisfacción	Satisfacción

Todos esos elementos deben tenerse en cuenta a la hora de contemplar la calidad de la interfaz. Revisando los modelos recogidos en el apartado anterior (McCall, 1977; Boehm, 1978; ISO 9126, etc.), y en función de las definiciones dadas a los diferentes factores que se contemplan en ellos, puede concluirse que en uno de esos factores se aglutinan la mayor parte de los elementos que conlleva la interfaz de usuario, es la usabilidad, pero antes de que surgieran algunos de los estándares presentados otros autores habían definido el concepto de usabilidad identificando algunas de esas

mismas características que serían recogidas finalmente en los estándares, por ejemplo (Shackel, 1991; Schneiderman, 1992, Nielsen, 1993; Preece et al., 1994; Constantine et al., 1999) han caracterizado la usabilidad utilizando diferentes atributos que se muestran en la Tabla 3-1.

Pero, la usabilidad no es solamente interfaz de usuario, y así queda reflejado en la evolución que ha sufrido el término a lo largo de su caracterización, ya sea en las definiciones ofrecidas por autores representativos ya sea en los estándares internacionales. En los siguientes apartados se ofrecen pistas que permiten seguir dicha evolución.

3.3.1 Usabilidad es presentación

No hace mucho el término *interfaz amigable* estuvo absolutamente extendido dentro de la comunidad dedicada al desarrollo de productos software, y con él se remarcaba el papel que debía jugar la interfaz de usuario asociada a un producto software. Este término dejo de utilizarse, y a principios de los ochenta fue sustituido por el término que ahora nos ocupa, la usabilidad.

Inicialmente dicha propiedad adquiría una connotación puramente dependiente del producto, de su interfaz, así podemos encontrar definiciones en las que se asocia usabilidad y facilidad de uso, estando esa facilidad de uso ligada, principalmente, a la interfaz que el producto ofrece, es decir, a su facilidad para ser utilizado, aprendido o entendido. Este punto de vista es el recogido en los estándares ISO 9126, visto anteriormente, o el ISO 14598, dedicado a la evaluación de productos software.

Tradicionalmente, la IS ha asociado la usabilidad con la interfaz de usuario y más concretamente con la facilidad con la que un producto puede ser utilizado, pero eso no deja de ser una visión parcial del término (Bevan, 1999). Para la usabilidad se han propuesto diferentes definiciones y descomposiciones en criterios y subcriterios de menor nivel de abstracción. Si hacemos un repaso por los estándares podemos identificar diferentes definiciones:

- según la IEEE Std.610.12-1990 la usabilidad es la facilidad con la cual un usuario puede aprender a utilizar, preparar las entradas e interpretar las salidas de un sistema o componente,

- según la (ISO/IEC 9126-1, 2001) la usabilidad es la capacidad de un producto software para ser entendido, aprendido, utilizado y atractivo para el usuario, cuando se utiliza bajo condiciones concretas,

Hasta aquí, como hemos reseñado, la usabilidad se considera parte del producto y es aquella parte que lo hace más fácil o difícil de usar.

3.3.2 Usabilidad es utilización

Paralelamente, con la visión mostrada en el apartado anterior existe otra concepción de usabilidad en la que se consideran conceptos relacionados con el uso, sin perder de vista la utilidad del propio producto (Bevan, 1995), según la (ISO 9241-11, 1998), dedicada a la recogida de requisitos ergonómicos para trabajar con terminales de presentación visual, el concepto de usabilidad puede definirse como *el nivel con el que un producto se adapta a las necesidades del usuario y puede ser utilizado por el mismo para lograr unas metas con efectividad, eficiencia y satisfacción en un contexto especifico de uso*.

En la definición anterior, por *efectividad* se entiende que cualquier software debe tener unos objetivos claros y estos deben ser alcanzables. Un software puede ser más o menos eficaz para sus objetivos, bien porque estos no son explícitos, están mal planteados o el desarrollo del software se haya desviado hacia otros objetivos diferentes. Para su evaluación se requiere que los diseñadores expliciten los objetivos que se persiguen y se compruebe si los usuarios pueden alcanzarlos.

La *eficiencia* depende de las destrezas del usuario y de las posibilidades del software, por lo que para su análisis se impone el estudio de diferentes tipos de usuarios. La eficiencia resulta difícil de medir directamente aunque es posible encontrar índices indirectos, sobre todo aquellos factores que inciden en ella incrementándola o haciéndola descender como por ejemplo: la facilidad de aprendizaje, la facilidad para ser recordado por el usuario, el nivel de retroalimentación en la interacción o el control de errores.

Finalmente, el nivel de *satisfacción* es un estado subjetivo que se alcanza cuando el usuario logra el objetivo de su actividad y lo hace de forma satisfactoria.

El marco de trabajo en el que se considera la usabilidad es el mostrado en la Fig. 3-11, donde se encuentra un producto en un contexto de uso caracterizado por un usuario, con unos objetivos; unas tareas; un equipamiento y un entorno de utilización. Fruto de una interacción entre todos esos elementos el usuario debe lograr los objetivos deseados con efectividad, eficiencia y satisfacción.

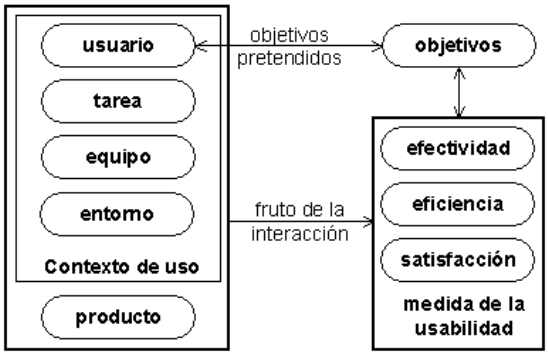


Figura 3-11. Marco de trabajo en el que se considera la usabilidad

3.4 Análisis y consideración del concepto de calidad

Lo que se concluye después de la retahíla de definiciones de usabilidad recogida en los apartados anteriores, con sus criterios y subcriterios respectivos, es una mezcolanza de conceptos que tienen que ver unas veces con características dependientes de la mera presentación de la información, y otras en las que hay una dependencia de la funcionalidad de la aplicación.

La aparente discrepancia de definiciones para un mismo concepto ha venido a converger con la introducción del término que aparecía al final del apartado 3.2, la *calidad en uso*. Este término está mucho más próximo al de la usabilidad, de hecho se consideran equivalentes en el propio estándar ISO 9126-4, será por ello que, a partir de este momento, usabilidad y calidad en uso se entenderán equivalentes y como tal serán utilizados en lo que resta del documento.

3.4.1 Cuándo considerar la calidad en uso

Las diferentes facetas que ofrece la usabilidad ha implicado que sean muchas las aportaciones que se han hecho para su logro y que éstas se hayan hecho a muy diversos niveles de abstracción y asociadas a muchas de las fases que constituyen el ciclo de vida de un producto software. Parece claro que la usabilidad ha de ser considerada siempre y desde el primer momento, para ello existe experiencia a muy diversos niveles de abstracción, que será tratada en un capítulo posterior, y que engloba desde principios generales hasta guías de estilo muy concretas.

Por otro lado, la puesta en práctica de un DCU exige que antes de iniciar un proyecto sea esencial caracterizar tanto a usuarios como a los aspectos

del producto de interés y necesidad. Teniendo en cuenta estas consideraciones de forma temprana se ahorra tiempo y dinero, dado que la posterior implementación de nuevos aspectos o, de simplemente, nuevas interfaces de usuario implican un enorme esfuerzo adicional. Incluso una vez que el producto está en el mercado se debe preguntar a los usuarios acerca de sus necesidades y actitud respecto del mismo.

El principal inconveniente que tiene la consideración de la usabilidad en el proceso de desarrollo es que ésta no se lleva a cabo de forma sistemática aunque esté acuñado el término de Ingeniería de la Usabilidad (Nielsen, 1991) desde hace más de veinte años, como se concluyó en el capítulo anterior al tratarse las diferentes metodologías propuestas bajo esta denominación. La principal razón para no considerar la calidad en uso a lo largo del proceso de desarrollo viene dada por la falta de experiencia a la hora de elaborar y caracterizar los modelos de calidad y, asociado con ello, a la disponibilidad de unos criterios establecidos que determinen la calidad de un producto software y, por ende, la calidad en uso del mismo. En estos momentos, disponemos de factores, criterios y métricas que además convergen a una calidad común establecida en calidad en uso, y lo que falta es integrar dichos elementos en el propio proceso de desarrollo.

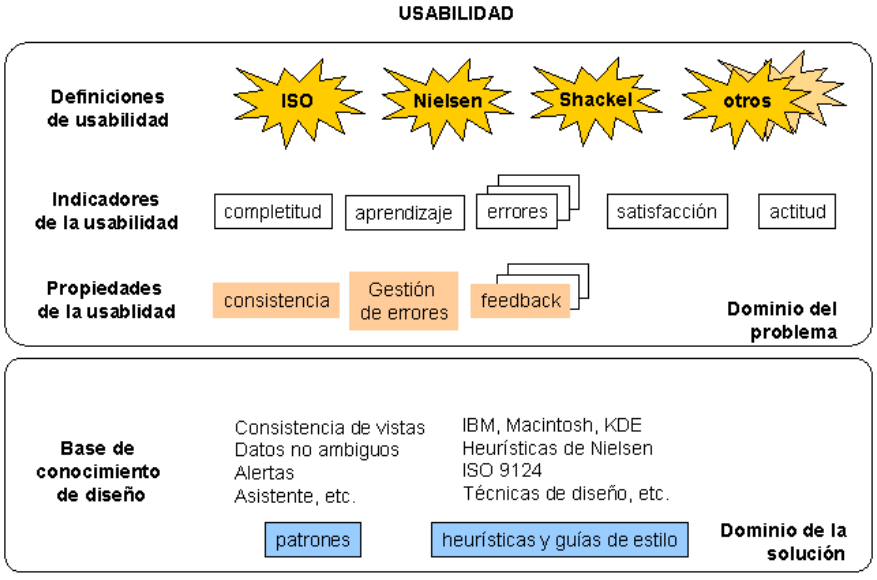


Figura 3-12. Ingeniería de la usabilidad según (Folmer et al., 2004)

Hay autores que abogan por considerar la calidad en uso desde el principio y que además han detectado que existen elementos a nivel arquitectónico que determinan el posterior desarrollo y mantenimiento del produc-

to una vez éste haya sido desarrollado (Bass et al, 2001; Folmer et al., 2004), es decir, han caracterizado la usabilidad en base a requisitos (Fig. 3-12).

Otras propuestas, también en el mismo sentido, apuestan por hacer una especificación de un producto partiendo de la consideración de la calidad en uso y de la utilización de diferentes niveles en un modelo de calidad asociado, y junto con dicha especificación del modelo de calidad existen datos (Fig. 3-13) que permiten la estimación de las métricas y que estas sean el punto de partida para el desarrollo del producto software (Seffah et al., 2001).

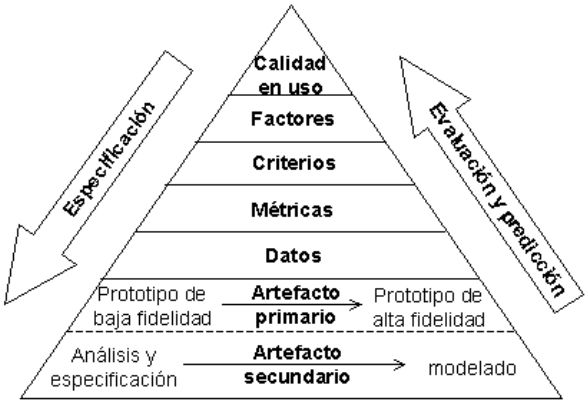


Figura 3-13. Ingeniería de la usabilidad según (Seffah et al., 2001)

En estas contribuciones viene a reflejarse lo que quedó recogido en la Fig. 3-2 de este mismo capítulo, donde se mostraban las relaciones entre las necesidades de un usuario al realizarse un producto software y la calidad de uso del mismo y, entre unas y otras, la presencia de procesos de evaluación (validación y verificación) y la existencia de métricas internas y externas de calidad con las que llevar a cabo dicha evaluación.

Por tanto, sabemos cuándo establecer criterios de usabilidad y qué factores y criterios es necesario establecer. Éstos últimos serían los identificados en los diferentes estándares internacionales recogidos, y en función de ellos tenemos identificadas métricas para comprobar el logro de algunos de ellos, siendo posible hacer uso de los métodos de evaluación de la usabilidad. Pero, aún así, al ingeniero con poca experiencia en desarrollos le falta dirección a la hora de seleccionar qué y cómo desarrollar su proyecto, además algunos de los criterios establecidos en los estándares siguen siendo de alto nivel y su consecución no resulta sencilla si se carece de la experiencia necesaria para el desarrollo en cualquiera de sus fases.

Seguidamente comentaremos algunos de los métodos de evaluación de la usabilidad más utilizados, y veremos cómo, en determinados casos y pa-

ra comprobar el cumplimiento de muchos de los criterios de calidad, puede no ser necesaria la disponibilidad de valores cuantitativos para determinar su cumplimiento y puede bastar con información cualitativa de su logro. No en vano los métodos de evaluación más usados son el de la evaluación heurística (Nielsen, 2005) y la realización de informes informales frente a otros más elaborados.

3.4.2 Evaluación de la calidad en uso

Los métodos de evaluación de la usabilidad pueden clasificarse atendiendo a diferentes criterios (Horn, 1996; Nielsen, 1994; Ivory, 2001). Para llevar a cabo nuestra recopilación apostaremos por aquel criterio que asocia método de evaluación con fase del ciclo de vida al que va asociado. Atendiendo a este criterio, cuando menos, el lector tendrá respuesta a la pregunta cuándo aplicar dicho método de evaluación. Las fases del ciclo de vida consideradas son las mostradas en la Fig. 3-1 de este mismo capítulo.

Habrá que tener en cuenta que en las etapas tempranas del proceso de desarrollo y durante el proceso de diseño, los cambios son relativamente baratos de hacer. Cuando más haya progresado el proceso de desarrollo, y cuando más completamente esté definido, más cara será la introducción de cambios. Así pues, es importante comenzar la evaluación lo más antes posible. En el mismo sentido, aumentar la usabilidad de un producto software, según la IPO, pasa por aplicar un DCU, analizaremos seguidamente qué conlleva dicho diseño y qué métodos de evaluación se utilizan en cada fase de su aplicación y extraigamos las conclusiones oportunas de llevar a cabo esta revisión.

3.4.2.1 Evaluación y requisitos

Una primera labor básica a llevar a cabo al poner en práctica una metodología que se diga centrada en el usuario pasa por entender y especificar el contexto de uso. El propósito es el de identificar, clarificar y registrar las características de los implicados, sus tareas y el entorno físico y organizacional en el que el sistema operará. Para contemplar estos propósitos se consideran:

i. las **entrevistas de campo** estructuradas (*Contextual Inquiry*), este método se basa en tres principios fundamentales: entender el contexto en el que un producto es utilizado es esencial para un diseño elegante, el usuario es parte del proceso de diseño y el proceso de diseño para la usabilidad, incluyendo los métodos para su mejora como la indagación en el contexto y test de usabilidad, deben tener un objeto o focus.

ii. la **observación** (*Naturalistic Observation, Etnographic Study/Field Observation y Proactive Field Study*), que consideran a un investigador que ve a los usuarios trabajar y toma nota de la actividad que tiene lugar. La observación puede ser directa o indirecta. Las observaciones ayudan a entender los requisitos del usuario así como otros elementos a incorporar en diseños preliminares.

iii. la **aproximación a los grupos** (*Focus Groups, Brainstorming*), el interés en la aproximación a los grupos es distinto al de la aproximación al contexto de uso habitual y el modo en el que se van a relacionar durante determinadas sesiones los integrantes del mismo van a ser clave para la obtención de valiosa información cualitativa. Porque ahora ya no interesa tanto el modo en que los implicados con el sistema interactúan con el mismo como sus intereses, motivaciones, expectativas, etc. enfrentados a los de otros usuarios.

iv. y la aproximación al individuo con la **realización de encuestas, cuestionarios y entrevistas** (*surveys, questionnaires* e *interviews*), donde se recoge un obvio interés por las actitudes, percepciones y motivaciones individuales de los usuarios. Se pretende profundizar en matices y particularidades. La confrontación y generación de ideas que se buscaban con anterioridad han podido limitar la expresividad de algunos usuarios. Mediante una aproximación al individuo se busca un mejor entendimiento del modelo mental de los usuarios y los implicados en general, algo que con las aproximaciones anteriores ya se había comenzado a perfilar. Algunos ejemplos de cuestionarios relacionados con la usabilidad son: SUS (Brooke, 1996). WAMMI (WAMMI, 2004) o el cuestionario para la evaluación de la usabilidad (Schneiderman, 1998).

El objetivo, en todos los casos, es entrar en contacto directo con los implicados en el sistema y hacerlo en el contexto en el que interactúan con el.

3.4.2.2 Evaluación y diseño

De los métodos de indagación en los procesos de desarrollo centrado en el usuario se ha obtenido la suficiente información en cuanto a los requisitos de usuario, producto y organización como para abordar la materialización de esa información en un prototipo del sistema a desarrollar. Sin embargo, diseñadores y usuarios tienen distintos modelos mentales del sistema o producto, pues hablan un lenguaje distinto. Hablar del desarrollo del concepto y de las especificaciones del sistema significa hablar de la puesta en común de esos lenguajes bien distintos para unificarlos poste-

riormente en un prototipo a partir del cual va a evolucionar el diseño, siempre con la participación del usuario.

Dentro de esta sección pueden incluirse diferentes técnicas unas buscarán el desarrollo del concepto y de las especificaciones del sistema, otras la categorización y el prototipado:

i. El concepto del producto es una descripción preliminar del producto futuro para ser evaluada como tal y para ser usada como la base de las siguientes fases del desarrollo del producto. Dentro del grupo de métodos para caracterizar el concepto del producto pueden incluirse: **técnicas para la composición de interfaces** (análisis de vínculos, análisis de *layouts*, análisis de tareas), **técnicas de diseño paralelo**, **elaboración de escenarios** (*Storyboarding, Scenarios*), cuadros de asignación de tareas (*Task Allocation Charts*), **análisis de tareas** (*Task analysis*) y **matrices de funcionalidad** (*Functionality Matrix*).

ii. Mediante la categorización y el prototipado va a ser posible hacer las decisiones de diseño más explícitas. Los diseñadores explorarán los diversos conceptos de diseño antes de optar por uno en particular, pero siempre posibilitando la incorporación de la retroalimentación del usuario en las etapas tempranas del proceso de desarrollo así como la evaluación de diversas iteraciones de un diseño y de diseños alternativos.

Las técnicas de **categorización por tarjetas** (*Card Sorting*) y mediante **diagramas de afinidad** (*Affinity Diagram*) se encuentran a mitad de camino entre la generación de ideas y la necesidad de organizarlas y clasificarlas para hacer uso de las mismas.

Un *prototipo* es un modelo (representación, demostración o simulación) modificable de un sistema planificado, probablemente incluyendo su interfaz y su funcionalidad de entradas y salidas. El prototipo modela el producto final y permite efectuar un test sobre determinados atributos del mismo sin necesidad de que esté disponible. Se trata, simplemente, de testar utilizando un modelo. Se pueden hacer múltiples clasificaciones de los productos atendiendo a diferentes criterios (funcionalidad, fidelidad, propósito, etc.) en base a ello se habla de **prototipado horizontal** (*Horizontal Prototyping*), **prototipado vertical** (*Vertical Prototyping*), prototipado de alta fidelidad (*High-Fidelity Prototyping*), **prototipado de baja fidelidad** (*Low-Fidelity Prototyping*), **prototipado de papel** (*Paper prototyping*), **prototipado rápido** (*Rapid Prototyping*), **prototipado por vídeo** (*Video Prototyping*), **prototipado por Mago de Oz** (*Wizard of Oz Prototype*), etc.

3.4.2.3 Evaluación e implementación

Una vez la implementación del producto ya ha comenzado y existen versiones disponibles del mismo es posible seguir evaluando las implementaciones realizadas frente a los requisitos. El objetivo es el de capturar información de los usuarios finales y otras fuentes representativas sobre la validez del diseño y la implementación realizada. Esta retroalimentación contribuye a la mejora del proceso. Se evalúa si los objetivos sobre la organización y los implicados han sido logrados o no y se monitoriza el uso a largo plazo del sistema. En este ámbito puede hablarse de inspecciones de usabilidad y de test de usabilidad.

Habitualmente, se habla de inspecciones en lugar de evaluaciones, si bien las variantes más conocidas de las **Inspecciones Formales** de usabilidad (*Formal Usability Inspection*) son las **Evaluación Heurística** (*Heuristic Evaluation*) y los **Análisis cognitivos** (*Walkthrough*). Las inspecciones pueden llevarse a cabo de forma específica sobre estándares o bien sobre las características específicas del sistema o la consistencia entre los distintos elementos o partes en caso de que las hubiera.

Frecuentemente el término test de usabilidad, se utiliza de forma indiscriminada para referirse a cualquier técnica utilizada para evaluar un producto o sistema. (Rubin, 1994) llama test de usabilidad al proceso que emplea a participantes representativos de la población objetivo para evaluar el grado en el que un producto se encuentra con los criterios de usabilidad. La inclusión de usuarios representativos, entonces, permite eliminar la etiqueta de valuación experta, revisión cognitiva y demás, que no requieren, aunque pueden considerar, la participación del usuario en el desarrollo de las metodologías en cuestión. El objetivo con los test de usabilidad es la identificación y resolución de definiciones de usabilidad existentes en un sistema.

Hay cuatro tipos de test de usabilidad: el exploratorio, el de valoración, el de validación y el de comprobación, aunque todos ellos no son propios de las últimas fases del ciclo de vida.

El **test exploratorio** se conduce en etapas tempranas del ciclo de desarrollo, cuando el producto se encuentra aún en las etapas preliminares a su definición y diseño. El objetivo es examinar o explorar la efectividad de los conceptos de diseño preliminares.

El **test de valoración** es, probablemente, el más habitual entre los tests de usabilidad. De entre todos ellos, es el que resulta más simple y directo para los profesionales noveles de la usabilidad en cuanto a dirección y procedimiento. Los tests de valoración se conducen en etapas tempranas o medias del proceso de desarrollo en el ciclo de desarrollo del producto, normalmente tras establecer el diseño de alto nivel del producto.

El **test de validación** o verificación se suele llevar a cabo en etapas finales del proceso de diseño con el propósito de certificar la usabilidad del producto. Al contrario que los tests anteriores, este test se realiza muy próximo al lanzamiento del producto. El propósito es establecer que el producto alcanza tal estándar de forma previa a su lanzamiento y, en caso de no hacerlo, establecer las razones por las que no lo hace, las referencias se originan habitualmente de los objetivos de usabilidad desarrollados en las etapas tempranas del proceso, que, a su vez, se fundamentan en encuestas de mercado, entrevistas con usuarios o simples estimaciones llevadas a cabo por un equipo de desarrollo.

El **test de comparación** no está asociado con ningún punto específico del ciclo de desarrollo. En las etapas tempranas, puede ser utilizado para comparar estilos de interfaz radicalmente distintos a través de un test exploratorio para ver cuál tiene un mayor potencial con la población objetivo. Hacia las etapas medias del proceso de desarrollo, el test de comparación puede ser utilizado para medir la efectividad de un elemento individual en la interfaz. Hacia el final del ciclo de vida, el test de comparación puede ser utilizado para comprobar cómo se presenta el producto a lanzar frente al de la competencia. Se trata, pues, de comparar dos o más alternativas de diseño.

Algunas variantes de test de usabilidad son: **el método tutorado** (*Coaching Method*), **el método de seguimiento** (*Shadowing Method*), **el método de instrucción previa** (*Teaching Method*), **el aprendizaje por descubrimiento conjunto** (*Co-discovery Learning*) o **el test retrospectivo** (*Retrospective Testing*).

El **protocolo de pensamiento manifestado** (*Thinking Aloud Protocol*) es una técnica popular utilizada durante el test de usabilidad. Durante el transcurso del test, donde el participante está realizando una tarea como parte de un escenario de usuario, se solicita de este que exprese en voz alta sus pensamientos, sensaciones y opiniones mientras interactúa con el producto.

El **descubrimiento conjunto** (*Co-discovery Protocol*) es una variante del test de usabilidad en la que dos participantes intentan realizar las tareas juntos mientras están siendo observados. La ventaja de este método sobre el del pensamiento manifestado se refleja por partida doble, primero porque en el lugar de trabajo, la mayoría de las personas cuentan con alguien que pueda ayudarles y, segundo porque la interacción entre los dos participantes puede revelar más información que un único participante expresando en voz alta sus pensamientos.

En la Tabla 3-2 se recoge un resumen de los métodos recopilados en las anteriores secciones. Los mismos se asocian con diferentes fases en el ciclo de vida de un producto software (R: requisitos, D: diseño, C: codifica-

ción o implementación, T: testeo y M: mantenimiento o depuración). La información contenida en la Tabla 3-2 debe tomarse como una mera sugerencia de utilización de cada método en cada fase ya que algunos de los métodos pueden utilizarse, en función de cómo sean elaborados en otras fases del desarrollo además de en las mostradas.

Tabla 3-2. Tabla resumen de diferentes métodos de evaluación de la usabilidad

Método de Evaluación	Fases del ciclo de vida				
	R	D	C	T	M
Estudios de campo inducidos	✔				
Revisión cognitiva conjunta		✔			
Método de instrucción previa		✔	✔	✔	
Método de seguimiento		✔	✔	✔	
Descubrimiento conjunto		✔	✔	✔	
Protocolo de preguntas		✔	✔	✔	
Escenario basado en checklists		✔	✔	✔	✔
Evaluación heurística		✔	✔	✔	✔
Pensamiento manifestado		✔	✔	✔	✔
Revisión cognitiva		✔	✔	✔	✔
Método tutorado		✔	✔	✔	✔
Realización de medidas		✔	✔	✔	✔
Entrevistas		✔	✔	✔	✔
Testeo retrospectivo		✔	✔	✔	✔
Testeo remoto		✔	✔	✔	✔
Inspección de características			✔	✔	✔
Grupos orientados				✔	✔
Cuestionarios				✔	✔
Estudio etnográfico				✔	✔
Sesiones de registro reales				✔	✔

La correcta puesta en práctica de la gran mayoría de los métodos reseñados requieren de experiencia para llevarlos a cabo con éxito y obtener información útil. Una experiencia que, en algunos casos, está soportada por la utilización de herramientas software, algunas de esas herramientas serán presentadas en la siguiente sección.

3.5 Evaluación cuantitativa

La evaluación cuantitativa de un producto software viene determinada por la disponibilidad de métricas que, asociadas al modelo de calidad, permitan la ponderación y estimación de las características ofrecidas por los productos software desarrollados. Dicha disponibilidad es una realidad y, en la actualidad, hay definidas gran cantidad de métricas que se pueden clasificar atendiendo a diferentes criterios.

Una taxonomía posible es la presentada por (Constantine et al., 1999). En ella las métricas se dividen en diferentes grupos atendiéndose a:

- métricas **estructurales**, que atienden a propiedades superficiales. Estas métricas son las más simples de computar (p. e., número de componentes visuales o *widgets* en la pantalla o diálogo, cantidad y distribución de espacios en blanco entre *widgets*, alineación de *widgets* entre ellos, número de pantallas adyacentes o diálogos directamente alcanzables, la más larga cadena de transiciones posible entre pantallas o diálogos, etc.).
- métricas **semánticas**, que son sensibles al contenido y tienen en cuenta la naturaleza de los componentes de interfaz de usuario y sus características en términos de su función, significado y operación. Las métricas semánticas sirven para estimar aspectos de diseño de la interfaz de usuario que dependen de los conceptos y acciones que representan los propios componentes visuales y de cómo los usuarios interpretan componentes e interrelaciones.
- métricas **procedurales**, que son sensibles a la tarea y, por tanto, se basan en aspectos relacionados con las tareas y escenarios que pueden llevarse a cabo con una interfaz de usuario.

Las métricas propuesta, por su forma de obtención, son básicamente procedurales, y no tanto estructurales, como pudiera pensarse fruto de pretender asociar usabilidad y mera interfaz de usuario. En este sentido, una colección de métricas puramente estructurales son las propuestas en (Ivory, 2001) donde se recogen 157 métricas asociadas a una metodología que pretende aportar métodos con los que abordar la evaluación automática de la usabilidad de sitios web. En la misma se responde a un modelo de calidad que se basa en el establecimiento de un modelo de calidad centrado en la usabilidad donde, la misma, se descompone en seis criterios de los cuales cinco son concretos y están relacionados con el contenido del sitio, su estructura y navegación, su diseño visual, su interactividad y su funcionalidad, el último criterio es uno de evaluación general. Las métricas asocia-

das con este modelo de calidad son estructurales y, por ello, pueden evaluarse automáticamente simplemente inspeccionando el código HTML que tienen detrás. Las ventajas que permite la evaluación automática de interfaces de usuario son evidentes al no precisarse de usuarios, de experimentos o de expertos para llevarse a cabo, por el contrario la simple evaluación automática de interfaces de usuario no es suficiente y así lo reconocen los propios presentadores de esta propuesta. Una idea similar, pero centrada en la usabilidad y aportando la definición de un Lenguaje de Descripción de Guías de estilo útil para la evaluación de aspectos relacionados con la usabilidad de interfaces de usuario Web, ha sido la propuesta en (Bereikdar et al., 2004). Las herramientas que dan soporte a los análisis mencionados son son respectivamente WebTango y Kwaresmi.

Otra posible taxonomía, más reciente, es la presentada con la propuesta WQM (Calero et al., 2004). En ella las métricas se clasifican atendiendo a tres criterios determinados por factores de calidad (extraídos de estándares internacionales), características del producto software, en este caso concreto se consideran características, propias de sitios y páginas web, y, como novedad, las fases del ciclo de vida de un producto software.

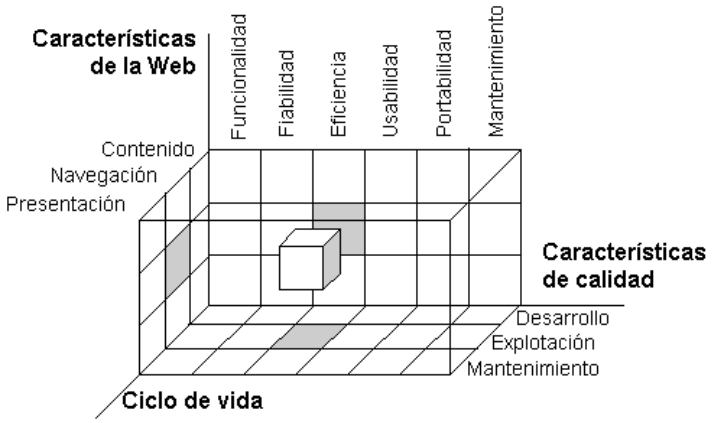


Figura 3-14. Modelo de calidad WQM (Coral et al., 2004)

Tabla 3-3. Métricas asociadas a la usabilidad en el estándar ISO 9241-11

Consideración de la usabilidad	Medidas de efectividad	Medidas de eficiencia	Medidas de satisfacción
General	Porcentaje de objetivos logrados Porcentaje de usuarios que completaron la tarea con éxito Media de terminación de tareas	Tiempo para completar una tarea Tareas completadas por unidad de tiempo Coste monetario de realización de las tareas	Grado de satisfacción Índice de uso Frecuencia de quejas
Usuarios experimentados	Número de tareas esenciales realizadas Porcenataje de funciones relevantes utilizadas	Eficiencia relativa comparada con usuarios expertos	Grado de satisfacción con características habituales
Uso habitual	Porcentaje de tareas realizadas con éxito en su primer intento	Tiempo necesario en el primer intento Eficiencia relativa en un primer intento	Tasa de utilización voluntaria
Uso poco frecuente		Tiempo invertido en reaprender funciones Número de erroes reiterados	Frecuencia de reutilización
Requisitos de soporte	Número de referencias a la documentación Número de llamadas para solicitar soporte Número de invocaciones a la ayuda	Tiempo productivo Tiempo de aprendizaje	
Facilidades de aprendizaje	Número de funciones aprendidas Porcentaje de usuarios capaces de realizar las tareas	Tiempo de aprendizaje Tiempo necesario para volver a aprender Eficiencia relativa mientras se aprende	Grado de facilidad de aprendizaje
Tolerancia a fallos	Porcentaje de errores corregidos o de los que informó el sistema Núemro de errores permitidos	Tiempo invertido en subsanar errores	Grado de manejo de errores
Legibilidad	Porcentaje de palabras leídas correctamente a una distancia de visualización normal		

Tabla 3-4. Tabla de métricas propuestas en el estándar ISO 9126-4

Nombre de la métrica	Método de evaluación	Fórmula
Métricas relacionadas con la efectividad		
Efectividad de la tarea	Test con usuarios	$M1 = \left\|1 - \sum A_i\right\|$ donde A_i es el valor proporcional de cada error en la salida obtenida al realizar la tarea.
Completitud de la tarea	Test con usuarios	$X = A/B$ donde A es el número de tareas completadas y B el número de tareas intentadas.
Frecuencia de error	Test con usuarios	$X = A/T$ donde A es el número de errores cometidos por el usuario y T es el tiempo o número de tareas.
Métricas relacionadas con la productividad		
Tiempo en llevar a cabo la tarea	Test con usuarios	$X = T_a$ donde T_a es el tiempo invertido en la tarea.
Eficiencia de la tarea	Test con usuarios	$X = M1/T$ donde M1 es la efectividad de la tarea y T es el tiempo invertido en llevarla a cabo.
Productividad económica	Test con usuarios	$X = M1/C$ donde M1 es la efectividad de la tarea y C es el coste total de la tarea.
Proporción de tiempo productivo	Test con usuarios	$X = T_a/T_b$ donde T_a es el tiempo productivo y T_b es el tiempo en completar la tarea incluyendo tiempo en que se consulta la ayuda, se subsanan errores o se determina cómo llevar a cabo la tarea.
Eficiencia relativa del usuario	Test con usuarios	$X = A/B$ donde A es la eficiencia en llevar a cabo la tarea por usuarios corrientes y B es el invertido por usuarios expertos.
Métricas relacionadas con la seguridad		
Seguridad y salud del usuario	Registro de utilización	$X = 1 - A/B$ donde A es el número de problemas de salud reseñados por los usuarios y B es el número total de usuarios.
Seguridad de la gente afectada por el uso del sistema	Registro de utilización	$X = 1 - A/B$ donde A es el número de personas escogidas al azar y B es el número total de personas potencialmente afectadas por el sistema.
Daños económicos	Registro de utilización	$X = 1 - A/B$ donde A es el número de ocurrencias de daños económicos y B es el número total de usos.

Daños software	Registro de utilización	$X = 1 - A/B$ donde A es el número de ocurrencias de defectos software y B es el número total de utilizaciones.
Métricas relacionadas con la satisfacción		
Escala de satisfacción	Cuestionarios al usuario	$X = A/B$ donde A es el valor de escala producido por el cuestionario y B es la media de la población.
Cuestionario de satisfacción	Cuestionarios al usuario	$X = \sum A_i / n$ donde A_i es la respuesta a una cuestión y n es el número de respuestas.
Aceptación del producto	Observación	$X = A/B$ donde A es el número de veces que funciones de software específico son utilizadas y B es el número de veces que las mismas son invocadas.

Tabla 3-5. Métricas relacionadas con la usabilidad de (Constantine et al., 1999)

Métrica	Descripción	Fórmula	
Essential Efficiency (simplicidad)	Es la relación entre el número de pasos necesarios para abordar un caso de uso idealmente y los pasos necesarios que deben darse para abordar ese caso de uso dada una interfaz de usuario concreta	$EE = 100\frac{S_{essential}}{S_{enacted}}$	
Task Concordance (eficiencia y simplicidad)	Estima la distribución de la complejidad de las tareas usando una interfaz donde se ha considerado la frecuencia de realización esperada de las distintas tareas posibles	$TC = 100\frac{D}{P}$	donde N es el número de tareas que están siendo consideradas y P = N(N-1)/2
Task visibility (visibilidad)	Estima la relación entre visibilidad de las características y las capacidades necesarias para completar una tarea o conjunto de tareas dada	$TV = 100\left(\frac{1}{S_{total}}\sum_{\forall i} V_i\right)$	
Layout Uniformity (organización espacial)	Es una métrica estructural que nos pondera cuán organizados están los elementos que determinan una interfaz de usuario	$LU = 100\left(1 - \frac{\left(N_h + N_w + N_t + N_l + N_b + N_r\right) - M}{6N_c - M}\right)$ donde es el número de componentes visuales en la pantalla y el resto de índices se corresponden con los diferentes alineamientos. M es $2 + 2\lceil 2\sqrt{N_c} \rceil$	
Visual Coherente (comprensión, aprendizaje y uso)	Estima la bondad de los agrupamientos de componentes, es decir, mide la proximidad entre componentes relacionados y la separación entre componentes no relacionados	$VC = 100\left(\frac{\sum_{\forall k} G_k}{\sum_{\forall k} N_k (N_k - 1)/2}\right)$	donde $G_k = \sum_{\forall i,j \mid i \leftarrow j} R_{i,j}$ y N_c es el número de componentes visuales

En las tablas anteriores, Tabla 3-3 a 3-5 se recogen las aportaciones que consideramos más interesantes que han sido realizadas en materia de métricas relacionadas con usabilidad. Las mismas han sido presentadas en (Constantine et al., 1999) y en estándares internacionales relacionados con usabilidad, concretamente en la ISO 9126-4 y el ISO 9241-11. Pero, adolecen de las carencias ya comentadas: falta de relación con la experiencia, validación, puesta en práctica y utilidad cuando el producto ya está avanzado.

3.6 Evaluación cualitativa

Las métricas del apartado previo encajan con los modelos de calidad presentados anteriormente, pero, por ello, por provenir de estándares, tienen el inconveniente de que son recogidas de forma muy general y se echa en falta cierta concreción en lo que a su uso y puesta en práctica se refiere. Además, no se asocian con los métodos necesarios para su obtención, análisis y estimación de sus respectivas contribuciones a cada uno de los criterios de los que depende el factor de calidad que constituye principal interés para esta tesis doctoral, la usabilidad o calidad en uso.

Las consideraciones anteriores se unen a que:

- el hecho de que los métodos de evaluación más utilizados pasen por las inspecciones heurísticas, los recorridos cognitivos y las listas de comprobación. Así como que la puesta en práctica de estos métodos cuesta menos que otros métodos basados en la evaluación utilizando usuarios reales y permite, igualmente, identificar mayor cantidad de fallos de usabilidad en menor tiempo (Nielsen, 2003),

- el hecho de que la experiencia en desarrollo de interfaces, por ejemplo, la recogida utilizando guías de estilo, persista casi sin alteraciones, en un 90%, aunque haya sido elaborada hace más de quince años (Nielsen, 2005),

- y si, además, es necesario seguir trabajando en documentar y validar la experiencia de forma que la documentación de la misma redunde positivamente en los productos software realizados (Nielsen, 2004).

Todo ello hace que lleguemos a la conclusión de que, en muchos casos, una evaluación cualitativa de la usabilidad es tanto, o más potente inicialmente, que lo puede ser una evaluación realizada con usuarios reales y ba-

sada en la realización de experimentos. Si además, se cuenta con un modelo de calidad basado en criterios de calidad próximos al usuario y con ello se hace referencia a aquellos criterios que el usuario puede manejar y comprender, y dicho modelo lleva asociado experiencia junto con, y en caso de disponibilidad, métricas se consigue mayor potencia de actuación.

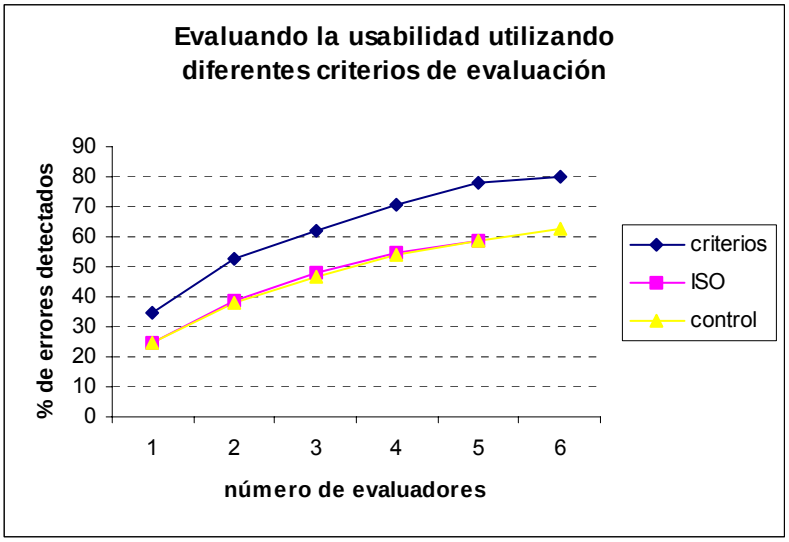


Figura 3-15. Resultado de evaluar la usabilidad utilizando criterios ergonómicos y estándares internacionales (Bastien et al, 1996)

Hay experimentos que respaldan las conclusiones recogidas anteriormente, por ejemplo véanse (Bastien et al., 1996), la Fig. 3-15 y (Chevalier et al., 2003). En la Fig. 3-15 se muestra gráficamente los resultados de un experimento recogido en (Bastien et al., 1996), de él se desprende que hay criterios, los criterios ergonómicos, que no se contemplan en estándares, al menos directamente, y que permiten identificar más problemas que otros criterios más generales y que sí están recogidos en los estándares internacionales. Cabe resaltar también que estos criterios no se han utilizado en la elaboración de modelos de calidad. Una de las propuestas recogidas en esta tesis doctoral pasa por la consideración de esos criterios cualitativos en la elaboración de un modelo de calidad, pero antes de comentar esta propuesta recogeremos qué herramientas representativas están desarrollándose en la actualidad y cuáles son las tendencias en este momento, con ello refrendaremos la idoneidad y el hueco que ocupa nuestra propuesta en lo que se refiere al ámbito de la consideración y estudio de la calidad.

3.7 Herramientas disponibles

Ante la necesidad de disponer de herramientas de soporte a gran parte de los métodos y conceptos recogidos en este capítulo se han desarrollado y se están desarrollando en la actualidad diferentes herramientas software que dan soporte a la elaboración de modelos de calidad, permiten la elaboración de prototipos, y asisten en la evaluación de la usabilidad, ya sea mediante el registrado de acciones y su posterior procesamiento o ya sea calculando métricas desarrolladas específicamente para ser evaluadas. Las iniciativas más reseñables serán recogidas seguidamente.

3.7.1 Elaboración de modelos de calidad

El editor QUIM (Harkikat et al, 2003) es una herramienta de usabilidad utilizada para manipular el marco de trabajo QUIM (*Quality in Use Integrated Map*), propuesto por el Grupo de Ingeniería del Software centrado en el usuario (HCSE en la Universidad de Concordia). QUIM es un editor (Fig. 3-16) útil para todos aquellos que quieran aprender cómo evaluar la usabilidad, cómo predecir o especificar la calidad y cómo integrar la usabilidad y el ciclo de vida de desarrollo de software.

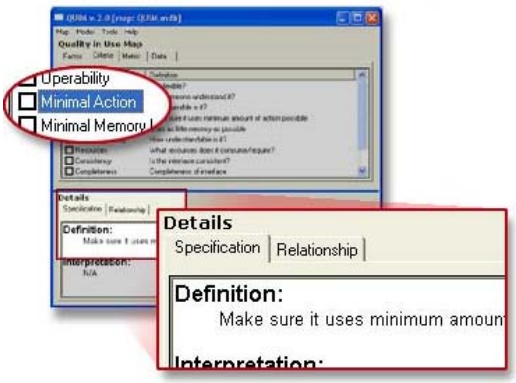


Figura 3-16. Editor QUIM (Harkikat et al., 2003)

QUIM consta de una colección de métricas de usabilidad que se han recopilado en una base de conocimiento desarrollada por diferentes organizaciones. Con este editor se pueden seleccionar factores, criterios y métricas que pueden ser relevantes en función del proyecto. Los elementos seleccionados pueden ser utilizados para evaluar la usabilidad siguiendo un protocolo basado en *checklists*.

Otra herramienta interesante es QM (Carvallo et al., 2004), la misma proporciona la funcionalidad necesaria para definir un modelo de calidad jerárquico, basado en el estándar ISO 9126-1, y las métricas para sus factores de calidad. Una de las ventajas en la usabilidad de la herramienta es que muestra una ventana que permanentemente muestra el modelo de calidad que se está construyendo utilizando una estructura de árbol. QM (Fig. 3-17) permite construir el modelo a partir de un árbol, y seleccionar aquellos factores de calidad que el usuario desee considerar. La definición del factor seleccionado aparece en una ventana debajo de modelo de calidad.

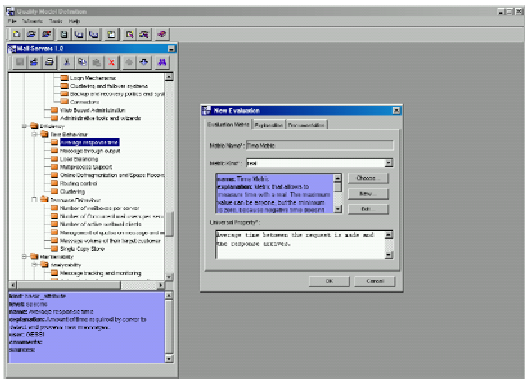


Figura 3-17. Editor de modelos de calidad QM (Carvallo et al., 2004)

3.7.2 Realización de prototipos

Otro conjunto de herramientas que permiten la evaluación temprana de interfaces de usuario son aquellas que facilitan labores de elaboración de prototipos, algunas de las que más han podido influir en nuestra propuesta ya fueron mencionadas en el capítulo anterior, en el apartado destinado a comentar herramientas que permitían diseñar interfaces de usuario utilizando algún tipo de experiencia. En este sentido, Denim, Suede, Damask y UIPilot dentro de las no comerciales y OlivaNova dentro del apartado de herramientas comerciales permiten la realización de prototipos de interfaz de usuario.

3.7.3 Soporte a la evaluación de la usabilidad

Dentro del grupo de herramientas destinadas a facilitar labores de evaluación podemos hacer diferentes grupos en función de los criterios utilizados para realizar dicha evaluación. Los dos grupos que se identifican son, por

un lado, aquellos destinados a registrar y analizar las acciones que el usuario realiza utilizando el sistema, fundamentalmente, lo que se registra es la navegación que el usuario lleva a cabo y, bien se estudia dicha navegación, bien se compara con el modelo de tareas asociado con el caso de uso ligado a la tarea propuesta al usuario o usuarios en la actividad de evaluación. Por otro lado, estaría un grupo importante de propuestas destinadas a dar soporte a labores de evaluación automática de interfaces de usuario.

3.7.3.1 Registrado y tratamiento de acciones

RemUSINE (Paterno et al., 1999; 2000) es un entorno diseñado y desarrollado en el Grupo de Interacción Persona-Ordenador del ISTI-CNR. La idea presentada con RemUSINE ha sido utilizada, y extendida al ámbito Web (Paganelli et al., 2003), para la evaluación de la usabilidad mediante el tratamiento de registros reales y su comparación con los modelos de tareas asociados a los sistemas evaluados. En este sentido, esta herramienta es la única que encontramos en la que se utilicen modelos para llevar a cabo labores de evaluación. WebRemUSINE (Paganelli et al, 2003) realiza evaluaciones automáticas de un sitio web basándose en la utilización de métricas, con ello se pueden identificar problemas potenciales de usabilidad (Fig. 3-18).

Las entradas a la herramienta son el modelo de tareas y los ficheros de registro almacenados durante una sesión de test. La herramienta está compuesta de tres módulos: un editor de modelos de tareas (CTTE), una herramienta de registro y visualización de la interacción llevada a cabo por parte del usuario y un elemento de cotejado de modelo y registrado de interacciones.

Figura 3-18. Ventana principal ofrecida en WebRemUsine (Paganelli et al., 2003)

Otras herramientas que permiten registrar las acciones que lleve a cabo el usuario manipulando un sistema y facilitan su posterior tratamiento y manipulación, aunque no estén basadas en la utilización de ningún modelo, permitiendo detectar problemas relacionados con la usabilidad del sistema son: DRUM (Macleod et al., 1993) que permite la manipulación de registros basados en la utilización de vídeo; o Kaldi (Al-Qaimari et al., 1999), Guitester (Okada et al., 1999), ObSys (Gellner et al., 2001) y WebQuilt que permiten registrar y analizar la interacción del usuario con el sistema utilizando el ratón o el teclado.

3.7.3.2 Evaluación automática de calidad

Un estudio más detallado de las propuestas realizadas en el ámbito de la evaluación automática de la calidad puede encontrarse en (Ivory, 2001). Las herramientas más interesantes, y que consideramos dignas de mención en este ámbito ámbito WebTango (Ivory, 2001) y Kwaresmi (Bereikdar et al., 2002).

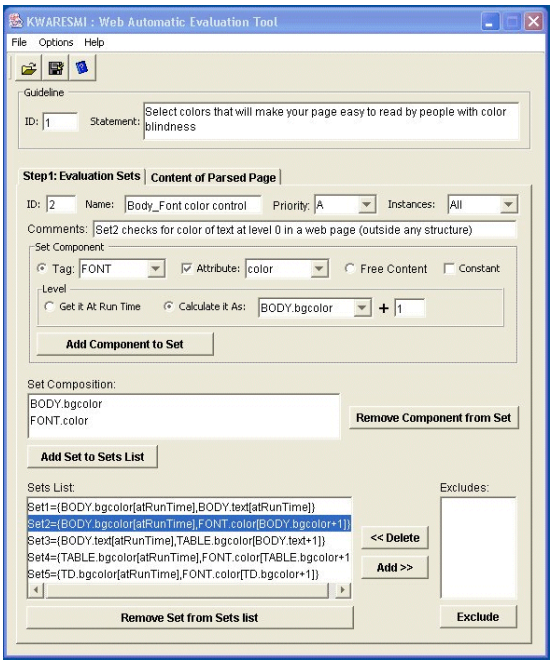


Figura 3-19. Evaluación de la accesibilidad: Kwaresmi (Bereikdar, et al., 2002)

WebTango evalua sitios y páginas web comparando el valor obtenido por un conjunto de métricas con el valor guardado para esas mismas métricas fruto de una evaluación previa de un conjunto extenso de páginas y sitios web. Las métricas con las que trabaja son estructurales y reciben su

valor de un análisis exhaustivo del código HTML asociado con cada página que constituya el sitio.

Kwaresmi (Fig. 3-19) es otra herramienta propuesta para la evaluación automática de sitios web que, de forma similar a la anterior, analiza el código HTML asociado con cada página web que pretenda evaluarse. La evaluación se basa en la comprobación de guías de estilo formuladas mediante un Lenguaje de Descripción de Guias de Estilo (GDL).

Los objetivos de evaluación de WebTango y Kwaresmi son diferentes, la primera se centra en la usabilidad, mientras que la segunda tiene como objetivo la accesibilidad.

3.8 Propuesta de consideración de la calidad

El principal problema que se observa al considerar la calidad, o cualquiera de los factores de los que depende, es que según el colectivo al que pertenezca, ya sea usuario, ingeniero o experto en usabilidad, quien maneje dicho concepto tiende a asociarlo a aquellos intereses que más afines le sean. Esa maleabilidad del concepto calidad lo hace poco preciso y condiciona su logro. En IS la tendencia es la industrialización del software y lo importante, al menos en primera instancia, es la facilidad de mantenimiento. En IPO lo que prima es la usabilidad.

No hay un modelo de calidad cerrado, pero si que existe un primer nivel de descomposición de calidad extendido y aceptado, es el asociado al estándar 9126. A partir de ahí se apuesta por la utilización de modelos de calidad abiertos donde cada uno considere aquello que más le pueda interesar. La solución, como en otros ámbitos, pasa por la consideración del contexto o del dominio donde se considere la solución.

Siendo las dimensiones que ofrece la interfaz de usuario tanto proceso como producto es necesario no perder de vista ninguna de ellas, para ello consideramos que los criterios ergonómicos son los idóneos para proseguir con el desarrollo de un modelo de calidad que continuando abierto permita tener presentes una serie de características que permitan abordar tanto el desarrollo como el proceso de evaluación e indirectamente el mantenimiento.

Los criterios que, en primera instancia, juzgamos dignos de tener en cuenta son los criterios ergonómicos de (Bastien et al., 1993; Apéndice D). Los mismos pueden integrarse con el modelo de calidad propuesto en el estándar 9126.

En la Tabla 3-5 se recoge el modelo de calidad inicial del que se ha partido para la elaboración de nuestra propuesta.

Tabla 3-6. Propuesta de modelo de calidad centrado en la usabilidad

Calidad	Factor[1]	Criterio[2]	Cont.[3]
Usability	Understandability	compatibility	H
		Legibility	M
		Prompting	M
		immediate feed-back	H
		significance of codes and behaviour	H
		helpfulness	M
	Learnability	Grouping	H
		minimal actions	H
		conciseness	L
		information density	M
		consistency	H
	Operability	explicit user action	H
		user control	H
		user experience's	H
		flexibility	M
		error protection	H
		quality of error messages	L
		error correction	M
		privacy policies	L
		accesibility	H

Las principales características que ofrece el modelo de calidad recogido en la Tabla 3-6 son:

- **Está basado en estándares internacionales**, como la ISO 9126 o la ISO 9241 relacionados con la calidad y en otros como la ISO 13407 y la 18529.
- **Es abierto**, pueden incorporarse nuevos criterios de calidad, ligándose a los ya identificados en el mismo,

[1] A nivel de factores el modelo de calidad propuesto considera aquellos criterios establecidos en el estándar ISO 9126 para la usabilidad

[2] A nivel de criterios las características que se consideran son los criterios ergonómicos de (Bastien et al., 1993) a los que se han añadido otros criterios adicionales (privacy policies, accessibility y helpfulness)

[3] Hace referencia a la contribución estimada por cada criterio a la usabilidad final

- **Es cerrado**, y puede utilizarse para dar asistencia a la puesta en practica de técnicas de evaluación de interfaces de usuario (Bastien et al., 1999; Chevalier et al., 2003).
- La **utilización de criterios ergonómicos** se ha demostrado eficiente no solamente para abordar el diseño de interfaces de usuario destinados a la Web (Scapin et al., 2000), asi como para otras interfaces: interfaces tradicionales (Scapin et al., 1997) y entornos virtuales (Bach et al., 200).
- **Permite la puesta en práctica de técnicas de diseño centradas en el usuario** ya que es posible utilizando criterios ergonómicos describir la calidad y así ha quedado demostrado empíricamente con la elaboración de diferentes experimentos, llevados a cabo con usuarios reales (Scapin et al., 1997; Chevalier et al., 2003).
- A su vez los criterios ergonómicos se han demostrado útiles para labores de **organización de guías de estilo** relacionados con la usabilidad (Scapin et al., 2000), eso nos permite su utilización para organizar otros tipos de experiencia.

Las características anteriores justifican la elección de los criterios ergonómicos como un elemento que puede integrarse dentro de una propuesta de modelo de calidad centrada en la usabilidad, donde se consideran los aspectos relacionados con la concepción del producto software desde las dimensiones que ofrece como producto y como proceso. Los criterios ergonómicos aparecen recogidos en la Fig. 3-20.

En nuestra propuesta, a los criterios ergonómicos, se han añadido nuevos criterios, asociándolos a los criterios de calidad definidos en el estándar 9126 y ligados, directamente, con la usabilidad, es decir, con las características de *understandability*, *learnability* y *operability*. Los criterios añadidos tienen en consideración aspectos relacionados con las facilidades de ayuda ofrecidas (*helpfulness*), con las políticas de privaciadad que determinan el tratamiento de la información gestionada a través de la interfaz de usuario (*policy privacy*) y con la accesibilidad de la propia interfaz ofrecida (*accessibility*).

El modelo de calidad propuesto es un elemento vertebrador de calidad y de experiencia de diseño y, en función de ello, puede utilizarse en dos direcciones: (a) para dar soporte a labores de evaluación de las interfaces generadas teniendo presente siempre características de calidad, que además están próximas al usuario y a su forma de percibir la calidad; y (b) a los criterios, obviamente pueden ligarse métricas, nuestro siguiente propuesta pasará por no solamente limitarse a esto sino a utilizar el modelo de calidad para organizar la experiencia disponible, en cualquiera de sus forma-

tos, para afrontar labores de desarrollo de interfaces de usuario, en este sentido, habrá que análizar qué tipo de información puede ser la más idónea, ésta será una labor tratada en el capítulo siguiente, dedicado a tratar la documentación de la experiencia y la importancia de su disponibilidad para dar lugar a interfaces de calidad.

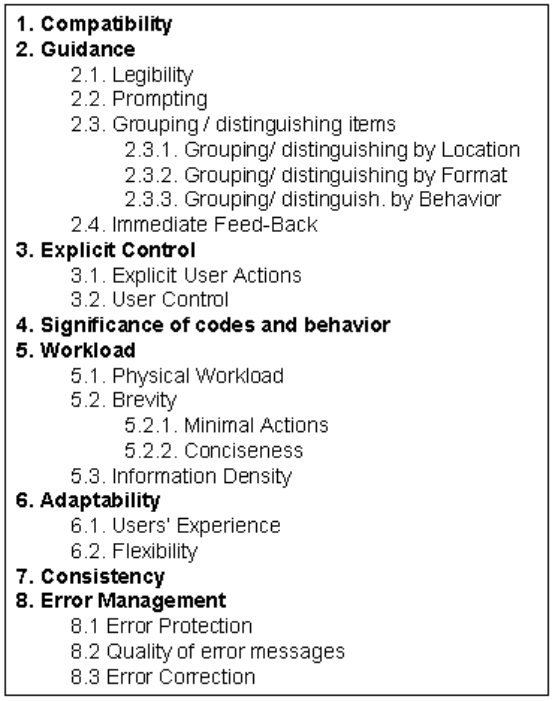


Figura 3-20. Criterios ergonómicos propuestos en (Bastien et al., 1993)

Por último, comentar también que la herramienta que se desarrolle estará basada en este modelo de calidad y, por tanto, deberá considerar la calidad desde el principio y facilitar las labores de gestión del modelo que al tratarse de un modelo de calidad abierto podrá ser modificado.

3.8.1 Criterios relacionados con la *understandability*

Describiremos seguidamente cada uno de los criterios ergonómicos tomados de (Bastien et al., 1993) y aquellos criterios que consideramos relevantes de ser considerados, para la elaboración del modelo de calidad sugerido en la sección anterior y relacionados con la facilidad para entender una aplicación o producto software (*understandability*).

Los criterios considerados en está sección son: la compatibilidad (compatibility), la legibilidad (*legibility*), la dirección y asistencia al usuario (*prompting*), el *feedback* inmediato, el significado de código y comportamiento (*significance of codes and behaviour*) y el suministro o disponibilidad de la ayuda considerada necesaria (*helpfulness*).

La asociación de estos criterios con la dimensión de la usabilidad que ligada a la facilidad para entender el producto software se ha realizado de forma empírica, atendiendo y analizando las guías de estilo que van asociadas con el logro de dichos criterios y los beneficios respecto a la usabilidad que se pueden conseguir con su uso.

Por compatibilidad (compatibility) se entiende el establecimiento o búsqueda de una adecuada relación entre las características del usuario y la organización de las entradas, salidas y diálogo por otro. Por ejemplo, los diálogos deberían reflejar las estructuras de datos o de organización que percibe el usuario de forma natural.

Por legibilidad (legibility) se denotan las características léxicas de la información presentada en pantalla. Por ejemplo las etiquetas utilizadas, los títulos o los cursores.

Por asistencia y dirección del usuario (*prompting*) entendemos aquellas facilidades que se proporcionan para facilitar la realización de operaciones de entrada y, junto con ellas, aquellas otras facilidades por las que se hace saber al usuario las diferentes alternativas cuando varias acciones son posibles. Un ejemplo seria visualizar las unidades de medida para cada valor que tenga que ser proporcionado o indicarle cómo tiene que proporcionar determinados datos que deben seguir un formato.

El *feedback* inmediato tiene relación con la respuesta que ofrece el sistema a las acciones que realiza el usuario. Un ejemplo de este criterio lo constituye aquellas guías de estilo que recomienda que ante entrada del usuario la misma se traduzca en un elemento visible, si los datos introducidos fueran confidenciales o debieran estar ocultos se deberán mostrar asociados a un símbolo que impida su visualización por ejemplo mediante el uso de asteriscos.

El criterio de ofrecer significado a códigos y comportamiento (*significance of codes and behaviour*) implica que debe existir una correspondencia entre cada término y el símbolo metafórico con el que se hace referencia al mismo. Por ejemplo, en este sentido, los códigos debieran ser significativos y familiares más que arbitrarios.

Finalmente, en esta sección dedicada a hacer referencia a aquellos criterios que sean significativos respecto a proporcionar facilidades relacionadas con la comprensión del producto software hemos incluido el criterio *helpfulness*, con él se hace alusión a aquella facilidad por la que cuando se

estime adecuado incluir una sección que ofrezca ayuda o información adicional sobre la tarea o tareas disponibles de forma explícita.

3.8.2 Criterios relacionados con la *learnability*

En este apartado se recogerán las definiciones correspondientes a aquellos criterios de calidad que tienen que ver con la consecución de facilidades adicionales relacionadas con el esfuerzo necesario para aprender el uso de un producto software (*learnability*).

Dentro de este grupo de criterios consideramos adecuado tener presentes subcriterios como son: el agrupamiento (*grouping*) en cualquiera de sus modalidades (forma o localización) de información y controles, logro de acciones mínimas (*minimal actions*), la concisión (*conciseness*), la densidad de información (*information density*) y la consistencia (*consistency*).

Con el subcriterio agrupamiento (*grouping*) recalcamos la necesidad de que para conseguir que el usuario aprenda y recuerde el funcionamieno de un producto software es necesario que se haya meditado sobre la organización de los elementos de interfaz que lo constituyen, ya sean estos elementos de información o de acción. Para ello se debe proporcionar una clara distinción visual de áreas que ofrezcan diferentes funciones, por ejemplo.

La reducción a un número de acciones mínimas (*minimal actions*) para acometer tareas y objetivos del usuario es esencial para aprender y recordar las tareas que el usuario puede hacer utilizando el producto software. Por ejemplo, la reducción del número de pasos necesarios para hacer una selección de un menú será una característica deseable.

La concisión (*conciseness*) es el criterio por el que se considera el uso de todas aquellas recomendaciones para lograr que sobre el usuario recaiga la mínima sobrecarga cognitiva y perceptiva posible cuando utilice un producto software. Por ejemplo cuando un usuario deba proporcionar información que la misma pueda proporcionarse utilizando mnemotécnicos o abreviaturas, o que determinados valores de relleno no tengan que facilitarse y aparezcan automáticamente.

La densidad de información (*information density*) sería un criterio que, relacionado con el anterior, guarda repercusión sobre la carga cognitiva y perceptiva que debe ejercer el usuario al utilizar un producto software, pero en este caso se hace hincapié en la información como conjunto, en lugar de como elemento individual. Por ejemplo, el suministro de la información necesaria y útil para realizar una transacción y la no sobrecarga con información adicional no necesaria es una recomendación en esta dirección.

La consistencia (*consistency*) es el criterio por el que se utilizan directrices de diseño similares para abordar el diseño de contextos similares. Por

ejemplo, el hecho de que los títulos de las diferentes ventanas asociadas a un producto software siempre aparezcan en una misma lugar o que los formatos de fuente, pantalla o acceso a la funcionalidad son algunos ejemplos significativos en la dirección del logro de la consistencia.

3.8.3 Criterios relacionados con la *operability*

El tercer grupo de criterios deseables de un producto software viene marcado por otra de las componentes que pueden asociarse, en función de las definiciones disponibles en los estándares internacionales, con la usabilidad. Este criterio no es otro que la facilidad para operar con el propio producto software (*operability*).

Asociado a la facilidad de operación de un producto software hemos asociado los siguientes subcriterios: disponibilidad de acciones de usuario de forma explícita (*explicit user action*), control de actividades por parte del usuario (*user control*), consideración de la experiencia que pueda adquirir o tener el usuario (*user experience*), la flexibilidad del producto software (*flexibility*), la gestión de los errores (error protection, quality error messages y error correction), las políticas de privacidad y seguridad (*privacy policies*) y la accesibilidad (*accessibility*).

Con la disposición de acciones al usuario de forma explícita (*explicit user action*) se recalca la asociación entre el procesamiento que realiza el artefacto y las acciones que realiza con él el usuario. Por ejemplo, la necesidad de que el usuario presione la tecla ENTER para iniciar un procesamiento que desencadene una actualización o una impresión es una recomendación en este sentido.

El control del usuario (*user control*) se refiere al hecho de que los usuarios debieran siempre ser los que controlen el procesamiento que se lleve a cabo con el artefacto. La disponibilidad de un botón CANCEL que ofrezca la posibilidad de borrar cualquier cambio hecho por el usuario y restaurar un estado anterior es un ejemplo de consecución de este criterio.

La flexibilidad y la experiencia del usuario son aquellos criterios por los que se consideran recomendaciones que redundan en las facilidades de adaptatividad y adaptación de un producto software. La primera característica sería inherente al producto software, es decir, la adaptación la llevaría a cabo o la sugeriría al usuario el producto automáticamente. La experiencia del usuario, por el contrario, pasaría porque el usuario pudiera modificar el artefacto adaptándolo en función de sus preferencias o experiencia.

La gestión de errores, es decir, su protección (*error protection*), información (*quality of error messages*) y corrección (*error correction*) es otro

criterio que redunda en la calidad de operación que el usuario percibe al hacer uso de un producto software.

Por último, dos criterios más como son las políticas de privacidad y seguridad, y las facilidades relacionadas con el uso de técnicas que redunden en la accesibilidad de acciones y contenidos son también considerados en el modelo de calidad en esta sección, aunque no estén consideradas dentro de la propuesta de criterios ergonómicos.

3.9 Análisis y conclusiones

En este capítulo se ha tratado el concepto de calidad de un producto software y se ha hecho, como viene siendo tónica general en este documento, desde los puntos de vista que más pueden aportar a nuestro interés por desarrollar interfaces de usuario de calidad: IS e IPO.

La calidad se ha demostrado un concepto confuso en su descripción y caracterización, y aún existiendo estándares internacionales los mismos no son completos y quedan a un nivel bastante superficial. En cualquier caso la comunidad internacional reconoce la necesidad y las ventajas consiguientes que tendría la disponibilidad de un modelo de calidad universal.

Una de las contribuciones recogidas en esta tesis doctoral pasa por la proposición de un modelo de calidad centrado en la usabilidad y donde criterios ergonómicos sirvan para ofrecer un modelo abierto, a posteriores identificaciones de criterios relevantes, basado y coherente con en los estándares internacionales disponibles y ligado a la experiencia de diseño. Con esta propuesta se integra conocimiento IPO y técnicas de industrialización de software, es decir, aquellos elementos más idóneos para su utilización de cada una de las disciplinas consideradas.

Las herramientas que se están desarrollando consideran bien la experiencia y su documentación, bien el desarrollo de interfaces de usuario o bien la elaboración de modelos de calidad, pero no permiten la utilización de todas y cada una de estas características. Esta consideración se tendrá en cuenta para seguir incorporando elementos a esta tesis doctoral en posteriores capítulos.

El capítulo finaliza con la presentación y propuesta de un modelo de calidad centrado en la usabilidad. El modelo está elaborado mediante la utilización de criterios de primer nivel de descomposición de la usabilidad, tomados como factores de usabilidad, que están basados en estándares internacionales, y el uso de criterios ergonómicos para potenciar la descomposición. La ventaja de este segundo nivel de descomposición está en la disponibilidad de un conjunto de criterios que ofrecen una versión de eva-

luación y que pueden utilizarse, con éxito probado, para abordar labores de evaluación de interfaces de usuario. Los criterios ergonómicos utilizados han sido integrados en el modelo de calidad propuesto y estarán ligados, en función de los intereses de esta tesis doctoral, con experiencia y métricas (cuando sea disponible). Esta forma de proceder proporcionará un doble beneficio:

- permitirá organizar la experiencia disponible,
- permitirá anticipar el proceso de evaluación y,
- finalmente, el diseñador o analista encontrará soporte en la experiencia documentada y organizada para el logro de características deseables desde el punto de vista de la calidad centrada en la usabilidad.

3.10 Contribuciones relacionadas con este capítulo

En lo relativo al tratamiento del concepto de calidad, de sus modelos y su consideración y logro cabe destacar las siguientes publicaciones:

Capítulos de libro:

- **Montero, F.**, Lozano, M., González, P. Calidad en interfaces de usuario. Capítulo 5 del libro Calidad en el desarrollo y mantenimiento del software. Coordinadores: Mario G. Piattini, Felix O. García. Editorial Ra-Ma. 2003. (ISBN: 84-7897-544-6).
- Lozano, M., **Montero, F.**, García, F.J., Gonzalez, P. Calidad en el desarrollo de aplicaciones web: modelos, métodos de evaluación y métricas de calidad. Capítulo 8 del libro Ingeniería de la web y patrones de diseño. Coordinadores de la obra: Paloma Díaz, Susana Montero e Ignacio Aedo. Pearson Prentice Hall. 2005. (ISBN: 84-205-4609-7)

Participación en congresos y conferencias:

- Fernández-Caballero, A., López-Jaquero, V., **Montero, F**. Métricas de usabilidad y sistemas multiagente en hipermedia adaptativa. IV Congreso Internacional de Interacción Persona-Ordenador. Interacción 2003. Vigo. España. (ISBN: 84-932887-4-8)
- López-Jaquero, V., **Montero, F.**, Fernández-Caballero, A., Lozano, M. Usability metrics in Adaptive Agent-Based Tutoring Systems. 10th International Conference on Human - Computer Interaction (HCI, 2003). 22 a 27 de Julio, Creta, Grecia. 2003.

- **Montero, F.**, Lozano, M., López-Jaquero, V., González, P. A Quality Model For Testing The Usability Of The Web Sites. 10th International Conference on Human - Computer Interaction (HCI, 2003). 22 a 27 de Julio, Creta, Grecia. 2003. Human-Computer Interaction: Theory and Practice (part 1). J. Jacko, C. Stephanidis (Eds.). Lawrence Erlbaum Associates. Londrés, Reino Unido, 2003. (ISBN: 0-8058-4931-9)
- **Montero, F.**, López-Jaquero, V., Lozano, M., González, P. Usability and Web Site Evaluation: Quality Models and User Testing Evaluations. ICEIS (1) 2003: 525-528

Referencias bibliográficas

Bach, C., Scapin, D. L.: Adaptation of Ergonomic Criteria to Human Virtual Environment, in : Ninth IFIP TC13 International Conference on Human-Computer Interaction - INTERACT'03. 2003 Amsterdam: IOS Press, p. 880-883, Zurich, Switzerland, September, 2003

Basili, V.R., Rombach, H.D.: The TAME Project: Towards Improvement-Oriented Software Environments, IEEE Transactions on Software Engineering, vol. SE-14, no. 6, June 1988, pp.758-773

Bastien J.M.C., Scapin, D., Leulier, C.: The ergonomic criteria and the ISO/DIS 9241-10 dialogue principles: a pilot comparison in an evaluation task. Interacting with Computers 11(3): 299-322. 1999

Bastien, J. M. C., & Scapin, D. L. Ergonomic criteria for the evaluation of human-computer interfaces (Report No. 156). Rocquencourt, France: Institut National de Recherche en Informatique et en Automatique. 1993

Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture, M. A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. In Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces. 2002

Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture, M.: A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. In Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002, Kluwer Academics Pub., Dordrecht, 2002

Boehm, B. W. Software Engineering Economics. Prentice-Hall, 1981

Boehm, B.W., Brown, J.R. Lipow, M. MacLeod, G.J., Merritt, M.J.: Characteristics of Software Quality, North-Holland, N.Y., 1978

Brajnik G. Towards valid quality models for websites, in Proc. Human Factors and the Web, 7th Conference, Madison, WI, June 2001

Brooke, J. SUS: A Quick and Dirty Usability Scale. In: P.W. Jordan, B. Thomas, B.A. Weerdmeester & I.L. McClelland (Eds.), *Usability Evaluation in Industry*. London: Taylor & Francis. (Also see http://www.cee.hw.ac.uk/~ph/sus.html). 1996

Calero, C., Ruiz, J., Piattini, M. A Web Metrics Survey Using WQM. ICWE 2004: 147-160

Carvallo, J.P., Franch, X., Grau, G., Quer. C. QM: A Tool for Building Software Quality Models. Demonstration to be presented during the International Requirements Engineering Conference (RE'04), Kyoto (Japan), September 2004

Chevalier, A., Ivory, M.: Can Novice Designers Apply Usability Criteria and Recommendations to Make Web Sites Easier to Use?. In Proceedings of the 10th International

Conference on Human-Computer Interaction, Vol. 1, Crete, Greece, June 22-27, 2003, pp. 773-777

Chevalier, A., Ivory, M.I.: Can Novice Designers Apply Usability Criteria and Recommendations to Make Web Sites Easier to Use?. In Proceedings of the 10th International Conference on Human-Computer Interaction, Vol. 1, Crete, Greece, June 22-27, 2003

Constantine, L.L. and Lockwood, L.A.D, Software for Use: A Practical Guide to the Models and Methods of Usage Centered design, Addison-Wesley, 1999.

De Marco T. Controlling software projects, Yourdon Press, New York, 1982

Dromey, R. G.: A Model for Software Product Quality. IEEE Trans. Software Eng. 21(2): 146-162 1995

Folmer, E., Bosch, J. Architecting for usability, Journal of systems and software issue 70-1, Pages 61-78, January 2004.

Fenton N.E. and Lawrence Pfleeger S. Software metrics, 2nd ed., International Thompson Publishing Company, 1997

Grady, Robert B. Practical Software Metrics for Project Managementand Process Improvement. Englewood Cliffs, N.J.: Prentice-Hall,1992.

Hom, J. The usability methods toolbox. http://www.hest.com/~jthom/usability. 1996

ISO/IEC Standard, ISO-9126 Software Product Evaluation - Quality Characteristics and Guidelines for Their Use. 1991

Ivory, M. An Empirical Foundation for Automated Web Interface Evaluation. Computer Science Division, UC Berkeley, December 2001.

Ivory, M. Y., Chevalier, A.: A Study of Automated Web Site Evaluation Tools. Technical Report UW-CSE-02-10-01, University of Washington, Department of Computer Science and Engineering, 2002

Lin, J.: Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In Conference Supplement of UIST 2003: ACM Symposium on User Interface Software and Technology, Vancouver, BC, Canada, Nov. 2–5, 2003

McCall, J.A., Richards, P.K. and Walters, G.F. "Factors in Software Quality", RADC TR-77-369, US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055, 1977

Newman, M.W., Lin, J., Hong, J.I., Landay, J.A.: DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. In Human-Computer Interaction, 2003

Nielsen, J., and Mack, R. L. (Eds.). Usability Inspection Methods. John Wiley & Sons, New York, NY, ISBN 0-471-01877-5. 1994

Nielsen, J. Jakob Nielsen's Alertbox, September 13, 2004:The Need for Web Design Standards

Nielsen, J. Jakob Nielsen's Alertbox: Durability of Usability Guidelines. January 17, 2005

Pastor, O., Molina, J.C., Iborra, E.: Automated production of fully functional applications with OlivaNova Model Execution, ERCIM News No.57, April 2004. http://www.care-t.com/

Paulk, M. C.; Curtis, B.; Chrissis, M., Chrissis, B., Weber, C., Capability Maturity Model for Software, Version 1.1Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993

Pressman, Roger S, Ingeniería del software, un enfoque práctico, McGraw Hill 1995

Puerta, A., Michelete, M., Mak, A.: The UI pilot: a model-based tool to guide early interface design. International Conference on Intelligent User Interfaces. Proceedings of the 10th international conference on Intelligent user interfaces. 2005

Scapin, D.L., Bastien, J.M.C., Ergonomic criteria for evaluating the ergonomic quality of interactive systems, Behaviour & Information Technology, 16, 1997, pp. 220–231

Scapin, D.L., Vanderdonckt, J., Farenc, Ch., Bastide, R., Bastien, Ch., Leulier, C., Mariage, C., and Palanque, P., Transferring Knowledge of User Interfaces Guidelines to the

Web, in Proceedings of the International Workshop on Tools for Working with Guidelines, Springer-Verlag, London, 2000

Scapin, D.L., Vanderdonckt, J., Farenc, Ch., Bastide, R., Bastien, Ch., Leulier, C., Mariage, C., and Palanque, P., Transferring Knowledge of User Interfaces Guidelines to the Web, in Proceedings of the International Workshop on Tools for Working with Guidelines, Springer-Verlag, London, 2000

Seffah, A., Kline, R., Donyaee, M. An Integrated Framework for Usability Measurement. 12th International Conference on Software QualityOttawa, Canada October 28-30, 2002

Sinha, A., Klemmer, S., Chen, J., Landay, J.A., Chen, C.: SUEDE: Iterative, Informal Prototyping for Speech Interfaces. Video poster in Extended Abstracts of Human Factors in Computing Systems: CHI 2001, Seattle, WA, March 31-April 5, 2001

Schneiderman, B. (1997). Designing the user interface. Reading, MA:Addison-Wesley. 1997

SPICE: The Theory and Practice of Software Process Improvement and Capability Determination. Khaled El Emam (Editor), Jean-Normand Drouin (Editor), Walcélio Melo (Editor), Alec Dorling (Foreword by). Wiley-IEEE Computer Society Press. 1997

WAMMI: http://www.wammi.com

Waterson, S.: WebQuilt: A Visual Analysis Tool for Understanding Web Usability Clickstream Data. Masters Report, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA. May 2002

Zeist, B.v an , Hendriks, P.: Quint2: The Extended ISO Model of Software Quality. http://www.serc.nl/quint-book/. 1996

Capítulo 4 La experiencia y el diseño de interfaces de usuario

La experiencia no tiene valor ético alguno,
es simplemente el nombre que damos a nuestros errores.
Oscar Wilde (Dramaturgo y novelista irlandés)

4.1 La experiencia

La experiencia es el conocimiento adquirido en el transcurso de nuestra vida, nos ayuda a tomar mejores decisiones ponderando posibilidades y riesgos; aprendemos en la intimidad de nuestro ser, en la familia, con los amigos, a través de la lectura, en la escuela, en el trabajo. A pesar de todo esto, muchas veces seguimos tomando decisiones a la ligera, cometiendo los mismos errores y cerrando nuestros oídos a los consejos que nos brindan personas con más visión que nosotros.

Donald A. Norman, profesor y director del Instituto de Ciencias Cognitivas en la Universidad de California, resaltó la relación entre experiencia y aprendizaje (Norman et al., 1986). También señaló que el aprendizaje y la memoria están muy relacionados, aunque el aprendizaje es más que el simple recuerdo; involucra la habilidad en la ejecución de una tarea con destreza.

Norman tomó como punto importante la obtención de información mediada por la experiencia, aspecto frecuentemente utilizado en el discurso sobre desarrollo cognitivo, en función de lo cual la experiencia obtenida por una persona *puede llevarle a un mejor desempeño* (potencial) al realizar una tarea, tal como caminar, solucionar problemas, conducir un automóvil o una silla de ruedas, no importa sus capacidades o limitaciones.

La construcción de interfaces de usuario es un proceso complejo, el cual requiere la utilización de métodos y técnicas ingenieriles y de experiencia multidisciplinar (Downey, 2003). Estos métodos y técnicas no deben ser solamente aplicados a la construcción del software que las implementa, sino también al proceso de diseño de las mismas. Este diseño no solamente consiste en la determinación de la disposición de los elementos componentes de una interfaz, con un sentido estético. Es más importante que dicho diseño incluya también todo el proceso de identificación de las tareas a realizar por el usuario y la evaluación de la forma en que la interfaz de una aplicación permite llevar a cabo eficientemente dichas tareas.

A lo largo del presente capítulo se realiza un recorrido por las diferentes formas de recopilar la experiencia y, actualmente, disponibles para abordar el diseño y desarrollo de interfaces de usuario. Conjuntamente, se identificarán limitaciones en los métodos disponibles en función de cómo y para qué se utilizan. Principios, reglas de diseño y estándares constituirán la primera parte del capítulo, como principal exponente de unas guías de estilo que en unos casos necesitan experiencia, por parte del diseñador, para su puesta en práctica y en otros son demasiado específicas y concretas. Se tratará, posteriormente, los *patrones* como una forma de documentación de la experiencia, que habiendo dado muy buenos resultados en el terreno del diseño y desarrollo orientado a objetos (Gamma et al., 1995), también se utiliza en IPO desde hace unos años, pero en su uso existen todavía muchas preguntas abiertas que afectan a su puesta en práctica y adopción con garantías.

4.2 Las guías de estilo

Para poder asegurar consistencia a través de las diferentes partes de un sistema o a través de una familia de sistemas, es fundamental para los ingenieros basar sus diseños en un conjunto de principios y directrices. Esto les permite transferir sus conocimientos a los usuarios de la interfaz, dentro de un producto y a todas las aplicaciones en que trabajan. Por este motivo es tan importante para las organizaciones que desarrollan software disponer de una guía que puedan seguir sus desarrolladores. Estas guías se denominan *guías de estilo* y varían mucho en sus objetivos.

Las guías de estilo permiten a los diseñadores tener marcos generales de diseño que les pueden ayudar a tomar decisiones correctas en sus diseños. Estas guías pueden adoptar una gran variedad de formas y se pueden obtener en diferentes sitios como, por ejemplo en artículos de revistas académicas, profesionales o comerciales que dan una buena referencia del estado actual en cuanto a práctica y experiencia.

La distinción que podemos establecer, desde el principio, en la experiencia que podemos recopilar utilizando guías de estilo vendrá dada por la consideración que se puede hacer de la interfaz como producto y como proceso.

Además, continuando con la dicotomía seguida en los capítulos previos, dos objetivos distintos se han perseguido en IS y en IPO en lo que se refiere a recopilar la experiencia. Por un lado, la experiencia recopilada en IS no se ha limitado a la propia documentación o recopilación de la misma, sino que se ha pretendido que esta experiencia pudiera ser incorporada en

el proceso, con intenciones de reutilización, ya fuera a nivel de implementación, de diseño o de análisis de requisitos. Sin embargo, en IPO no parece, en función de las iniciativas que se irán presentando en los apartados pertenecientes a este capítulo, que ese haya sido el objetivo. Más bien al contrario, se ha apostado tradicionalmente por la mera documentación de la experiencia y la posibilidad de que la misma esté disponible, incluso, para que el usuario final pueda involucrarse más en un proceso participativo de diseño o desarrollo.

Seguidamente, principios, reglas de diseño y estándares serán comentados, como elementos propios que pueden incluirse dentro de lo que se entiende por guías de estilo.

4.2.1 Principios

Los diseñadores difieren en el número de principios de diseño que deben considerarse importantes. Un *principio* es una sentencia en un sentido muy amplio que normalmente está basado en la investigación hecha de cómo las personas aprenden y trabajan. En cualquier caso, lo realmente importante en cuanto a lo que ofrecen los principios de diseño, está en la consideración y en el establecimiento de una filosofía de búsqueda de la unificación que se traduce en la intención de lograr retos relacionados con la calidad que, en función de su labor, se plantean diseñadores y desarrolladores.

En función de su definición, muchos principios de diseño son similares y, en algunas ocasiones, aparentemente contradictorios, pudiendo ser útiles para determinados diseños y no para otros. Es por ello, que la aplicación de principios de diseño es una labor complicada para diseñadores sin experiencia, resultando más fácil su aprendizaje por la observación de su uso que por la propia definición asociada al principio.

En cuanto a los principios asociados con el desarrollo de interfaces de usuario, contemplando esta labor como proceso (Gould et al., 1985) establecieron tres principios para el logro de usabilidad:

(1) resaltaron la necesidad del diseñador de poner especial interés en conocer al usuario y sus tareas,

(2) el diseñador debe realizar evaluaciones empíricas con usuarios a través de la utilización de prototipos y simulaciones, recogiendo y analizando reacciones e información asociada a dicho proceso de evaluación empírica,

(3) el diseñador debe eliminar los posibles errores detectados mediante la puesta en práctica de un proceso de diseño iterativo. Posteriores aportaciones, recogidas en el capítulo anterior (Nielsen, 1993; Mayhew, 1999;

Constantine et al., 1999), han venido a refinar estos principios básicos en los que descansa el diseño para el logro de la usabilidad.

Tabla 4-1. Recopilación de principios de diseño y autores

Norman (1990)	Nielsen (1990)	Mayhew (1992)	Shneiderman (1992)	Tognazzini (2003)
Visibility	Visibility of system status	User compatibility	Strive for consistency	Anticipation
Good conceptual model	Match between system and the real world	Product compatibility	Enable frequent users to use shortcuts	Autonomy
Good mapping	User control and freedom	Task compatibility	Offer informative feedback	Color Blindness
Feedback	Consistency and standards	Consistency	Design dialog to yield closure	Consistency
	Error prevention	Familiarity	Offer simple error handling	Defaults
	Recognition rather than recall	Simplicity	Permit easy reversal of actions	Efficiency of the user
	Flewibility and efficiency of use	Direct manipulation	Support internal locus of control	Explorable interfaces
	Aesthetic and minimalist design	Control	Reduce shor-term memory load	Fitts's law
	Help users recognize, diagnose and recover from errors	WYSIWYG		Human interface objects
	Help and documentation	Flexibility		Latency reduction
		Responsiveness		Learnability
		Invisible technology		Use of metaphors
		Robustness		Protect user's work
		Protection		Readability
		Easy of learning and ease of use		Track state
				Visible navigation

Las principios de diseño que debe aportar el producto guardan analogía con lo que, en el capítulo anterior, se venía denominando modelo de calidad. Así, (Norman, 1990; Nielsen, 1990; Mayhew, 1992; Shneiderman, 1992; Tognazzini, 2003) han propuesto diferentes principios que deben tenerse en cuenta en el desarrollo de interfaces de usuario. Como se puede observar (*véase* Tabla 4-1) muchas de los principios recogidos no son otros que los comentados en el capítulo anterior, y están relacionados con los factores y criterios de calidad y usabilidad deseables en un producto soft-

ware, por ejemplo, *feedback*, visibilidad, gestión de errores, consistencia, facilidad de ser aprendido o leído, eficiencia, etc. Debido a esta asociación directa podremos asociar modelos de calidad y experiencia en sus diferentes formas y no sólo con métricas que es lo habitual al especificar modelos de calidad. La realidad es que al final estos principios necesitan ser puestos en práctica y es ahí donde el diseñador necesitará saber *cómo*.

La principal limitación de los principios está en que no siempre hay información directamente asociada con su puesta en práctica, de ahí que la experiencia del diseñador o desarrollador que finalmente pretenda utilizarlos será indispensable para su adecuado uso. Los principios son recomendaciones deseables, pero altamente abstractas y generales, directrices de diseño relacionadas con dichos principios son las *reglas de diseño*, las mismas ayudan a conseguir los logros de calidad asociados con los principios.

4.2.2 Reglas de diseño

Las reglas de diseño recomiendan acciones basándose en un conjunto de principios de diseño. Generalmente, las reglas de diseño son más específicas que los principios y requieren menos experiencia para entenderlas e interpretarlas que éstos, debido a su menor nivel de abstracción.

Tabla 4-2. Ejemplo de guías de estilo recogidas en (Smith et al, 1986)

(1.0/4) **Fast Response**: *ensure that the computer will acknowledge data entry actions rapidly, so that users are not slowed or paced by delays in computer response; for normal operation, delays in displayed feedback should not exceed 0.2 seconds.*
(…)
(4.0/5) **Only Necessary Information Displayed**: *tailor the display for any transaction to the current information requirements of the user, so that only relevant data are displayed.*

Las reglas de diseño, recopiladas de esta forma, son una herramienta de documentación y gestión de experiencia. La organización de las mismas está determinada, como se recogía anteriormente, por atribuciones funcionales asociadas con la interfaz de usuario. Recientemente, existen propuestas de organización de las reglas de diseño mediante criterios de calidad, concretamente *criterios ergonómicos* (Scapin et al., 2000), el objetivo perseguido con ello es facilitar labores de evaluación de la calidad a través del uso de las propias guías de estilo. Indirectamente, también se logra, de esta forma, aportar experiencia al logro de criterios o principios de calidad.

Específicamente relacionadas con el diseño de interfaces de usuario, (Smith and Mosier, 1986; Brown, 1988) propusieron una recopilación de guías de estilo relacionadas con aspectos funcionales atribuibles a la interfaz de usuario como son: la entrada de datos, la visualización de datos, las secuencias de control, la asistencia al usuario, la transmisión de datos y la protección de datos. Esta recopilación de guías de estilo no estaba asociada directamente con principios de diseño, sino con aspectos funcionales, y la documentación de las guías seguía la forma mostrada en los ejemplos recogidos en la Tabla 4-2. En algunos casos, junto con la regla de diseño, se incluyen ejemplos concretos de utilización de la guía de estilo y excepciones a su utilización.

El principal reto en lo que respecta a la gestión de la experiencia recopilada utilizando reglas de diseño no está en la mera documentación, p.e. GuideBook (Ogawa et al., 1995), sino en su recuperación y utilización efectiva, es decir en trabajar con ellas de forma eficiente. Iniciativas en este sentido han pasado por la utilización de sistemas hipertexto, como HyperSAM (Ianella, 1995), sistemas hipermedia, como SIERRA (Vanderdonckt, 1995); por la disponibilidad de bases de datos web distribuidas donde las reglas de diseño se agrupen y se asocien con métodos de evaluación de interfaces de usuario, como facilita MetroWeb (Mariage et al, 2004) recopilando experiencia útil en ámbitos web; y por sistemas que utilicen razonamiento basado en casos (Kolodner, 1992) para la recuperación de la regla/s de diseño más adecuadas en función del contexto, como Ask-Jef (Barber et al, 1992) o GUIDE (Henninger et al., 1995). Con todo y con ello, resultan si cabe, desde el punto de vista asociado a esta tesis doctoral, más interesantes aquellas propuestas en las que se asocia interfaz de usuario con conocimiento proveniente de otros elementos de diseño de la aplicación, entiéndase como tal el diseño conceptual, la estructura del diálogo, la elección de los elementos de interacción concreta, y la caracterización del usuario, como ejemplo de herramientas en esta dirección destacan EXPOSE (Gorny, 1995) y Diades-II (Dilli et al., 1995).

4.2.3 Estándares

Un *estándar* es un requisito, regla o recomendación basada en principios probados y en la práctica. Representa un acuerdo de un grupo de profesionales oficialmente autorizados a nivel local, nacional o internacional. Se han desarrollado estándares relacionados con el proceso de diseño de la interacción persona-ordenador y con las características que debe ofrecer un producto en ese mismo terreno, estos últimos son más abundantes que los primeros.

Algunos ejemplos destacados de estándares relacionados con la línea de investigación en la que se centra esta tesis doctoral, junto con las instituciones internacionales, nacionales, militares o comerciales que los han propuesto se recogen en la Tabla 4-3, recogida seguidamente.

Tabla 4-3. Distintas organizaciones y ejemplos de estándares

Carácter del estándar	Organiación (Ejemplo)
Internacional	- Organización Internacional para la Estandarización. ej. ISO9241: Requisitos ergonómicos para trabajar en una oficina con terminales, ISO9126: Calidad de un producto (modelo de calidad, métricas internas, externas y de calidad en uso), ISO13407: diseño de sistemas interactivos siguiendo procesos centrados en el usuario, ISO14598: evaluación de productos software
	- Consorcio WorldWide Web (W3C) – desarrollo de especificaciones, herramientas, recomendaciones para su uso en el desarrollo de productos para la Web, ej. accesibilidad
Nacional	- Instituto de Estándares Nacionales Americanos (ANSI)
	- Sociedad de Ergonomía y Factores Humanos (HFES)
	- Institución de Estándares Británica (BSI)
	- Ente Nacional Italiano para la Unificación (UNI)
	- Instituto Alemán para la normalización (DIN)
Militar	- Departamento de defensa (DoD) (ej. ESD-TR-86-278)
	- Administración del espacio y aeronáutica nacional (NASA)(ej. NASA-STD-3000)
	- US Military (ej. MIL-STD-1472D, MIL-STK-1472D)
Comercial	- Macintosh Human Interface Guidelines de Apple Computer
	- Common User Access (CUA) de IBM
	- OSF/Motif Style Guide de Open Software Foundation
	- The Windows Interface Guidelines for Design de Microsoft

En cualquier caso, los estándares cuentan con importantes limitaciones a la hora de ponerse en práctica y de esas limitaciones se deriva que no se pueda asegurar que el producto final desarrollado, aunque se hallan considerado estándares en su diseño y desarrollo, sea finalmente usable. Las limitaciones están relacionadas directamente con el nivel de abstracción al que se especifican los estándares y con la especificidad que impone y exige el contexto, ya sea éste el usuario final que utilice la aplicación, el entorno de trabajo donde la aplicación sea utilizada o la estructura y el contenido de la información que la aplicación manipule.

4.3 Los patrones

Además de las diferentes formas de recopilar la experiencia comentadas en los apartados anteriores, el uso del concepto de patrón ha ido apareciendo, de forma natural, en todas aquellas disciplinas donde, llegado el momento, se ha dispuesto de *conocimiento reutilizable* y donde la utilización del mismo de forma reiterada es garantía de éxito.

Un *patrón* es conocimiento recopilado sobre una determinada actividad y, en lo que concierne a esta tesis doctoral, el *conocimiento* estaría relacionado con la elicitación de requisitos, el análisis, el diseño o la implementación de interfaces de usuario. Pero, ¿qué diferencias tienen los patrones con las guías de estilo comentadas en apartados anteriores?. Fundamentalmente, las diferencias vienen dadas por la forma de recopilar o recoger el conocimiento asociado con la experiencia que se desee documentar. Si en el caso de las guías de estilo se recogen junto con la guía, aunque sea de forma excepcional, ejemplos de su uso o se documentan situaciones en las que la utilización de esa recomendación es contraproducente, la documentación del patrón es diferente.

Sea cual sea el formato elegido para la documentación de un patrón, en él, la sección dedicada al contexto en el que la aplicación del patrón resulta significativa se torna relevante, y junto con ella otras secciones (problema y solución) deben documentarse también con especial cuidado.

El concepto de patrón no es de uso exclusivo en Informática, ni siquiera su origen puede asociarse a esa disciplina sino al de la Arquitectura. En cualquier caso, en Informática, concretamente en Ingeniería del Software, los patrones han cobrado otra dimensión distinta a la inicialmente perseguida (*véanse* Tidwell, 1999; Borchers, 2000). Y es que los patrones en Ingeniería del Software son *generativos*, frente a los presentados en Arquitectura o en Interacción Persona-Ordenador donde se centran en proveer facilidades de documentación de la experiencia. Por generativo se denota aquella cualidad que puede adquirir la experiencia y que le confiere capacidad para ser puesta en práctica utilizando, para ello, información proporcionada con la propia experiencia.

4.3.1 Origen y evolución de los patrones

El concepto de patrón tiene su origen en los trabajos realizados por Christopher Alexander dentro del ámbito de la arquitectura (Alexander, 1977; 1979). Alexander afirma que cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada

más de un millón de veces sin tener que presentarse ni siquiera dos veces de la misma forma. Esta misma noción de patrón está presente tanto en desarrollo de software como en interacción persona-ordenador, en ambas actividades hay experiencia disponible sensible de ser documentada.

Una definición más formal del concepto de patrón proporcionada por Alexander (Alexander, 1979) es la siguiente:

Como un elemento en el mundo, cada patrón es una relación entre cierto contexto, cierto sistema de fuerzas que ocurre repetidas veces en ese contexto y cierta configuración espacial que permite que esas fuerzas se resuelvan. Como un elemento de lenguaje, un patrón es una instrucción que muestra la forma en que esta configuración espacial puede usarse, una y otra vez, para resolver ese sistema de fuerzas, donde quiera que el contexto la torne relevante. El patrón es, en suma, al mismo tiempo una cosa que pasa en el mundo y la regla que nos dice cómo crear esa cosa y cuándo debemos crearla. Es tanto un proceso como una cosa; tanto una descripción de una cosa que está viva como una descripción del proceso que generará esa cosa.

En Ingeniería del Software y bajo las consideraciones que ofrece el paradigma orientado a objetos el término patrón ha sido utilizado con gran éxito de resultados. El ejemplo más significativo de ese éxito es la publicación del libro *Design Patterns* de (Gamma et al, 1995), pero antes de esa aparición otros autores ya utilizaron esa misma filosofía para documentar la experiencia. Por ejemplo Coad (Coad, 1992) en el ámbito del diseño orientado a objetos entendía como patrón:

… an abstraction of a doublet, triplet, or other small grouping of classes that is likely to be helpful again and again in object-oriented development (Coad, 1992).

En esta última definición, además de las analogías que se pudieran establecer con las ideas originales asociadas con el término manejado, se advierte una característica distintiva, como es el agrupamiento o *mapping*, que debe existir entre entes que formen parte del contexto en el que ocurre un problema y de cuya asociación se derive la solución propuesta. Si estos entes son clases y las relaciones entre ellas son las propias de una metodología orientada a objetos, estaríamos en el caso de los patrones propuestos en (Gamma et al., 1995; Coad et al., 1997).

En Interacción Persona-Ordenador también se está utilizando, aunque desde hace menos tiempo, el concepto de patrón y con el mismo significado intrínseco. Por patrón se denota experiencia, pero en IPO no se ve, o no quiere verse, la vertiente generativa (Tidwell, 1999; Borchers, 2000) sino, principalmente, se presta atención a las características que aporta un patrón para facilitar la comunicación y para permitir su uso en el logro de una

Lingua franca (Erickson, 2000) o *Common ground* (Tidwell, 1998), que pudiera ser utilizada por aquellas personas involucradas en el desarrollo de un producto software, incluida la interfaz de usuario. El peligro que se corre de esta manera es caer en la duplicidad de fuentes de información, es decir, en que entre patrón y guía de estilo no se encuentren diferencias significativas. En los siguientes apartados estos últimos comentarios se justificarán en mayor detalle con el uso de algunos ejemplos representativos.

4.3.2 Patrones en Ingeniería del Software

El principal éxito en el ámbito informático del concepto de patrón se ha logrado con su utilización para documentar la experiencia relacionada con la programación orientada objetos, y es que cualquier desarrollo de software orientado a objetos que pretenda un buen diseño se traduce en una tarea tremendamente exigente. El problema principal radica en la complejidad inherente de los problemas que se pretenden solucionar y la necesidad de analizar y organizar esta complejidad de cara a poder llegar a un diseño que la alivie. En cualquier caso, no sólo bajo la metodología orientada a objetos se ha utilizado el concepto de patrón, el lector interesado puede consultar (Appleton, 1998) si desea tener una visión más pormenorizada del uso del concepto de patrón en Ingeniería del Software.

El patrón se encuentra íntimamente ligado con el análisis y el diseño orientado a objetos (ADOO). En este ámbito los patrones han sido una importante contribución que, efectivamente, presta una ayuda sustancial en el alcance de ese objetivo facilitando la definición de estrategias y de lenguajes y notaciones que sirvan de apoyo en el proceso de desenmarañar, con un alto nivel de abstracción, la complejidad del problema, evolucionando hacia su solución.

Los patrones de diseño no son una forma más de ADOO, sino una recopilación de los esfuerzos hechos en ese ámbito que sirve como perfecto complemento. Surgen de diseños cuya utilidad o necesidad se ha manifestado repetidamente en unos y otros proyectos y que han ido evolucionando hasta conseguir una cierta estabilidad y genericidad que los convierten en núcleos de solución altamente reutilizables.

Dicho de otra forma, si el ADOO exige una alta creatividad y capacidad de abstracción, los patrones de diseño son todo un repositorio de soluciones a problemas específicos que ayudan al desarrollador a afrontar este proceso con el mayor éxito posible recopilando en buena medida la experiencia obtenida por otras personas.

Pero un catálogo de patrones no sólo ayuda en el proceso de diseño de una solución, sino también a la hora de analizar y entender el problema.

Un desarrollador con un buen conjunto de patrones en su cabeza será capaz de reconocerlos cuando estudie un problema al que se enfrente, razón suficiente para tener presentes algunos de los más interesantes.

Se puede pensar que, por lo expuesto, los patrones son inherentemente geniales o, al menos, complejos. Pero muchas veces ocurre lo contrario: la mayoría son de una simplicidad sorprendente que a veces invita incluso a no hacerles el caso que se merecen.

En definitiva, los patrones proporcionan al desarrollador un bagaje importante para enfrentarse a muchos problemas que aparecen con frecuencia en su trabajo, a la vez que le facilitan la labor de organizar sus ideas, proporcionándole piezas genéricas de soluciones particulares que allanan el camino hacia la solución total del problema. No en vano, el desarrollador dotado de un buen número de patrones los reconocerá continuamente en las distintas librerías de calidad que pueda utilizar, por ejemplo, las JFCs de Sun son ejemplo de ello.

Los patrones de diseño se describen utilizando un formato consistente. Cada patrón se divide en secciones de acuerdo a diferentes plantillas (*Alejandrina*[1]*, Canónica, Coplien, GoF, Compacta, Portland, Fowle*r, etc.). Éstas dan una estructura uniforme a la información, haciendo que los patrones de diseño sean más fáciles de aprender, comparar y usar.

Un elemento que distingue a los patrones que se especifican en el terreno del análisis y diseño orientado a objetos, y de la Ingeniería del Software, de los presentados en otras disciplinas es la sección estructura de la solución. En la estructura se utiliza una representación gráfica de las clases del patrón usando una notación basada en la Técnica de Modelado de Objetos (OMT). También se puede hacer uso de diagramas de interacción para mostrar secuencias de peticiones y colaboraciones entre objetos. En concreto, los patrones de diseño usan notaciones más formales para denotar relaciones e interacciones entre clases y objetos:

- Con los diagrama de clases se representan clases de objetos, su estructura y las relaciones estáticas entre ellas.
- Con los diagrama de objetos se muestra una determinada estructura de objetos en tiempo de ejecución.
- Con los diagrama de interacción se muestra el flujo de peticiones entre objetos.

Cada patrón de diseño incluye al menos un diagrama de clases. Las otras notaciones se usan cuando son necesarias para completar la descripción.

[1] denominado así por asociación con su autor: Christopher Alexander

Pero, las notaciones gráficas, aunque importantes, no son suficientes. Con ellas, simplemente, se consigue representar el producto final del proceso de diseño, en forma de clases, objetos y de relaciones entre ellos. Para reutilizar el diseño debemos hacer constar las decisiones, las alternativas y las ventajas e inconvenientes que dieron lugar al mismo. También son importantes los ejemplos concretos, porque nos ayudan a ver el diseño en acción.

A modo de ejemplo utilizaremos un patrón, el *Adapter* (Gamma et al, 1995), para mostrar cómo se formula y documenta el conocimiento asociado con un patrón en Ingeniería del Software. Al tratarse de un patrón definido por la *banda de los cuatro* utiliza el formato GoF para su documentación (véase Gamma et al., 1995).

Nombre: *Adapter* (alías *Wraper*)

Tipo: patrón estructural

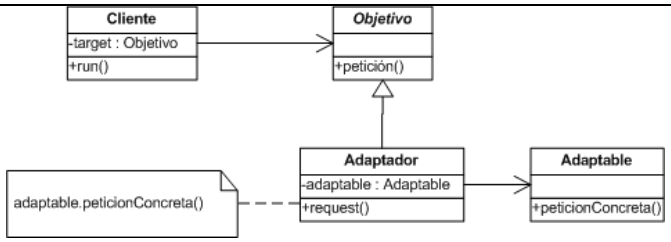
Propósito: convierte la interfaz de una clase en otra interfaz que es la que esperan los clientes. Permite que cooperen clases que de otra forma no podrían por tener interfaces incompatibles.

Motivación: A veces una clase de un *toolkit* que ha sido diseñada para reutilizarse, no puede hacerlo porque su interfaz no coincide con la interfaz específica que requiere la aplicación. (...)

Aplicabilidad: Debería usarse el patrón *Adapter* cuando:

- Se requiere usar una clase existente y su interfaz no concuerda con la que necesita.
- Se requiere crear una clase reutilizable que coopere con clases no relacionadas o que no han sido previstas, es decir, clases que no tienen por qué tener interfaces compatibles.
- (solamente en el caso de un adaptador de objetos) es necesario usar varias subclases existentes, pero no resulta práctico adaptar su interfaz heredando de cada una de ellas. Un adaptador de objetos puede adaptar la interfaz de su clase padre.

Estructura: Un adaptador de clases usa bien la herencia múltiple, bien la composición de objetos en la figura se muestra está segunda posibilidad.



Estructura del patrón *Adapter*

Los participantes en el diagrama de clases anterior son:

- **Objetivo** define la interfaz específica del dominio que usa el Cliente.
- **Cliente** colabora con objetos que se ajustan a la interfaz Objetivo.
- **Adaptable** define una interfaz existente que necesita ser adaptada.
- **Adaptador** adapta la interfaz de Adaptable a la interfaz Objetivo.

Colaboraciones: Los clientes llaman a operaciones de una instancia de Adaptador. A su vez, el adaptador llama a operaciones de Adaptable, que son las que satisfacen la petición.

Consecuencias: Los adaptadores de clases y de objetos tienen diferentes ventajas e inconvenientes. Un adaptador de clases

- Adapta una clase Adaptable a Objetivo, pero se refiere únicamente a una clase Adaptable concreta. Por tanto, un adaptador de clases no nos servirá cuando lo que queremos es adaptar una clase y todas sus subclases.
- Permite que un Adaptador redefina parte del comportamiento de Adaptable, por ser Adaptador una subclase de Adaptable.
- Introduce un solo objeto, y no se necesita ningún puntero de indirección adicional para obtener el objeto adaptado.

Por su parte, un adaptador de objetos

- Permite que un mismo Adaptador funcione con muchos Adaptables –es decir, con el adaptable en sí y todas sus subclases, en caso de que las tenga-. El Adaptador también puede añadir funcionalidad a todos los Adaptables a la vez.
- Hace que sea más difícil redefinir el comportamiento de Adaptable. Se necesitará crear una subclase de Adaptable y hacer que el Adaptador se refiera a la subclase en vez de a la clase Adaptable en sí.

Implementación: Aunque la implementación del patrón *Adapter* suele ser sencilla, éstas son algunas cuestiones que hay que tener en cuenta:

1. Implementación de adaptadores de clases en C++. En una implementación

C++ de un adaptador de clases, Adaptador debería heredar públicamente de Objetivo y privadamente de Adaptable. Así, Adaptador sería un subtipo de Objetivo, pero no de Adaptable.
2. Adaptadores conectables. Utilizando operaciones abstractas, usando objetos delegados y adaptadores parametrizados.

Usos conocidos: ET++Draw, una aplicación de dibujo basada en ET++. En ella se reutilizan las clases de ET++ para editar texto mediante una clase adaptadora TextShape. (...)
El *Matrimonio de conveniencia* de Meyer es una forma de adaptador de clases. Meyer describe cómo una clase PilaFija adapta la implementación de una clase Array a la interfaz de una clase Pila. El resultado es una pila que contiene un número fijo de entradas.

Patrones relacionados: El patrón *Bridge* tiene una estructura similar a un adaptador de objetos, pero con un propósito diferente: está pensado para separar una interfaz de su implementación, de manera que ambos puedan cambiar fácilmente y de forma independiente uno del otro, mientras que un adaptador está pensado para cambiar la interfaz de un objeto existente.
El patrón *Decorador* decora otro objeto sin cambiar su interfaz. Un decorador es por tanto más transparente a la aplicación que un adaptador. Como resultado, el patrón Decorador permite la composición recursiva, lo que no es posible con adaptadores puros.
El patrón *Proxy* define un representante o sustituto de otro objeto sin cambiar su interfaz.

A la vista de la descripción recogida anteriormente se puede constatar la potencia que ofrece una descripción de esta forma con la utilizada en otras disciplinas, por ejemplo con las descripciones que se utilizan en Interacción Persona-Ordenador utilizando el mismo concepto de patrón, las mismas serán tratadas en el siguiente apartado de este mismo capítulo.

El propósito, la aplicabilidad, la estructura y algún ejemplo son, sin duda elementos altamente deseables, a la hora de conocer en profundidad lo qué puede ofrecer un patrón a su lector, sobre todo si el que presenta el problema es un diseñador o desarrollador y carece de la experiencia necesaria.

El patrón utilizado está tomado del catálogo de patrones elaborado por (Gamma et al, 1995), pero estos no son los únicos patrones que podríamos tildar de *generativos* aunque no tengan asociado directamente código alguno.

Tabla 4-4. Taxonomía de patrones (Sarver, 2000)

Tipo de patrón	Comentario	Problemas	Soluciones	Fase de Desarrollo y referencia
Patrones de arquitectura	Relacionados con la interacción de objetos dentro o entre niveles arquitectónicos	Problemas arquitectónicos, adaptabilidad a requisitos cambiantes, prestaciones, modularidad, acoplamiento	Patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de funcionalidad	Diseño inicial (Buschmann et al., 1996)
Patrones de Diseño	Conceptos de ciencia de computación en general, independiente de aplicación	Claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, etc	Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC)	Diseño detallado (Gamma et al., 1995)
Patrones de Análisis	Usualmente específicos de aplicación o industria	Modelado del dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes	Modelos de dominio, conocimiento sobre lo que habrá de incluirse (p. ej. logging & reinicio)	Análisis (Fowler, 1996)
Patrones de Proceso o de Organización	Desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización	Productividad, comunicación efectiva y eficiente	Armado de equipo, ciclo de vida del software, asignación de roles, prescripciones de comunicación	Planificación (Ambler, 1998; Coplien, 2004)
Idioms	Estándares de codificación y proyecto	Operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo. Legibilidad, predicción	Sumamente específicos de un lenguaje, plataforma o ambiente	Implementación, Mantenimiento, Despliegue (Coplien, 1991)

Los patrones se pueden clasificar atendiendo a diferentes niveles de abstracción. En la Tabla 4-4 se han recogido diferentes tipos de patrones: arquitectónicos (Buchmann et al., 1996), de diseño (Gamma et al., 1995), de análisis (Fowler, 1996), de proceso u organización (Ambler, 1998; Coplien, 2004) e *idioms* (Coplien, 1991). Al fin y al cabo, diseñar requiere atender a 6los principios que gobiernan y sustentan la arquitectura de software, como dice (Meyer, 1997), los aspectos a gran escala de la arquitectura de software, se encomiendan o están sustentados en los aspectos de bajo nivel o escala detallada (código fuente). Si estos aspectos no obedecen a principios coherentes y de estilo, difícilmente se logran los aspectos a gran escala.

Nuestra experiencia personal impartiendo clases de programación orientada a objetos, en las que se introduce el concepto de patrón de diseño, nos confirma la necesidad de aportar una estructura que sirva de base para recordar y poner en práctica un patrón. El alumno, con ciertas nociones de programación orientada a objetos, es capaz de entender mejor el patrón si combina ejemplo y estructura, que si se le ofrece la mera descripción textual del problema y la solución que documenta un patrón. Obviamente, el alumno en este último caso, y el diseñador o desarrollador en general, debe tener conocimiento de una serie de conceptos a la hora de entender y utilizar un patrón. La ventaja con la que se cuenta en programación orientada a objetos es que, aunque los conceptos asociados son enormemente potentes, no son excesivos y están perfectamente identificados. Así, restringiéndonos a un nivel de abstracción asociado con el diseño, se puede decir que, ante todo, se busca incrementar el principio de cohesión y disminuir el de acoplamiento entre clases, para ello se dispone, haciendo uso de los patrones de diseño, de facilidades para la definición de clases y de relaciones entre ellas, así como de características como el polimorfismo, la ligadura dinámica, la herencia y la composición.

Si la estructura es importante en la documentación de patrones directamente relacionados con la experiencia disponible en Ingeniería del Software, no lo es menos la disponibilidad de un lenguaje unificado para denotarla. En programación orientada a objetos se dispone de UML, en Interacción Persona-Ordenador se recogerá seguidamente la panorámica general disponible para la formulación de patrones, que dista bastante, en unificación de criterios, de la que se disfruta en Ingeniería del Software.

Cabe mencionar, por último, antes de finalizar este apartado que existen propuestas de utilización de patrones de diseño para dar soporte al desarrollo de interfaces de usuario (Gamma et al., 1995; Noble, 1997; Rossi et al., 1997; 2000), las mismas se apoyan en la utilización de paradigmas orientados a objetos para dar respuesta a problemas habituales que surgen de

forma reiterada en el desarrollo de interfaces de usuario y de sistemas hipermedia respectivamente.

4.3.3 Patrones en Interacción Persona-Ordenador

La disciplina relacionada directamente con el tratamiento de la interacción entre persona y ordenador no se encuentra al mismo nivel de unificación de criterios del que se hace gala en Ingeniería del Software, esto no es nuevo y ya ha quedado reseñado en este documento. Algunos estados del arte relacionados con la utilización de patrones para recopilar experiencia en el diseño de interfaces de usuario pueden encontrarse en (Erickson, 2004; Fincher; 2004).

Obviamente, aunque con objetivos muy próximos entre ambas disciplinas, los aspectos que se ve obligado a considerar la IPO son, en parte, subjetivos y dependientes del usuario final que haga uso del producto software que se quiera diseñar y desarrollar, especialmente si lo que termina considerándose es únicamente el aspecto final que, a través de la interfaz de usuario, ofrecen dichos productos.

En IPO no se dispone de un lenguaje unificado, como es UML en Ingeniería del Software, pero si que existen otras notaciones. Unas están destinadas a considerar las tareas que realiza el usuario, entre ellas destaca CTT (Paternò, 1997), y otras tienen como objetivo describir la interfaz de usuario, ya sea específicamente (UIML, XUL, XAML, etc.) o junto con información adicional relacionada con tareas, dominio, usuario o contexto (XIML, UsiXML, TERESA-XML, etc.). Existe más información sobre estas notaciones en el capítulo segundo de esta tesis doctoral.

En cualquier caso, la tendencia que en la actualidad siguen IPO e IS es la misma, se apuesta por lo declarativo, es decir, por el modelo, abandonando o dejando en un segundo plano la utilización de lenguajes de programación. En IPO el Lenguaje de Marcado eXtensible (XML) ha cobrado una relevancia importante. Con él es posible describir a diferentes niveles de abstracción la interfaz de usuario y además ligar dicha descripción con la funcionalidad que debe acompañar, indiscutiblemente, cualquier producto software.

En un principio, XML se concibió para definir nuevos formatos de documentos para la Web. XML se deriva del lenguaje de marcado generalizado estándar (SGML, *Standard Generalized Markup Language*) y se puede considerar como un *metalenguaje*, es decir, un lenguaje para definir lenguajes de marcado.

Actualmente, tras haberse extendido el uso de XML, se acepta de manera general que este lenguaje no sólo es útil para describir nuevos formatos

de documentos Web, sino que también es adecuado para la descripción de datos estructurados. Entre los ejemplos de datos estructurados se incluye la información que habitualmente contienen las hojas de cálculo, los archivos de configuración de programas, los protocolos de red y la descripción de interfaces de usuario.

XML no está relacionado con ningún lenguaje de programación, sistema operativo o proveedor de software. De hecho, es bastante sencillo producir o hacer uso de XML utilizando diferentes lenguajes de programación. La independencia de la plataforma hace que XML sea muy útil para lograr la interoperabilidad entre diferentes plataformas de programación y sistemas operativos.

Además, con frecuencia, utilizando XML es necesario transformar dichos documentos de un vocabulario a otro. En ocasiones, esto se hace para poder representarlo en un formato adaptado a su posterior impresión o a un explorador Web; también puede ser para convertir documentos recibidos de una entidad externa a un formato con el que se esté más familiarizado. En este sentido, XSLT es el lenguaje de transformación de XML más popular. Una transformación expresada en XSLT describe reglas para transformar un árbol de origen en un árbol de resultados. La transformación se consigue mediante la asociación de patrones con plantillas. En este contexto, un patrón es una expresión regular que coincide con partes de un árbol de origen XML. En un proceso de transformación un patrón se hace coincidir con los elementos del árbol de origen. Si la coincidencia es correcta, se genera una instancia de una plantilla para crear parte del árbol de resultados. Al crear el árbol de resultados, los elementos del árbol de origen se pueden filtrar y cambiar de orden, además de poder agregar cualquier tipo de estructura.

Ésta es la tendencia en la descripción y posterior desarrollo de interfaces de usuario: el uso de lenguajes declarativos basados en XML. En función de ella, hace un par de años, en un *workshop* (*HCI patterns: concepts and tools*) celebrado paralelamente con la conferencia CHI del año 2003, un grupo de investigadores relacionados con el concepto de patrón propusieron una plantilla, mediante la especificación de un DTD de XML, denominado *Pattern Language Markup Language* (PLML) (Ficher, 2003; Borchers, 2003; Schümmer, 2004).

Dicha plantilla sólo persigue unificar criterios a la hora de documentar patrones de clara relación con Interacción Persona-Ordenador. Esta propuesta supone simplemente un nuevo formato inspirado en los múltiples disponibles. Esto es, si cabe, más patente si se hace un recorrido por los elementos caracterizados con la palabra reservada ANY en el DTD mostrado en la Tabla 4-5. Estos elementos podrían contener cualquier tipo de sub-elemento o dato (cadena de caracteres). Un elemento con la especifi-

cación de contenido ANY es completamente no estructurado. Los elementos *forces*, *diagram*, *example* y *rationale*, entre otros, no son estructurados y eso supone una puerta abierta a la hora de documentar la experiencia de alguien en forma de patrón.

Tabla 4-5. Propuesta PLML (2003)

```
<!ELEMENT pattern (name?, alias*, illustration?, problem?,
            context?, forces?, solution?, synopsis?, diagram?,
            evidence?, confidence?, literature?, implementation?,
            related-patterns?, pattern-link*, management?)>
<!ATTLIST pattern patternID CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT alias (#PCDATA)>
<!ELEMENT illustration ANY>
<!ELEMENT problem (#PCDATA)>
<!ELEMENT context ANY>
<!ELEMENT forces ANY>
<!ELEMENT solution ANY>
<!ELEMENT synopsis (#PCDATA)>
<!ELEMENT diagram ANY>
<!ELEMENT evidence (example*, rationale?)>
<!ELEMENT example ANY>
<!ELEMENT rationale ANY>
<!ELEMENT confidence (#PCDATA)>
<!ELEMENT literature ANY>
<!ELEMENT implementation ANY>
<!ELEMENT related-patterns ANY>
<!ELEMENT pattern-link EMPTY>
<!ATTLIST pattern-link
  type CDATA #REQUIRED
  patternID CDATA #REQUIRED
  collection CDATA #REQUIRED
  label CDATA #REQUIRED>
<!ELEMENT management (author?, revision-number?,
          creation-date?, last-modified?, credits?)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT creation-date (#PCDATA)>
<!ELEMENT credits (#PCDATA)>
<!ELEMENT revision-number (#PCDATA)>
<!ELEMENT last-modified (#PCDATA)>
```

En el capítulo siguiente se proponen limitaciones a este grado de libertad a la hora de documentar un patrón de interacción, tomando como referente la definición de Coad (Coad et al., 1992) y la intención de hacer de los patrones componentes de experiencia reutilizable. La aportación en el

terreno de patrones de interacción que se presenta en esta tesis doctoral está inspirada en los patrones de diseño propios de Ingeniería del Software, unos patrones en los que la motivación está limitada a una serie de posibilidades (creación, estructura y comportamiento) los ejemplos están comentados y encajan con la estructura, en forma de diagrama, asociado con el patrón y, finalmente, las *forces* (fuerzas) están directamente ligadas con la documentación de la aplicabilidad y las consecuencias de aplicación del patrón.

A la hora de documentar un patrón, si no se es más explícito que lo que se recoge es el formato PLML propuesto, se corre el riesgo de tener una versión algo más elaborada, en su documentación, que la que ya proporcionan algunas guías de estilo, o de conjuntos de guías de estilo asociadas entorno a un nombre, unos ejemplos y unas relaciones con otras guías de estilo o patrones. Surge así uno de los eternos dilemas asociados con dar respuesta a la pregunta ¿a quién van destinados los patrones? ¿qué se persigue con su formulación?

En este sentido hay partidarios de que los patrones puedan ser leídos y manejados por usuarios sin conocimientos específicos y, que así, a partir de ellos esos usuarios puedan participar en el diseño de productos software. Si es ese el ámbito en el que se presentan, el patrón no será *generativo*, será un elemento de documentación de experiencia, un componente de conocimiento que facilita la comunicación en la puesta en práctica de un diseño participativo, en el que tienen cabida diseñadores y desarrolladores junto con usuarios. Este uso corresponde con la idea original esbozada por Alexander y que, según el propio Alexander, no se ha mantenido en el uso que se ha dado del concepto patrón en la Ingeniería del Software (sus palabras fueron recogidas en (Borchers, 2000)):

Now, my understanding of what you are doing with patterns (...) It is a kind of a neat format, and that is fine. The pattern language that we began did have other features, and I don't know whether those have translated into your discipline. I mean, there was at root behind the whole thing a continuous mode of preoccupation with under what circumstances is the environment good. In our field that means something.

Dentro de este conjunto de propuestas coincidentes con la filosofía original, no generativa, de uso de un patrón están la mayoría de contribuciones elaboradas para recopilar experiencia en interacción: *Experiences* (Coram et al., 1996), *Common Ground* (Tidwell, 1999), *UI Patterns and Techniques* (Tidwell, 2004), *Patterns in Interaction Design* (Welie, 2005) y más específicamente en un contexto web: *The design of sites* (van Duyne et al., 2002), (Montero et al., 2002a, 2002b) y *Wu Patterns* (Graham,

2002). Como ejemplo de alguno de los patrones recogidos en estas colecciones se recoge seguidamente uno de ellos que es habitual en casi todos las colecciones propuestas (véase Tabla 4-6), se trata del patrón *Search Box*.

Tabla 4-6. Search Box: un ejemplo de patrón de interacción (Welie, 2005)

Search Box

From: www.tucows.com
Problem: The users need to find an item or specific information.
Use when: Any web site that already has primary navigation. User may want to search for an item in a category. User might want to further specify a query
Solution: Offer a search

** The search interface*
Offer search functionality consisting of a search label, a keyword field, a filter if applicable and a "go" button. Pressing the return key has the same function as selecting the "go" button. Also provide *Search Tips* and examples in a seperate page. A link to that page is placed next to the search functionality. The edit box for the search term is large enough to accomodate 3 typical user queries (typically around 20 characters). If the number of filters is more than 2, use a combobox for filterselection, otherwise a radiobutton.

Search -- editbox -- **for/in** -- filter -- **Go button**
or just...

-- editbox -- **Go button**

** Presenting search results*
The search results are presented on a new page with clear label containing at least "Searchresults" or similar. The search function is repeated in the top-part of the page with the entered keywords, so that the users know what the keywords were.

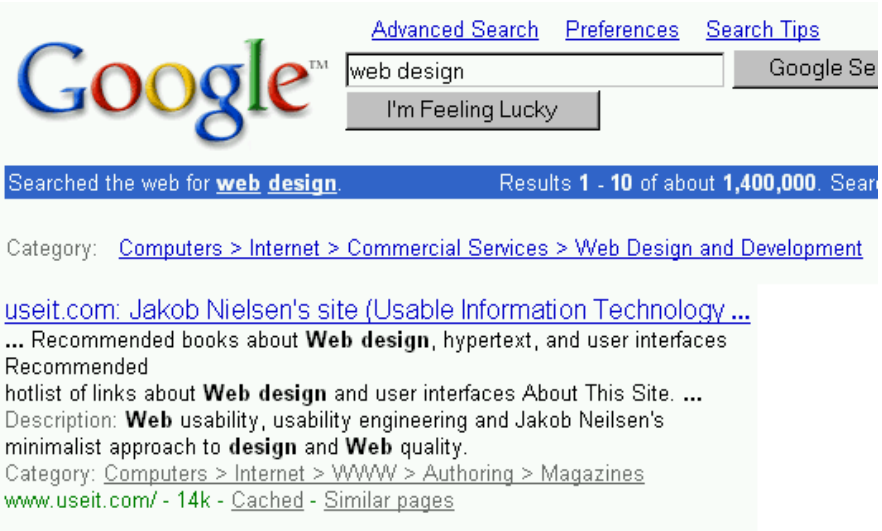
The number of "hits" is reported and the list of search results is organized; sorted or rated with the best matches at the top. When there are more than 10 results use a *Paging* mechanism. Each search result shows a link to the item itself and a snibbet of text to explain the item. Preferably that would a summary or abstract but can also be the first lines of text of the resulting item. The structure of a "result" typically shows:

1. Page Title
2. Description
3. Categorization
4. URL, Size, Date

* *Keyword matching*
If more than one search term is used the search engine must handle them as follows: if no special separators are used (not including the space), the search is interpreted as an OR function, the results that match both terms are listed first. If special separators are used the search engine must be able to handle more than one convention. For example, sometimes the "AND/OR" separators are used but using a "+" or a "-", include and exclude, must also be handled correctly. The engine must also be able to handle spelling mistakes of at least one character.

Why: By using this setup the whole search becomes a sentence that reads like the search query. More Examples

In this example from tucows, the designers actually were able to make the search read like a sentence. Users can "download sotware package X for Win2000"....



This example from Google shows how each result is presented.
Known Uses: www.amazon.com;

Los formatos utilizados en cada una de las colecciones referidas en el ámbito común de la interacción son diferentes, pero coinciden en utilizar lenguaje natural, adjuntar ejemplos y primar la organización y la relación entre patrones frente a realizar una formulación más formal. Las críticas que se pueden realizar a esta forma de recopilar la experiencia son:

(1) la experiencia así recopilada no resulta realmente útil para un ingeniero novato, si es que éste se ha acercado a una colección de patrones con la intención de conocer cómo diseñar e implementar la solución a un problema,

(2) el lector de estos patrones puede caer en la tentación de apostar por la elección de unos componentes de interacción concretos fruto de su identificación en los ejemplos normalmente asociados con los patrones,

(3) normalmente las ventajas obtenidas por la aplicación del patrón no se recogen con el propio patrón. En este sentido, de las colecciones reseñadas con anterioridad sólo una pequeña parte de los patrones recogidos en (Welie, 2005) asocian un principio de usabilidad con la descripción de los patrones, fundamentalmente porque no se dispone de una definición de calidad, o usabilidad extendida y plenamente aceptada como hacíamos notar en el apartado capítulo anterior, y

(4) salvo propuestas de repositorios de patrones de interacción como es el caso de Moudil (Gaffar et al., 2003), no existen herramientas que permitan gestionar y utilizar la experiencia asociada con los patrones reseñados anteriormente. Damask (Lin, 2003) es un intento en esa dirección, en él se utilizan algunos del los patrones recogidos en (van Duyne et al., 2002), pero sólo aquellos a los que se ha asociado en su documentación un boceto, a modo de esquema de utilización, ligado con la solución aportada y documentada en el patrón.

En definitiva, la solución documentada en los patrones mencionados, y que calificamos como *no generativos,* carece de la potencia suficiente para, a partir de ella, obtener la especificación de componentes *tangibles* de diseño o desarrollo como es el caso de los patrones *generativos* disponibles en Ingeniería del Software. Será esta limitación una de los retos a los que pretendemos dar respuesta en esta tesis doctoral.

Bajo esta misma consideración, dos colecciones de patrones: Just-UI (Molina, 2003) y WebML (Ceri, 2000) habrían recorrido una dirección similar a la que proponemos en esta tesis doctoral.

En ambas propuestas se habría estudiado y documentado un catálogo de patrones *propios*, que se han codificado (*hard coded*) y que mediante un proceso de inferencia facilitada por sendos entornos, OlivaNova Model Execution System® y WebRatio®, es posible obtener un prototipo de interfaz de usuario.

OlivaNova® hace uso de su catálogo de patrones para generar prototipos de interfaces de usuario, para diferentes plataformas (escritorio, móvi-

les, PDA), a partir de una especificación conceptual de las entidades involucradas en el problema (véase Fig. 4-1).

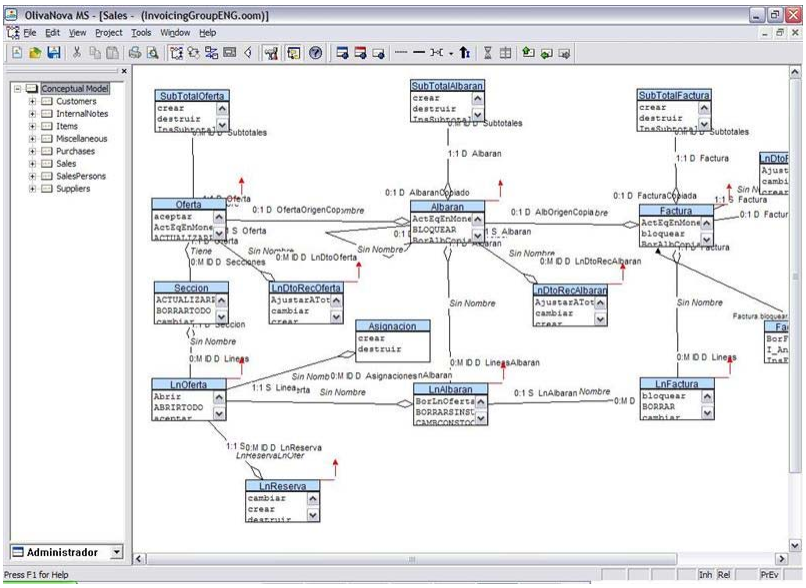


Figura 4-1. Herramienta OlivaNova Model Execution System®

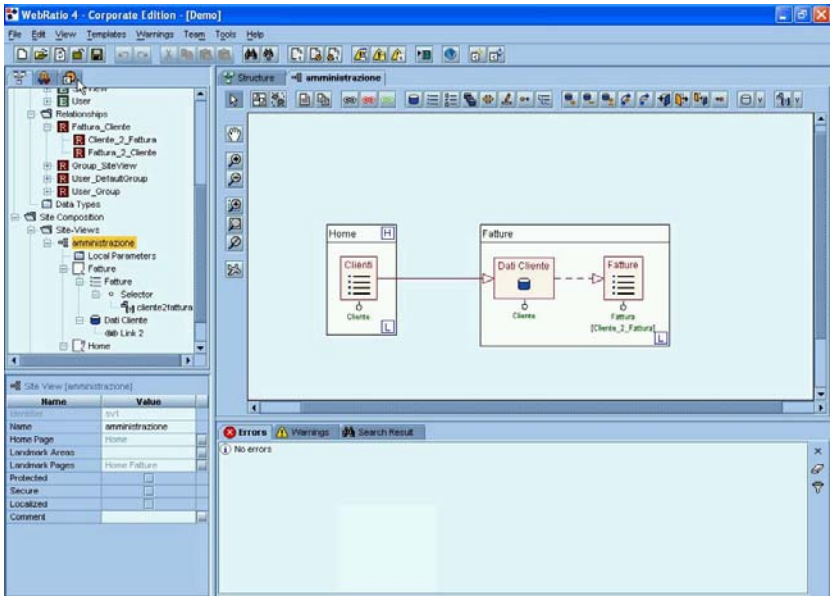


Figura 4-2. Herramienta WebRatio®

WebRatio® permite desarrollar sitios web complejos mediante del uso de un catálogo de patrones propuestos y disponibles en el entorno citado (véase Fig. 4-2).

En ambas herramientas se utilizan lo que venimos denominamos patrones generativos, al modo y manera de los encontrados en Ingeniería del Software, así, a partir de especificaciones conceptuales, haciéndose uso de diagramas de clases o de modelos entidad-relación respectivamente, se desarrollan aplicaciones completas infiriéndose la interfaz de usuario. Los modelos conceptuales, representados gráficamente se almacenan utilizando un lenguaje basado en XML, y haciendo uso de transformaciones XSL se consigue la apariencia final de la interfaz asociada a la aplicación. Siendo ambas herramientas muy interesantes desde todos los puntos de vista, tienen la desventaja de que son comerciales y eso afecta, en parte a su estudio. Los patrones, que llevan asociados cada una de estas herramientas, están *camuflados*. En OlivaNova® la interfaz de usuario se infiere automáticamente a partir del catálogo de patrones propuestos. El modelo de presentación, en OlivaNova®, consta de diversos conceptos y patrones. El uso combinado de estos elementos permite la construcción completa de una especificación concreta de interfaz de usuario.

Los patrones en Just-UI se organizan en tres niveles:

(1) *Árbol de jerarquía de acciones* donde se define el acceso a la funcionalidad del sistema,

(2) *Unidades de interacción* que presentan al usuario escenarios donde se produce el diálogo con el sistema, pueden ser de servicio, de instancia de población y maestro/detalle; y

(3) *Patrones auxiliares* que funcionan como piezas base para crear las unidades de interacción y complementan a estas aportando semántica precisa relacionada con actividades ligadas a la interacción como introducción, selección, información complementaria, dependencia, agrupación de argumentos, filtrado, visualización, navegación, acción y ordenación.

En WebRatio® la idea es similar, los patrones se proporcionan mediante facilidades dispuestas, en el entorno de desarrollo, en una barra de herramientas en un entorno de desarrollo de interfaces de usuario. En esa barra de herramientas pueden encontrarse elementos abstractos de descripción de sitios y páginas web, es decir, modelos que podemos considerar patrones. Los patrones más representativos propuestos en WebML se recogen en el Apéndice D.

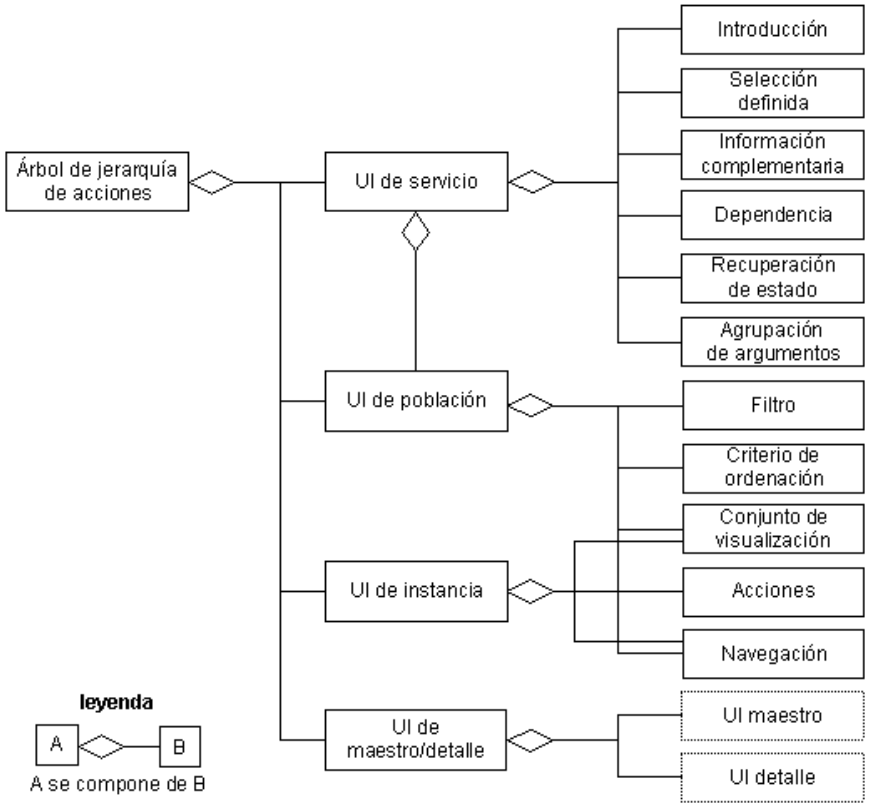


Figura 4-3. Catálogo de patrones Just-UI (Molina, 2003)

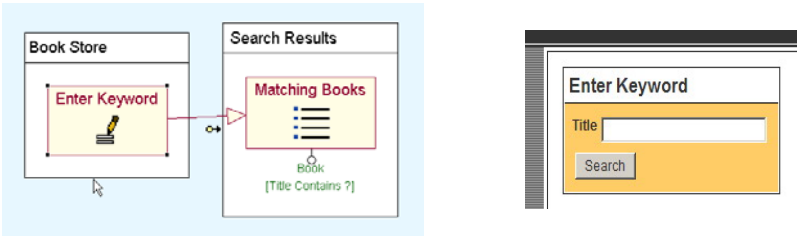


Figura 4-4. Posible especificación del patrón Search utilizando WebML (a) y presentación asociada fruto de aplicación de transformaciones XSLT (b)

En la Fig. 4-4a se muestra, utilizando los patrones propuestos en WebML (véase el Apéndice E para más detalles), un posible diagrama que correspondería al patrón *Search* recogido anteriormente como ejemplo de los patrones de interacción que se proponen en la actualidad (Welie, 2005; van Duyne, 2002; Graham, 2002; Montero, 2002). Este tipo de informa-

ción, relacionada con la estructura asociada a la solución que documenta un patrón, es la que se pretende incorporar en la sección diagrama propuesto en PLML, potenciando, de esta forma, la descripción ofrecida de un patrón. De esta forma el ingeniero obtendría información de cómo aplicar la solución que el patrón lleva asociado y no sólo se limitaría a poder identificar, tras leer una colección de patrones, un problema y ejemplos de soluciones concretas que otros diseñadores han dado a ese mismo problema, por ejemplo la mostrada en la Fig. 4-4b.

La clave de una y otra propuesta está en que los patrones documentados en los respectivos catálogos no están aislados, sino estrechamente ligados a elementos del modelo conceptual, en ello reside su potencia, flexibilidad y versatilidad, al igual que ocurre con los patrones de diseño disponibles en Ingeniería del Software. Los patrones, en este sentido, son asociaciones y dichas asociaciones se establecen entre un modelo primario, el de dominio, y otro secundario o dependiente como es el de presentación. Las herramientas citadas demuestran la potencia que ofrece el concepto de patrón para dar soporte a actividades de desarrollo de interfaces de usuario, otro aspecto es la calidad ligada a las interfaces así generadas.

4.4 Análisis y conclusiones

En este capítulo se ha considerado la experiencia en el diseño y desarrollo de interfaces de usuario, y especialmente la documentación y reutilización eficiente de la misma. Se han identificado las diferentes características de las guías de estilo y los diferentes niveles de abstracción que se pueden considerar en las mismas: principios, estándares y reglas de diseño. Entre las principales limitaciones que ofrecen las guías de estilo está su volumen, autoridad y, ante todo, su puesta en práctica cuando se identifica un problema o una necesidad en la que pudieran aplicarse. Estas limitaciones se hacen más apreciables cuando el que pretende hacer uso de las guías de estilo es un diseñador sin la suficiente experiencia.

En lo que se refiere a la documentación de la experiencia, concretamente a aquella relacionada con aspectos arquitectónicos y, en Ingeniería del Software, con programación orientada a objetos se utiliza el concepto de patrón y con él se logran buenos resultados.

En IPO desde hace unos años se está tratando de incorporar la misma idea a la hora de documentar la experiencia asociada a esta disciplina, pero el resultado es que la experiencia así documentada no dista mucho de la que se puede lograr utilizando las tradicionales guías de estilo. Las limitaciones que se identifican, en el uso de patrones en IPO, están asociadas

con: la carencia de un lenguaje unificado con el que describir los patrones que se presentan; la reseñada falta de caracterización de la calidad de uso de un producto software; y la determinación del destinatario de la experiencia documentada utilizando patrones: usuario *versus* ingeniero.

Las colecciones de patrones disponibles en IPO han sido recogidas en este capítulo y dos propuestas, localizadas a medio camino entre IS e IPO, han resultado de especial interés: Just-UI (Molina et al., 2003) y WebML (Ceri et al., 2000). La clave de estas propuestas está en la consideración de las implicaciones que el modelo de presentación tiene con otros modelos, concretamente, en los catálogos comentados, con el de dominio.

Uno de los retos que queda pendiente y que está relacionado con el concepto de patrón no pasa tanto por su utilización para documentar la experiencia y facilitar la comunicación entre los diferentes individuos involucrados en el desarrollo, donde hay pruebas de su validez que han sido recogidas a lo largo del capítulo, sino por facilitar la comunicación entre teoría y práctica independientemente de la disciplina donde se desee utilizar. Otro reto lo constituye el establecimiento de relaciones entre calidad y experiencia. Ambos constituyen el nudo fundamental de esta tesis doctoral y centrará la discusión en los capítulos restantes de este documento.

En la Tabla 4-7 se recogen las características que consideramos más reseñables de las colecciones de patrones de interacción más interesantes e importantes disponibles en la actualidad. Las mismas han sido encontradas en sitios web elaborados para su difusión y en diferentes libros editados recientemente. Tras una búsqueda y lectura detenida de algunos de los patrones disponibles en dichas colecciones se observan diferentes formas de documentación de los patrones de interacción, siendo a su vez también distintas de la forma de documentar los patrones de diseño (Gamma et al., 1994). En cualquier caso, en todas las colecciones recogidas se recurre al uso de ejemplos, en los que se visualizan imágenes donde se muestran soluciones particulares que implementan el núcleo de solución que el patrón documenta. Los patrones de interacción recogidos en la Tabla 4-7 tienen diferente ámbito de aplicación (Web, dispositivos móviles, GUI tradicionales, etc.), pero desde esta tesis doctoral no estamos de acuerdo con esta característica y en ella se es más partidario de la opinión por la que los patrones son generales e igualmente aplicables a muy diferentes entornos, siendo independientes de la plataforma y del modo de interacción.

Tabla 4-7. Recopilación de lenguajes y catálogos de patrones de interacción

Author	Coram et al.	Tidwell		Borchers	Welie	Van Duyne et al.,	Graham
Name	Experiences (1996)	Common Ground (1999)	UI patterns and techniques (2002)	A pattern approach to Interaction design (1999)	Interaction design patterns (2002)	The design of sites (2002)	A pattern language for web usability (2002)
nº. patterns	26	60	51	17	99 / 26 / 7	89	79
Scope	GUI	GUI	GUI	GUI	Web/ GUI / mobile	Web	Web
Purpose	design	design	design	design	design	design	usability
Form	Alexandrian	Canonical	Compact	Alexandrian	Canonical	Canonical	Canonical
Organization	DAG	Multiple DAG	Multiple DAG	DAG	Multiple DAG	Multiple DAG	DAG

Será en el capítulo siguiente, capítulo quinto, donde se presentará en detalle la principal propuesta asociado a esta tesis doctoral. Asociada a la misma se detallarán modificaciones al formato acordado, PLML (Fincher et al., 2003), para llevar a cabo la documentación de los patrones de inter-

acción. Con dichas modificaciones se contribuirá a parte de las recomendaciones recogidas en el capítulo segundo, especialmente a aquellas que están relacionadas con la documentación, gestión, manipulación, utilización, aprendizaje y diseminación de la experiencia.

Finalmente, comentar que la documentación de los patrones de interacción cuenta con la disponibilidad de pocas herramientas que faciliten dicho proceso, dentro de ellas cabe destacar *Collaborative Pattern Editor* (COPE, Fig. 4-5) descrito en (Schümmer, 2004). Con la citada herramienta puede describirse textualmente cualquier patrón de interacción, así como visualizar gráficamente la estructura del catálogo de patrones descritos y las relaciones establecidas entre ellos.

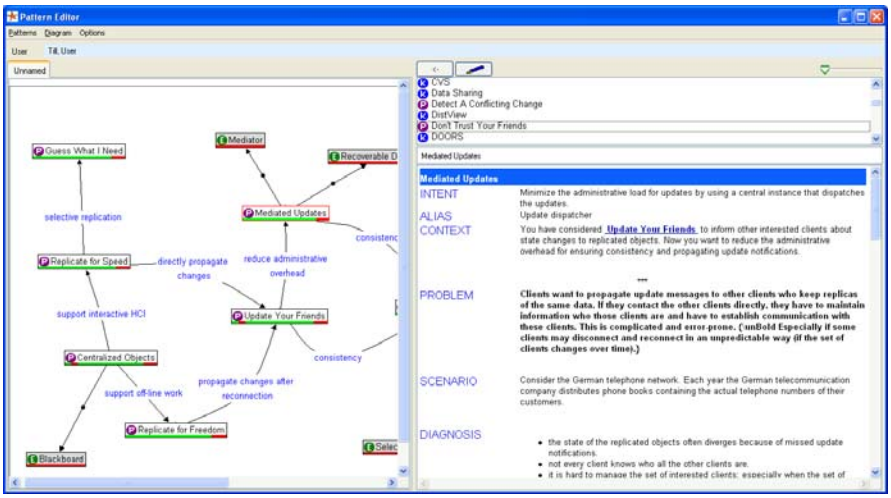


Figura 4-5. Edición de patrones con CoPE (Schümmer, 2004)

4.5 Contribuciones relacionadas con este capítulo

En el estudio y consideración de los patrones, sus limitaciones, ventajas y aportaciones respecto a otras formas de documentar la experiencia se ha participado en la elaboración de las siguientes publicaciones:

- **Montero, F**., Lozano, M., González, P. Patrones de Interfaz para entornos de trabajo en grupo. II Jornadas de trabajo Dolmen. Universidad Politécnica de Valencia. 12 y 13 de marzo de 2002.

- **Montero, F**., Lozano, M., González, P. Patrones de interacción: Taxonomía y otros problemas. Congreso Internacional de Interacción 2002. Universidad Carlos III, Madrid, 8 a 10 de mayo de 2002.
- **Montero, F.**, López-Jaquero, V., Lozano, M., González, P. A user interfaces development and abstraction mechanisms. Artículo seleccionado en el V Congreso Interacción Persona Ordenador para su publicación en Springer-Verlag, Berlin, 2005. (to appear)
- García, F.J., Lozano, M., **Montero, F.**, Gallud, J.A., Ruiz, V. Survey on Quality Models to Measure the Quality of Web Sites and Applications. 11th International Conference on Human-Computer Interaction (HCI, 2005). 22-27 July, 2005, Las Vegas, Nevada, USA. 2005.
- **Montero, F**., López-Jaquero, V., Ramírez, Y., Lozano, M., González. P. Patrones de Interacción: para usuarios y para diseñadores. VI Congreso de Interacción Persona-Ordenador. 13-16 de septiembre, Granada, España. 2005.

Se han propuesto, igualmente, diferentes catálogos de patrones aplicables en diferentes ámbitos, como son los entornos de trabajo en grupo o el desarrollo de sitios web, fruto de dichas propuesta se han identificado refrendado limitaciones en el terreno de la consideración de la experiencia y de la calidad. Las publicaciones más reseñables son:

- **Montero, F**., Lozano, M., González, P. Patrones de Interfaz para entornos de trabajo en grupo. II Jornadas de trabajo Dolmen. Universidad Politécnica de Valencia. 12 y 13 de marzo de 2002.
- **Montero, F**., Lozano, M., González, P., Ramos, I. Designing web sites by using design patterns. The second latin american conference on pattern languages of Programming. Itaipava- Rio de Janeiro, Brasil. 5 a 7 de agosto de 2002. (ISBN: 85-87837-07-9)
- **Montero, F**., Lozano, M., González, P., Ramos, I. A first approach for design web sites by using patterns. First Nordic conference on Pattern Languages of Program. Hojstrupgard, Dinamarca. 20 a 22 de septiembre de 2002. (ISBN: 87-7849-769-8)

Referencias bibliográficas

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. Angel, S. A Pattern Language. Oxford University Press, New York, 1977.

Ambler, S. Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press, 1998

Apple Computer. *Macintosh human interface guidelines.* Addison and Wesley, Reading, MA, 1992

Appleton, B. Patterns and Software: Essential Concepts and Terminology. http://www.bradapp.net/

Barber, J. AskJef: Integration of Case-Based and Multimedia Technologies for Interface Design Support, *Artificial Intelligence in Design '92,* J.S. Gero (Ed.), Kluwer Academic, pp. 457-474, 1992

Barber, J., Bhatta, S., Goel, A., Jacobson, M., Pearce, M., Penberthy, L., Shankar, M., Simpson, R., Stroulia, E.: AskJef: Integration of Case-Based and Multimedia Technologies for Interface Design Support. In Proc. Of the Second International Conference on AI in Design (AID'92), pp.457–475, Pittsburgh, PA. 1992

Borchers, J. HCI patterns page: http://www.hcipatterns.org/tiki-index.php

Borchers, J. Interaction Design Patterns: Twelve Theses**,** Position Paper, Workshop "Pattern Languages for Interaction Design: Building Momentum", CHI 2000 (The Hague, Netherlands, April 2–3, 2000)

Borchers, J. Interaction design patterns: Twelve theses. Position paper in BHCI Workshop "Patterns in Human Computer Interaction", London, Nov.16–17,2000.

Brown, C. M. (1988). Human-computer interface design guidelines. Norwood, NJ: Ablex Publishing.

Brown, C. Marlin, Human-Computer Interface Design Guidelines, Ablex Publishing Co., Norwood, NJ (1988).

Ceri, S., Fraternali, P., Bongio, A. Web Modeling Language (WebML): a modeling language for designing Web sites. Proc. of WWW9, and *Computer Networks*, 3 (**1-6**), 137-157, 2000

Coad, P. Object-oriented patterns. Communications of the ACM. Volume 35, Number 9. Sept. 1992

Constantine, L. L., & Lockwood, L. A. D: Software for use: A practical guide to the models and methods of usage-centered design. New York, NY: ACM Press. 1999.

Coplien, J. Advanced C++ Programming Styles and Idioms. Addison-Wesley. 1991

Coplien, J., Harrison, N. Organizational Patterns of Agile Software Development. Prentice Hall. 2004

Coram, T. Lee, J. Experiences -- A Pattern Language for User Interface Design. http://www.mindspring.com/%7Ecoram/papers/experiences/Experiences.html

Dilli, I., Hoffmann H.J.: Diades-II, a multi-agent user interface design approach with an integrated assessment component. ACM SIGCHI Bulletin 27/2: 33 – 34. 1995

Dilli, I., Hoffmann, H. Diades-II. A Multi-Agent User Interface Design Approach with an Integrated Assessment Component. SIGCHI Bulletin. Volume 27, Number 2. April. 1995

Downey, L.: *Usability engineers: who do too much!!!!*. In Interactions, 10 (5) p. 12-17. 2003

Erickson, T. Lingua Francas for Design: Sacred Places and Pattern Languages. In *The Proceedings of DIS 2000* (Brooklyn, NY, August 17-19, 2000). New York: ACM Press, 2000, pp 357-368.

Erickson, T. The Interaction Design Patterns Page http://www.visi.com/~snowfall/InteractionPatterns.html

Ficher, S. The pattern gallery. http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html

Fowler, M. Analysis Patterns: Reusable Object Models. Addison-Wesley. 1996

Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns - Elements of Reusable Object-Oriented Software. Addison Wesley, 1995

Gorny, P. EXPOSE – An HCI-counseling tool for User Interface Designers. Tools for Working with Guidelines. SIGCHI Bulletin. Volume 27, N. 2 April. 1995

Gorny, P.: EXPOSE, An HCI-counseling tool for user interface designers. ACM SIGCHI Bulletin 27/2: 35 – 37. 1995

Gould, J. D., & Lewis, C.. Designing for usability: Key principles and what designers think. Communications of the ACM, 28, 3, 300-311. Reprinted in Baecker, R. M., & Buxton, W. A. S. (Eds.) (1987), Readings in human-computer interaction: A multidisciplinary approach, 528-539. San Mateo, CA: Morgan Kaufmann Publishers. 1985

Graham, I. A Pattern Language for Web Usability. Addison Wesley. 2002. http://www.wupatterns.com/

Henninger, S., Haynes, K., Reith, M.: A Framework for Developing Experience-Based Usability Guidelines. Symposium on Designing Interactive Systems 1995: 43-53. 1995

Henninger, S., Haynes, K., Reith, M.W., A Framework for Developing Experience-Based Usability Guidelines, *Symposium on Designing Interactive Systems* (DIS '95), Ann Arbor, MI, August 23-25, 1995, pp. 43-53

Ianella, R. HyperSAM: a management tool for large user interface guideline sets, Special Interest Group on Computer Human Interaction SIGCHI, vol. 27, pp. 42-43, 1995

IBM Corporation. Object–oriented interface design: IBM common user access guidelines. Que Corp., Carmel, IN, 1992

Jason I. Hong, and James A. Landay, WebQuilt: A Framework for Capturing and Visualizing the Web Experience. In *Proceedings of The Tenth International World Wide Web Conference (WWW10)*, Hong Kong, May 2001, pp. 717-724.

Kolodner, J., Mark, W.: Case-Based Reasoning - Guest Editors' Introduction. IEEE Expert 7(5): 5-6. 1992

Lin, J., Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In Conference Supplement of *UIST 2003: ACM Symposium on User Interface Software and Technology*, Vancouver, BC, Canada, Nov. 2–5, 2003, pp. 13–16.

Lin, J., M.W. Newman, J.I. Hong, and J.A. Landay, DENIM: Finding a tighter fit between tools and practice for web site design. CHI Letters (Human Factors in Computing Systems: CHI 2000), 2000. 2(1): p. 510-517.

Mahemoff, M. J. and Johnston, L. J. Principles for a Usability-Oriented Pattern Language In Calder, P. and Thomas, B. (Eds.), *OZCHI '98 Proceedings* , 132-139. Los Alamitos, CA. 1998

Mariage, C., Vanderdonckt, J.: MetroWeb: a Tool to Support Guideline-Based Web Evaluation Methods, in Proc.of 10th Int. Conf. on Human-Computer Interaction HCI International2003 (Heraklion, 22-27 June 2003), C. Stephanidis (Ed.), Lawrence Erlbaum Associates, Mahwah, 2003.

Mayhew, D. J. Principles and guidelines in software user interface design. Englewood Cliffs, NJ: Prentice Hall, 1992

Mayhew, D. J. The usability engineering lifecycle: A practitioner's handbook for user interface design. San Francisco, California: Morgan Kaufmann Publishers. 1999.

Mayhew, D. J.: Principles and guidelines in software user interface design. Englewood Cliffs, NJ: Prentice Hall. 1992

Meyer, B. Construcción de Software Orientado a Objetos (2ª edición). Prentice Hall. 1999

Meyer, B. Object-Oriented Software construction, Second Edition. Prentice Hall Professional Technical reference. 1997

Microsoft Corporation. *The Microsoft Windows user experience*. Microsoft Press, Redmond, WA, 1999

Molich, R., Nielsen, J. Improving a human-computer dialogue: Communications of the ACM, 33, 3, 338-348, 1990

Molina, P., Belenguer, J., Pastor, O.: Describing Just-UI Concepts Using a Task Notation. DSV-IS 2003: Madeira, Portugal. 2003

Molina, P., Pastor, O., Martí, S., Fons, J., Insfrán, E.: Specifying Conceptual Interface Patterns in an Object-Oriented Method with Automatic Code Generation. UIDIS 2001

Molina, P: A Review to Model-Based User Interface Development Technology. MBUI 2004

Nielsen, J. Usability engineering. Boston, MA: AP Professional. 1993.

Nielsen, Jakob. Hypertext and Hypermedia, Academic Press, 1990

Noble, J. GOF Patterns for GUI Design. In *Proceedings of the European Conference on Pattern Languages of Program Design*. Irsee, Germany. 1997

Norman, D. A. The Design of Everyday Things: Doubleday and Company, 1990

Norman, D. A. The psychology of everyday things. New York, NY: Basic Books. Subsequently published under the title The design of everyday things. 1988

Norman, D., Draper, S.: User Centered System Design: New Perspectives on Human-Computer Interaction. Lawrence Erlbaum Associates. 1986

Ogawa, K., Useno, K. GuideBook: design guidelines, Special Interest Group on Human Computer Interaction SIGCHI, vol. 27, pp. 38-39, 1995

Rossi, G., Schwabe, D., Garrido, A. Design Reuse in Hypermedia applications development. Hypertext. Southampton. 1997

Rossi, G., Schwabe, D., Lyardet, F. User Interface patterns for hypermedia applications. Proceedings of the working conference on Advanced visual interfaces. 2000

Sarver, T. Pattern refactoring workshop. Position paper, OOSPLA 2000, http://www.laputan.org/patterns/positions/Sarver.html, 2000.

Scapin D., Leulier C., Vanderdonckt J., Mariage C., Bastien C., Farenc C., Palanque P. and Bastide, R.: Towards Automated Testing of Web Usability Guidelines In. Tools for Working with Guidelines London Springer; pp. 293-304. 2000.

Schneiderman, B.: Designing the user interface. 2a. Edition. Addison-Wesley. Reading, Ma. 1992

Shneiderman, B. Designing the User Interface - Strategies for Effective Human-Computer Interaction, second edition, Reading, MA: Addison-Wesley Publishing Company, 1992

Smith, S.L., and Mosier, J.N. (1986). *Design Guidelines for Designing User Interface Software*. Technical Report MTR-10090 (August), The MITRE Corporation, Bedford, MA 01730, USA.

Sun Microsystems Inc *Java look and feel guidelines*, 2ª edición. Addison and Wesley, Reading, MA, 2001

The OlivaNova Model Execution System®. Software that delights. Care Technologies. http://www.care-t.com. 2005

Tidwell, J. .The Gang of Four Are Guilty. http://www.mit.edu/˜jtidwell/gofareguilty.html, 1999.

Tidwell, J. Interaction Design Patterns (aka. Common Ground). http://www.mit.edu/%7Ejtidwell/interaction_patterns.html

Tognazzini, B.: First principles of interaction design. AskTOG. Interaction design solutions for the real world. http://www.asktog.com/

van Duyne, D.K., J.A. Landay, and J.I. Hong, *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience*. Reading, MA: Addison-Wesley, 2002

van Welie, M. The Amsterdam Collection of Patterns for User Interface Design. http://www.welie.com

Vanderdonckt, J. SIERRA – An interactive System for Ergonomic Realization of Applications. Tools for Working with Guidelines. SIGCHI. Bulletin. Volume 27, N. 2. April. 1995

Vanderdonckt, J., Accessing Guidelines Information with SIERRA, in Proc. of IFIP TC.13 International Conference on Human-Computer Interaction Interact'95 (Lilleham-mer, 27-29 June 1995), Chapman & Hall, London, 1995, pp. 311-316.

WebRatio®. a CASE tool innovating your Web applications development. http://www.webratio.com. 2005

Capítulo 5 Una propuesta integradora de teoría y práctica para la especificación abstracta de interfaces de usuario

Largo es el camino de la enseñanza por medio de teorías; breve y eficaz por medio de ejemplos.
Lucio Anneo Séneca (Filósofo latino)

Los que se enamoran de la práctica sin la teoría son como los pilotos sin timón ni brújula, que nunca podrán saber a dónde van.
Leonardo da Vinci (Pintor, escultor e inventor italiano)

5.1 Introducción

Hasta este capítulo, Ingeniería del Software e Interacción Persona-Ordenador han centrado el dualismo en la presentación de contenidos seguida en esta tesis doctoral. Primeramente, se han identificado las aportaciones y tendencias pujantes en cada disciplina en cuanto a metodologías, lenguajes y herramientas se refiere. Después, y desde los puntos de vista ofrecidos desde cada disciplina reseñada, se han considerado, a su vez, dos elementos que influyen decisivamente en el desarrollo de productos software como son la calidad y la experiencia, caracterizándose cada uno de esos ingredientes que se antojan fundamentales e identificándose una falta de consideración conjunta de ambos.

Pero, ésta no ha sido la única consideración dual que se ha manejado hasta el momento, también se ha reflexionado con la dualidad que supone lo abstracto y lo concreto, es decir, con la identificación de la realidad asociada al desarrollo de interfaces de usuario, identificándose una necesidad de búsqueda del conocimiento de esa realidad inestable y de la necesidad de aprenderla aunque se encuentre en continuo cambio. Una realidad que se muestra plural, efímera, múltiple, y, en definitiva, cambiante. En esta consideración de lo único y lo plural, de lo efímero y lo persistente, otra disciplina, la Filosofía, se ha ocupado de discriminar entre apariencia y realidad y ese estudio ha sido un desafío constante en ella a lo largo de toda su evolución, desde la filosofía clásica hasta la contemporánea. Se comentan, seguidamente, algunas de las reflexiones que se han hecho en la filosofía clásica, identificándose las relaciones que se identifican con las diferentes tendencias de desarrollo del software. En realidad, toda la filo-

sofía, según Descartes, está pendiente de Platón o Aristóteles y serán ellos los referentes, con ellos seguirá la dualidad en la presentación de contenidos también en este capítulo.

Servirá esta forma de comenzar la andadura en este capítulo quinto para resaltar la necesidad de considerar lo abstracto y esencial al documentar la experiencia y alcanzar conocimiento reutilizable, y con ello valorar la consideración conjunta de calidad y experiencia. En este capítulo se unificarán algunas de las contribuciones que han sido introducidas en capítulos anteriores y que ahora serán completadas e integradas. El capítulo siguiente abundará, con la utilización de algunos ejemplos, en la propuesta presentada en este capítulo. En este capítulo se caracterizará la calidad centrada en la usabilidad que una interfaz de usuario puede ofrecer. Se abundará en las especificaciones propuestas para documentar la experiencia utilizando patrones. Se integrarán calidad y experiencia y se apostará por integrar la propuesta con la actual tendencia de desarrollo software utilizada en Ingeniería del Software. Con ello, e indirectamente, se disminuirá el gap semántico existente entre las dos disciplinas que han centrado la presentación de contenidos y de la propia tesis doctoral; IS e IPO.

5.2 Filosofía y desarrollo de software

Dos preguntas crearon incertidumbre y preocupación a los máximos representantes de la filosofía clásica en su día: Platón y Artistóteles. Las mismas tenían que ver con *cómo conocer la realidad y poder aprenderla*, cuando ésta se muestra inestable y *cómo puede explicarse el movimiento o cambio* del que hace gala esa misma realidad a través de los seres sensibles que la constituyen. Estas preguntas y la propia analogía con la Filosofía Clásica no resultan peregrinas o gratuitas a estas alturas del documento que nos ocupa. Mas al contrario, el desarrollo de interfaces de usuario (véase Fig. 5-1); la apariencia, el modo de interacción, su evolución, cambio y mantenimiento, tiene que dar respuesta a esas mismas preguntas y, por ello, conviene tener presente las consideraciones epistemológicas y ontológicas que aquellos pensadores hicieron por trasnochadas que pudieran parecer. Sus reflexiones servirán, cuando menos, para identificar la necesidad de considerar las características que determinan la calidad de una interfaz y si las mismas son inherentes al producto o proceso de desarrollo y si, por lo tanto, pueden ser caracterizadas y establecidas previamente.

Figura 5-1. La realidad se nos muestra efímera, plural y cambiante

5.2.1 Platón y la Ingeniería del Software tradicional

El centro de la Filosofía Clásica lo constituye la *teoría de las formas* o *de las ideas* de Platón. En el fondo, la filosofía de Platón; su idea del conocimiento, su teoría ética, su psicología, su concepto del estado y su concepción del arte deben ser entendidos a partir de dicha perspectiva. El problema que trató Platón fue el de la relación de lo uno y lo múltiple. La realidad se nos muestra en un devenir constante de seres heterogéneos e, incluso, contrarios. *¿Cómo poder conocer esa realidad tan inestable*? *¿Cómo poder aprenderla si se halla en continuo cambio*?

La *teoría de las ideas* de Platón y su *teoría del conocimiento* están tan interrelacionadas que habitualmente se tratan de forma conjunta. Platón estaba convencido de que el conocimiento se podía alcanzar y que éste tiene dos características esenciales. La primera, el conocimiento debe ser certero e infalible. La segunda, el conocimiento debe tener como objeto lo que es en verdad real, en contraste con lo que es sólo apariencia, para Platón lo que es real tiene que ser fijo, permanente e inmutable. Así, Platón identificó lo real con la esfera ideal de la existencia en oposición al mundo físico del devenir. Una consecuencia de este planteamiento fue el rechazo de la afirmación de que todo conocimiento se deriva de la experiencia. Pensaba que las proposiciones derivadas de la experiencia tienen, a lo sumo, un grado de probabilidad. No son ciertas. Mas al contrario, los objetos de la experiencia son fenómenos cambiantes del mundo físico, por lo tanto, los objetos de la experiencia no son objetos propios del conocimiento.

Esta misma situación es la que hemos identificado en el capítulo anterior cuando, presentando el concepto de patrón en IPO, se apreciaba que

los mismos se basaban fundamentalmente en la identificación de elementos comunes que aparecen en ejemplos concretos de interfaces de usuario. En IS, al contrario que en IPO, la tendencia es a la identificación de modelos independientes de la plataforma de los que más tarde, y por transformación, se obtienen componentes dependientes de ella de forma automática o semiautomática.

La *teoría del conocimiento* de Platón quedó expuesta principalmente en La República (Roser, 2001), en concreto en su discusión sobre *la imagen de la línea divisible* (véase Fig. 5-2) y *el mito de la caverna*. En la primera, Platón distingue entre dos niveles de saber: *opinión* y *conocimiento*. Las declaraciones o afirmaciones sobre el mundo físico o visible, incluyendo las observaciones y proposiciones de la ciencia, son sólo opinión. Algunas de estas opiniones están bien fundamentadas y otras no, pero ninguna de ellas debe ser entendida como conocimiento verdadero. El punto más alto del saber es el *conocimiento*, porque concierne a la razón en vez de a la experiencia. La *razón*, utilizada de forma debida, conduce a ideas que son ciertas y los objetos de esas ideas racionales son los *universales verdaderos*, las *formas eternas* o *sustancias* que constituyen el mundo real.

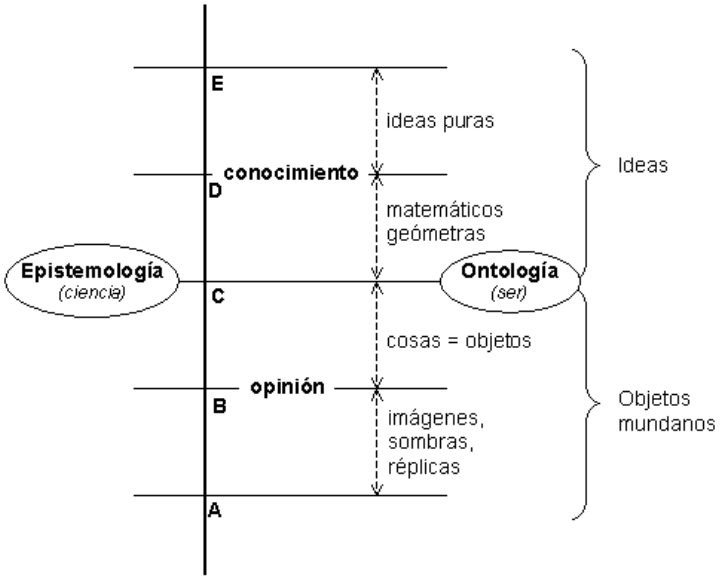


Figura 5-2. Divisiones de la Línea platónica (Roser, 2001)

En el mito de la caverna, Platón describe a personas encadenadas en la parte más profunda de una caverna. Atados de cara a la pared, su visión está limitada y, por lo tanto, no pueden distinguir a nadie. Lo único que se ve es la pared de la caverna sobre la que se reflejan las sombras producidas por estatuas de animales y objetos que pasan delante de una gran hoguera

resplandeciente. Uno de los individuos huye y sale a la luz del día. Con la ayuda del sol, esta persona ve por primera vez el mundo real y regresa a la caverna diciendo que las únicas cosas que han visto hasta ese momento son sombras y apariencias y que el mundo real les espera en el exterior si quieren liberarse de sus ataduras. El mundo de sombras de la caverna simboliza para Platón el mundo físico de las apariencias. La escapada al mundo soleado que se encuentra en el exterior de la caverna simboliza la transición hacia el mundo real, el universo de la existencia plena y perfecta, que es el objeto propio del conocimiento.

La teoría de las ideas se puede entender mejor en términos de entidades matemáticas. Un círculo, por ejemplo, se define como una figura plana compuesta por una serie de puntos, todos equidistantes de un mismo lugar. Sin embargo, nadie ha visto en realidad esa figura. Lo que la gente ha visto son figuras trazadas que resultan aproximaciones más o menos acertadas del círculo ideal. De hecho, cuando los matemáticos definen un círculo, los puntos mencionados no son espaciales, sino lógicos. No ocupan espacio. No obstante, aunque la forma de un círculo no se ha visto nunca, los matemáticos y otros sí saben lo que es. Para Platón, por lo tanto, la forma de círculo existe, pero no en el mundo físico del espacio y del tiempo. El círculo existe como un objeto inmutable en el ámbito de las ideas, que sólo puede ser conocido mediante la razón. Las *ideas* tienen mayor entidad que los objetos en el mundo físico tanto por su perfección y estabilidad como por el hecho de ser modelos, semejanzas que dan a los objetos físicos comunes lo que tienen de realidad. Las formas circular, cuadrada y triangular son excelentes ejemplos de lo que Platón entiende por *idea*. Un objeto que existe en el mundo físico puede ser llamado círculo, cuadrado o triángulo porque se parece (*participa de* en palabras de Platón) a la idea de círculo, cuadrado o triángulo.

Platón concibió, como muestra la Fig. 5-2, las ideas de manera jerárquica: la idea suprema es la de Dios que, como el Sol en el mito de la caverna, ilumina todas las demás ideas. La idea de Dios representa el paso de Platón en la dirección de un principio último de explicación. En el fondo, la teoría de las ideas está destinada a explicar el camino por el que uno alcanza el conocimiento y también cómo las cosas han llegado a ser lo que son. En lenguaje filosófico, la teoría de las ideas de Platón es tanto una tesis epistemológica (teoría del conocimiento) como una tesis ontológica (teoría del ser).

La Ingeniería del Software tradicional puede considerarse platónica[1]. En ella se distinguen dos mundos, uno de ellos estaría asociado con la opinión

[1] *La IS tradicional puede considerarse platónica* en el sentido de que modelos y objetos perviven de forma aislada. En última instancia, la actividad de progra-

y otro con el conocimiento, el primero está ligado a los objetos mismos y el otro con su esencia en forma de modelos.

El problema principal que surge en IS es el del mantenimiento del propio software producido y de su gestión posterior. El problema, en este contexto, es el que planteó Aristóteles: *¿cómo explicar el movimiento o cambio de los seres sensibles si las ideas no causan en ellas ni movimiento ni ningún cambio?*

5.2.2 Aristóteles y la tendencia actual en Ing. del Software

Las objeciones que Aristóteles formuló contra la teoría de las ideas de Platón se pueden reducir a las siguientes:

(a) La duplicación innecesaria de las cosas. Aristóteles demuestra que ese mundo de las ideas que Platón construye con objeto de justificar las cosas sensibles es una duplicación del mundo de las realidades que resulta totalmente innecesaria, es decir, no hay un mundo inteligible de las ideas contrapuesto o distinto del mundo sensible.

(b) El mundo de los conceptos tendría que ser infinito, lo cual es absurdo: en efecto, si dos cosas son semejantes, son semejantes porque ambas participan de una misma idea, entonces para que se diera la semejanza haría falta otro concepto, el de la semejanza, y, para advertir la semejanza entre este tercer concepto y la cosa, otro tercer concepto también sería necesario, y así sucesivamente. Esto resulta absurdo.

(c) Si hay conceptos de cada cosa, también deberá haberlos de las relaciones existentes entre las cosas, puesto que también las percibimos intuitivamente.

(d) Si hay ideas de lo positivo, por ejemplo, la calidad, también deberá haberlas de lo negativo, así como de los diferentes rangos o dimensiones que puedan adquirir las mismas.

(e) La teoría de las ideas no explica la producción, la génesis de las cosas: las ideas platónicas son conceptos personificados, y éstos para lo único que pueden servir es para dar razón de lo que las cosas son, pero en ningún momento para explicar cómo las cosas llegan a ser.

Pero Aristóteles, con su crítica, no se contenta con traer los conceptos platónicos del cielo a la tierra, una de las principales criticas que hace a la

---

mación está ligada al uso de lenguajes de programación de alto nivel, mientras que los modelos, cuando se contemplan, son meros elementos de documentación.

filosofía platónica consiste en reprocharle que los conceptos no tienen fuerza genética, son inmóviles e inmutables; pero la inmutabilidad en lo físico es algo mortal y monstruoso. Por ello, considera que pretender conocer la naturaleza sin preocuparse del movimiento o cambio de los seres es condenarse al fracaso.

La idea de movimiento que tiene Aristóteles, no es la de un físico, sino la de un biólogo. En este sentido, el movimiento es toda alteración cuantitativa o cualitativa que se produce en cualquier sustancia, y es una propiedad de los seres naturales. Aristóteles clasifico los seres en:

- Seres *naturales*: son todos aquellos que llevan dentro de sí mismos la capacidad de cambio o movimiento, y la ciencia que les es propia es la Física-Biología.

- Seres *artificiales*: son aquellos que no llevan dentro de sí este principio de cambio, y su ciencia más afín es la Tecnología.

- Seres *conceptuales*: son las entidades matemáticas y su ciencia es la Matemática.

- *Ser primero*: es el primer Motor Inmóvil, el que da sentido a la naturaleza, y su ciencia propia es la Metafísica.

La actual tendencia de la Ingeniería del Software está más próxima a las ideas de Aristóteles que a las de Platón. El problema que se ha encontrado la Ingeniería del Software, con la crisis crónica que adolece, es el del mantenimiento, que también podría entenderse como evolución o movimiento ante un cambio en las especificaciones o requisitos. Para dar respuesta a ese *movimiento* se ha propuesto como tendencia el *metamodelado*.

El *metamodelado* es un mecanismo que permite definir formalmente lenguajes de modelado. Así, un metamodelado de un lenguaje es una definición precisa de sus elementos, mediante conceptos y reglas de cierto metalenguaje, necesarios para crear modelos en ese lenguaje. Básicamente se trata de usar modelos para describir otros modelos.

De una forma curiosamente similar a la propuesta aristotélica en la literatura reciente disponible, en IS también se disponen de cuatro niveles o capas arquitectónicas que pueden asociarse con las diferentes entidades o seres aristotélicos:

- Capa de instancias: constituida por los datos que se desean describir.

- Capa de modelo del sistema. En ella se representa el modelo de un sistema software. Los conceptos de este nivel representan categorías de las intancias del nivel anterior.

- Capa de metamodelo: compuesta de las descripciones que definen la estructura y la semántica de los metadatos, es decir, lenguajes abstractos para describir diferentes clases de datos.

- Capa de Meta-metamodelo: constituida por la estructura y semántica de meta-metadatos. Es decir, es un lenguaje abstracto para definir diferentes clases de metadatos.

Un resumen de las relaciones que pueden establecerse entre las propuestas filosóficas clásicas y la actual tendencia en desarrollo del software es la que se recoge en la Tabla 5-1.

Tabla 5-1. Filosofía platónica *vs* Ingeniería del Software

	Filosofía (seres)	**Ingeniería del Software (capas)**
Opinión	Naturales	Información
↑	Artificiales	Modelo
↓	Conceptuales	Metamodelo
Conocimiento	Ser primero	Meta-Metamodelo

En relación con esta tendencia, el desarrollo de software tiende hacia la industrialización. El desarrollo de productos software se desea obtener fruto de un proceso automático o semiautomático donde la especificación basada en la utilización de modelos tome un papel relevante.

En esta tesis doctoral se apuesta por una visión aristotélica de la realidad. En ella los modelos residen junto a los objetos y se utilizan no sólo con la intención de documentar sino que adquieren condición generativa y pueden ser utilizados para crear esos mismos objetos en su versión virtual. La pregunta inmediata es *dónde queda la calidad* en esa concepción

5.3 Retomando la usabilidad y el desarrollo de software

Teniendo presente la tendencia de desarrollo manifestada, si denotamos un producto software elaborado, concretamente su interfaz de usuario, utilizando el lenguaje de alto nivel *L*, de la forma: α_L^{UI}. Muchas de las características que constituyen esta interfaz de usuario solamente pueden ser evaluadas cuando el producto software ya está desarrollado en su totalidad o

en una fase muy avanzada de desarrollo. Esto se ha constatado en función de los métodos de evaluación (inspección: inspección heurística, recorridos cognitivos, etc. y testeo y observación: evaluación con usuarios, *thinking-aloud*, etc.) y las métricas (Constantine et al., 1999; Ivory et al., 2001) recogidas en el capítulo tercero de este documento.

Actualmente, el desarrollo del software tiende a abordarse apoyándose en la realización de transformaciones, siguiendo una formula como la siguiente:

$$p(\sigma_{\Sigma}, \mu_{\Sigma I}) = \beta_I$$

donde σ_{Σ} es una especificación utilizando el lenguaje Σ, $\mu_{\Sigma I}$ denota una asociación entre una especificación utilizando el lenguaje de especificación Σ y una implementación realizada utilizando el lenguaje de alto nivel *I*, β_I es una implementación en la que se utiliza el lenguaje de alto nivel *I*.

Ante esta coyuntura, cabe preguntarse si las características de calidad en general, y de usabilidad en particular, que son inherentes de la interfaz de usuario son, siguiendo la tendencia mostrada en el actual desarrollo de software, dependientes o independientes de la plataforma.

Si las características de usabilidad fueran independientes de la plataforma podrían ser especificadas utilizando el lenguaje Σ. Si, por el contrario, fuesen dependientes de ella debieran ser recogidas utilizando características de alto nivel utilizando el lenguaje de alto nivel *I*. Se analizará, seguidamente, cada uno de estos supuestos, justo antes de introducir la principal propuesta de esta tesis doctoral en la que se unificarán experiencia y calidad, considerando ambas en el proceso de desarrollo de software en general, y de interfaces de usuario en particular, y donde parte de la calidad se considerará a un nivel independiente de la plataforma.

Previamente, si se retoma la filosofía aristotélica en ella se consideran los seres vivos como unidades. Estas unidades, a su vez, estaban constituidas de dos elementos: *materia* y *forma*. La *materia* es aquel sustrato fundamental o principio informe e indiferente del que están hechas todas las cosas, es decir, es aquel sustrato que permanece cuando algo ha dejado de ser lo que era y se ha transformado en otra cosa distinta. Por el contrario, la *forma* es el principio capaz de organizar la *materia*; es el principio de configuración y ordenación de la materia.

Si esta distinción de elementos se extrapola al terreno de las interfaces de usuario. También podríamos identificar dos elementos, unos dependientes y otros independientes de la plataforma, al modo y manera de *forma* y de *materia* respectivamente. En esta tesis doctoral, y fruto de elegir trabajar a un nivel de abstracción superior al habitual, nos interesará documen-

tar el conocimiento de la *materia* frente al que puede ofrecer el de la *forma*. La siguiente pregunta es si en esa materia tienen cabida características de calidad.

5.3.1 La interfaz de usuario es *materia*

El hecho de que la interfaz de usuario conlleve una parte de componente visual es indudable y con ello todos estamos de acuerdo, pero junto con ella se constata que la interfaz está determinada por características como la usabilidad que tiene una componente relacionada con la facilidad de uso y la utilidad (*véase* el capítulo tercero) que ofrece el producto software al va asociada. Si esto es así, debe poder expresarse esta vertiente utilizando un lenguaje de especificación. Esa especificación debiera realizarse a un nivel de abstracción suficiente para permitir su asociación a diferentes dispositivos o plataformas.

Tradicionalmente, en las propuestas presentadas y relacionadas con el desarrollo de interfaces de usuario y su prototipado rápido, la interfaz se infiere o se deriva de la especificación conceptual elaborada para especificar el modelo de dominio y de tareas, pero en ese proceso de traducción o asociación no se consideran características de calidad (*véanse* las propuestas recogidas en el capítulo cuarto), así se da lugar a interfaces de usuario pobres en ese ámbito. En este terreno, la consideración de la experiencia y de características de calidad asociadas con la interfaz de usuario pueden ser determinantes para lograr la calidad desde el punto de vista de la interfaz de usuario, concretamente, desde el que determina la usabilidad de la misma. En este capítulo, nuestra propuesta asociará modelo de calidad centrado en la usabilidad y experiencia en forma de patrones. El fin último será el de la especificación de aquellos requisitos funcionales y no funcionales que debe aportar la interfaz como *materia*.

En este sentido, se han identificado pocas contribuciones que centren su objetivo en recoger qué características deben considerarse al documentar la experiencia. Entre las reseñables, (Mahemoff et al., 1998) especificaron qué características de calidad debe considerar la experiencia documentada cuando se pretende desarrollar un lenguaje de patrones orientado a la usabilidad. Dicha relación se muestra de forma resumida en la Fig. 5-3.

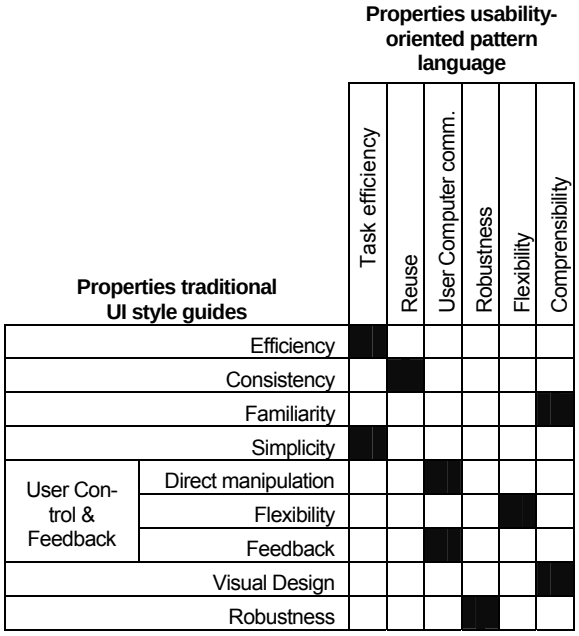


Figura 5-3. Características deseables en la experiencia documentada (derivado de (Mahemoff et al., 1998))

5.3.2 La interfaz de usuario es *forma*

El establecimiento de la asociación entre interfaz de usuario y *forma* es lo habitual. Lo meramente visual es la principal preocupación que se ha manifestado y buscado, desde el principio, con la concepción que se tiene de interfaz de usuario. De ello pueden surgir carencias fruto de no tener en cuenta otras consideraciones importantes que deben contemplarse. A nivel puramente visual, las guías de estilo ofrecen mecanismos para tener presentes características que debieran evitarse al diseñar las interfaces de usuario, indicando qué está bien y qué mal, veamos como ejemplo algunas recomendaciones para diseñar sitios web[1] (véase Tabla 5-2).

Como se observa, tras una lectura detenida de la información recogida en la Tabla 5-2, las características reflejadas están relacionadas con el aspecto final que ofrece la interfaz de usuario (en el caso recogido una interfaz de usuario relacionada con sitios web). En definitiva, aspectos de orga-

[1] Williams, R., Tollett, J.: The non-designer's web book (2nd edition). Peachpit Press. 2000.

nización, selección de componentes concretos, fuentes, colores, tamaños, entre otras consideraciones dependen de la plataforma, pero pueden ser derivados de una especificación abstracta independiente de la misma.

Tabla 5-2. Características asociadas con buenos y malos diseños Web

Características de un mal diseño	Características de un buen diseño
- Elegir un color de fondo gris	- El color de fondo no interfiere en el texto
- Elegir una combinación de colores para texto y fondo que dificulten la lectura	- El texto es lo suficiente para leerse, pero no demasiado grande
- Utilizar tamaños de fuente demasiado pequeños	- La jerarquía de la información es clara
- Demasiado texto agrupado en la parte izquierda	- Los botones de navegación son fáciles de entender y de usar
- Texto distribuido por toda la página	- La navegación es consistente a través del sitio web
- Utilizar mayúsculas, cursiva o negrita para elaborar los párrafos	- Los frames, si se utilizan, no son demasiados
- Subrayar texto que no sean enlaces	- El color de los enlaces de navegación es consistente con el resto de colores
- Utilizar bordes de enlaces de color azul	- Los enlaces están subrayados, así se distinguen claramente
- Disponer enlaces muertos	- Toda imagen tiene su equivalente textual
- Disponer enlaces que no dejan claro dónde llevarán	- Las páginas se descargan rapidamente
- Ficheros gráficos excesivamente pesados	- Las páginas de un sitio son consistente en cuanto a diseño y aparición de elementos coincidentes
- Imágenes sin etiquetas alternativas	- Los gráficos animados se detienen por si solos
- Abusar de animaciones	
- Introducir scrolls	
- Diseñar navegaciones complejas	
- Abusar del uso de marcos	

5.4 Tres direcciones para mejorar el desarrollo de IU

En esta tesis doctoral se propone mejorar la calidad de las interfaces de usuario desarrolladas gracias a hacer énfasis en tres direcciones:

(1) caracterizando la propia calidad,
(2) utilizando la experiencia en diseño y desarrollo disponible, e
(3) integrando calidad y experiencia en un entorno de desarrollo que posibilite su utilización con eficiencia, efectividad y satisfacción.

Antes de comenzar con el desarrollo pormenorizado de cada una de estas vías de mejora del proceso y producto de una interfaz de usuario conviene dejar claros una serie de conceptos. (Booth, 1989) define *interacción* como el intercambio de símbolos entre dos o más partes, asignando los

participantes en el proceso comunicativo los significados a esos símbolos. Se veía ya en el capítulo segundo como la Interacción Persona-Ordenador puede ser analizada en función de su estilo, de su estructura, de su contenido y de su comportamiento. Algunas de estas componentes, especialmente aquellas relacionadas con el estilo y la estructura, pueden considerarse y tratarse fácilmente, por ejemplo en un entorno Web, mediante el uso de dos técnicas: las hojas de estilo y las transformaciones asociadas al uso de lenguajes XSL.

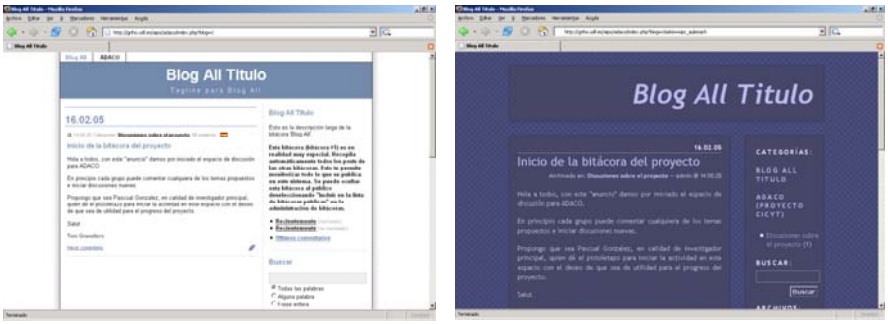


Figura 5-4. Apariencia *vs* realidad en los desarrollos para la Web

En la Fig. 5-4 pueden verse diferentes versiones de una misma página logradas con la utilización de diferentes hojas de estilo. La idea de las hojas de estilo, aplicadas a las páginas web no puede ser más simple. Igual que ocurre en un programa de autoedición o procesador de texto, la hoja de estilo tiene definido el formato del texto, los encabezados, subtítulos, etc. y se puede detallar tanto como se quiera. Cualquier cambio que se haga en la hoja de estilo se aplicará globalmente en todas las páginas web basadas en aquella hoja de estilo.

El poder que tiene esta forma de definir el formato de las páginas es enorme. Así, es posible cambiar el aspecto de una web entera, con todas las páginas que tenga, con sólo modificar una sola hoja de estilo. Estaríamos actuando directamente sobre la *consistencia* de un sitio web, una característica recogida dentro del modelo de calidad propuesto en el capítulo tercero de este documento.

A cada elemento de código de la página (por ejemplo, texto normal, encabezados, enlace...) se le atribuyen unas propiedades referentes al tipo de texto, colores, tamaño, posición, etc.. Se pueden crear tantas variantes o *clases* como se deseen para cada elemento. Por ejemplo, pueden definirse varios tipos de texto: uno blanco sobre fondo negro y otro inverso, y otros a mayor tamaño y color diferente para subtítulos, citas, etc., otro pequeño para pies de imagen, etc.

Lo más atractivo de usar CSS es que, una vez definido el estilo, el desarrollador no debe preocuparse de especificar el tipo de letra, tamaño, color, alineación, o cualquier otro atributo que tenga que ver con la *forma* que ofrezce el producto software. Al contrario, sólo con marcar el texto e indicar el estilo que le corresponde se aplica todo automáticamente y sin fallos.. Aún así, las hojas de estilo presentan algunas limitaciones:

- CSS es eficaz para describir formatos y presentaciones, pero no sirve para decidir qué tipos de datos deben ser mostrados y cuáles no.

- Esto es, CSS se utiliza con documentos XML en los casos en los que todo su contenido debe mostrarse sin mayor problema.

- En función de las limitaciones anteriores surge XSLT, que no sólo permite especificar cómo queremos presentar los datos de un documento XML, sino que también sirve para filtrar los datos de acuerdo a ciertas condiciones.

- Su modo de operar se parece a un lenguaje de programación tipo PHP. Permite incluir en una página web operaciones con cálculos numéricos, ordenación de datos, condicionales, funciones lógicas y otras funciones típicas de las aplicaciones profesionales.

Como complemento a las CSS, El W3C creó un nuevo lenguaje de hojas de estilo llamado XSL (*eXtended StyleSheet Language*) que además de lo que ofrece el CSS tiene la capacidad de funcionar como un lenguaje de transformación, alcanzándose la funcionalidad que se describía más arriba. XSL es en verdad dos estándares muy diferentes:

- XSLT: es un lenguaje que describe cómo debe transformarse un archivo XML en otro archivo XML. Por ejemplo, un archivo conteniendo *tags* que describen una receta puede convertirse mediante este sistema a otro archivo conteniendo *tags* que solamente describen la posición del texto en una página impresa. Obviamente, en semejante transformación hay información que se pierde. La idea es que aplicando diferentes archivos XSLT se puedan obtener archivos XML diferentes, adecuados a diferentes presentaciones o usos de la información. Esta parte del estándar está ya publicada en su versión 1.0 final, y ya existen varias implementaciones de diferente calidad.

- XSL FO[1]: Un archivo XSLFO es una especie de HTML muy sucio de *tags* de colores, tamaños y posición. Es un archivo que no preserva nada de la semántica de la información original, solamente describe cómo debe mostrarse en pantalla, o en papel. Es similar, en concepto, al lenguaje *PostScript*, o quizá al T_EX. Este formato, que obviamente es un formato XML, se puede usar generándolo mediante una transformación XSLT. Siguiendo el ejemplo que se utilizo antes ya podemos describir un circuito bastante completo de lo que pasa con la información:

$$XML + XSLT \approx objetosFO$$

  Esta parte del estándar es un trabajo en progreso y todavía ningún *browser* está cerca de soportarlo. El tiempo dirá si se convierte en un estándar de la industria o no. Lo que sí se está empezando a usar mientras tanto como lenguaje de presentación es XML+CSS.

Otras tecnologías asociadas son DOM (*Document Object Model*) y SAX, que son APIs estándares para acceder a los *árboles* de información XML.

Recogiendo estas alternativas se han considerado dos aspectos relevantes asociados con la interacción como son la presentación y la estructura. Ambos tienen mucho que ver con la *forma* e influyen en la calidad final de la interfaz del usuario cuando un usuario hace uso del producto software, pero su descripción es fundamentalmente temporal, transitoria y cambiante, ya que lo que establecen son características tangibles y correspondientes al mundo físico, a diferencia de lo que correspondería con una descripción inteligible e imperecedera fruto de trabajar a un nivel superior, a un nivel de descripción de interfaces de usuario abstracta donde también cabe contemplar las características de calidad de un producto software (Bass et al., 2001).

Esta tesis doctoral busca conseguir aprovechar la flexibilidad que permiten los desarrollos basados en transformaciones a nivel concreto y asociar este mecanismo con otro que aporte una visión abstracta independiente de la plataforma, donde prime la *materia* frente a la *forma*.

5.4.1 Considerar la calidad

En el Capítulo tercero de esta tesis doctoral se consideró el concepto de calidad y sus implicaciones, así como los estándares disponibles que caracte-

[1] FO es el acrónimo de *formatting objects*.

rizaban dicho concepto. Hay diferentes estándares internacionales disponibles que tratan con la definición y caracterización de la calidad. En dos de esos estándares, el ISO 9241-11 (ISO, 1998) y el 9126 (ISO, 1991; revisado en 2001), se han tratado conceptos relacionados con la interacción entre usuario y ordenador considerando el concepto de usabilidad. La diferencia entre uno y otro estándar está en su objetivo, el primero de ellos considera la usabilidad como proceso y el segundo como producto (véase el capítulo tercero y el Apéndice C). Otro estándar posterior fruto de la revisión del estándar ISO 9126, ha venido a aproximar ambos puntos de vista, es el estándar ISO 9126-4 que presenta y caracteriza el concepto de calidad en uso, con él se hace hincapié en el hecho de la necesidad de considerar facilidad de uso y utilidad.

Una de las principales limitaciones que tiene la caracterización de la calidad de un producto software es la profundidad ofrecida por los modelos de calidad propuestos. Una caracterización plena de la calidad ofrecida por un producto software, probablemente, no es necesaria ni posible (Olsina, 2001), además de ser subjetiva y dependiente de las necesidades concretas que se establezcan para cada producto software que se desarrolla influenciado por el contexto de utilización y por las características de los usuarios finales que utilizan dicho producto.

También en el capítulo tercero hacíamos referencia a una serie de criterios ergonómicos (Bastien et al., 1993) que, respaldados por una serie de experimentos reseñados y a través de los que se justifica su utilización para labores de evaluación de interfaces y de clasificación de las guías de estilo, pueden pasar a ser utilizados para formar parte de una propuesta de modelo de calidad centrado en la usabilidad. Seguidamente se recoge tal propuesta en la Tabla 5-3.

En esta tesis doctoral los criterios ergonómicos se utilizan tanto para labores de evaluación como para abordar tareas de organización de la experiencia. Para conseguir abordar ambos retos los criterios se utilizan como criterios y subcriterios de calidad asociados con los factores de calidad establecidos y propuestos en estándares internacionales.

Las ventajas que aporta disponer de un modelo de calidad pasan por dar soporte al diseñador y desarrollador, especialmente al poco experimentado. El mismo dispone de mucha experiencia en diferentes formatos y niveles de abstracción. Una de las más extendidas formas de documentar la experiencia es mediante el uso de guías de estilo. Muchas guías de estilo son útiles para asegurar la consistencia en el desarrollo de aplicaciones para ciertas plataformas, compañías (Microsoft, Apple, IBM, etc.) o grupos de usuarios. Dicha consistencia es uno de los principios de calidad deseables en cualquier producto software y puede conseguirse a través del uso de técnicas como las vistas en el apartado anterior, donde se trabajaba a nivel

de objetos de interacción concretos utilizando estilos arquitectónicos. Pero otros muchos principios, factores y criterios deben considerarse también. Una decisión crucial es la determinación de qué principios aplicar, ya que diferentes situaciones y contextos aconsejan una elección diferente de los mismos.

Tabla 5-3. Propuesta de modelo de calidad centrado en la usabilidad

Factor	Criteria	Sub-criteria	C[1]
Usability	Understandability[2]	compatibility	H
		legibility	M
		prompting	M
		Imm. feed-back	H
		Sig codes & beh.	H
		helpfulness	M
	Learnability[3]	grouping	H
		physical workload	H
		minimal actions	H
		conciseness	L
		information density	M
		consistency	H
	Operability[4]	explicit user action	H
		user control	H
		user experience's	H
		flexibility	M
		error protection	H
		quality error msg	L
		error correction	M
		privacy policies	L
		accesibility	H

Si se tienen presentes técnicas asociadas al Diseño Centrado en el Usuario y a la Ingeniería de la Usabilidad, además del estudio de las tareas y de los usuarios objetivo de satisfacción por parte del producto software, es

[1] C≈contribución. H es una contribución alta (*high*), M es una contribución media (*medium*) y L es una contribución baja (*low*)

[2] Atributos del software que hacen referencia al esfuerzo que debe hacer el usuario para reconocer la lógica y aplicabilidad de la aplicación.

[3] Atributos que hacen referencia al esfuerzo que debe realizar el usuario para aprender a usar el producto software (p. e.: controles, entradas, salidas).

[4] Atributos software que hacen referencia al esfuerzo necesario para operar y controlar la operación realizada al utilizar el producto software.

necesario caracterizar la calidad. Para ello hay que elaborar un modelo de calidad cada vez, para cada aplicación o reutilizarlo de una vez para la siguiente en función de que se disponga de un modelo de calidad abierto para su enriquecimiento, pero cerrado para su utilización.

La consideración de un modelo de calidad caracterizado haciéndose uso de criterios de diseño permite ligar evaluación y proceso de desarrollo (Mahemoff et al., 1998). La inclusión de un modelo de calidad en un proceso metodológico de desarrollo de software y de interfaces de usuario pasa por (*véase* Fig. 5-5):

(1) detectar fallos relacionados con la calidad de una interfaz de usuario,
(2) diagnosticarlos e identificar sus causas siguiendo un proceso de evaluación basado en la inspección heurística (Nielsen, 1993) y en la realización de experimentos con usuarios reales según proceda,
(3) establecer prioridades en cuanto a su tratamiento y subsanación,
(4) determinar cómo abordar el tratamiento de las deficiencias identificadas haciendo uso de experiencia documentada y probada y,
(5) estimar el beneficio y coste del tratamiento de las deficiencias identificadas en el paso anterior.

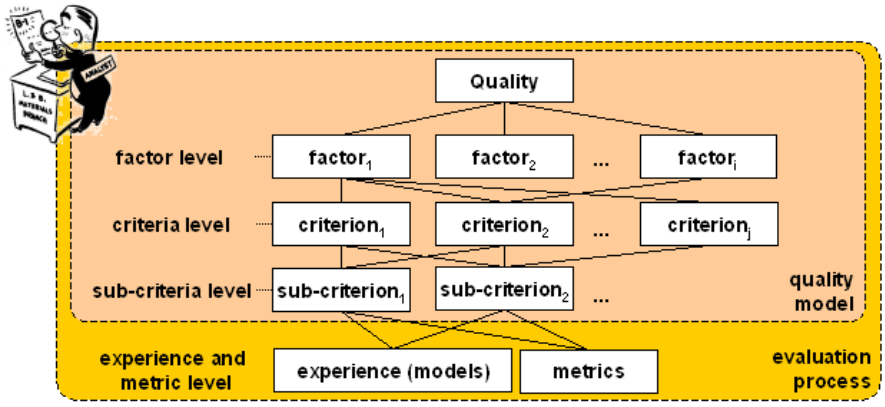


Figura 5-5. Modelo de calidad y proceso de evaluación

En cualquier caso, la calidad de un sistema, aunque el punto de vista considerado en esta tesis doctoral sea el ofrecido desde la interacción entre persona y ordenador y desde la visión que ofrece la interfaz de usuario, viene determinada por muchas más consideraciones que la mera presentación o la selección y disposición de unos componentes de interacción u otros, como se ha reflejado en los apartados anteriores. En la calidad también influyen factores relacionados con las tareas que afectan al usuario fi-

nal a través de aspectos relacionados con la navegación, la calidad de la presentación, y la adecuación de funciones y contenidos; factores relacionados con las prestaciones que afectan a la eficiencia y consideraciones económicas directa e indirectamente logradas a través de su utilización, así como aspectos dependientes de tiempos de respuesta, productividad, fiabilidad y robustez; y también todos aquellos factores relacionados directamente con el proceso de desarrollo en forma de complejidad del código, legibilidad, flexibilidad, facilidad de mantenimiento, cohesión, acoplamiento, etc.

En nuestra propuesta, a la hora de caracterizar la calidad, se ha apostado por la elección de criterios ergonómicos por las ventajas que ofrecen y que fueron recogidas en el capítulo tercero. Además, siendo la *ergonomía* el estudio del trabajo en relación con el entorno en que se lleva a cabo (el lugar de trabajo) y con quienes lo realizan (los trabajadores), hemos apostado por criterios ergonómicos como elementos definitorios en nuestra propuesta de modelo de calidad ya que con ellos es posible determinar cómo diseñar o adaptar el lugar de trabajo al trabajador a fin de evitar distintos problemas de salud y de aumentar la eficiencia. En otras palabras, para hacer que el trabajo se adapte al trabajador en lugar de obligar al trabajador a adaptarse al lugar de trabajo.

En dichos criterios se tienen en cuenta tanto características relacionadas con el proceso como con el producto. Además, el estudio de aspectos ergonómicos ha estado ligado al propio estudio de la calidad de un producto software y de su uso y en él se tienen en cuenta consideraciones relacionadas con el usuario y con el contexto en el que se utiliza el producto software, aspectos de reiterada aparición en las definiciones recogidas en el Capítulo 3 relacionadas con calidad y usabilidad. Los criterios ergonómicos siguiendo una descomposición jerárquica, típica en las propuestas de modelo de calidad, se sitúan en los niveles destinados a la consideración de criterios y subcriterios y llevarían asociados experiencia y/o métricas según disponibilidad. La experiencia considerada podría ser cualquiera de la recogida en el Capítulo 4 pero especialmente se tratará de experiencia formulada en forma de patrones y asociada a modelos en sus diferentes dimensiones.

5.4.2 Considerar la experiencia

La Filosofía, desde la antigüedad, ha reconocido la importancia del lenguaje. Con él los humanos hacen referencia a objetos, pero esa referencia no es natural sino convencional. De esta forma se cuestiona si el lenguaje sir-

ve para describir realmente al mundo real y si las oraciones y las palabras designan propiedades reales de los objetos, procesos y personas.

Tradicionalmente, para la recopilación de la experiencia se utiliza el lenguaje natural, así se han recogido gran cantidad de guías de estilo: principios, estándares y recomendaciones de diseño que han sido tratadas en el Capítulo cuarto. También se utiliza el lenguaje natural en la redacción y escritura de patrones de interacción, pero a falta de otras consideraciones *generativas*, estos últimos adolecerán de las mimas carencias que presentaban aquellas otras formas de documentación.

En esta tesis doctoral se complementan los formatos de documentación propuestos, concretamente PLML (Fincher et al., 2003), eliminando aquellas secciones que puedan tener cualquier contenido (ANY) o que no estén adecuadamente estructuradas. En este sentido, se proponen modificaciones sobre los secciones: *forces*, *diagram*, *examples* y *authors.* Aparte, se añade un sección adicional que relaciona el uso del patrón y los criterios de calidad sobre los que influye la solución que éste aporta.

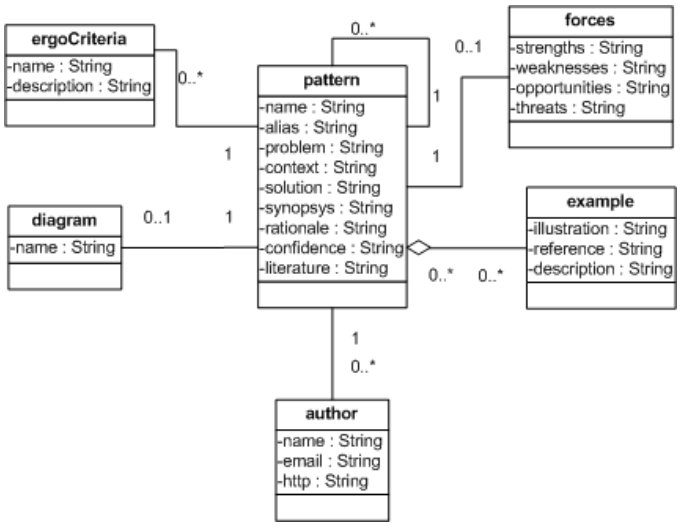


Figura 5-6. Modelado conceptual de un patrón

La Fig. 5-6 muestra el esquema conceptual del concepto de patrón que se considera en esta tesis doctoral. La clase *ergoCriteria* recoge los criterios considerados en el modelo de calidad y, en definitiva, las posibles características de calidad que el uso de cada patrón puede suponer, es decir, si la solución aportada por un patrón es un mecanismos que abunda en proporcionar un feedback inmediato será esa característica la que quedará recogida en la redacción del patrón. Más tarde esas características ergonómicas proporcionarán dos facilidades, por un lado la posibilidad de buscar

directamente recursos para potenciar el logro de calidad. Por otro, y una vez el producto ya está suficientemente elaborado para elaborar pruebas con usuarios reales, las taras identificadas por usuarios reales en su interacción con el producto podrán ser caracterizadas con los criterios ergonómicos y el diseñador dispondrá de experiencia que ofrece solución a esas taras.

Las fuerzas (*forces*) o condicionantes internos y externos en la aplicación del patrón, y del núcleo de solución que le acompaña se caracterizan utilizando, un concepto de marketing, un análisis DAFO (Johnson et al., 1989; Hill et al., 1997). En marketing, los análisis DAFO sirven para identificar aquellas características, ya sean éstas internas, externas, positivas o negativas que ofrece un producto o una empresa en el mercado. Las siglas DAFO se corresponden con Debilidades, Amenazas, Fortalezas y Oportunidades, siendo su dimensión la mostrada en la Tabla 5-4.

Tabla 5-4. Ámbito de un análisis DAFO (Johnson et al., 1989)

	Positivas	**Negativas**
Exterior	Oportunidades	Amenazas
Interior	Fortalezas	Debilidades

En la documentación de un patrón, la sección dedicada a las fuerzas sirve para caracterizar el contexto en el que se puede aplicar el patrón, así como el contexto resultante de su aplicación.

La clase *diagram*, que aparece en la Fig. 5-6, sirve para potenciar la vertiente generativa que puede ofrecer un patrón, aunque la solución que éste aporte no sea directamente aplicable y se trate únicamente de un núcleo de solución. En nuestra propuesta, a través de esta sección se enlaza una especificación, utilizando un modelo, que muestra cómo poner en práctica la solución descrita previamente de forma textual. Para elaborar estas especificaciones o modelos se utilizan aquellas notaciones ligadas con actividades de especificación de modelos disponibles. Estas notaciones, a falta de unificación, se corresponden con aquellas más extendidas en la comunidad IPO para modelar el conocimiento que llevan asociado.

En la Fig. 5-7, se resaltan aquellos componentes que constituyen la especificación de un modelo ligado a un patrón. Nuestro principal interés es la documentación de patrones de interacción para ser utilizados en la confección y desarrollo de interfaces de usuario, en este sentido, la información almacenada en la sección *diagram* estará dirigida a describir un modelo de interfaz (*uiModel*). El mismo estará constituido por diferentes modelos asociados con la información relacionada con el dominio, las tareas y la presentación en su dimensión abstracta. El almacenamiento de es-

ta información se realizará en XML utilizando el lenguaje propuesto en usiXML (Limbourg et al., 2004).

Esta información, así dispuesta, permite la utilización de dichos modelos durante el proceso de desarrollo, como información *know-how*. Si cada patrón aparece asociado con factores, criterios y subcriterios de calidad, a través de la sección *ergoCriteria*, el diseñador dispondrá de información ligada al proceso, a través de la sección *diagram*, es decir, información que le indicará cómo especificar la solución asociada con el patrón y, a la vez, cómo lograr potenciar diferentes criterios de calidad. Además, si se dispone de un modelo de calidad que ofrece soporte al proceso de evaluación, como es el caso, con el que identificar taras en la interacción facilitada a través de la interfaz ofrecida, y puesto que los patrones van asociados a esos mismos criterios de evaluación de la calidad, el diseñador dispondrá automáticamente de información sobre cómo resolver la carencia identificada.

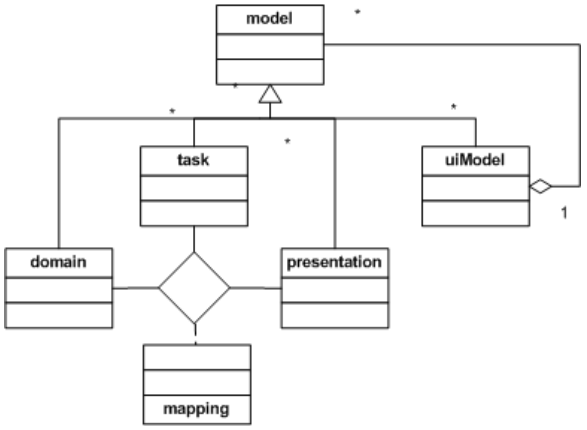


Figura 5-7. Modelos asociados con la descripción de un patrón

Las notaciones que se utilizan para describir cada uno de los modelos recogidos en la propuesta y reflejados en la Fig. 5-7, son los diagramas de clases de UML para el modelo de dominio, la notación CTT de (Paternò, 1999) para el modelo de tareas y una variante de la notación propuesta en usiXML (Limbourg et al., 2004) para especificar el modelo de presentación abstracta de la interfaz de usuario. Para documentar el posible mapping que surja entre los diferentes modelos anteriores se utiliza parte de la especificación propuesta en usiXML destinada al tratamiento de ese fin. Se utilizará sólo parte de las relaciones propuestas ya que el ámbito de aplicación de nuestra propuesta está circunscrito al ámbito abstracto y por lo tanto las relaciones entre abstracto y concreto no se documentan con el patrón. Junto con cada patrón, y en aquellos donde sea posible, se incluirá en la misma sección, *diagram*, información sobre patrones de diseño (Gamma

et al., 1995) que resultaría conveniente considerar una vez se alcance la fase de implementación.

Llegado el momento de la selección de componentes de interacción concreta, y disponiendo de una especificación en forma de componentes de interacción abstracta, se considerarán tablas de equivalencia como la recogida en la Tabla 5-5. En ella a cada elemento de especificación se le asocian diferentes componentes de interacción concreta con los que se puede lograr ofrecer aquella faceta o facetas perseguidas. La discriminación entre uno u otro componente de interacción concreto de entre los disponibles vendrá condicionada por la plataforma final.

Tabla 5-5. Asociación entre elementos de interacción abstracta y concreta GUI

Objetos de interacción abstracta (AIO) y facetas asociadas	Objetos de interacción concreta (Java Swing)
	
	
	
	
	
	

Una sección adicional, como es la representada a través de la clase *example* en la Fig. 5-6, está destinada al almacenamiento de un elemento fundamental para la puesta en práctica de los patrones en procesos de diseño centrados en el usuario. Con ellos el usuario, jugando el papel de lector del catálogo de patrones, identificará necesidades y requisitos que desea ofrecer a través de la interfaz que precisa. Esta labor se verá facilitada con la presencia de figuras en las que se muestran ejemplos asociados con la puesta en práctica de cada patrón. A estos ejemplos se les adjuntarán imágenes para facilitar su descripción.

Una última sección relacionada con información del autor o autores del patrón (clase *author* en la Fig. 5-6) está también disponible en la descripción que acompaña a cada patrón.

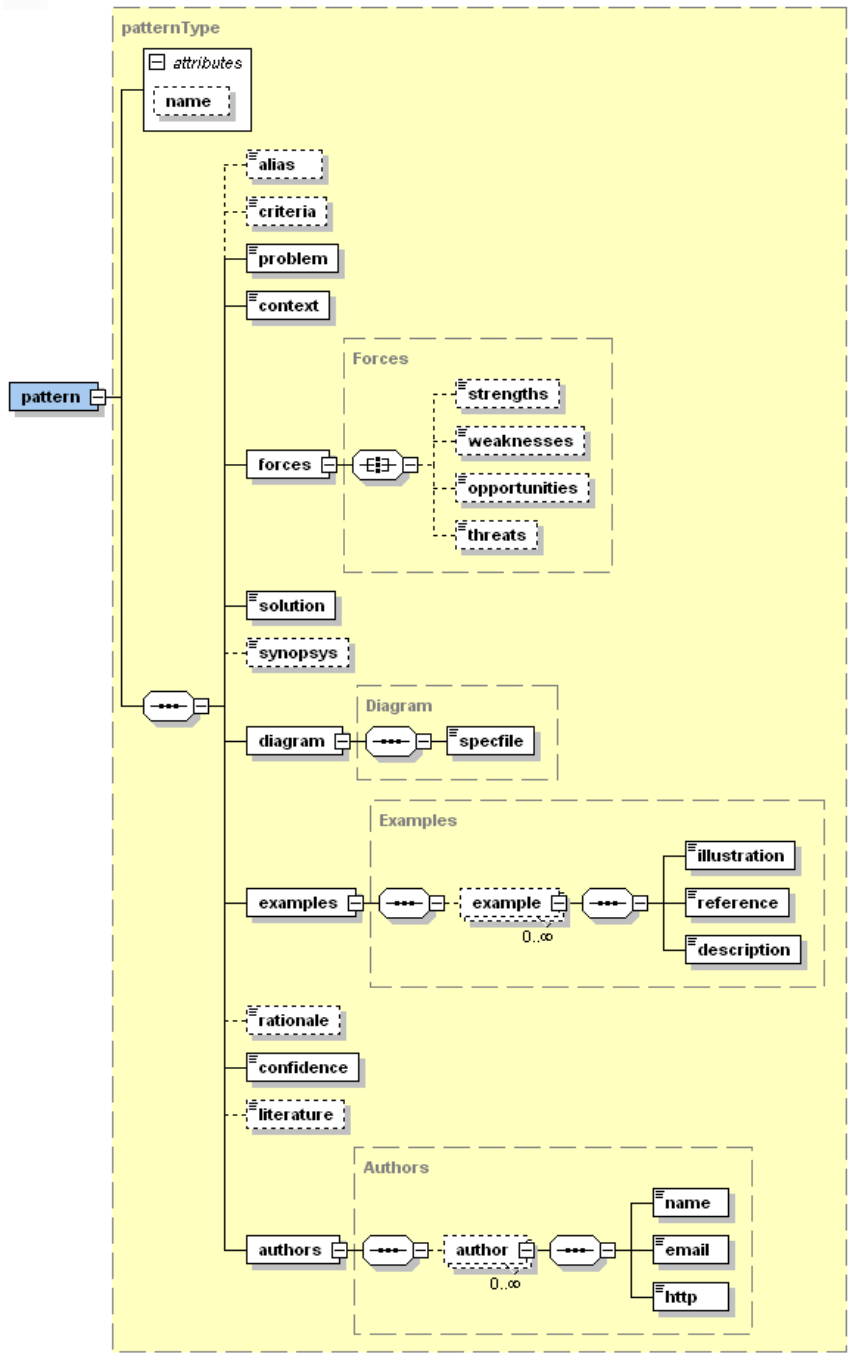


Figura 5-8. Esquema XML asociado con la descripción de un patrón

Con la intención de resolver las limitaciones relacionadas con una adecuada estructuración de algunas secciones utilizadas para la documentación de un patrón surgidas de la propuesta PLML, y fruto de la inclusión de las anteriores consideraciones se propone la sustitución del DTD por un esquema XML como el recogido en la Fig. 5-8.

La principal ventaja de la que hace gala la nueva descripción ligada a cada patrón está en que con la descripción adicional, fruto de la consideración de modelos, el patrón adquiere una dimensión añadida respecto a la tradicional, orientada al suministro de facilidades de documentación o mera descripción textual. El patrón adquiere dinámica y ésta la proporciona la documentación utilizando notaciones generativas. Esta propuesta es en esencia compatible con la actual tendencia en Ingeniería del Software (MDA) y con la tendencia de desarrollo basada en modelos utilizada para elaborar interfaces de usuario.

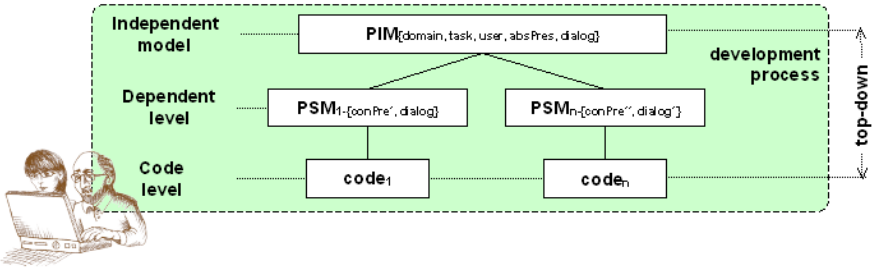


Figura 5-9. Desarrollo dirigido por modelos y experiencia en el desarrollo de interfaces de usuario (proceso)

En la Fig. 5-9 se muestra cómo los patrones pueden utilizarse en el proceso de desarrollo. A través de los modelos asociados con cada patrón en la sección *diagram*, los patrones se convierten en especificaciones independientes de la plataforma, y sirven para aunar dos procesos, en la práctica dependientes, como son el de desarrollo y el de evaluación.

5.4.3 Considerar el soporte

Los métodos y propuestas más inverosímiles pueden adquirir aceptación, o perderla, en función de las herramientas disponibles para facilitar su uso y viceversa. En la propuesta recogida en este capítulo dos elementos: calidad y experiencia, tienen necesidad de recibir soporte. Éste debe plasmarse en aplicaciones y productos software que faciliten su gestión. Tanto la calidad como la experiencia deben caminar conjuntamente a la hora de abordarse el problema de desarrollar productos software.

Los patrones, independientemente de su contrastada valía, no han contado con abundantes herramientas con las que facilitar su utilización con garantías, cabe preguntarse si herramientas como las destacadas en el capítulo cuarto, y asociadas con la utilización de catálogos de patrones, son suficientes y aprovechables para ser utilizadas en la puesta en práctica de nuestra propuesta. En principio, la respuesta es negativa debido a que los patrones se encuentran ligados a distintas notaciones de especificación, identificándose asociaciones entre ellas, y en las herramientas disponibles no se contemplan mecanismos con los que facilitar este proceso de reutilización.

Las características deseables que debieran disfrutar aquellas herramientas que pretendan dar soporte al desarrollo y gestión de patrones, en cualquiera de sus formas o asociados a diferentes disciplinas, pasan por:

(a) tener la flexibilidad suficiente para admitir descripciones en diferentes formatos o al menos permitir la visualización en los formatos más habituales disponibles,
(b) las herramientas desarrolladas debieran proporcionar mecanismos para la edición de cada uno de los campos relevantes asociados a la documentación de un patrón,
(c) un patrón es un elemento catalizador de comunicaciones y por lo tanto debiera ser accesible y manipulable. Un patrón no se trata de conocimiento reservado o situado fuera del alcance de la comunidad de usuarios independientemente de los *roles* que desempeñen,
(d) deben facilitar mecanismos con los que reutilizar la experiencia que en un patrón se recoge y de ésta potenciar la reutilización de las soluciones que aportan los patrones,
(e) el repositorio encargado de recopilar el conocimiento asociado con los patrones debe ser dinámico y no estático, debe ser posible actualizar dicho repositorio y enriquecerlo con nuevas aportaciones o revisiones de la experiencia allí almacenada,
(f) debe recogerse explícitamente en qué dirección, desde el punto de vista de la calidad, incide el uso de un patrón o conjunto de patrones,
(g) deben recogerse las relaciones entre patrones y deben proporcionarse facilidades de navegación entre los mismos,
(h) los patrones recogidos y manipulados corresponderán a diferentes intereses y en función de ello las notaciones utilizadas para documentar la estructura de la solución que proponen deberá ser diferente y la herramienta debe dar capacidad para editar haciendo uso de cada una de esas notaciones,

(i) la herramienta desarrollada no puede dar la espalda a las tendencias de desarrollo actuales y deben estar dirigidas por la presencia de modelos. Dichos modelos se constituirán en diagramas asociados con la documentación de los patrones y resultarán de mayor importancia, en función de los objetivos perseguidos en esta Tesis Doctoral, aquellos que resulten independientes de la plataforma, es decir pertenecientes en el mundo inteligible.

Este conjunto de características deseables para las herramientas desarrolladas para la gestión de patrones no desmerece, en ningún caso, las herramientas destacadas en el capítulo cuarto (especialmente OlivaNova® o WebRatio®) y asociadas con la gestión de conocimiento relacionado con interacción. No hay que pasar por alto que en ambos casos se trata de herramientas comerciales y donde los principales objetivos perseguidos se centran fundamentalmente en dar soporte al proceso de desarrollo, generando en unos casos la interfaz de usuario de forma automática a partir de una especificación conceptual del problema y, en otros, facilitando el desarrollo estructurado de un sitio web deseado.

Otra consideración reseñable y distintiva de la propuesta presentada en este documento pasa por aumentar el nivel de abstracción y no decantarse prematuramente por objetos de interacción concretos que puedan influir en la elección final realizada por parte del usuario, si es que el mismo participa en procesos de evaluación realizados paralelamente con el proceso de desarrollo. Además, en esta tesis doctoral se considera que la experiencia recopilada en forma de patrones se encuentra por encima de disquisiciones posteriores relacionadas con la plataforma o el modo de interacción que deberá ser considerado posteriormente, es decir, la experiencia de diseño será independiente del dispositivo final en el que se haga uso de dicha experiencia. Se persigue, de esta forma, favorecer los atributos característicos en el desarrollo de software, desde el punto de vista de la Ingeniería del Software, como son la cohesión y el acoplamiento.

El patrón se contempla como algo inherentemente cohesivo, y tendrá valor en si mismo, estableciendo relaciones de acoplamiento con otros patrones para conseguir lograr el propósito de refinar las soluciones que cada uno de ellos aporta. Los patrones así formulados serán complejos en su documentación ya que el desarrollo automático o semiautomático de interfaces de usuario implica la consideración de aspectos relacionados con el dominio, las tareas y el contexto, obteniéndose la información asociada con el modelado de la presentación y del diálogo.

Para asegurar la pervivencia de los patrones de interacción elaborados, otra de las características deseables que deben presentar los mismos pasará por realizar su formulación en función de aspectos independientes de la

plataforma, dejando aquellas consideraciones dependientes de la plataforma para un proceso de refinamiento posterior que si que podrá se abordado, en función de transformaciones realizadas en los modelos generados, mediante los mecanismos presentados en este mismo capítulo (CSS y XSL).

5.5 Unificación de la propuesta

Las diferentes propuestas consideradas en esta Tesis Doctoral se unifican en la elaboración de una herramienta donde se conjugan las distintas trayectorias de investigación sugeridas. IDEALXML (Montero et al., 2005a; 2005b) es un metodología de documentación de la experiencia, donde se conjuga teoría y práctica. Dicha metodología está sustentada en un entorno de gestión de experiencia donde se posibilita la edición y manipulación de patrones de interacción dirigidos y basados en modelos, y asociados con el proceso de evaluación a través de un modelo de calidad. Fruto de esta unificación es posible considerar tanto el proceso de evaluación como el de desarrollo, y el nexo de unión entre uno y otro lo constituye la experiencia en forma de patrones (véase Fig. 5-10).

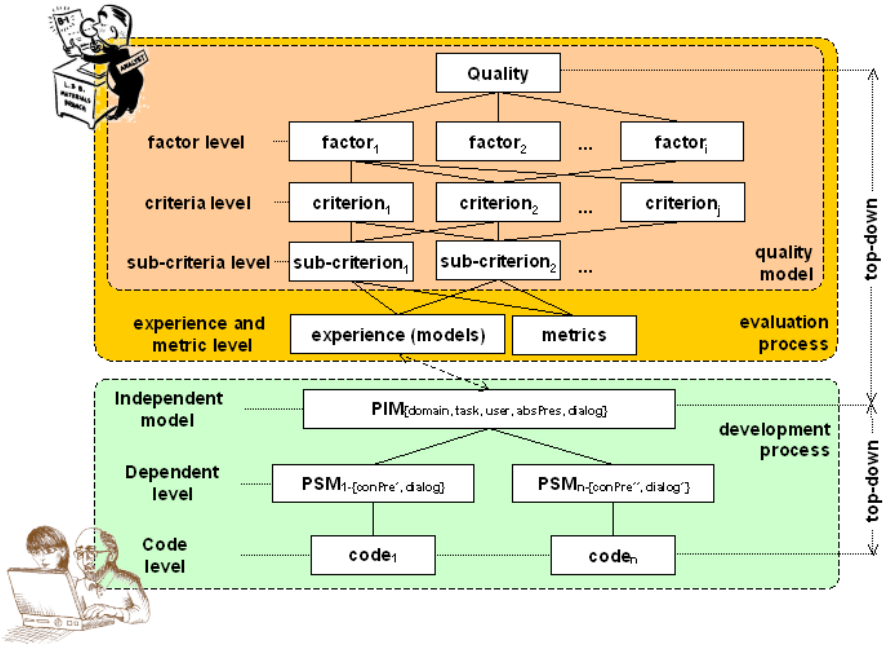


Figura 5-10. Marco de trabajo propuesto con IDEALXML (Montero et al., 2005)

En la Fig. 5-10 se muestra el enlace entre los dos procesos a los que se da soporte; evaluación y desarrollo. En la parte superior de la imagen aparece reflejado el proceso de evaluación caracterizado mediante la disponibilidad de métricas (en función de disponibilidad) y experiencia en forma de modelos. Unas y otros se encuentran ligados a la caracterización jerárquica de la calidad elaborada, en la que se contemplan ambas perspectivas: producto y proceso. Los niveles considerados en el modelo de calidad son tres: factores, criterios y subcriterios. El nivel de factores está ligado a la especificación realizada en los estándares internacionales disponibles. Los niveles de criterio y subcriterios se establecen haciéndose uso de los criterios ergonómicos comentados en apartados anteriores.

La experiencia está asociada con diferentes formas de lograr muchos de los criterios y subcriterios considerados en el modelo de calidad, y esa experiencia puede ser documentada de diversas formas. La propuesta presentada en esta tesis doctoral apuesta por el uso del concepto de patrón a la hora de documentar la experiencia, por lo menos en lo que se refiere a la experiencia inmutable, es decir a aquella experiencia que es independiente de la plataforma y del modo de interacción y que está relacionada con la especificación de cualquier problema a muy diversos niveles; tanto conceptual ligada al dominio del problema, de tareas y de presentación abstracta.

Los patrones que se utilizan en esta propuesta, además de describirse textualmente y seguir las recomendaciones que se recogen en el lenguaje PLML, acordado en el *workshop* celebrado paralelamente con la conferencia internacional CHI del año 2003, incluyen secciones relacionadas con diagramas elaborados con diferentes notaciones, fruto de la no disponibilidad de un lenguaje unificado de modelado en la disciplina IPO que unificaría dicho extremo. En estos patrones tendría cabida experiencia relacionada con el dominio, con las tareas, con el usuario, y, fruto de las relaciones que muchos de estos modelos tienen entre si, con el modelo de mapeo (*mapping*) que también estaría presente y sería, en parte, independiente de la propia plataforma.

Mediante el uso de transformaciones, realizadas a diferentes niveles, es posible obtener de los modelos independientes de la plataforma (PIM) modelos dependientes de la misma (PSM) tantos como plataformas, contextos o grupos de usuarios se deseen considerar. Un segundo proceso de elaboración de transformaciones conduciría las especificaciones realizadas en formas de modelos a código necesario para la confección de la interfaz de usuario utilizando un lenguaje de especificación de interfaces concreto.

5.5.1 Aspectos distintivos de la propuesta

Las diferencias más reseñables respecto a la forma de trabajar en el desarrollo de interfaces de usuario que se realizan en esta tesis doctoral, utilizando IDEALXML, pasan por:

(1) estar basada en una definición de interacción que se asienta en la experiencia recogida en forma de patrones,
(2) trabajar a un nivel de abstracción superior al tradicional también en lo que se refiere a la consideración de la presentación,
(3) considerar la calidad tanto en el proceso de desarrollo como en el de evaluación de un producto software,
(4) los patrones recogidos no se circunscriben a un único aspecto como puede ser el dominio o las tareas, sino que al contrario consideran todos ellos y junto con ellos la propia especificación abstracta de la presentación y las relaciones entre las distintas especificaciones consideradas,
(5) reutilizar la experiencia modelándola e integrándola utilizando patrones, y
(6) organizar dicha experiencia en función de factores de calidad.

es decir, las indicaciones anteriores se resumen en que la calidad se considera ligada a la experiencia y la experiencia es reutilizable. Frecuentemente, el tratamiento que se realiza en la actualidad de los aspectos relacionados con la interfaz de usuario se localiza a un nivel de abstracción menor del que se propone en esta tesis doctoral. En ese tratamiento, la calidad no ocupa un papel destacado y se traduce en un proceso dependiente de la posterior y tardía disposición y elección de los objetos de interacción concretos que se seleccionen durante el proceso de desarrollo. El aumento en el nivel de abstracción, junto con el de la cohesión y la disminución del acoplamiento, ha sido una constante en la evolución de los diferentes paradigmas propuestos para el desarrollo de productos software y dichas características se logran utilizando patrones de diseño en IS.

Al centrar la atención en el reto que supone el desarrollo de interfaces de usuario de calidad, la cohesión y el acoplamiento se utilizan, pero el objetivo es totalmente diferente y está relacionado con la apariencia, o disposición espacial asociado con objetos de interacción concreta. Así por ejemplo se definen métricas como las recogidas en el capítulo cuatro y elaboradas por (Constantine et al., 1999) donde se estiman valores relacionados con la concordancia de las tareas (*task concordance*), la visibilidad de las tareas (*task visibility*), la uniformidad del diseño (*layout uniformity*)

y la coherencia visual (*visual coherence*). Bajo esta perspectiva la consideración de la usabilidad debe relegarse a etapas intermedias y finales.

En las propuestas de especificación de interfaces de usuario y, en su caso, de desarrollo automático o semiautomático de las mismas, rara vez se integra la experiencia, y si se hace se consideran guías de estilo también en las etapas de desarrollo finales. En ese sentido, esa experiencia basada en guías de estilo, frente a la elaborada utilizando patrones, no es *generativa* y, aunque abundante, está formulada atendiendo a aspectos de pura presentación y de características de bajo nivel de abstracción.

El proceso de evaluación de la interfaz de usuario se basa en el desarrollo de prototipos y test con usuarios, pero la evaluación heurística es otro de los elementos que solapado con el propio proceso el diseñador utiliza, consciente o inconscientemente, cuando elabora una interfaz de usuario asociada a un producto software. En la propuesta formulada la experiencia está documentada y disponible para ser reutilizada, al modo y manera que haría un diseñador, independientemente, del nivel de conocimientos que le acompañen.

En definitiva, la propuesta está influenciada por una política propedéutica y donde se da por sentado que la calidad no sucede por accidente y debe considerarse desde el comienzo. Dicha calidad está estrechamente ligada con la experiencia y es ésta la que nos impide a través de su utilización cometer errores que de otra forma aparecerían frecuentemente.

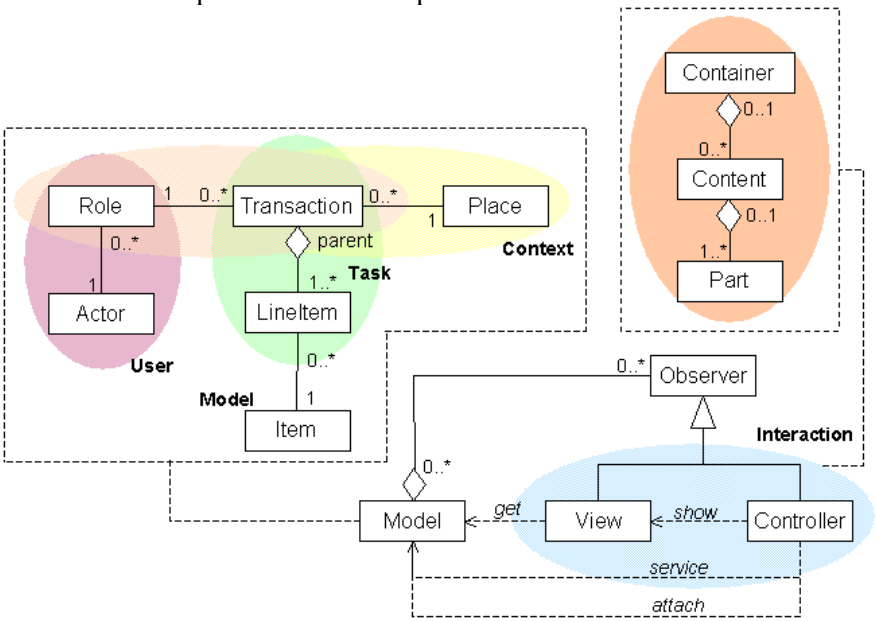


Figura 5-11. La interacción y el modelado de su alcance

5.5.2 Puesta en práctica de la propuesta

Las ideas recogidas en esta tesis doctoral se sintetizan en una única que está relacionada con la *reutilización de la experiencia*. Para ello la misma debe estar formulada de manera que pueda ser reutilizada y adaptada a problemas distintos, aunque semejantes en esencia, a aquellos en las que se evidenció con éxito y probada valía.

En la Fig. 5-11 se muestra gráficamente la definición de *interacción* en la que descansa la propuesta. La misma está formulada utilizando patrones. En ella un usuario (Actor), desempeñando un role (Role), realiza transacciones (Transaction) en un determinado contexto (Place). Las transacciones se traducen en el acceso (Controller) y modificación, en su caso, a los objetos del dominio conceptual (Model) presentados a través de una interfaz (Vista). La relación entre el modelo, la vista y el controlador se gestiona a través de un patrón de diseño como es el patrón Observer (Gamma et al., 1995). Para la especificación de esta definición se han utilizado patrones de diferentes colecciones y propósitos (arquitectónicos, conceptuales y de diseño).

En función de esta definición, y partiendo de que la experiencia está disponible y no hay que reinventarla en todo momento, se propone aprovecharla en su justa medida y aprovechando todo su potencial. La experiencia formulada utilizando patrones y disponible en varias colecciones, en lo que se refiere a interacción se citó en el capítulo cuarto, servirá como base para alimentar repositorio inicial de experiencia reutilizable para nuestra propuesta. Previamente las colecciones de patrones disponibles deben ser estudiadas detalladamente sometiéndolas al siguiente análisis:

(1) distinguir los diferentes niveles de abstracción a los que trabajan los patrones recogidos en las distintas colecciones, estableciendo aquellos patrones para los que sea posible identificar un modelo que describa la solución que les acompaña. Serán estos patrones los que más interesan a esta tesis doctoral. Hay otros patrones que se aplican sobre objetos de interacción concreta y que tienen que ver con la apariencia o presentación de dichos componentes para lograr un fin, los mismos quedan fuera del ámbito principal de esta propuesta ya que presentan una afinidad con las guías de estilo que raya la equivalencia,

(2) estudiar las soluciones aportadas por los patrones objeto de estudio y, empírica o heurísticamente, considerar su posible asociación con el modelo de calidad definido en esta misma propuesta,

(3) una vez modelada la estructura de una solución aportada por cada patrón perteneciente a las colecciones disponibles utilizando aquella o aquellas notaciones que le sean idóneas e identificadas las conse-

cuencias que su aplicación supone para la usabilidad, ya sean éstas directas o indirectas, se considerará la utilización de dichos criterios para seleccionar aquellos patrones que aporten la experiencia necesaria al diseñador en cada momento. También se utilizará, y antes posiblemente del criterio del factor de calidad, aquellos patrones que permiten dar respuesta a las tareas que el usuario requiere, o puede requerir, y que el diseñador pretenderá facilitar,

(4) los patrones se utilizarán a través del modelo o modelos que tenga asociado. Los modelos asociados a los patrones serán generales y constituirán núcleos de solución que deberán adaptarse al problema concreto y a sus características. Estas características vendrán determinadas por el contexto en cualquiera de sus dimensiones. El diseñador utilizará los patrones como si se tratase de su propia experiencia, en ella identifica semejanzas con otros problemas presentados anteriormente o identificados en otras situaciones y adapta aquellas soluciones que utilizó con éxito a los problemas que ahora se le presentan. Esta forma de trabajar guarda cierta semejanza con lo que supone una propuesta asociada con la Inteligencia Artificial: el razonamiento basado en casos (Kolodner, 1992).

(5) Finalmente, una vez se utilicen varios patrones, los modelos asociados con ellos se integrarán y mezclarán unos con otros y podrá ocurrir, como ocurre con los patrones de diseño en Ingeniería del Software, que los patrones sólo algunas veces son identificables directamente en los diagramas de clases asociados con las aplicaciones especificadas y finalmente desarrolladas.

Aunque la propuesta será desarrollada en el próximo capítulo utilizando algunos ejemplos significativos, previamente, se debe tomar partido por una colección de patrones de entre los disponibles, es decir, los reseñados en el capítulo anterior. Los patrones, obviamente, estarán relacionados con labores de interacción y a los mismos habrá que añadirles aquella dimensión generativa que hemos justificado como interesante desde el punto de vista de la reutilización.

La colección de patrones de la que se parte en esta propuesta es: Common Ground de Jenifer Tidwell (Tidwel, 1999). Tidwell recogió sesenta patrones en esta primera colección. La misma pese a no estar organizada de forma genérica, si que lo están diferentes propuestas de sublenguajes de patrones que se pueden formar haciendo uso de los patrones recogidos en dicha colección. Dicha colección aunque presenta con cada patrón ejemplos varios de su utilización y ocurrencia no hace referencia en ningún caso a elementos de interacción concretos.

Otra ventaja que presenta, a nuestro juicio, la colección de patrones de Tidwell es que pueden aplicarse a diferentes plataformas o modos de interacción, es decir, no son patrones específicos para la web o para entornos móviles como ocurre en otras propuestas (Welie, 2000; Montero et al., 2002; van Duyne et al., 2003; Graham, 2003). Al contrario, los patrones de Tidwell son generales y pueden considerarse a la hora de diseñar interfaces persona-ordenador sin ninguna otra limitación. Los patrones de Tidwell están pensados para facilitar la realización y el logro de diferentes tareas por parte de usuarios y desarrolladores. Y aunque, directamente, no estén ligados con criterio de calidad alguno, en ellos si que se consideran criterios de distribución de contenidos y acciones, anticipación de acciones, gestión de espacios, facilidades de navegación, modificación y claridad visual, en definitiva direcciones con las que influir en la calidad de las interfaces especificadas y posteriormente desarrolladas.

En las Tablas 5-6 y 5-7 se muestran los criterios de calidad, centrados en la usabilidad que constituyen el modelo de calidad propuesto, y los patrones recogidos en la colección Common Ground (Tidwell, 1999; véase el Apéndice C para tener una panorámica más extensa de esta colección de patrones). El eje de abscisas está ocupado por los criterios ergonómicos identificados y dispuestos jerárquicamente, ocupando la posición más elevada la usabilidad. Los patrones de interacción seleccionados ocupan el eje de ordenadas. Las celdas de intersección entre unos y otros elementos adquieren sombreado para indicar que utilizando un patrón se contribuye al logro de ese criterio, o viceversa, es decir, para el logro de un determinado criterio de calidad se debe actuar del modo documentado en los patrones, así como, teniendo presente aquello que se desee ofrecer desde el punto de vista del diseñador o lograr desde el punto de vista del usuario.

Tabla 5-6. Patrones de interacción y modelo de calidad (a)

	Usability																					
	Understandability						Learnability						Operability									
	compatibility	legibility	prompting	imm. feedback	sig. Codes & beh.	helpfulness	grouping	Physical workload	Minimal actions	conciseness	Information density	consistency	Explicit user action	User control	User experience's	flexibility	Error protection	Quality error msg.	Error correction	Privacy policies	Accessibility
Narrative	■									■	■											■
High-density Information Display		■						■			■											■
Status Display	■			■				■									■					■
Form	■		■	■	■	■		■	■	■			■	■		■	■	■	■	■	■	■
Control Panel	■				■	■	■		■				■	■	■							■
WYSIWYG Editor	■	■		■	■							■	■									■
Composed Command									■				■	■	■	■						■
Social Space (*unwritten*)																						
Navigable Spaces	■		■	■	■		■					■	■				■	■	■			■
Overview Beside Detail		■					■	■		■	■			■								■
Step-by-step Instructions	■		■	■	■	■	■		■		■		■	■			■	■	■			■
Small Groups of Related Things	■		■	■			■				■											■
Series of Small Multiples (*unwritten*)																						■
Hierarchical Set					■		■	■			■	■										■
Tabular Set					■		■	■			■	■										■
Chart or Graph	■	■					■				■	■										■
Optional Detail On Demand		■					■	■	■	■	■		■	■								■
Disabled Irrelevant Things	■			■			■	■	■				■									■
Point Shows Affordance	■			■									■		■		■					■
Short Description	■			■						■		■	■				■					■
Sovereign Posture						■		■	■				■	■			■		■			■
Helper Posture						■		■	■				■	■			■	■	■			■
Background Posture						■		■	■				■	■			■	■	■			■
Central Working Surface		■		■			■				■											■
Tiled Working Surfaces		■					■	■														■
Stack of Working Surfaces		■	■		■		■	■				■		■								■
Pile of Working Surfaces		■					■					■		■								■
Map of Navigable Spaces	■		■		■		■	■				■					■					■
Clear Entry Points	■		■		■			■					■	■								■
Color-Coded Sections	■			■	■	■	■	■				■										■
Go Back One Step						■							■	■					■			■

Tabla 5-7. Patrones de interacción y modelo de calidad (b)

	Usability																					
	Understandability						Learnability						Operability									
	compatibility	legibility	prompting	imm. feedback	sig. Codes & beh.	helpfulness	grouping	Physical workload	Minimal actions	conciseness	Information density	consistency	Explicit user action	User control	User experiences	flexibility	Error protection	Quality error msg.	Error correction	Privacy policies	Accessibility
Go Back to a Safe Place						■		■					■	■					■			■
Convenient Environment Actions	■		■	■								■	■				■					■
Localized Object Actions	■											■	■	■		■						■
Actions for Multiple Objects				■			■	■	■			■										■
Choice from a Small Set	■			■			■				■		■									■
Choice from a Large Set	■			■			■	■			■		■									■
Sliding Scale	■			■	■		■						■	■								■
Editable Collection	■		■	■			■					■	■	■		■	■	■				■
Forgiving Text Entry						■		■	■					■			■	■	■			■
Structured Text Entry	■			■		■		■				■					■	■				■
Toolbox	■						■		■				■	■		■						■
User Preferences	■													■	■	■						■
Personal Object Space	■													■	■							■
Scripted Action Sequence	■								■						■	■						■
User's Annotations	■													■	■						■	■
Bookmarks	■						■		■	■			■	■								■
Iconic Reference (unwritten)																						
Calm Grid (unwritten)																						
Repeated Framework				■				■				■	■	■		■						■
Few Hues, Many Values (unwritten)																						
Good Defaults	■			■		■		■				■					■					■
Remembered State						■		■				■		■					■			■
Interaction History						■		■	■			■					■					■
Progress Indicator				■				■					■	■								■
Important Message	■		■		■													■				■
Reality Check				■									■	■			■		■			■
Demonstration	■					■		■			■						■					■
Quick access (unwritten)																						
Familiar Quantity (unwritten)																						

5.6 Análisis y conclusiones

La capacidad de los patrones para documentar la experiencia es indudable y de ahí su acogida en muy diversas disciplinas. Ingeniería del Software e Interacción Persona-Ordenador no son disciplinas escépticas en este sentido, pero lo que se entiende en cada una de esas disciplinas, cuando se utiliza el concepto de patrón, es diferente.

Por otro lado, tanto experiencia como conocimiento sirven para facilitar las comunicaciones. Para ello, dicha experiencia debe ser accesible a todos

los interlocutores involucrados en la misma. Cuando, además, fruto de la comunicación entre interlocutores se identifica una labor de desarrollo o elaboración de un producto software resulta tan importante la respuesta a la pregunta *qué hacer* como a las preguntas *cómo hacerlo* o *por qué hacerlo*. En este capítulo se han considerado todas las preguntas anteriores y se han ofrecido respuestas a las mismas. Las respuestas pasan por la elaboración de estructuras y especificaciones generativas que, mediante modelos, permitan documentar la solución asociada a cada patrón con piezas de conocimiento reaprovechable como núcleos de solución.

La calidad debe también quedar reflejada a la hora de documentar el patrón. Dicha calidad se entiende centrada en la usabilidad. En Interacción la usabilidad tiene un papel estelar, pero, tradicionalmente, este papel se traduce en la realización de actividades de evaluación. En esta propuesta, la usabilidad se considera desde el comienzo y dirige el propio proceso de desarrollo.

La experiencia en interacción, a través de los patrones que persiguen documentar experiencia asociada con ese propósito, guarda relaciones posteriores para su logro y puesta en práctica con patrones de diseño propuestos desde la Ingeniería del Software, esto último se constatará en los ejemplos que se recogerán en el próximo capítulo destinado a poner en práctica la propuesta presentada.

En la Fig. 5-12 se muestra la metodología asociada con la puesta en práctica de esta tesis doctoral. En ella, se parte de una identificación de requisitos en la que diseñadores y usuarios se ponen de acuerdo realizando entrevistas, prototipos, inspecciones de documentos y de entornos de trabajo. En esta primera fase se documentan los requisitos identificados utilizando diagramas de casos de uso esenciales (Constantine et al., 1999) y la comunicación entre usuarios y diseñadores puede verse facilitada mediante la utilización de patrones. La descripción tradicional de los patrones, puramente textual y donde se enfatiza las secciones dedicadas a recoger ejemplos característicos de su puesta en práctica ayuda al usuario a decir qué quiere y al diseñador a identificar qué necesita el usuario. Además, el diseñador utilizando los patrones de interacción puede, también utilizando los patrones de interacción facilidades adicionales, como *undo*, *redo* o determinados mensajes de error, mecanismos para protegerse ante ellos o corregirlos o políticas de seguridad, etc., pueden mostrarse al usuario y percibir si serán necesarias o no en función del trabajo que quiera desarrollarse utilizando la interfaz y la aplicación proporcionada.

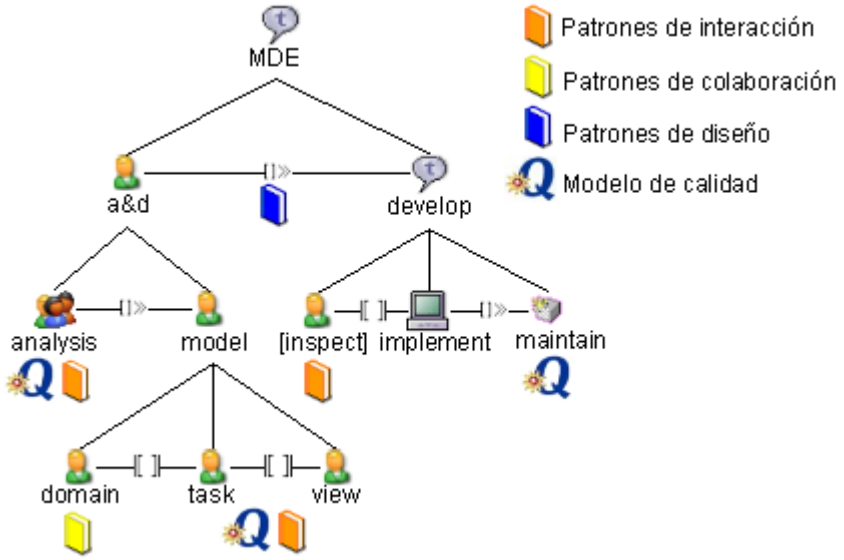


Figura 5-12. Tareas e información asociadas con el marco de trabajo presentado

Identificados los casos de uso, comienza una labor que depende del grupo de diseñadores, estos tendrán como principal labor la de elaborar una especificación en el que tendrán cabida los diferentes modelos asociados con la puesta en práctica de un entorno de desarrollo basado en modelos. Específicamente, los modelos de dominio, de tareas o de vista son indispensables en esta tesis doctoral. Las características de las que haga gala el diseñador se traducen en la necesidad de aportar experiencia, en su defecto, los diseñadores podrán recurrir a la experiencia documentada en forma de patrones. Respectivamente, y para abordar la especificación de cada uno de los modelos citados, se podrán considerar las colecciones de patrones siguientes:

- Para la elaboración de modelos de dominio desde esta tesis doctoral se recomienda la consideración de los patrones conceptuales propuestos por Coad (Coad et al., 1997), adaptados más tarde a UML en la colección de patrones de colaboración de (Nicola et al., 2001).
- Los patrones de interacción serán de utilidad en la comunicación entre diseñadores cuando se aborden labores relacionadas con el modelado de tareas y de presentación abstracta. El modelo de dominio también se verá influido por las consideraciones y requisitos que se desprenden de la especificación del modelo de tareas y de presentación. El catálogo de patrones de interacción volverá a aparecer en las fases de revisión previa a la fase de implementación. En esta última los patrones de

diseño serán los que entrarán en escena según la metodología propuesta. Aunque, como se mostraba en la Fig. 5-12 esta labor se llevaría a cabo de forma automática, será necesario, por tanto, adaptar los patrones de diseño, y todos en general, a la situación concreta en la que sea recomendable su uso.

- Un catálogo de patrones de gran éxito y aceptación en Ingeniería del Software como son los patrones de diseño (Gamma et al., 1995) serán utilizados en la fase de diseño como elemento de refinamiento para dar soporte al modelado de la solución al problema que se le presente al ingeniero.

En definitiva, los patrones se convierten en un ente generativo, es decir, en elementos de primer orden que sirven para la especificación y documentación de la experiencia. Los patrones aparecen integrados en la metodología propuesta y presentada en la Fig. 5-12, y sirven, por un lado, para facilitar la comunicación entre diseñador y usuario, y, por otro, para facilitar ese mismo proceso comunicativo que deben mantener los diseñadores entre sí cuando abordan un proyecto. Los diseñadores pueden reutilizar experiencia gracias a la disponibilidad de modelos ligados a la experiencia documentada.

Otra característica distintiva de esta propuesta está en la asociación entre el concepto de modelo y la calidad centrada en la usabilidad. La misma ha sido caracterizada con un modelo de calidad abierto y asociada con colecciones de patrones de interacción representativos, en especial con la propuesta *Common Ground* de Jenifer Tidwell (Tidwell, 1999). Los patrones, de esta forma, se presentan de forma transparente, son *cajas blancas*, a diferencia de las colecciones disponibles en las que se hace gala de una vertiente generativa, como pueden ser WebML (Ceri et al., 2000) o Just-UI (Molina et al., 2002), pero sin consideraciones de calidad consideradas directamente, ni disponibilidad explicita de los modelos ligados con cada patrón y de su posterior consideración en el proceso de elaboración.

5.7 Contribuciones relacionadas con este capítulo

En este apartado se hace recogen expresamente aquellas publicaciones con las que se ha presentado y dado a conocer la principal propuesta recogida en esta tesis doctoral. Entre ellas hay congresos y conferencias de prestigio específicas del tema objeto de estudio de ámbito tanto nacional como internacional. Fruto de la difusión realizada con dichas publicacio-

nes se ha obtenido una impagable y eternamente agradecida ayuda que ha posibilitado el desarrollo de esta tesis doctoral.

- **Montero, F.**, Lozano, M., González, P. User Interfaces Development by using Patterns. VIIP 2001: 39-43
- Sendin, M., Lores, J., **Montero, F.**, López-Jaquero, V. Using Reflection on Dynamic Adaptation of User Interfaces. 4th International Workshop on Mobile Computing. IMC Workshop. Assistance, Mobility, Applications. Rostock, Germany, 2003.
- Sendin, M., **Montero, F**., Lozano, M., Lorés, J. El prototipo interactivo del Montsec. Hacia la automatización de diversos aspectos adaptativos en el desarrollo de la interfaz de usuario. IV Congreso Internacional de Interacción Persona-Ordenador. Interacción 2003. Vigo. España. (ISBN: 84-932887-4-8)
- Sendín, M., Lorés, J., **Montero, F.**, López-Jaquero, V. Towards a Framework to Develop Plastic User Interfaces. Mobile HCI 2003: 428-433
- **Montero, F.**, López-Jaquero, V., Molina, J.P., González, P. An Approach to Develop User Interfaces with Plasticity. DSV-IS 2003: 420-423
- López-Jaquero, V., **Montero, F.**, Molina, J.P., Fernández-Caballero, A., González, P. Model-Based Design of Adaptive User Interfaces through Connectors. DSV-IS 2003: 245-257
- **Montero, F.**, López-Jaquero, V., Lozano, M., González, P. De Platón al desarrollo de Interfaces de Usuario. V Congreso Interacción Persona Ordenador. 3 - 7 de mayo de 2004 Universitat de Lleida. 2004.
- López-Jaquero, V., **Montero, F.**, Molina, J. P., González, P. A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties. The 9th IFIP Working Conference on Engineering for Human-Computer Interaction *Jointly with The 11th International Workshop on* Design, Specification and Verification of Interactive Systems (EHCI-DSVIS, 2004). Tremsbüttel Castle, Hamburg, Germany, July 11-13, 2004.
- **Montero, F.**, Lozano, M., González, P.: IDEALXML: an Experience-Based Environment for User Interface Design and pattern manipulation. Technical Report DIAB-05-01-4. Universidad de Castilla-La Mancha, Albacete (2005).
- **Montero, F.**, López-Jaquero, V., Lozano, M., González, P. A user interfaces development and abstraction mechanisms. Artículo seleccionado en el V Congreso Interacción Persona Ordenador para su publicación en Springer-Verlag, Berlin, 2005. (to appear)

Referencias bibliográficas

Bach, C., Scapin, D.L. Recommandations ergonomiques pour l'inspection d'environnements virtuels. (Rapport de contrat). Projet EUREKA-COMEDIA, INRIA Rocquencourt, France, 2003

Basili V.R. and Weiss D. A methodology for collecting valid software engineering data, *IEEE Trans. on Software Engineering*, SE-10(6), pp. 728-738, 1984

Bass, L., John, B., Kates, J. Achieving Usability Through Software Architecture. Technical Report CMU/SEI-2001-TR-005. Carnegie Mellon Software Engineering Institute

Bastien, C., Scapin, D. Ergonomic criteria for the evaluation of human-computer interfaces. Rapport technique de l'Inria. RT-0156. 1993

Bastien, C., Scapin, D. Evaluating a user interface with ergonomic criteria. Rapport de recherche de l'Inria. RR-2326. 1994

Brajnik, G. Quality Models based on Automatic Webtesting. In Proc. CHI2002 workshop. Automatically evaluating usability of Web Sites, Minneapolis (MN), April 2002

Brajnik, G. Towards valid quality models for websites. In Proc. of *7th Human Factors and the Web Conference*, Madison, Wisconsin, June 2001

Ceri, S., Fraternali, P., Bongio, A. *Web Modeling Language (WebML): a Modeling Language for Designing Web Sites*. WWW9 Conference, Amsterdam, May 2000.

Chevalier, A., Ivory, M. Web site designs: Influences of designer's expertise and design constraints. Int. J. Hum.-Comput. Stud. 58(1): 57-87 2003

Chomsky, N. Language and Mind. New York: Harcourt Brace & World, Inc., 1968. (Based on the Beckman lectures delivered at the University of California at Berkeley, January 1967. Reprint. Enlarged edition. New York: Harcourt Brace Jovanovich, 1972; reprinted as El Lenguaje y el Entendimiento. Barcelona: Planeta-Agostini, 1992.

Fenton N.E. and Lawrence Pfleeger S. *Software metrics*, 2nd ed., International Thompson Publishing Company, 1997

Hill, T. and R. Westbrook. *SWOT Analysis: It's Time for a Product Recall, Long Range Planning*, Vol.30, No.1, pp. 46-52. 1997

Johnson, G., K. Scholes and R.W. Sexty. *Exploring Strategic Management*. Prentice Hall, Scarborough, Ontario. 1989

Mahemoff, M. J. and Johnston, L. J. Principles for a Usability-Oriented Pattern Language In Calder, P. and Thomas, B. (Eds.), *OZCHI '98 Proceedings* , 132-139. Los Alamitos, CA. [In Adelaide, Australia, November 30 to December 4, 1998]. http://www.cs.mu.oz.au/~moke/papers/principles/

Molina, P., Melia, S., Pastor, O. *Just-UI: A User Interface Specification Mode.l* En Computer-Aided Design of User Interfaces III, Proceedings of the 4th International Conference on Computer- Aided Design of User Interfaces. 2002

Nielsen, J. Durability of Usability Guidelines. Jakob Nielsen's Alerbox, January, 17, 2005. http://www.useit.com/alertbox/20050117.html

Norman, D.A. The Design of Everyday Things, Basic Books, New York, 1990

Paternò, F., Model Based Design and Evaluation of Interactive Applications, Springer-Verlag, London, 1999.

OMG – Unified Modeling Language (UML). http://www.uml.org/. 1997 – 2005.

Roser, C. Platón. Libro VII de la República. Editorial Diálogo. 2001

Scapin, D. L., and Bastien, J. M. C. Ergonomic criteria for evaluating the ergonomic quality of interactive systems. Behaviour & Information Technology, 16, 220-231. 1997

Scapin, D.L., Leulier, C., Bastien, C., Vanderdonckt, J., Mariage, C., Farenc, C., Palanque, P., and Bastide, R. (2000) *A Framework for 0rganizing Web Usability Guidelines.* 6th. Conference on Human Factors and the Web. June 19, 2000

Tidwell, J. (1999). Common ground: A pattern language for human-computer interface design. http://www.mit.edu/~jtidwell/interaction_patterns.html.

Capítulo 6 IDEALXML: un entorno gráfico con el que poner en práctica la propuesta metodológica

Aquella teoría que no encuentre aplicación práctica en la vida, es una acrobacia del pensamiento.
Swami Vivekananda (Líder espiritual y reformador hindú)

Teoría es cuando se sabe todo y nada funciona; práctica, cuando todo funciona y nadie sabe por qué.
Anónimo

En el capítulo anterior se han presentado las bases teóricas y se han establecido las directrices prácticas con las que aunar experiencia y calidad en una metodología de desarrollo de productos software dirigida por modelos. Partiendo de todo ello, este capítulo tiene como principal pretensión ofrecer una visión general de cómo unas y otras consideraciones pueden ponerse en práctica. En este capítulo se utilizará IDEALXML, un entorno que facilita la gestión y manipulación de experiencia recopilada en forma de patrones y la posterior utilización de la misma en labores de especificación de productos software. Asociado a la experiencia, en esta tesis doctoral, está la calidad y, por tanto, esta otra cualidad también tendrá cabida en IDEALXML.

6.1 Hoja de ruta de la propuesta metodológica

En los últimos apartados del capítulo anterior se introducía, utilizando una notación similar a la propuesta en CTT[1] (Paternò, 1999), los pasos que, integrados dentro de una hoja de ruta, se consideran necesarios para llevar a cabo el desarrollo de productos software utilizando modelos según la propuesta integradora de teoría y práctica introducida en el capítulo anterior. Dicha hoja de ruta se vuelve a recoger ahora en la Figura 6-1 y el lector deberá tenerla presente a lo largo de todo este capítulo.

[1] El lector familiarizado con esta notación sólo apreciará unos iconos ligeramente diferentes a los utilizados en la notación CTT para hacer referencia a los diferentes tipos de tareas. Aquel lector que no conozca la notación CTT puede revisar el apartado dedicado al modelado de tareas en el capítulo segundo de este mismo documento.

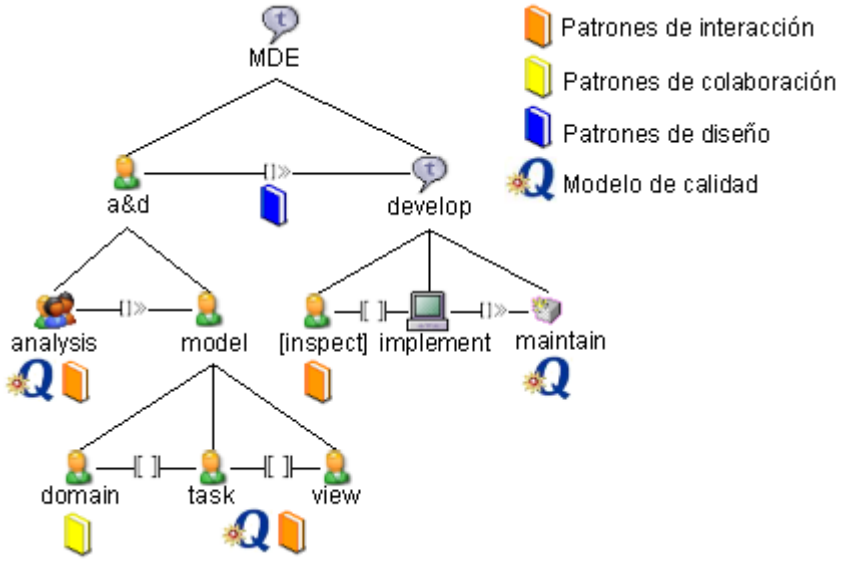


Figura 6-1. Tareas y puesta en práctica de la metodología presentada

En la hoja de ruta mostrada en la Fig. 6-1 aparecen recogidas las fases habituales de análisis, diseño, implementación y mantenimiento asociadas con el desarrollo de productos software. La calidad y la experiencia vienen consideradas explícitamente en todas ellas, y lo hacen en los distintos libros asociados con las múltiples especificaciones necesarias al utilizarse el Entorno de Desarrollo dirigido por Modelos (MDE) propuesto. Los libros que aparecen asociados a las tareas realizadas en las etapas mostradas en la Fig. 6-1 simbolizan experiencia, en diversos ámbitos (dominio, tareas y presentación), que se encuentra disponible en forma de patrones. Dicha experiencia puede tenerla el propio ingeniero o puede encontrarse disponible en muy diversas fuentes que serán referidas a lo largo del capítulo. En cualquier caso, dicha experiencia verá facilitada su utilización mediante el uso del entorno IDEALXML, donde una y otra –la experiencia propia y la prestada- podrá ser documentada y utilizada. Por otro lado, la calidad de un producto software está relacionada con la disponibilidad de experiencia cuando fue desarrollado, como se ha visto en el capítulo anterior, especialmente con la disponibilidad de aquella experiencia relacionada con la interacción, aunque esa no sea la única experiencia disponible y útil a la hora de abordar la elaboración de un producto software.

En este capítulo se hará un recorrido por cada una de las tareas y relaciones identificadas entre ellas y recogidas en la hoja de ruta mostrada en la Fig. 6-1. La primera fase, la de análisis de requisitos, se abordará utili-

zando los diagramas de casos de uso, propios de Ingeniería del Software, y a los que se someterá a un primer refinamiento En este apartado, cabe destacar la utilización de la notación CTT para enriquecer las descripciones realizadas con los diagramas de casos de uso y abordar labores de análisis de tareas. En la fase de modelado se recurrirá al uso de diferentes notaciones, la mayoría de ellas provendrán de la IS. De la fase de análisis y diseño se saldrá con una especificación conceptual lo más completa posible, que permite una posterior implementación automática o semi-automática dirigida por modelos.

6.2 Punto de partida: el análisis de requisitos

La fase inicial de análisis de requisitos (Fig. 6-2) está soportada por la utilización y elaboración de diagramas de casos de uso (Cockburn, 1997; 2004). Un hecho distintivo de esta tesis doctoral es la consideración e identificación de características de calidad, si los diagramas de casos de uso presentan limitaciones en la consideración de la misma habrá que complementarlos de la forma más adecuada posible.

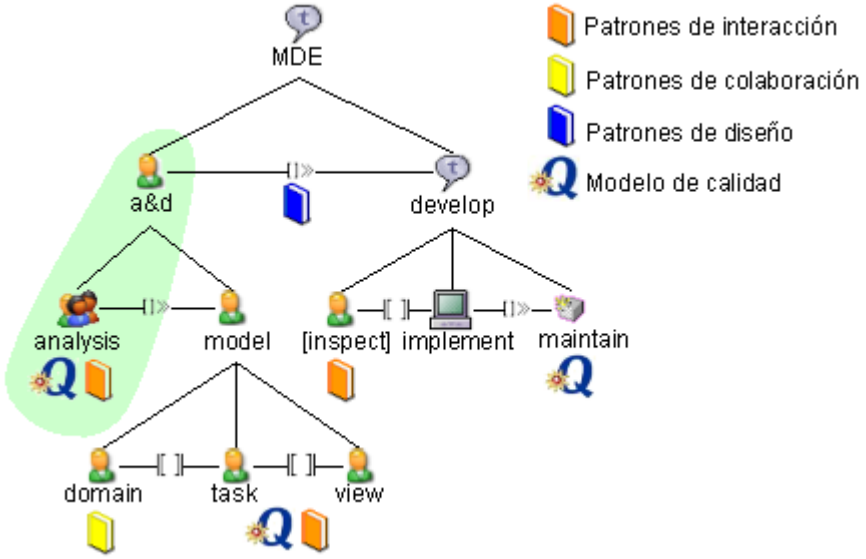


Figura 6-2. Fase de análisis

Los diagramas de casos de uso están constituidos por *casos de uso* y por *actores*. Un caso de uso es lo que sucede cuando diferentes actores interactúan con un sistema y un actor es una entidad externa (una persona u otro sistema) que usa un sistema para lograr un objetivo deseado.

Mediante la recopilación de información, con la utilización de diagramas de casos de uso y de técnicas de observación, realización de entrevistas o llevando a cabo trabajos de campo con usuarios, tal y como se describía en el capítulo tercero, es posible identificar todos los requisitos y objetivos funcionales que un sistema debe satisfacer. En este sentido, un caso de uso es una colección de posibles secuencias de interacción entre el sistema objeto de discusión y sus usuarios o actores, considerando un objetivo particular. La colección de todos los casos de uso debiera recoger el comportamiento relevante que debe ofrecer el sistema y no debiera considerar aquel comportamiento considerado irrelevante.

En definitiva, las características principales que presentan los casos de uso pasarían por:

- Recoger los requisitos funcionales de forma que sean fáciles de leer y entender.
- Representar el objetivo de una interacción entre un actor y un sistema, donde el objetivo se supone significativo y mensurable para el actor.
- Almacenar el conjunto de escenarios que atraviesa un actor desde que dispara un evento hasta que alcanza su objetivo.
- Almacenar el conjunto de escenarios que atraviesa un actor desde que dispara un evento tratando de alcanzar un objetivo y falla en su intento.
- Tener presente que un caso de uso puede utilizar o extender la funcionalidad de otro.

De este modo como se ha mencionado, los casos de uso se utilizan en las primeras fases del desarrollo para capturar los requisitos funcionales del sistema. Fruto de esa captura, posteriormente, es posible validar los resultados de otras fases frente a los requisitos identificados. Los casos de uso se describen gráfica y textualmente utilizando diferentes plantillas en las que quedan recogidas diferentes secciones, valga como ejemplo de las mencionadas plantillas la recogida en la Tabla 6-1.

Pero, habitualmente, junto a la documentación, gráfica y/o textual, de los casos de uso se acompañan *escenarios*. Un escenario no es sino una instancia de un caso de uso que representa una traza concreta de ese caso de uso. El principal inconveniente o limitación que presenta esta labor de refinamiento de los casos de uso es que con ella se pierde el grado de abstracción logrado con los casos de uso. Los casos de uso, y los casos de uso esenciales (Constantine et al., 2000) son ejemplo de ello, son independientes de la plataforma, es decir, en ellos no aparece referencia alguna al modo de interacción o plataforma concreta con el que el actor interactúa. En cualquier caso, y asumida la premisa de recurrir al uso de los escenarios, se

hace patente la necesidad de abundar en la información recogida utilizando los diagramas de casos de uso ya que es considerable el *gap* existente entre las fases de análisis y de diseño.

Tabla 6-1. Plantilla de documentación de un caso de uso

Sección	Descripción
Nombre	Nombre del caso de uso. Debería estar relacionado con el resultado, propósito o evento del caso de uso.
Propósito	El principal objetivo del caso de uso y aquel que los participantes esperarán obtener.
Descripción	Un párrafo que describa tanto los objetivos como escenarios que ilustren el caso de uso. Esta sección debería incluir información sobre el tipo de transacción y el proceso de negocio, el contexto, la fase y aquella información relevante ligada al contexto de aplicación del caso de uso.
Actores	A quién involucra el caso de uso (individuos, organizaciones, funciones, aplicaciones, máquinas, etc.)
Dominio asociado	Datos manipulados en este caso de uso, es decir, qué información se intercambia en las transacciones ligadas al caso de uso.
Tareas involucradas	A qué manipulaciones se somete a los datos involucrados en las transacciones.
Tareas de aplicación	Qué aplicaciones aparecen en escena manipulando, recibiendo o facilitando datos para el correspondiente caso de uso.
Objetivos perseguidos	Qué objetivos relevantes se requieren en este caso de uso. En este punto puede condicionar cada actor y aplicación definida.
Precondiciones	Qué condiciones deben darse antes de comenzar con el caso de uso.
Comienzo	Qué pone en marcha el caso de uso.
Finalización	Cuándo finaliza el caso de uso.
Excepciones	Qué situaciones excepcionales pueden acontecer durante el desarrollo de un caso de uso.
Post-condiciones	Cuál es el estado del sistema una vez el caso de uso se ha completado.
Referencias	Qué otros casos de uso están relacionados con el actual.
Otras consideraciones	Cualquier otro detalle digno de tener en cuenta en la documentación de un caso de uso.

6.2.1 Limitaciones

La definición y ventajas que ofrecen los casos de uso se han recogido en el apartado anterior. En este se hará referencia a aquellas limitaciones que identificamos en los diagramas de casos de uso y que justifican un refinamiento de la información que en ellos se documenta. Previamente a la identificación de las limitaciones que presentan los casos de uso debe tenerse presente que:

- Los casos de uso no especifican características de diseño de la interfaz de usuario. En los casos de uso se documentan motivaciones y no detalles.
- Los casos de uso tampoco documentan detalles de implementación, a no ser que éstos sean de singular importancia para el logro del objetivo perseguido.

Así las limitaciones que hemos identificados y por las que sugerimos seguir abundando en la documentación y enriquecimiento de la información recopilada con los casos de uso son:

- Siendo excelentes para la identificación de requisitos funcionales de un sistema no son apropiados para capturar requisitos no funcionales, ni aquellos que hemos denominado requisitos no explícitamente funcionales y que estarían directamente ligados con la calidad que percibe el usuario cuando interactúa con un producto software.
- Los diagramas de casos de uso son una técnica cuya documentación y elaboración (ya sea gráfica o textual) depende notablemente de la experiencia que presente el ingeniero o equipo de desarrollo.
- Una vez elaborados los diagramas o cumplimentadas las plantillas relacionadas con los casos de uso identificados, dicha información no es de aplicación inmediata sino que deben, a su vez, demostrarse habilidades y conocimientos para su posterior paso y transformación a notaciones propias de fases posteriores.
- Los casos de uso se han identificado como técnicas centradas en el documento, de esta forma hay propuestas que prefieren trabajar con documentaciones más ligeras y simples.
- En muchos casos, y aún no siendo aconsejable la consideración tan temprana de modos y maneras concretas de interacción, en la elaboración de diagramas de casos de uso y en su documentación se incluyen referencias a tales modos de interacción lo cual es perjudicial en fases tan tempranas. No hay que caer en la tentación de confundir un caso de uso y un escenario. Un caso de uso es un escenario genérico, y el objetivo con ellos es representar la esencia de lo que el usuario quiere conseguir, un objetivo completo y significativo para el usuario.

En este sentido, desde esta tesis doctoral se defiende y recomienda la utilización de otras notaciones, concretamente de la notación CTT (Paternò, 1999), para continuar con la identificación, especificación y análisis de las tareas identificadas como objetivos que el usuario desea llevar a cabo utilizando un producto software. El uso de la notación CTT, provenien-

te de IPO, permite enriquecer la información documentada utilizando los casos de uso en dos direcciones, por un lado abundando en labores de identificación de tareas y de actividades adicionales y, por otro, distinguiendo quién o qué lleva a cabo cada una de esas tareas y actividades. Esas dos labores se realizan garantizando el nivel de abstracción al que se puede trabajar haciéndose uso de los casos de uso y, por ello, permitiendo una especificación y un modelado posterior independiente de la plataforma y del modo de interacción. A este enriquecimiento en la descripción de los objetivos y casos de uso de un producto software se volverá más adelante en el apartado destinado al modelado de tareas, donde se incluirá información de un primer refinamiento de los casos de uso llevado a cabo con el uso de la notación CTT.

6.2.2 Resultados de esta primera etapa

En función de las ventajas y limitaciones comentadas con anterioridad es indudable que la elaboración de diagramas de casos de uso efectivos no resulta una labor sencilla y que la experiencia del ingeniero en la identificación de los casos de uso y en su documentación, es determinante para el éxito de dichas labores. Además de la experiencia, otro elemento resulta objetivo primordial en esta tesis doctoral, la consideración de la calidad. La calidad que percibe el usuario tiene poca cabida en la plantilla asociada a la documentación de los casos de uso que anteriormente se recogía (Tabla 6-1). Solamente una sección, aquella dedicada a los *objetivos perseguidos*, puede dar cabida a la consideración explícita de la calidad y de sus factores y criterios asociados. Es en este momento en el que la experiencia documentada y disponible en forma de patrones de interacción puede hacer acto de presencia actuando de forma positiva en la identificación de requisitos tanto funcionales no explícitos como en características de calidad asociadas a esos requisitos mencionados. Es decir, las características de calidad contempladas en un modelo de calidad ofrecidas en el capítulo tercero de esta tesis doctoral y que más tarde, en el capítulo quinto, se ligaron con la experiencia, pueden utilizarse en este momento con tres propósitos:

- Primero, para la identificación de requisitos tanto funcionales como no explícitamente funcionales aunque deseables por el usuario. Para ello, en primera instancia, se utilizarán aquellas capturas o imágenes que muestran la puesta en práctica de cada patrón de interacción disponible y, con ellas, usuarios e ingenieros podrán identificar soluciones similares (al menos en apariencia) a las que se desean lograr. Se estaría de esta manera capturando y validando los requisitos funcionales.

- Segundo, indirectamente y fruto de la relación identificada y que mantienen los patrones de interacción con la calidad (asentada en la disponibilidad de un modelo de calidad), puede precisarse, con mayor conocimiento de causa, la sección relacionada en la documentación de un caso de uso con los objetivos perseguidos. En ella se incluirán tanto los objetivos funcionales como los objetivos de calidad. Se estaría caracterizando de una manera más precisa la caracterización habitualmente considerada en las propuestas metodológicas relacionadas con el Diseño Centrado en el Usuario y la Ingeniería de la Usabilidad. También, de esta manera, se identifica y elabora un modelo de calidad deseado por el usuario y que estaría compuesto por criterios de calidad seleccionados de entre un conjunto definido y que además se encuentran ligados a la experiencia y al *know-how* que determina su logro.
- En tercer y último lugar, el modelo de calidad elaborado será utilizado en fases posteriores como un documento activo sobre el que contrastar los criterios de calidad identificados como necesarios. Además, una vez elaborado el producto software, si los usuarios identifican taras en cuanto a la calidad ofrecida por el producto software se podrán proponer mejoras relacionadas con el fin de superar las taras identificadas gracias a la disponibilidad de experiencia que puede ser utilizada y que está relacionada con mejoras que tienen incidencia sobre la calidad.

En función de los puntos anteriores, y utilizando la experiencia disponible y el modelo de calidad propuesto, se estaría considerando conjuntamente la experiencia y la calidad, facilitándose la captura, en esta primera fase, tanto de requisitos funcionales (habituales en la elaboración de diagramas de casos de uso) como de requisitos no explícitamente funcionales y relacionados con el modelo de calidad deseado por el usuario. Con la propuesta metodológica presentada se ofrece, por tanto, la posibilidad, en base a la experiencia disponible, de identificar qué requisitos de calidad, de entre un conjunto dado a través de un modelo de calidad, resultan significativos para el usuario.

Los propósitos mencionados pueden verse facilitados por el entorno IDEALXML que se ha desarrollado. Con él es posible documentar patrones, de entre ellos, en esta fase, interesarán especialmente los patrones de interacción que el usuario puede revisar con la compañía del ingeniero e identificar aquellos patrones que puedan resultar afines a las labores que el usuario pretende llevar a cabo con el producto software. El ingeniero, en función de los patrones identificados por el usuario, podrá recomendar la visualización de otros patrones con los que estos estén relacionados y podrá construir una propuesta de modelo de calidad que, a su vez, podrá ser validada frente al diseño e implementación posterior del producto softwa-

re. La Fig. 6-3 muestra la ventana principal de IDEALXML, donde ingenieros y usuarios pueden aprender, gestionar, manipular y utilizar diferentes tipos de patrones, en estas fases iniciales los patrones utilizados son los de interacción.

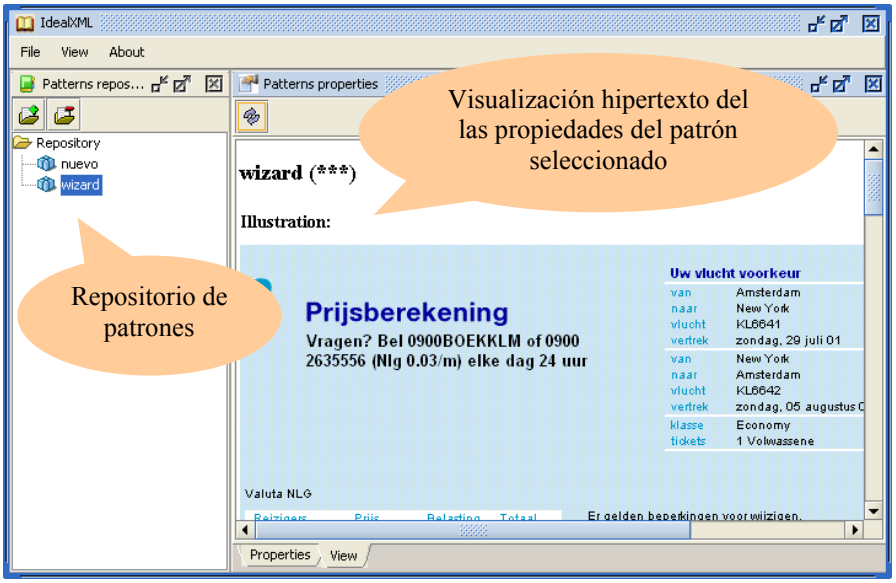


Figura 6-3. Visualización de las propiedades de un patrón en IDEALXML

En IDEALXML, para cada patrón se almacenan diferentes propiedades con el fin de describirlo y documentarlo. Las mismas responden al esquema XML presentado en el capítulo quinto que estaba, a su vez, inspirado en la propuesta PLML (Fincher et al., 2003), y respecto al cual se ha potenciado especialmente la vertiente generativa que puede ofrecer un patrón y de asociación con un modelo que puede presentar el mismo patrón. En este sentido, un elemento reseñable de dicha propuesta es la sección *diagram*, en la que, utilizando el lenguaje usiXML (Limbourg et al., 2004), se pueden almacenar diferentes modelos de un patrón que facilitan su puesta en práctica en la fase de análisis y de diseño. No hay que olvidar que la propuesta presentada se enmarca dentro de un entorno de desarrollo dirigido por modelos, entorno que pasamos seguidamente a tratar más en detalle.

6.3 El entorno de desarrollo dirigido por modelos

Nos encontramos en una fase intermedia del proceso de desarrollo, concretamente en la fase de diseño y modelado. En ella las tareas a las que ahora se presta especial consideración son las que aparecen resaltadas en la ya habitual hoja de ruta mostrada en la Fig. 6-4.

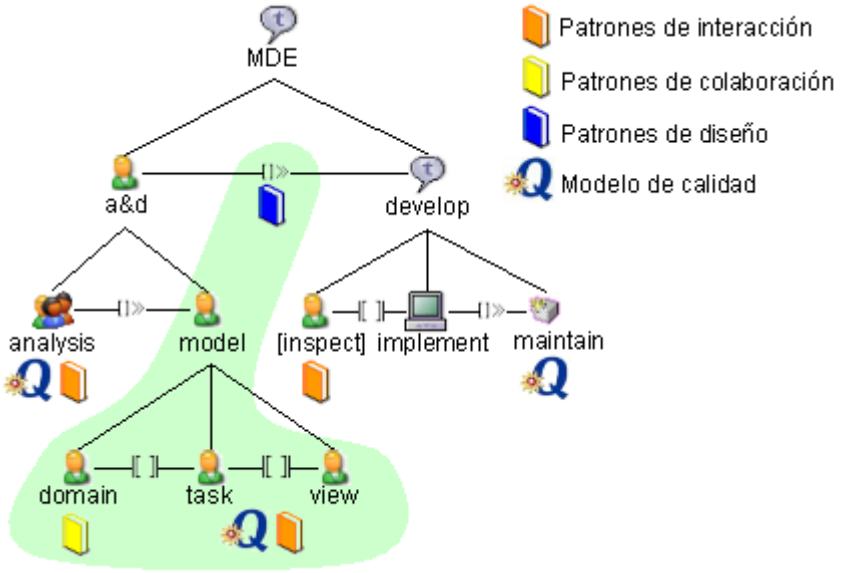


Figura 6-4. Tareas que involucra la fase de diseño

Partiendo de que un *patrón es un modelo* (Bezivin, 2005) durante la realización de esta tesis doctoral se han revisado las colecciones de patrones disponibles y se han identificado aquellas que pueden ser más útiles para la puesta en práctica de la propuesta asociada a esta tesis doctoral. En esta tesis se considera especialmente la calidad y se hace desde el punto de la calidad que percibe el usuario al hacer uso de un producto software. Dicha calidad no se entiende únicamente determinada por el aspecto visual que ofrece la interfaz, más bien ese mero aspecto visual se considera superfluo y secundario. La máxima de que la calidad hay que lograrla y que para ello debe ser considerada es la se ha tenido presente en la elaboración de esta tesis. Para lograr calidad hay que buscarla, identificarla y modelarla. La calidad no es fruto de la improvisación o de la fortuna. En la fase de análisis ayudándonos de los patrones de interacción y junto con la participación del usuario se han identificado necesidades funcionales y facilidades no explícitamente funcionales pero que si que repercuten en el uso que del producto software hace el usuario. Dichas facilidades tendrán que ser ahora diseñadas y modeladas para que aparezcan finalmente en el producto final que

se quiere elaborar. Además, dichas facilidades aparecen de forma reiterada en distintos productos software de distinta aplicación y el usuario puede conocer de su existencia fruto de tenerlas presentes en otros productos software de los que hace uso.

En la fase, que ahora se describe, se realizan fundamentalmente tres actividades de modelado relacionadas con el dominio, las tareas y la especificación de la presentación a nivel abstracto de la interfaz. Parte de las especificaciones pueden llevarse a cabo de forma semiautomática gracias a la utilización del repositorio de patrones de diferentes tipos que se encuentran analizados y a disposición del ingeniero. La especificación de los modelos de dominio, tareas y presentación no se realiza de forma aislada e independiente, al contrario, estos modelos están relacionados y de dicha relación se obtienen facilidades para la generación automática que pueden explotarse llegado el momento, gracias a la identificación de mappings entre los distintos modelos y de la entrada en escena de las correspondientes transformaciones de modelos, habituales en un entorno de desarrollo basado en modelos. Algunas de dichas transformaciones están disponibles en usiXML (Limbourg et al., 2004; Montero et al., 2005).

En la zona resaltada correspondiente a esta fase de diseño mostrada en la Fig. 6-4 se ha incluido también la relación de activación con paso de información (representada con el símbolo []>>) que marca el punto de transición entre la fase de modelado y la fase de diseño. Se comentará en un apartado posterior qué información se produce en esta fase de modelado y con qué información se llega, por tanto, a la fase de implementación que hace posible la generación automática o semiautomática del producto software deseado por el usuario.

La presentación de las actividades realizadas en cada uno de los modelos abordados en esta fase de diseño se compaginará con las facilidades que el ingeniero puede encontrar en el entorno IDEALXML. Dicho entorno ofrecerá experiencia disponible en cada uno de los ámbitos (dominio, tareas y presentación) que el ingeniero debe modelar. Las notaciones utilizadas para modelar la experiencia asociada a cada patrón serán gráficas y se traducirán en el uso de diagramas de clases para el modelado de dominio, análisis CTT para la especificación del modelo de tareas (derivadas de los casos de uso y elaboradas en la fase de análisis de requisitos) y una notación gráfica asociada a la especificación de la presentación abstracta definida en usiXML para realizar el modelado abstracto de la presentación de la interfaz de usuario. Como ya se ha mencionado, todas las descripciones citadas pueden almacenarse utilizando el lenguaje basado en XML usiXML (Limbourg et al., 2004).

6.3.1 Modelo de dominio

La experiencia más reseñable que se ha encontrado y con la que es posible abordar el modelado del dominio es la disponible, inicialmente, en (Coad et al., 1997) y posteriormente reelaborada en (Nicola et al., 2001). Concretamente, en la Fig. 6-5 aparecen recogidos los patrones de ámbito dominio disponibles en IDEALXML y con los que es posible abordar labores de especificación y modelado relacionadas con el modelo de dominio.

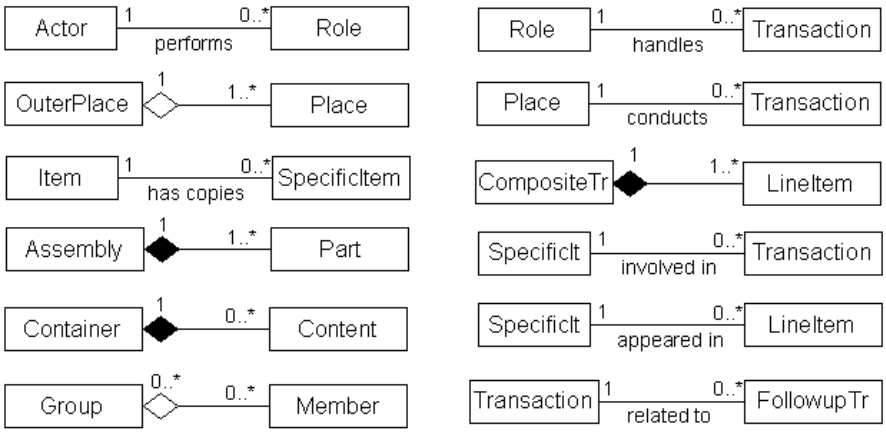


Figura 6-5. Experiencia útil para el modelado del dominio (Nicola et al., 2001)

Los patrones recogidos en la Fig. 6-5 son los denominados patrones de colaboración (Nicola et al., 2001; Apéndice A) y en ellos, mediante el uso de diferentes tipos de relaciones de asociación, se identifican relaciones entre entidades habituales que surgen a la hora de especificar actividades que involucran la realización de transacciones. Este tipo de operaciones, aquellas basadas en la realización de transacciones, pueden extrapolarse a muchos y muy diversos ámbitos ya que prácticamente cualquier operación que el usuario realiza utilizando un producto software puede considerarse como tal. De esta forma, consideramos que el usuario realiza una transacción cuando, por ejemplo en un sistema hipertexto[1], pulsa sobre un enlace que le lleva a información adicional sobre un tema que aparece reflejado en el texto que está visualizando.

[1] Un sistema **hipertexto** es aquel que contiene documentos hipertexto, es decir, documentos digitales que se pueden leer de manera no secuencial. Un documento hipertexto tiene los siguientes elementos: secciones, enlaces y anclajes. Las secciones o nodos son los componentes del hipertexto o hiperdocumento. Los enlaces son las uniones entre nodos que facilitan la lectura secuencial o no secuencial del documento. Los anclajes son los puntos de unión entre nodos.

Con los doce patrones de colaboración recogidos en (Nicola et al., 2001), que están disponibles en IDEALXML, es posible la elaboración de modelos de dominio. En estos patrones están identificadas relaciones habituales entre cuatro tipos de elementos: *personas*, *lugares*, *cosas* y *eventos*. Las clases que aparecen recogidas en los patrones de colaboración son superclases que serán especializadas a la hora de producir un modelo de dominio. Así, por ejemplo, un actor puede ser un cliente, un proveedor, un alumno, etc., y de igual forma una transacción puede ser cualquier operación de interacción realizada entre una persona y un sistema.

Los patrones de colaboración pueden utilizarse directamente o fruto de un proceso de refactorización (Fowler et al., 1999). En la Fig. 6-6 se muestra, como ejemplo, la especificación de un modelo de dominio elaborado utilizando los patrones de colaboración de (Nicola et al., 2001). Todas las clases y relaciones entre ellas responden a patrones de colaboración, como ejemplo en la figura se han resaltado algunas de ellas. De esta manera, la relación entre la clase *User* y *UserRole* responde al patrón Actor-Role, la relación entre la clase *UserRole* y *Order* responde, de forma similar, al patrón *Role-Transaction* o la relación entre *OrderItem* y *Product* está relacionada con el patrón *Item-SpecificItem*. Las anteriores asociaciones son sólo algunos ejemplos, ya que el resto de relaciones mostradas en la Fig. 6-6 también tienen su patrón de orden superior asociado.

En IDEALXML es posible gestionar una colección de patrones, independientemente de su ámbito de aplicación, y también es posible utilizar los patrones almacenados para llevar a cabo la especificación de nuevos productos software siguiendo un desarrollo dirigido por modelos y basado en la experiencia recopilada y disponible. Tomándose como ejemplo los patrones de colaboración mencionados, seguidamente mostraremos cómo es posible añadir cualquiera de estos patrones al entorno IDEALXML y posteriormente utilizar los modelos que lleva asociados sin tener que realizar otra operación que la de seleccionar el patrón deseado y arrastrarlo al correspondiente editor asociado al ámbito en el que el patrón documenta experiencia (dominio, tareas o presentación). El usuario de IDEALXML que quiera introducir un nuevo patrón no tiene más que pulsar sobre el botón  y proporcionar un nombre significativo al nuevo patrón que desee documentar, por ejemplo, si se supone que deseamos almacenar el patrón de colaboración que dicta la relación entre un actor y el role que desempeña en un sistema podemos denominar a este patrón *actorRole* (Fig. 6-7).

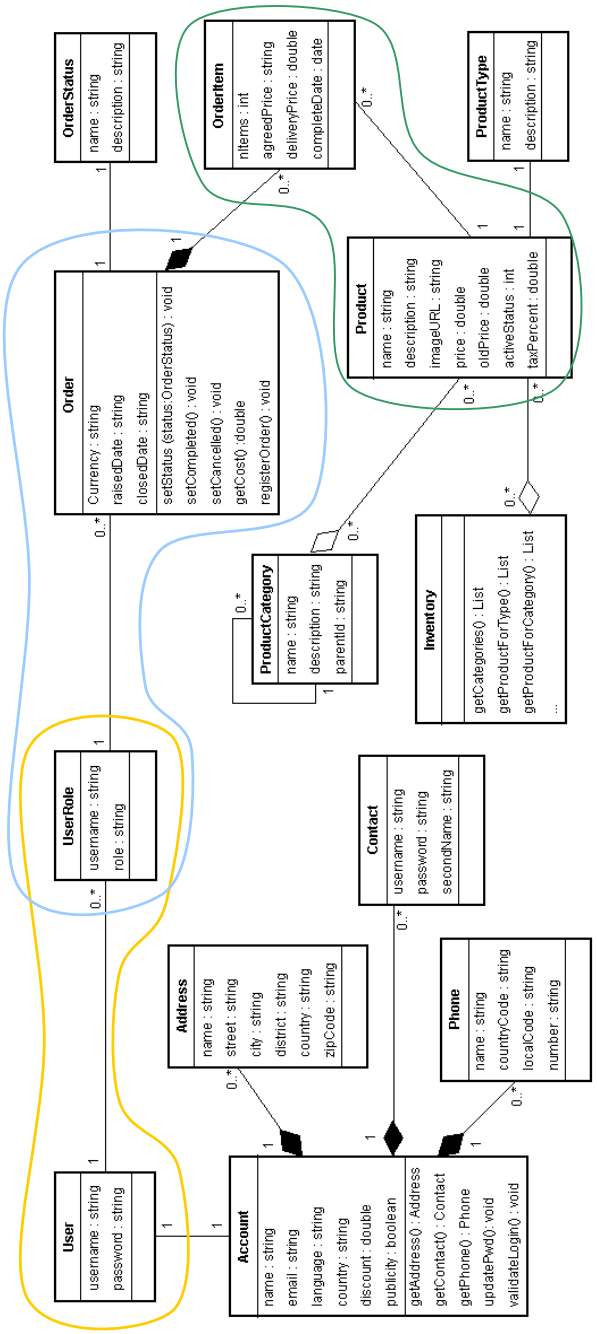


Figura 6-6. Ejemplo de modelo de dominio elaborado utilizando patrones de colaboración

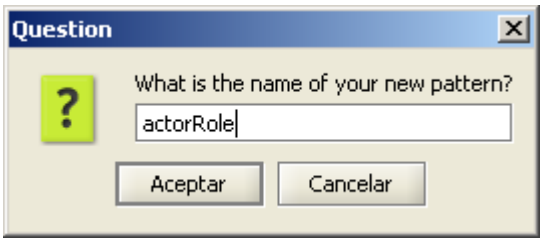


Figura 6-7. Añadiendo un nuevo patrón en IDEALXML

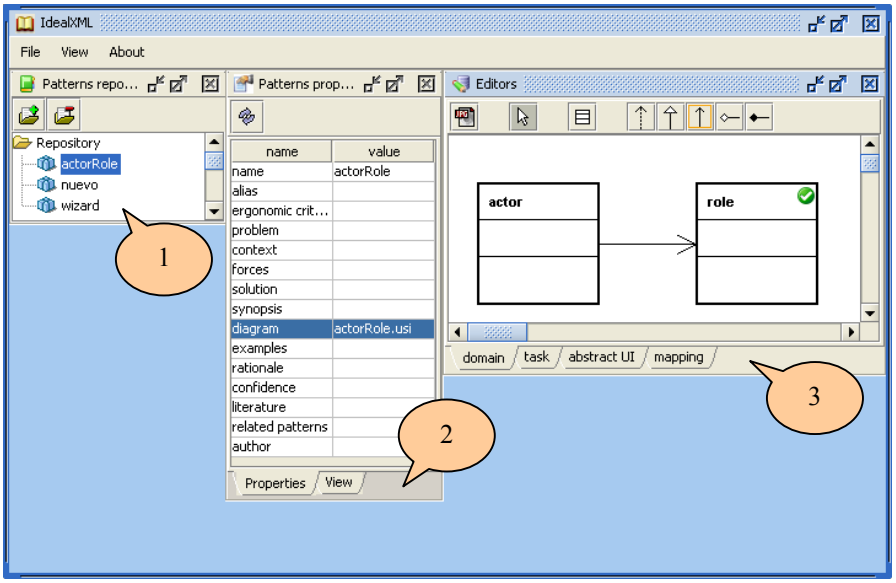


Figura 6-8. Ventanas de especificación de un patrón en IDEALXML

Después de asignar un nombre al nuevo patrón que se desea considerar en el repositorio de patrones disponible (paso 1 en la Fig. 6-8), será labor del ingeniero documentar adecuadamente el nuevo patrón introducido. Para ello será necesario cumplimentar cada una de las propiedades asociadas al mismo (paso 2 en la Fig. 6-8). Entre esas propiedades (descritas en el capítulo quinto de este documento) que documentan un patrón, dos de ellas merecen un tratamiento especial en esta tesis doctoral. Una de esas propiedades está destinada a almacenar un diagrama o modelo asociado al patrón documentado. En el caso del ejemplo que se está manejando el patrón documenta la relación que se establece entre un *actor* y el *role* que desempeña. Dicho patrón puede modelarse utilizando un diagrama de clases en el que se consideren dos clases, una para cada una de las entidades involucradas, es decir, una clase *actor* destinada a almacenar la información representativa y el comportamiento del que haga gala el actor y otra clase

destinada al mismo fin para la entidad *role* (paso identificado con 3 en la Fig. 6-8). Para elaborar el diagrama de clases asociado con un modelo de dominio el entorno IDEALXML ofrece un editor de diagramas de clases donde a través de los botones etiquetados con los iconos mostrados en la Tabla 6-2 es posible describir gráficamente cualquier modelo de dominio.

Tabla 6-2. Iconografía para la elaboración de diagramas de clases en IDEALXML

Icono	Descripción
	Iconos asociados a los botones disponibles en el editor de diagramas de clases que acompaña a IDEALXML y con los que es posible definir nuevas clases, añadir atributos y añadir métodos
	Iconos disponibles para el establecimiento de diferentes tipos de relaciones entre las clases definidas en el diagrama de clases (dependencia, generalización, asociación, agregación y composición respectivamente)

El modelo resultante, descrito mediante un diagrama de clases, es modificable y adaptable al problema concreto que deba ser modelado utilizando IDEALXML. El almacenamiento del diagrama de clases representado se realiza utilizando el lenguaje usiXML (Limbourg et al., 2004).

6.3.2 Análisis y modelado de tareas

Para la especificación del modelo de tareas, y directamente relacionado con los diagramas de casos de uso elaborados en la fase de análisis de requisitos, en esta tesis doctoral se utiliza la notación propuesta en (Paternò, 1999), dicha notación, la CTT, puede utilizarse mediante el editor CTTE, ya mencionado en el capítulo segundo. Aparte, IDEALXML incorpora su propio editor CTT en el que, al igual que en el original, se dispone de los diferentes tipos de tareas y de las relaciones temporales identificadas en la propuesta original de Paternò. Las diferencias entre el CTTE e IDEALXML, en lo que al uso de la notación CTT se refiere, están en el aspecto visual que ofrecen los iconos asociados a las tareas (véase Tabla 6-3) y en la posibilidad, más digna de mención, de almacenar la especificación realizada utilizando el lenguaje usiXML.

Sin embargo, la notación CTT no dispone de un tutorial o similar en el que se indique qué *modus operandi* debe seguir el ingeniero para realizar la especificación de las tareas identificadas en la fase de análisis de requisitos, y que facilitan la consecución de los objetivos documentados en los diagramas de casos de uso elaborados en esa fase. Tampoco en la notación CTT hay una consideración explícita de la calidad a la hora de abordar la especificación. En este sentido, hemos observado que muchos de los pa-

trones de interacción no son, en muchos casos, sino funcionalidad adicional no explícitamente definida, pero sí deseable por el usuario. Dicha funcionalidad adicional puede conocerla el usuario fruto de estar disponible en otros productos software que utiliza habitual o esporádicamente.

Tabla 6-3. Iconografía para la elaboración de modelos de tareas en IDEALXML

Icono	Descripción
	Iconos asociados a los botones del editor de modelos de tareas en IDEALXML con los que es posible definir diferentes tipos de tareas en función de su carácter (abstracta, aplicación, interacción, cooperación y usuario)
||| |[]| >> [ ]>> [ ] [> |> |=|	Iconos asociados con los diferentes tipos de tareas binarias que pueden definirse utilizando el editor de modelos de tareas en IDEALXML (concurrencia, sincronización, activación, activación con paso de información, elección, desactivación, suspensión/reinicio, independencia de orden)
[T] T*	Iconos asociados con los diferentes tipos de tareas de carácter unario que pueden definirse utilizando el editor de modelos de tareas que acompaña a IDEALXML

En función de estas dos limitaciones reseñadas, en esta tesis doctoral se propone un *patrón fundamental*, el recogido en la Fig. 6-9. Con él sería posible especificar las actividades necesarias para abordar la realización de un objetivo identificado como útil para el usuario o de parte de dicho objetivo. Con dicho patrón fundamental, y de forma general, se abordaría la realización de la tarea identificada como *task* en la propia Fig. 6-9. Para la realización de dicha tarea (*task*), y previamente, se habría mostrado al usuario aquella interfaz que le permite su realización (*showStatus*). En dicha interfaz se le informaría de forma constante del estado en el que se encuentra la tarea que se está realizando (información de estado). Además, y aquí es donde se comenzarían a considerar las características relacionadas con la calidad, al usuario siempre le estaría permitida la posibilidad de abandonar la ejecución de la tarea relacionada con la especificación realizada utilizando el patrón fundamental (*close*), no siendo ésta la única tarea que podría llevar a cabo de forma inesperada. Otro conjunto de tareas colaterales, identificadas como *collaction*, estarían también permitidas y es en ese conjunto en el que tienen cabida los patrones de interacción. Mucha de la experiencia documentada en esos patrones no pasa por ser otra cosa que funcionalidad adicional con un trasfondo y una repercusión directa en la calidad, relacionada con la usabilidad, de la que hace gala el producto software que el usuario utiliza. De esta forma, esta funcionalidad, que habitualmente no se considera o sólo se hace dependiendo de la experiencia que posea el ingeniero, está documentada en los patrones disponibles y puede especificarse en fases tempranas (concretamente en la fase de análi-

sis de requisitos), es decir, primero, elaborando un modelo de calidad en la fase de requisitos y luego en esta fase de diseño.

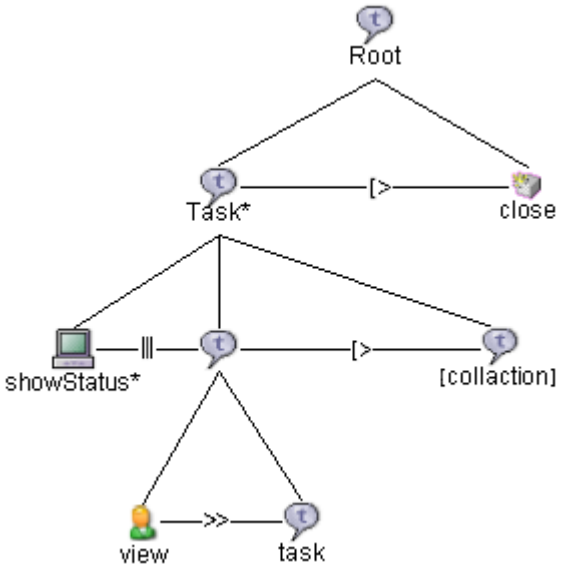


Figura 6-9. Patrón fundamental elaborado para facilitar el modelado de tareas

En la Fig. 6-10, a modo de ejemplo, se utiliza el patrón fundamental para especificar el comportamiento y analizar las tareas asociadas a un sistema hipertexto (que ha sido definido con anterioridad). En él, al usuario, tras habérsele mostrado la interfaz de usuario donde aparecen los diferentes elementos característicos de un sistema hipertexto se le ofrece la posibilidad de pulsar en determinados enlaces y, con ello, poder *conmutar de contexto* o simplemente obtener información adicional sobre el concepto asociado al enlace. En este ejemplo, los patrones de interacción documentan la necesidad de ofrecer una serie de facilidades relacionadas con la posibilidad de que el usuario pueda, entre otras cosas, volver a un lugar seguro, avanzar o retroceder en las acciones realizadas (y por tanto en los contextos visitados), imprimir la información que está visualizando, buscar de forma automática información en el contexto en el que se encuentra y ser informado de y gestionados los errores que puedan producirse mientras se está interactuando con el sistema. Todas esas características mencionadas son las documentadas en los patrones de interacción y se encuentran integradas en el modelo de calidad propuesto en el capítulo tercero mediante su asociación a criterios de calidad.

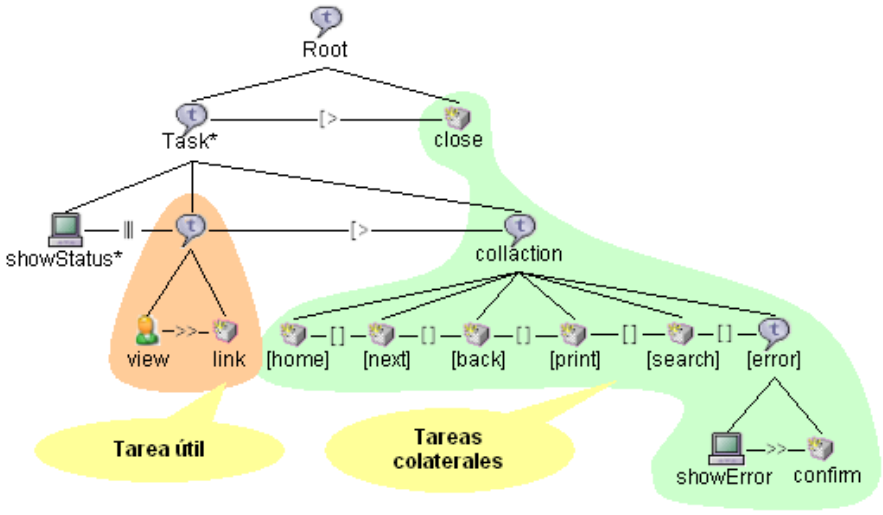


Figura 6-10. Modelo de tareas de un sistema hipertexto con el patrón fundamental

Las especificaciones de tareas realizadas utilizando la notación CTT se consideran en esta tesis doctoral directamente ligadas a los casos de uso y aportan respecto a ellos dos facilidades adicionales. Por un lado facilitan la descripción más detallada de estos últimos y, por otro, mantienen la especificación a un nivel de abstracción deseable en la fase de diseño de un producto software en la que se considera su utilización. Una observación que debe recalcarse es que aunque algunos iconos utilizados en IDEALXML para representar las tareas (concretamente el utilizado para representar las tareas de interacción) puedan hacer pensar que el modo de interacción está establecido, esto no es así y el modo de interacción y la independencia de la plataforma siguen perviviendo en la especificación de las tareas que se realiza cuando se utiliza la notación CTT.

IDEALXML permite, implementando un mecanismo de *drag-and-drop,* (véase la Fig. 6-11) la posibilidad de utilizar los patrones almacenados en el repositorio de patrones que tiene asociado. El ingeniero también puede editar directamente aquellos patrones que considere útiles, utilizando para ello el editor de análisis de tareas facilitado. Como ejemplo de esta posibilidad se ha modelado una colección, que también está disponible en el entorno de desarrollo facilitado con IDEALXML. Esta colección de patrones incorporada en el repositorio gestionado en IDEALXML consta de casi 60 patrones de interacción disponibles textualmente en una colección representativa dentro del ámbito de la interacción como es la colección *Common Ground* presentada por Jenifer Tidwell en 1999 (Tidwell, 1999). La especificación de dichos patrones está recogida en un apéndice que para tal propósito se incluye en esta tesis doctoral. En cualquier caso, el lector debe

ser consciente de que la colección de patrones de interacción que puede almacenarse en IDEALXML no se restringe a los patrones mencionados y el propio usuario de dicha herramienta puede incluir sus propios patrones, fruto de la experiencia de que disponga.

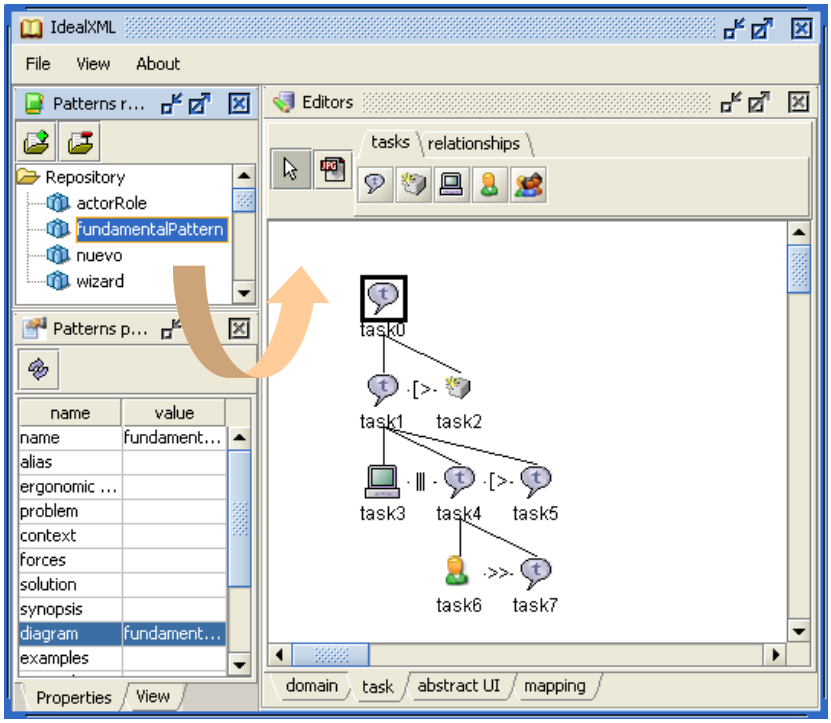


Figura 6-11. Editor de modelos de tareas en IDEALXML

Debe recalcarse, igualmente, que hasta el momento han aparecido, y que se tienen en cuenta, consideraciones de calidad relacionadas directamente con la usabilidad sin que por el momento se haya alcanzado el modelado de la presentación. Pero, con todo y con ello, se ha de tener presente también que muchos de los patrones de interacción disponibles en la colección mencionada en el párrafo anterior tienen una vertiente en su especificación que conlleva la consideración, junto con el análisis de tareas, del modelado de la presentación. Dicho modelado y especificación se puede hacer a un nivel abstracto sin que eso imponga cortapisa alguna al diseño o aspecto visual concreto y final que ofrezca la interfaz y el producto software generado. En definitiva, muchos patrones de interacción no circunscriben su especificación a disponer de un modelo de tareas y, junto a este último, también ofrecen un modelo de presentación que también debe estar documentado. Con ello se abunda en la idea de que la calidad de interacción no

es sólo fruto de la presentación. En cualquier caso, las facilidades identificadas en esta sección y a las que se hace referencia con la denominación de *actividades colaterales* deben ponerse a disposición del usuario del producto software y, por ello, dichas facilidades identificadas en este momento (transición del análisis de requisitos al diseño) implicarán consecuencias tanto en el modelo de dominio como en el modelo de presentación. Este tipo de implicaciones serán tratadas en una sección posterior dedicada al *mapping* entre modelos. Dichas relaciones entre modelos podrán en muchos casos elicitarse de forma automática mediante la transformación de modelos correspondiente.

Un elemento por el que se ha pasado de puntillas aunque su consideración resulta importante en esta tesis doctoral, cuando de la especificación del modelo de tareas se trata, es del concepto de *contexto de uso*. Por contexto de uso se entiende aquella especificación por la que se posibilita alcanzar al usuario un objetivo deseable por él, e identificado en los casos de uso, o, en su defecto, un sub-objetivo necesario para alcanzar dicho objetivo deseable. Puede utilizarse como ejemplo para clarificar el concepto anterior lo siguiente. Supóngase que se está elaborando una aplicación destinada a la realización de compras electrónicas y que para realizar una compra el usuario necesita estar registrado previamente. La tarea de registrado del usuario habría sido identificada como un caso de uso. Para la realización de esa labor el usuario debe facilitar diferente información, a partir de ese momento, el usuario registrado podrá realizar las compras de forma más rápida y eficiente al no tener que facilitar una misma información cada vez que realice una compra. Dicha labor de registro puede especificarse utilizando el patrón fundamental. En la sección *task* de dicho patrón se incluirían aquellas acciones destinadas a que el usuario facilitase la información requerida de una vez y para siempre (ofreciéndose al mismo tiempo la posibilidad de tener acceso a la misma y de poder modificarla en su caso). Esa especificación que conduce a la realización de un objetivo como es el de registrarse en el sistema de compra electrónica es lo que denominamos *contexto*. Si la labor que se pretende realizar es compleja y su realización conlleva la necesidad del paso por varios sub-objetivos (por ejemplo, piénsese en las habituales compras electrónicas asistidas y en las que el usuario avanza por diferentes fases con las que se comprueba la disponibilidad del producto, se constata su tarifa, el usuario facilita la información necesaria para proceder a formalizar el pago, al usuario se le informa del plan de envío y, finalmente, se confirma la compra) a cada uno de los sub-objetivos identificables se les asocia un contexto de uso y el paso de uno a otro lo hemos denominado *conmutación de contexto*. Estos conceptos, contexto y conmutación de contexto, darán pie a la incorporación en una fase posterior (la fase de mapping) de diferentes patrones de interacción que permiti-

rán profundizar en el diseño y modelado de la solución conceptual al problema que se le presenta al ingeniero al elaborar un producto software.

6.3.3 Modelo de presentación

Junto a las labores de modelado de dominio y de tareas, en la hoja de ruta mostrada en la Fig. 6-4 aparecía recogida la necesidad de modelar la presentación. El objetivo con cualquier modelado llevado a cabo en esta fase de diseño será desembocar más tarde en la obtención de algo concreto, algo con lo que el usuario pueda abordar el trabajo que desee acometer (Constantine, 2000).

En la realización del modelado de la presentación consideramos imprescindible la especificación explícita de aquellos mecanismos, que deben proporcionarse al usuario, y que facilitan la realización tanto de tareas directamente funcionales como de aquellas no explícitamente funcionales asociadas directamente con la calidad de la interacción ofrecida al usuario a través de la interfaz finalmente presentada. Dicho modelado de presentación es deseable que siga realizándose a nivel abstracto, para no poner cortapisa alguna a la necesaria y posterior obtención de una interfaz de usuario final realizada con criterio y creatividad. Pero, como se ha comentado la calidad y de la usabilidad, la presentación, su completitud y bondad, hay que buscarla y para ello hay que disponer de una especificación previa que establezca qué elementos debe tener presentes.

Tabla 6-4. Iconografía utilizada en IDEALXML para elaborar modelos de presentación

Icono	Descripción
	Icono asociado a un objeto Container. Objeto de interacción abstracto que permite aglutinar componentes en su interior.
	Icono asociado a un Componente. Objeto de interacción abstracto que mediante la incorporación de distintas facetas definidas en usiXML permite dar soporte a la interacción mantenida entre usuario y sistema.
	Iconos asociados a las diferentes facetas definidas en usiXML. A través de ellas es posible dotar a los componentes de interacción de facilidades para dar soporte a diferentes acciones de interacción, respectivamente, entrada de datos, salida de datos, operaciones de control y de navegación.

Para la especificación del modelo de presentación se puede utilizar IDEALXML. En dicho entorno se ofrece un editor que asocia una notación gráfica (véase la Tabla 6-4) al lenguaje de especificación de modelos de presentación propuesto en usiXML (Limbourg et al., 2004). Con la utilización del editor de modelos de presentación facilitado es posible especificar gráficamente y almacenar, utilizando usiXML, el modelo de presentación

que deseemos asociar a cada contexto de uso identificado inicialmente con la elaboración de los diagramas de casos de uso, en la fase de análisis, y, más tarde, refinado con los modelos de tareas asociados a cada caso de uso identificado utilizando la notación CTT también facilitada en el entorno.

Fruto de la relación directa que existe entre muchas de las tareas especificadas realizada en la fase de análisis de requisitos y de la posterior necesidad de proporcionar mecanismos para proceder al lanzamiento de esas mismas tareas por parte del usuario, se identifica una relación bidireccional entre parte del análisis de tareas (aquella dedicada a contemplar las actividades denominadas colaterales) y el modelado de la presentación. Es decir, si nos circunscribimos al ámbito de la especificación de aquellas tareas relacionadas con la realización de tareas no explícitamente funcionales, no bastará con el modelado de tareas, sino que será necesario complementarlo con una especificación paralela que conllevará consideraciones que afectarán tanto modelo de dominio como al de presentación. Las relaciones entre modelos son el verdadero motor del entorno de desarrollo basado en modelos propuestos, y no así tanto su especificación aislada e independiente. Será por ello por lo que seguidamente abordaremos en el siguiente apartado el mapping entre los modelos considerados hasta el momento.

6.3.4 Mapping entre los modelos especificados

En el apartado dedicado al modelado de tareas, se reseñaba que la puesta en práctica de la gran mayoría de los patrones de interacción, ya sean de los disponibles en (Tidwell, 1999) ya sea de los disponibles en otras colecciones también mencionadas, conlleva tener en cuenta consideraciones relacionadas con todos los ámbitos del análisis y el diseño. Algunos de esas consideraciones vienen directamente determinadas por el ofrecimiento de realización de una tarea al usuario y, por ello, se han tratado en el apartado dedicado al modelado de tareas. Pero, para llevar a cabo esas tareas es necesario tanto ofrecer al usuario los mecanismos de interacción necesarios para lanzar la ejecución de cualquier tarea como proporcionar aquellos recursos que permitan gestionar los datos y entidades que involucran las tareas identificadas. Unos y otros elementos (mecanismos de interacción y recursos) afectan, respectivamente, a los modelos de presentación y de dominio.

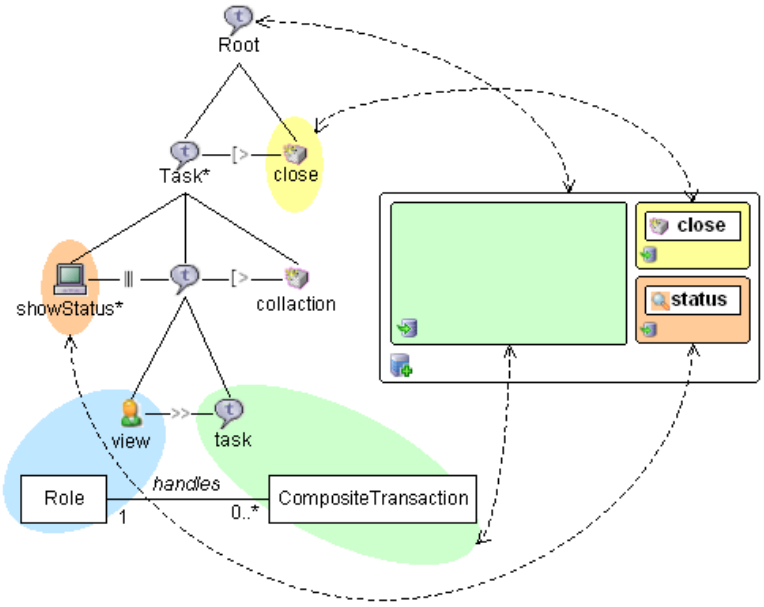


Figura 6-12. Identificación inicial de relaciones entre los distintos modelos

El establecimiento de relaciones entre los diferentes modelos viene, en esta tesis doctoral, dictado inicialmente por el uso de un patrón considerado tanto patrón arquitectónico como patrón de diseño. Dicho patrón es el Modelo-Vista-Controlador (MVC) (Buschmann et al., 1996; Reenskaug, 2003). Una posible representación gráfica de este patrón se muestra en la Fig. 6-12. En ella un modelo de tareas (simbolizado mediante el uso de una notación equivalente a la CTT) que representa el que se ha denominado en esta tesis doctoral el patrón fundamental, un modelo de dominio representado mediante un diagrama de clases (asociado al patrón de colaboración Role-Transaction) y un modelo de presentación (simbolizado con el uso de una especificación abstracta de una serie de objetos de interacción basados en la notación propuesta en usiXML y facilitada en IDEALXML) han sido recogidos y, junto a ellos, se muestran las relaciones entre ellos.

El patrón MVC descompone una aplicación interactiva en tres bloques: un *modelo* que contiene los datos y la funcionalidad de la aplicación y es independiente de la representación que se haga de los mismos, una *vista* encargada de mostrar la información almacenada en el modelo al usuario y un *control*, asociado a cada posible vista, que recibe las entradas en forma de eventos procedentes del usuario o de otros sistemas y traduce dichas entradas en peticiones que envía bien a la vista bien al modelo.

En el patrón MVC se recogen las principales relaciones que pueden identificarse entre las especificaciones recogidas en los apartados anterio-

res y que afectaban a los modelos de dominio, tareas y presentación. Parte de dichas relaciones quedan especificadas mediante el uso de un patrón de diseño como es el patrón *Observer* (Gamma et al., 1995), a través del que se representan las relaciones que mantienen las entidades recogidas en los modelos de presentación, control y dominio (véase la Fig. 6-13).

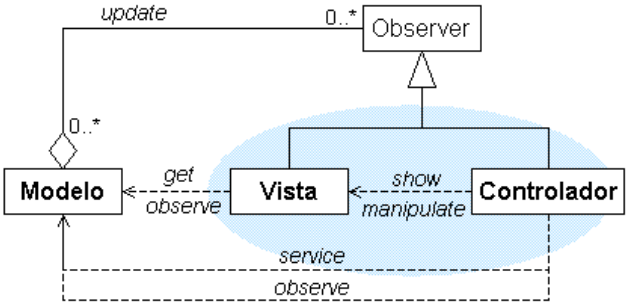


Figura 6-13. Relaciones identificables en el patrón MVC

La zona resaltada en azul en la Fig. 6-13 es a la que directamente afecta la interacción. En ella están recogidas tanto la vista como el controlador, siendo la vista la presentación ofrecida al usuario o a otros sistemas que interactúen con uno dado y el controlador la entidad encargada de responder a la interacción mantenida. En esta tesis el comportamiento del sistema se asocia al modelo de tareas y por ello, el comportamiento queda, al menos inicialmente, modelado mediante la notación CTT facilitada en el entorno IdealXML. Se hace hincapié en lo de modelado inicial ya que el verdadero comportamiento del que hagan gala las tareas especificadas con la elaboración del modelo de tareas utilizando CTT deberá ser posteriormente refinado mediante los diagramas asociados a UML y disponibles gracias a las contribuciones procedentes del mundo de la Ingeniería del Software. Se habría descrito en este momento una doble fase de refinamiento que habría comenzado con los diagramas de casos de uso, habría pasado posteriormente por someter a cada caso de uso a una especificación basada en el modelado de tareas que facilita la notación CTT y por último se habría especificado cada tarea con la elaboración de los diagramas de secuencia y colaboración necesarios que facilita y están disponibles en UML. La propuesta metodológica seguida en la hoja de ruta estaría siendo una propuesta dirigida por casos de uso y tareas. Esta forma de proceder está inspirada en la metodología IDEAS (Lozano, 2001) donde, partiéndose también de los casos de uso, se avanza en procesos de refinamiento sucesivos que conducen a la especificación de una interfaz de usuario asociada a una aplicación.

Además de esta identificación de patrones con la que dictar relaciones entre modelos, en usiXML (Limbourg et al., 2004) también se ha definido un modelo de mapping en el que se identifican distintas relaciones entre los elementos integrantes de los modelos de dominio, tareas y presentación. Muchas de las asociaciones definidas en ese modelo son las identificadas con el patrón MVC y comentadas anteriormente. Otras relaciones, sin embargo, vienen a recalcar el nivel de abstracción en el que puede hacerse la especificación utilizando usiXML y determinan las relaciones existentes entre especificaciones de mayor y menor nivel de abstracción (*isReifiedBy*) y viceversa (*isAbstractedInto*) (Limbourg et al., 2004b). Una tabla resumen de las relaciones de mapeo definidas en usiXML se recoge en la Tabla 6-5 (Montero et al., 2005).

Tabla 6-5. Diferentes asociaciones definidas en usiXML (Limbourg et al., 2004)

Nombre de la relación	Descripción
observes	Define una asociación entre un objeto de interacción y un elemento del modelo de dominio que es modificado (atributo) o ejecutado (método) y produce una variación en la información visualizada en la interfaz
updates	Define una asociación entre un objeto de interacción y un atributo del modelo de dominio, describiendo la situación en la que el valor de dicho atributo del dominio debe ser coherente con el valor mostrado en la interfaz
isReifiedBy	Define una relación de reificación entre un objeto de interacción concreto y otro abstracto
isAbstracteInto	Define una relación de abstracción entre un objeto de interacción abstracto y otro concreto
triggers	Define una asociación entre un objeto de interacción y un objeto del modelo de dominio, describiendo la posibilidad de que un objeto de la interfaz pueda disparar un método disponible en el dominio
isExecutedIn	Define una asociación entre una tarea y un objeto de interacción (container o component)
manipulates	Define una asociación entre una tarea y un elemento del modelo de dominio (atributo, conjunto de atributos, clase o conjunto de clases).
hasContext	Define una asociación entre un elemento de cualquier modelo y uno o varios contextos de uso

En el entorno IDEALXML es posible especificar las relaciones entre modelos establecidas en usiXML. Para ello, una vez especificados los modelos a los que afecta una posible relación habrá que ir al editor de mappings seleccionar los elementos involucrados y especificar la relación de asociación definida. El resultado de la especificación de mapeos entre elementos de los modelos podrá ser almacenado utilizando el lenguaje usiXML al utilizar el entorno IDEALXML.

Tabla 6-6. Reglas de transformación definidas en usiXML (Limbourg, 2004)

Regla	Descripción
No. 1	Por cada nodo hoja en la especificación del modelo de tareas se incluirá un elemento component en el modelo de presentación asociado. A la vez, por cada nodo padre de nodo hoja se incluirá un objeto container que contendrá al anterior componente.
No. 2	Crear la especificación del modelo de tareas de forma paralela con la especificación y descomposición del modelo de tareas.
No. 3	Por cada componente (objeto de interacción abstracto) asociado a una tarea cuya naturaleza consista en la activación de un método de una clase, añadir al objeto de interacción una faceta que posibilite la invocación al citado método.
No. 4	Establecer una relación de adyacencia entre objetos de interacción abstracta creados a partir de dos tareas entre las que hay una relación temporal secuencial (>>).
No. 5	Por cada par de tareas hermanas, de las que se hayan derivado objetos de interacción, definir relaciones entre sus containers que mantengan la misma semántica que se establecía entre las tareas originales.
No. 6	Por cada tarea que manipula o invoca un método, el objeto de interacción asociado mantendrá una relación trigger sobre el mismo método.
No. 7	Si dos tareas manipulan un mismo atributo y están relacionadas mediante una relación secuencial con paso de información ([ ]>>). Entonces el objeto de interacción asociado a la primera tarea definirá una relación update sobre el atributo, mientras que el segundo objeto de interacción definirá una relación observe sobre ese mismo atributo.
No. 8	Si dos tareas manipulan un mismo atributo y mantienen una relación entre ellos de concurrencia con paso de información (|[]|). Los objetos de interacción asociados a las tareas anteriores mantienen una relación observe y update sobre el atributo manipulado por las tareas.
No. 9	Cada container se transforma en objeto ventana. A la inversa todo container es reificado en un objeto box que, a su vez, está embebido en una ventana.
No. 10	Cada container contenido dentro de otro container que haya sido reificado en un objeto window se transforma en un objeto horizontal box embebido en un window.
No. 11	Cada faceta de entrada se reifica en un objeto de interacción concreto editable.
No. 12	Aquellos objetos de interacción abstractos que mantienen una relación de adyacencia, extrapolan esa relación a los objetos de interacción concretos a los que se reifican.
No. 13	Cada container relacionado con otro container a través de una relación “es anterior a” incluye un objeto de interacción con una faceta de navegación en él.
No. 14	Por cada par de containers que mantienen una relación de control de diálogos entre ellos extiende dicha relación a los objetos de interacción concretos asociados.
No. 15	Las relaciones de reificación entre objetos de interacción mantienen las relaciones update definidas sobre ellos.

Lo interesante de tener identificados un conjunto de asociaciones entre modelos es que permite, entre otras acciones relevantes, llevar a cabo la elicitación de parte o de un modelo en su totalidad a partir de la información especificada en otros modelos. Por ejemplo, de esta forma sería posible derivar el modelo de presentación a partir de los modelos de tareas y de

dominio. Esta labor, por ejemplo en la propuesta usiXML, se lleva a cabo mediante la especificación de un modelo de transformación con el que es posible definir un conjunto de reglas que determinan la existencia de unos elementos de especificación en función de la existencia de otros elementos pertenecientes a otros modelos. En usiXML hay definidas 15 reglas de transformación (recogidas en la Tabla 6-6) con las que es posible, primero, obtener, a partir de los modelos de dominio y de tareas, el modelo de presentación especificado a nivel abstracto y, en un segundo lugar, a partir de él es posible obtener una especificación concreta de la interfaz de usuario. Dicho modelo de transformación constituye una buena base sobre la que considerar tanto la experiencia como la calidad, verdaderos propósitos de esta tesis doctoral.

Las principales aportaciones respecto a la propuesta asociada a usiXML de esta tesis doctoral pasan por: revestir de una propuesta metodológica parte de los recursos facilitados en usiXML, incorporar en esta propuesta metodológica la experiencia identificada y considerada útil para la realización de labores de especificación y diseño, incorporar la calidad a través de la disponibilidad de un modelo de calidad relacionado con la experiencia disponible anteriormente considerada y, finalmente, incorporar e integrar todo lo mencionado dentro de un entorno de desarrollo dirigido por la existencia y elaboración de modelos.

6.3.5 Resultados de esta etapa

En la Fig. 6-14 se muestran parte de los refinamientos que se realizan en la fase de diseño. A dicha fase se llega habiéndose identificado en la fase anterior (fase de análisis) los requisitos funcionales deseables por el usuario y los no funcionales en función de llevar a cabo un primer refinamiento realizado sobre los casos de uso, utilizando el concepto de contexto de uso. En la fase de análisis se elaboraban diagramas de casos de uso y se utilizaba la experiencia  disponible en forma de patrones de interacción para lograr identificar los requisitos funcionales  y no funcionales  .

Fruto de la información obtenida en la fase de análisis, en la fase de diseño, se dispone de un modelo de calidad deseable por el usuario y derivado del análisis de tareas realizado. Dicho modelo de calidad está asociado a la experiencia  y a las necesidades requeridas por el usuario  . La notación utilizada para especificar dichos requisitos no explícitamente funcionales, junto con los requisitos funcionales es la notación CTT. Utilizán-

dose especialmente el patrón fundamental presentado con anterioridad (Fig. 6-9).

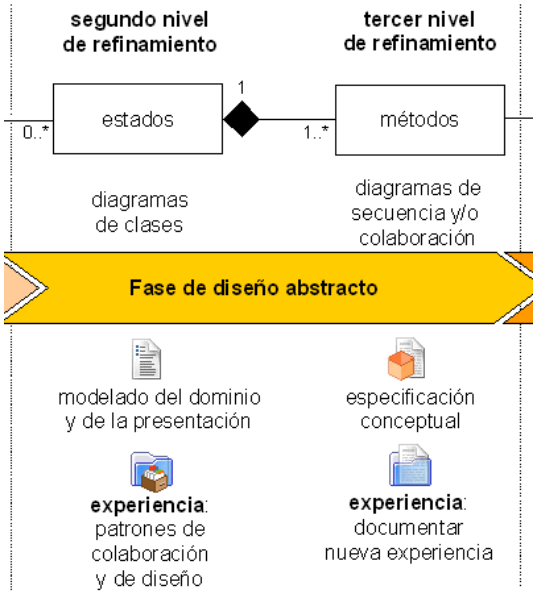


Figura 6-14. Fase de diseño abstracto y sus resultados

En la fase de diseño, a partir de las especificaciones realizadas utilizando la notación CTT, se consideran diferentes colecciones de patrones  (colaboración (Nicola et al., 2001), diseño (Gamma et al., 1995; Apéndice B) y de interacción (Tidwell, 1999)) con el fin de lograr una especificación conceptual completa de las partes estática y dinámica de la aplicación.

En la primera parte de la fase de diseño abstracto se modelaría el dominio y la presentación a un nivel abstracto  , dichas labores estarían facilitadas por el uso de los patrones de colaboración y de interacción. Posteriormente, a cada contexto de uso identificado en la fase de análisis se le asociaría un estado, patrón *State* (Gamma et al., 1995)), en el que se considerarían los métodos necesarios para modelar la funcionalidad explícita y otros métodos para lograr obtener los requisitos no explícitamente funcionales estarían también contemplados. En la Fig. 6-15 se muestran gráficamente dos contextos de uso simbolizados mediante sendos gráficos CTT. Junto a ellos aparecen dos estados que modelarían, utilizando diagramas de clases, los contextos antes reseñados y que se encargarían de proporcionar la funcionalidad deseada y deseable. Se habría cubierto el segundo paso de refinamiento de los tres que se llevan a cabo en esta fase. En este paso nos

hemos centrado fundamentalmente en el modelado y diseño asociado a las tareas, pero, paralelamente, se habría abordado el modelado del dominio y si así se desea el modelado de la presentación, aunque parte de ella se puede haber elicitado utilizando las reglas definidas en usiXML (Limbourg et al., 2004).

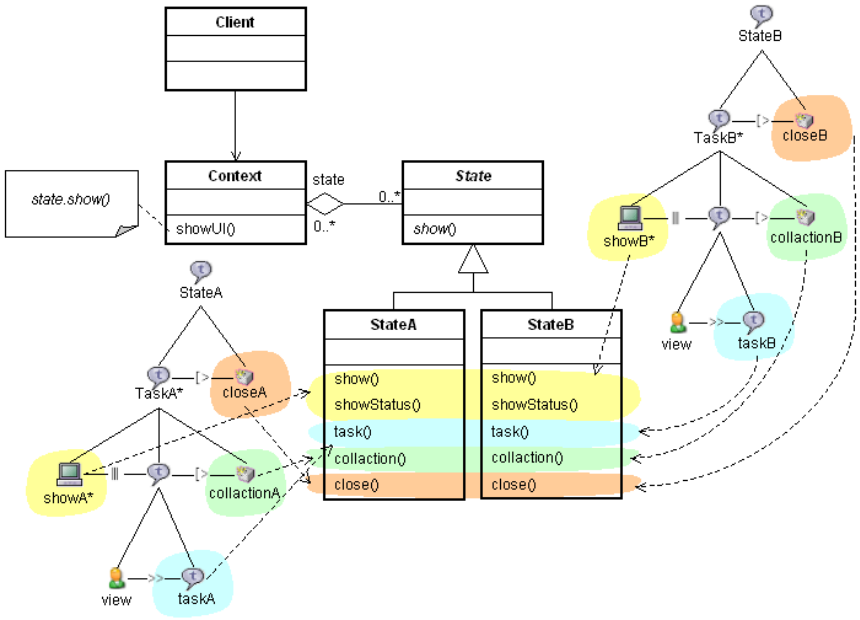


Figura 6-15. Segundo nivel de refinamiento (de CTT y diagramas de clases)

No es el patrón *State* (Gamma et al., 1995:305) el único patrón de diseño que puede utilizarse en este proceso. En la Tabla 6-7 aparecen recogidos diferentes patrones de diseño que se utilizan en este segundo nivel de refinamiento para conseguir diferentes propósitos y especificar conceptualmente cada uno de ellos. Uno de los patrones de diseño que aparece en la Tabla 6-7 es el patrón *Observer* (Gamma et al., 1995:293), con él es posible especificar relaciones de observación y actualización[1] entre los elementos constituyentes en los diferentes modelos de dominio, tareas y presentación. Junto con los patrones comentados los patrones *Abstract Factory*, *Command*, *Memento*, *Strategy* y *Decorator* (Gamma et al., 1995) también se utilizan para lograr diferentes objetivos. Con la utilización de los patrones de diseño mencionados se actúa en otras direcciones de la calidad, concretamente en la calidad interna de un producto software al ac-

[1] relaciones de asociación *observes* y *updates* en usiXML (Limbourg et al., 2004)

tuar sobre la cohesión, acoplamiento, encapsulado y ocultación de la información.

Con la aplicación de los patrones utilizados y la segunda fase de refinamiento se conseguiría alcanzar una especificación conceptual donde se habrían recogido los métodos necesarios para lograr los requisitos funcionales y los no explícitamente funcionales. Dichos métodos serán especificados y diseñados utilizando los diagramas interacción propios de UML (secuencia y colaboración). Dichas especificaciones junto a las anteriores proporcionan una especificación conceptual completa  .

Tabla 6-7. Patrones de diseño utilizados (Gamma et al., 1995)

Nombre del patrón de diseño	Descripción y utilización
Abstract Factory	Creación y gestión de los objetos de interacción. Con dicho patrón se crean las diferentes familias de objetos de interacción posibles para cada dispositivo. A los métodos de las diferentes clases consideradas en este patrón se invocará desde el método showStatus del patrón fundamental. Facilita la creación de la vista.
Observer	Facilita un mecanismo de comunicación entre el modelo de dominio y la vista y el controlador. Define relaciones de observación y actualización asociadas a los atributos definidos en el modelo de dominio.
State	Sirve para modelar el modelo de tareas que ya ha sido especificado utilizando el patrón fundamental y la notación CTT. Cada estado presenta una serie de métodos comunes y otros adicionales que dependen del contexto de uso y de las tareas que puedan desarrollarse en él.
Command	Sirve para modelar las tareas, tanto explícitas como implicitas, relacionadas con un contexto de uso dado. La consideración y logro de muchos de los criterios de calidad recogidos en el modelo de calidad presentado en esta tesis doctoral se traducen en funcionalidad adicional que se proporciona utilizando este patrón.
Memento	Junto al patrón anterior permite recopilar las acciones realizadas y, llegado el momento, deshacer o rehacer cualquier interacción llevada a cabo por el usuario.
Strategy	Permite encapsular los algoritmos que se ocupan de realizar las diferentes tareas y hacerlos intercambiables. La consideración de algoritmos que implementen mecanismos que aporten seguridad y que no la implementen está considerada en este patrón.
Decorador	Permite añadir funcionalidad dinámicamente sin necesidad de hace un uso abusivo del mecanismos de herencia, por ejemplo la funcionalidad asociada a las actividades colaterales identificadas en cada contexto de uso.

El proceso seguido está plenamente orientado a la tarea, pero también es posible orientarlo a la presentación y especificar el modelo de presentación y a partir de dicha especificación asociar los estados correspondientes al tratamiento y funcionalidad, de cualquier tipo, que quiera considerarse en cada presentación utilizada. La Fig. 6-16 muestra las asociaciones posibles en un proceso orientado a la presentación, en lugar de a la tarea. Obvia-

mente y de forma paralela, utilizando esta última orientación es necesario continuar con la especificación y el diseño del modelo de dominio y de tareas.

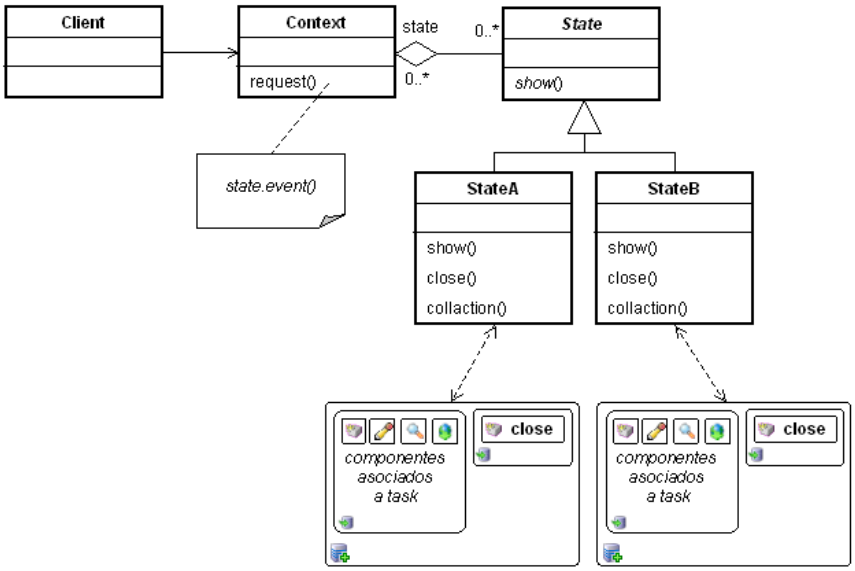


Figura 6-16. Segundo nivel de refinamiento (orientado a la presentación)

Esta fase de diseño se abandona habiendo elaborado una especificación conceptual completa derivada de las sucesivas etapas de refinamiento orientadas a la tarea de la aplicación realizada a nivel abstracto. En una fase posterior dicha especificación de la presentación deberá acomodarse a la plataforma y al modo de interacción elegido. También, fruto de esta fase se dispondría de un modelo de calidad con el que sería posible validar la disponibilidad de sus criterios y subcriterios en la fase de implementación y mantenimiento.

6.4 Implementación automática basada en usiXML

En la última fase de desarrollo, fase de implementación (véase Fig. 6-17a), incluye diferentes actividades. Las mismas pasan por inspeccionar la especificación realizada, implementar la solución obtenida, manteniéndola, evaluándola e introduciendo aquellas mejoras que se identifiquen necesarias. La inspección se llevaría a cabo utilizando patrones, todos los mencionados hasta el momento aunque especialmente seguirán estando presentes los patrones de interacción, con los que el ingeniero debe estar

familiarizado. La principal labor desarrollada en la fase de implementación está orientada al ámbito de la interfaz de usuario y de la presentación de la misma y a la obtención de una interfaz de usuario concreta a partir de la especificación abstracta elaborada en las fases previas. En esta fase se lleva a cabo otro tipo de refinamiento que está relacionado con la consideración del modo de interacción y de la plataforma que será finalmente utilizada por el usuario.

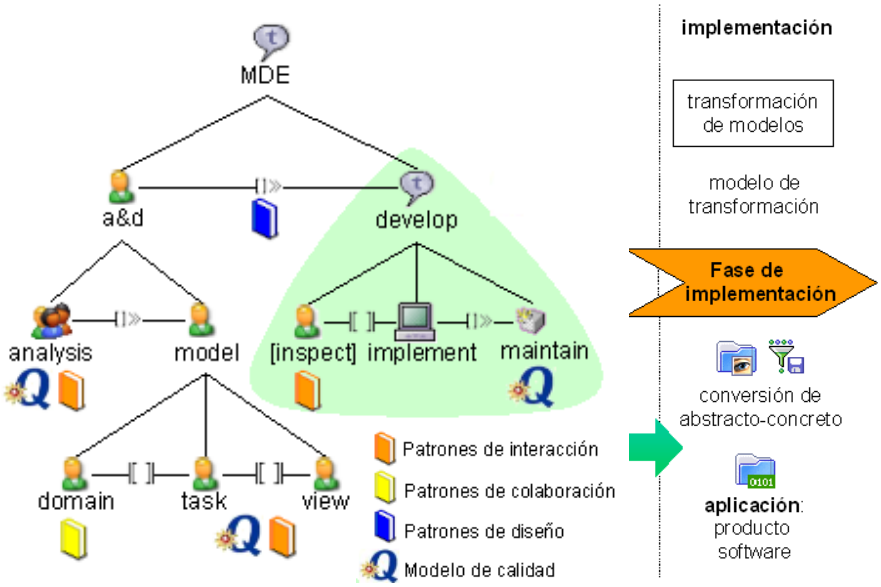


Figura 6-17. Fase de implementación dirigida por modelos (a) descripción del modelo de tareas y (b) actividades y resultados de la fase de implementación

En la Fig. 6-17b se recogen los elementos necesarios para llevar a cabo la fase de implementación. Como se observa las notaciones de especificación y diseño, propias de la fase anterior, han dado paso a los modelos de transformación y a un conjunto de reglas con las que es posible llevar a cabo una conversión de la especificación de la interfaz realizada a nivel abstracto y pasar a una presentación descrita a nivel concreto  , donde se apuesta por un modo de interacción concreto y se determina también la plataforma destino donde se ejecutará la aplicación final obtenida  .

Hasta el momento, en este documento, se ha hecho referencia a la especificación abstracta de la interfaz de usuario. Para realizar dicha especificación se ha utilizado la descripción de IU que posibilita el lenguaje de especificación usiXML (Limbourg et al., 2004). El hecho de trabajar con una

especificación abstracta hace que la misma sea independiente de la plataforma. Seguidamente, las figuras 6-18 y 6-19 recogen una posible descripción de diferentes objetos de interacción concretos, disponibles en diferentes lenguajes de programación de alto nivel, utilizando objetos de interacción abstractos. El proceso de traducción de objetos de interacción abstractos a otros concretos puede abordarse mediante la utilización de reglas de transformación disponibles en usiXML (Limbourg et al, 2004).

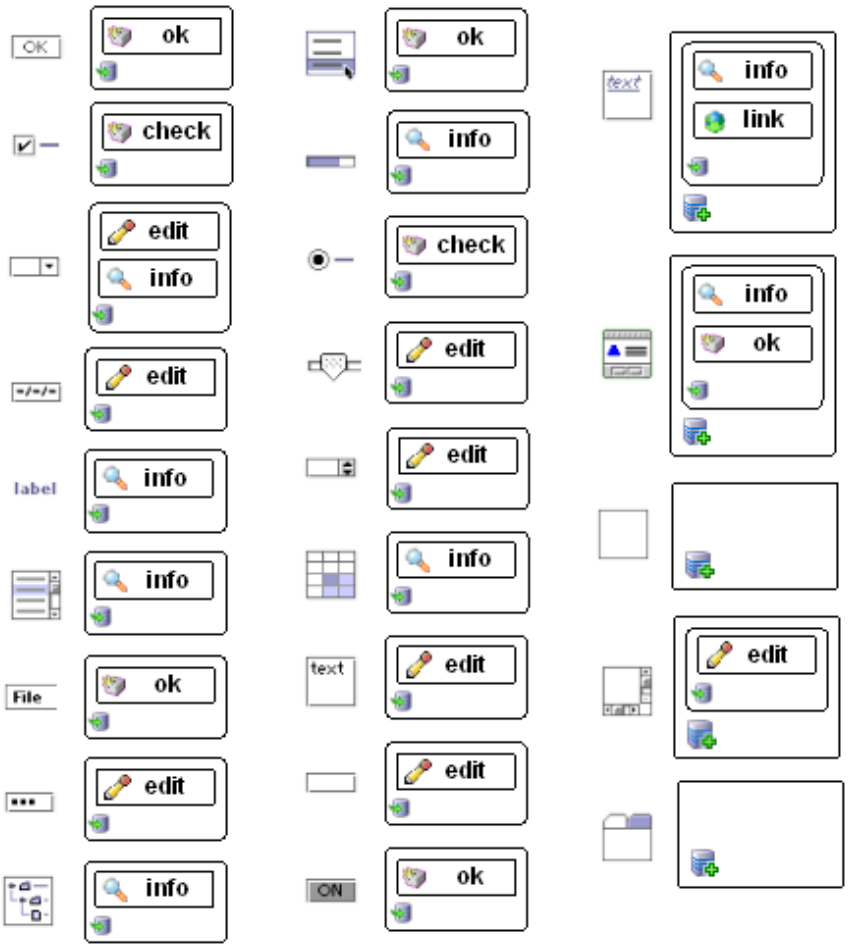


Figura 6-18. Objetos de interacción concreta y junto con su equivalente abstracto

La Fig. 6-18 muestra, en tres dobles columnas, objetos de interacción concretos y su correspondiente representación abstracta utilizando usiXML. Siguiendo las denominaciones utilizadas en el lenguaje de espe-

cificación usiXML, las dos primeras dobles columnas muestran componentes y la última doble columna muestra containers. Cada componente presenta una o varias facetas, específicamente aquellas necesarias para desempeñar su función (entrada, salida, navegación y control). Los objetos de interacción concretos mostrados en la Fig. 6-18 son componentes swing de interfaz de usuario.

En la Fig. 6-19 se muestra una figura similar a la Fig. 6-18, pero en ésta los objetos de interacción concretos son los definidos en MIDP para la elaboración de interfaces de usuario destinado a dispositivos móviles.

Figura 6-19. Componentes de interacción de MIDP y su equivalente abstracto

6.5 Posibilidad de adaptación

Otra posible consideración que puede abordarse utilizando similares recursos que la propuesta presentada y que añade la posibilidad de adaptación en tiempo de ejecución de la interfaz de usuario generada, adaptación llevada a cabo en función del dispositivo con el que interactúa el usuario, es la llevada a cabo utilizando agentes y presentada en otra de las tesis doctorales realizada dentro del grupo de investigación LoUISE. El método propuesto responde a la denominación de AB-UIDE (*Agent-Based User Interface Development Environment*). Dicho método se basa en una arquitectura multi-agente, que proporciona capacidades de adaptación diseñadas en tiempo de ejecución y en la cual diferentes agentes colaboran para proporcionar inteligentemente al usuario las adaptaciones más adecuadas en cada situación que se le presenta durante la interacción con la interfaz de usuario (López-Jaquero et al., 2003).

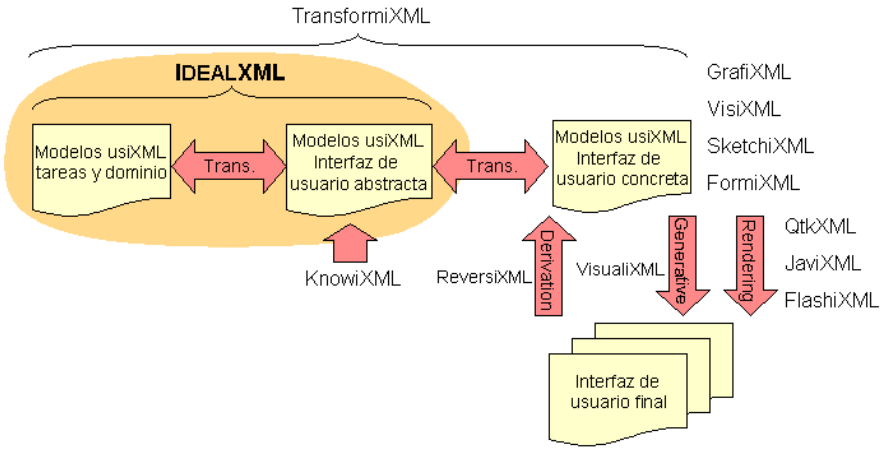


Figura 6-20. Integración con otras propuestas de la metodología presentada

6.6 Beneficios al estar integrada la propuesta dentro de UsiXML

La metodología presentada y la herramienta que le da soporte no está desarrollada de forma aislada sino que está integrada dentro de un desarrollo más ambicioso de *suite* de propuestas y herramientas destinadas a considerar diferentes labores como son la consideración de las guías de estilo (knowiXML), la generación de automática de la interfaz abstracta y a par-

tir de ella de la interfaz de usuario concreta (TransformiXML), la posibilidad de realizar especificaciones a nivel concreto (GrafiXML), herramientas para crear bocetos (SketchiXML, FormiXML) y la generación automática y directa de aplicaciones utilizando transformaciones y disponibles en java (JaviXML) o Flash (FlashiXML) e indirecta mediante el uso de reglas de derivación (ReversiXML).

El desarrollo de las propuestas y herramientas reseñadas y cuya localización y representación gráfica aparece recogida en la Fig. 6-20 responde a las actuales formas y maneras de trabajar englobadas dentro de lo que se conoce como MDA (Vanderdonckt, 2005) y se realizan de forma integrada dentro de un grupo de trabajo de carácter y ámbito internacional.

6.7 Análisis y conclusiones

En la Fig. 6-21 se muestra la puesta en práctica de la metodología presentada a lo largo de este capítulo. La misma centra su labor en las primeras fases del desarrollo (análisis y diseño), dejando caer en las fase posterior de implementación el peso de adaptar el diseño de la solución realizado a un modo de interacción y a una plataforma concreta.

La transformación de modelos es la base de la implementación, mientras que la calidad y la experiencia son los elementos que se consideran de forma recurrente en las fases de análisis y de diseño. La caracterización del modelo de calidad deseado por el usuario es el elemento más característico de la propuesta presentada y su obtención, a partir de la experiencia disponible ya sea ésta documentada o propia del ingeniero, es otro aspecto distintivo y novedoso frente a otras propuestas similares.

La figura anterior muestra (de arriba hacia abajo y de izquierda a derecha) las fases correspondientes por las que debe pasar el ingeniero. En la segunda de las filas, aparece recogido el recurso más destacado utilizado en cada fase, así partiendo de los casos de uso se avanza por una serie de refinamientos que conducen a través del modelado de tareas al logro de la especificación conceptual de la solución al problema de desarrollo que se le plantea al ingeniero. Junto a cada recurso utilizado se ha recogido la notación usada para su documentación y formulación, fundamentalmente UML aunque alguna notación representativa del terreno de la interacción persona-ordenador, como es la notación CTT, también se utiliza en las primeras fases para refinar los casos de uso e identificar el modelo de calidad deseado en función de uno general establecido (el presentado en el capítulo tercero de este documento). En la parte inferior de la Fig. 6-21 apa-

recen recogidos los resultados parciales y finales obtenidos a partir de la evolución por el ciclo de vida descrito.

La propuesta está integrada con otras propuestas y herramientas con las que se comunica gracias a utilizar el mismo lenguaje, usiXML (Limbourg et al., 2004). En el marco de esas propuestas TransformiXML y AB-UIDE especifican la base de la generación automática o semi-automática de la interfaz de usuario a través del uso de reglas de transformación recogidas en un modelo de transformación establecido.

En definitiva, la calidad y la experiencia se consideran y lo hacen de forma reiterada en el ciclo de vida mostrado. Su aparición se realiza de forma sistemática y precisa utilizándose la experiencia documentada y el modelo de calidad general establecido para el desarrollo de interfaces de usuario presentado en capítulos anteriores.

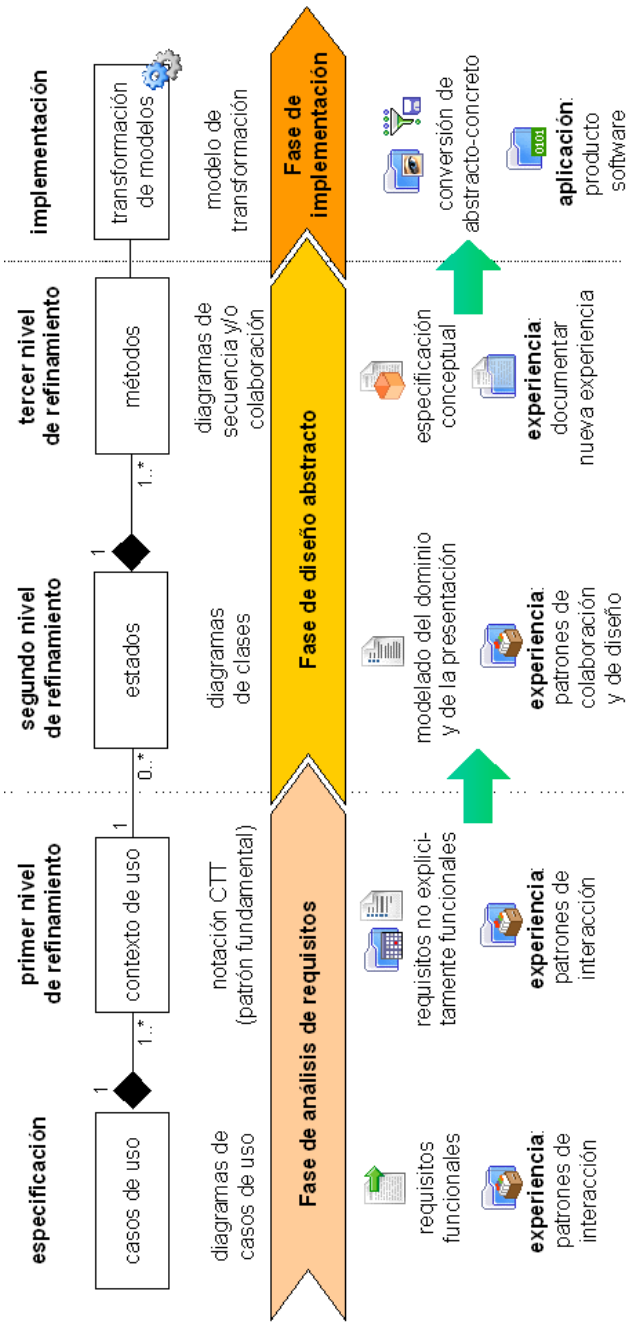


Figura 6-21. Fases, notaciones y resultados de la metodología presentada

6.8 Contribuciones relacionadas con este capítulo

Se recogen seguidamente aquellas publicaciones relacionadas con el capítulo que se acaba de presentar y que han permitido, con su publicación y presentación en diferentes congresos y conferencias específicas, difundir los extremos de la propuesta asociada a esta tesis doctoral y posibilitar un feedback que ha permitido enriquecer y mejorar diferentes aspectos que se han ido considerado en su elaboración.

- López-Jaquero, V., **Montero, F.**, Molina, J.P., Fernández-Caballero, A., González, P. Model-Based Design of Adaptive User Interfaces through Connectors. DSV-IS 2003: 245-257
- Sendín, M., Lorés, J., **Montero, F.**, López-Jaquero, V., González, P. User Interfaces: A Proposal for Automatic Adaptation. ICWE 2003: 263-266
- Lozano, M., **Montero, F.**, González, P. A usability and accessibility oriented development process. 8th ERCIM Workshop User Interfaces for all. Viena, 28-29 de junio de 2004.
- **Montero, F.**, Lozano, M., González, P. Calidad en interfaces de usuario. Capítulo 5 del libro Calidad en el desarrollo y mantenimiento del software. Coordinadores: Mario G. Piattini, Felix O. García. Editorial Ra-Ma. 2003. (ISBN: 84-7897-544-6).
- **Montero, F.**, Lozano, M., González, P.: IDEALXML: an Experience-Based Environment for User Interface Design and pattern manipulation. Technical Report DIAB-05-01-4. Universidad de Castilla-La Mancha, Albacete (2005).
- **Montero, F**., López-Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M.D., Solving the Mapping Problem in User Interface Design by Seamless Integration in IDEALXML. 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'2005), Newcastle upon Tyne, England, July 13-15, 2005. Springer-Verlag, Berlin, 2005 (to appear)

Referencias bibliográficas

Bézivin, J : Model Driven Engineering: Principles, Scope, Deployment and Applicability. In: Pre-proceedings of the Generative and Transformational Techniques in Software Engineering (GTTSE'05), Tutorials. Centro de Ciências e Tecnologias de Computação, Departemento de Informatica, Universidade do Minho, Braga, Portugal, pages 1--33.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. John Wiley & Sons; 1 edition. 1996

Coad, P., North, D., Mayfield, M. Object Models: Strategies, Patterns, and Applications (2nd Edition). Pearson Education; 2 edition (October 10, 1996)

Cockburn, A. Crystal Clear: A Human-Powered Methodology for Small Teams. (Agile Software Development Series). Addison-Wesley Professional. 2004

Cockburn, A. "Structuring Use Cases with Goals", *Journal of Object-Oriented Programming*, Sep-Oct, 1997 and Nov-Dec, 1997. Also available on http://members.aol.com/acockburn/papers/usecases.htm

Constantine, L. What Do Users Want? Engineering usability into software. Reprinted and revised (June 2000) from Windows Tech Journal, December 1995. http://www.foruse.com . 2000

Fincher, S. Perspectives on HCI patterns: concepts and tools (introducing PLML). *Interfaces*, (56):26-28, September 2003

Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional; 1st edition (June 28, 1999)

Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns - Elements of Reusable Object-Oriented Software. Addison Wesley, 1995

Limbourg, Q., Vanderdonckt, J., Addressing the Mapping Problem in User Interface Design with UsiXML, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA. 2004

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-Jaquero, V., UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems.2004

López-Jaquero, V., Montero, F., Molina, J.P., Fernández-Caballero, A., González, P. Model-Based Design of Adaptive User Interfaces through Connectors. DSV-IS 2003: 245-257

Lozano, M. Entorno metodológico orientado a objetos para la especificación y desarrollo de interfaces de usuario. Tesis doctoral realizada en la Universidad Politécnica de Valencia. 2001

Montero, F., López-Jaquero, V., Vanderdonckt, J., González, P., Lozano, M., Limbourg, Q. Solving the mapping problem in user interface design by seamless integration in IdealXML. 12th International Workshop on Design, Specification and Verification of Interactive Systems. (DSV-IS 2005) Devonshire Building, University of Newcastle-upon-Tyne, England. July 13-15, 2005. pg.123 -134

Nicola, J., Mayfield, M., Abney, M., Abney, M. Streamlined Object Modeling: Patterns, Rules, and Implementation. Prentice Hall PTR; 1st edition (September 21, 2001)

Paternò, F. Model-based Design and Evaluation of Interactive Applications. Springer Verlag, November 1999, ISBN 1-85233-155-0

Reenskaug, T. The Model-View-Controller (MVC). Its Past and Present. JavaZone, Oslo, 2003

Tidwell, J. Interaction Design Patterns (aka. Common Ground). http://www.mit.edu/%7Ejtidwell/interaction_patterns.html. 1999.

Vanderdonckt, J., A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005), O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31

Capítulo 7 Caso de estudio: elaboración de un sistema de compra electrónica

A quién va usted a creer, ¿A mi, o a sus propios ojos?
Groucho Marx (Actor estadounidense)

7.1 Introducción

Este capítulo está dedicado a la puesta en práctica, utilizando un ejemplo concreto, de la propuesta metodológica presentada en los capítulos anteriores y detallada especialmente en el capítulo anterior. En él se plasmó, de forma general, una propuesta metodológica indicando los resultados, documentos y especificaciones, elaboradas en cada una de las etapas que la constituyen y se prestó especial atención a la interfaz de usuario y a la calidad que ésta ofrece. La calidad considerada debe lograrse y para ello debe caracterizarse y contemplarse desde el principio. Los principales recursos que se utilizan para ello provienen tanto de la IS y como de la IPO.

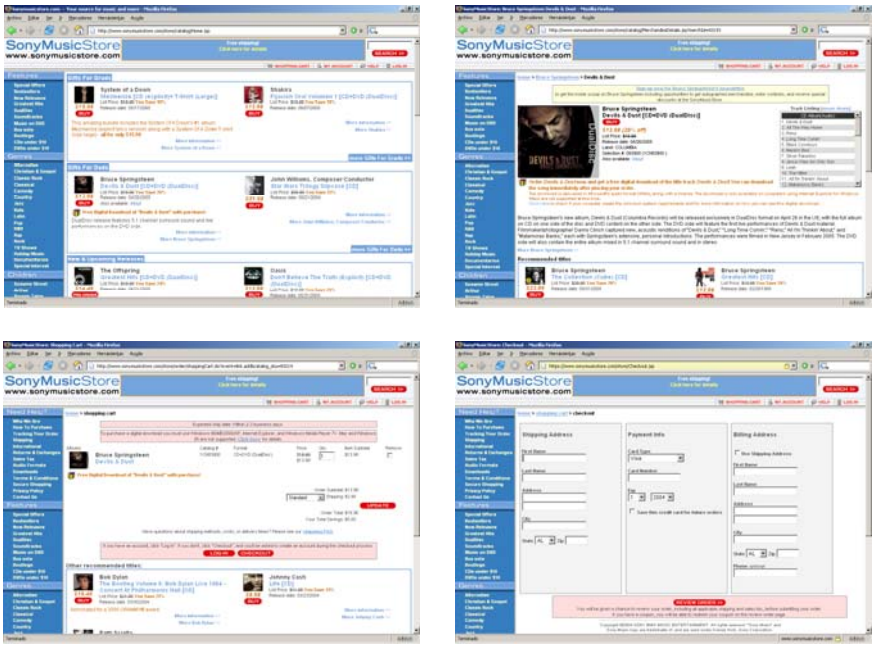


Figura 7-1. Ejemplo de aplicación de compra electrónica orientada a la Web

El ejemplo que utilizaremos para la realización de un recorrido concreto por la metodología mencionada será el de un producto software dedicado a la realización de compras electrónicas. En la Figura 7-1 se ha mostrado un sitio web, concretamente el ofrecido por Sony Music[1] (http://www.sonymusicstore.com) destinado a la venta de material de temática musical y tomado en cierta manera como referencia para la elaboración del caso de estudio.

Figura 7-2. Distintas fases consideradas en la propuesta metodológica

La presentación de contenidos llevada a cabo en este capítulo está marcada por el recorrido metodológico general introducido en el capítulo anterior (Fig. 7-2). Dicho recorrido no es secuencial, sino iterativo, presentando múltiples lazos y realimentaciones derivados tanto de la identificación de nuevos requisitos, ya sean éstos funcionales o no funcionales, como de taras relacionadas con la calidad que presente el ofrecimiento de los requisitos funcionales considerados a través de la interfaz proporcionada. De esta forma, se considerarán diferentes fases, dedicadas al análisis de requisitos, al diseño y a la implementación, en el recorrido que se prescribe. En este capítulo, junto a cada fase considerada se plasmarán aquellas especificaciones o modelos elaborados que permitan y busquen el desarrollo del producto software deseado.

Al final de este capítulo se llevará a cabo una labor de análisis y conclusiones del proceso seguido.

7.2 Análisis de requisitos

En esta fase inicial de análisis de requisitos y dado el ejemplo concreto que se ha elegido y que se pretende especificar debemos comenzar familiarizándonos con el sistema deseado y viendo qué debe ofrecerse, qué podemos facilitar en función de la calidad que debe considerar un sistema de venta electrónica y qué permite hacer la competencia o lugares de compra/venta electrónica que estén disponibles en el mercado.

La puesta en práctica de esta fase tiene como resultado inicial más representativo la elaboración y documentación de uno o varios diagramas de casos de uso en los que quedan recogidos, gráfica y textualmente, aquellos requisitos funcionales deseables. Para alcanzar una especificación lo más

[1] "Sony Music" y su logo son marcas registradas por Sony Corporation.

completa posible de estos requisitos se pueden poner en práctica, en función de la experiencia del equipo ingenieros ocupados de la misma, múltiples técnicas relacionadas con el análisis de requisitos (muchas de ellas fueron recogidas en el capítulo tercero de este documento, por ejemplo cuestionarios, entrevistas, observación, inspección de productos similares, etcétera). Volviendo a los diagramas de casos de uso la principal premisa que debe tenerse en cuenta es la no inclusión en su documentación de características que determinen posteriormente el desarrollo, en especial en lo que tiene que ver con el modo de interactuar con el sistema, con ello conseguiremos que el análisis realizado pueda servir para, posteriormente, poder utilizar esa misma especificación para abordar la elaboración de dicho producto software y destinarlo a otra plataforma o considerar otros modos de interacción.

El icono utilizado para denotar la documentación inicialmente elaborada en esta fase es el situado a la izquierda y con él se da idea de que los casos de uso no son únicamente gráficos, sino que ofrecen una versión textual que resulta tanto o más importante que la anterior. Dicha especificación textual recoge información importante a la hora de abordar tareas posteriores, y así se ha querido recoger incluyendo la flecha de color verde incluida en el icono utilizado.

7.2.1 Requisitos funcionales

Cualquier proceso de elicitación y análisis de requisitos casi nunca se corresponde con una labor directa o inmediata, sobre todo si se carece de la experiencia suficiente que permita llevarla a cabo de esa manera. Por lo general, cualquier proceso de análisis se traduce en un proceso de refinamiento sucesivo hasta que se considera, refrendando esta opinión con el usuario, que se ha identificado suficiente información para poder llevar a cabo fases posteriores de desarrollo y la obtención final de un producto software acorde con las necesidades del usuario.

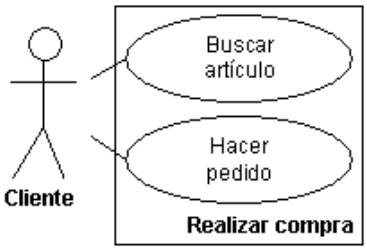


Figura 7-3. Diagrama de casos de uso simplificado de un sistema de compra electrónica

En la Fig. 7-3 se recoge un primer diagrama de casos de uso relacionado con el supuesto contemplado. En dicho diagrama se han considerado únicamente dos casos de uso: "Buscar artículo" y "Hacer pedido". Dichos casos de uso, según se refleja, puede llevarlos a cabo un usuario que juega el role de cliente. Sin embargo, los casos de uso reflejados, siendo en esencia los que deberían facilitarse a la hora de abordar el tipo de aplicaciones que se está considerando, son muy generales y no contribuyen a identificar y a poder considerar características de calidad. Resulta conveniente seguir refinando dicho diagrama y en este sentido la Fig. 7-4 recoge un diagrama algo más elaborado donde los anteriores casos de uso se han dividido, añadiéndose algunos otros casos de uso que redundan en las operaciones y tareas que el usuario (cliente) podrá llevar a cabo utilizando el sistema. En este otro diagrama de casos de uso algunas consideraciones relacionadas con características de calidad y con el modelo de calidad descrito en capítulos anteriores también se han tenido en cuenta aunque dichas características no queden reflejadas de forma explícita en el diagrama de casos de uso (no es éste tampoco su principal objetivo).

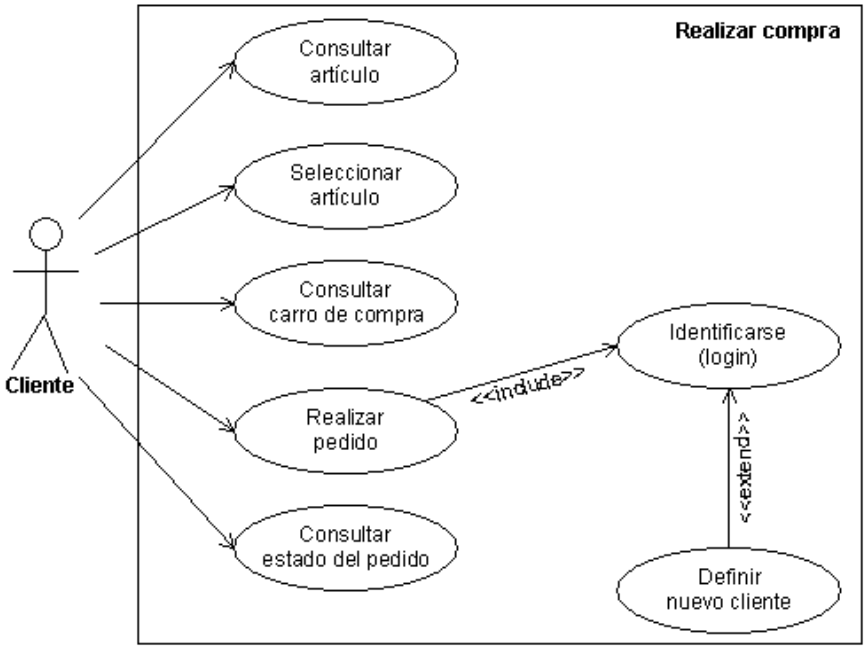


Figura 7-4. Diagrama de casos de uso asociado a un sistema de compra electrónica

Algunas de las características de calidad que pueden tenerse en cuenta desde estas primeras fases de desarrollo pasarían por aquellas que, recogidas en el modelo de calidad propuesto, contribuyen de forma decisiva a la

calidad y no tienen una puesta en práctica directamente relacionada con la presentación que ofrezca la solución asociada a su logro. Por ejemplo, la compatibilidad, la provisión de un feedback inmediato, el ofrecimiento al cliente de tareas ideadas para ser llevadas a cabo mediante un número mínimo de acciones, la posesión por parte del usuario del control sobre las acciones realizadas por el sistema y el hecho de que dichas acciones sean explícitas son ejemplos de dichas criterios de calidad que pueden considerarse paralelamente con la documentación de los casos de uso.

La realización del diagrama de casos de uso mostrado en la Fig. 7-4 puede llevarse a cabo, o al menos facilitarse, mediante un proceso de inspección de los patrones de interacción recopilados y disponibles (simbolizado mediante el icono adjunto). En ellas junto a la descripción de cada patrón se acompañan imágenes donde se muestra gráficamente su puesta en práctica e implementación en situaciones reales. Este proceso puede pasar por ser una técnica más de implementación de técnicas de Diseño Centradas en el Usuario, en cuya puesta en práctica pueden incluirse usuarios reales o potenciales del producto software que se está elaborando.

Habitualmente, la documentación de los casos de uso, recogidos en los diagramas de casos de uso elaborados, se realiza de forma textual y su estructura responde a diferentes plantillas. En la Tabla 7-1 se recoge, como ejemplo, la documentación asociada al caso de uso "Realizar pedido". Como se ha comentado con anterioridad, las características directamente relacionadas con la consecución y ofrecimiento de características de calidad pueden quedar recogidas en la especificación textual de los casos de uso que aparecen en todo diagrama de casos de uso. Pero, para que eso ocurra la experiencia y capacidad del ingeniero encargado de la elaboración de los diagramas juega un papel fundamental. En este sentido, la experiencia y la propia disponibilidad de un modelo de calidad universal, o medianamente consensuado que pueda ser utilizado y que sea conocido por el mayor conjunto de personas involucradas en el desarrollo de productos software, permite considerar en estas primeras fases la calidad y establecer las bases que permitan su logro.

En este sentido y en esta tesis doctoral, los patrones de interacción se han considerado asociados a la calidad y cada patrón está relacionado con un conjunto de criterios de calidad provenientes de un conjunto de criterios y subcriterios relacionados, a su vez, con la ergonomía, verdadera dimensión en la que la usabilidad de un producto software ha quedado caracterizada en esta tesis doctoral. El conocimiento de este modelo de calidad es necesario para identificar los posibles problemas que puedan presentársele al usuario, ya que responde a la pregunta *qué buscar*. Por otro lado, desde el punto de vista del ingeniero que dispone del modelo de calidad, éste de-

be plasmarse en el producto software elaborado y para ello necesita saber *cómo* lograr considerar el modelo. En este desafío el ingeniero puede ayudarse de los patrones disponibles, especialmente de los de interacción.

Tabla 7-1. Especificación textual del caso de uso "Realizar pedido"

Identificación: Realizar pedido	
Resumen	El usuario desempeñando, el role de cliente, ha seleccionado una serie de artículos en los que está interesado y, con este caso de uso, procede a la formalización de la compra identificándose y/o facilitando aquella información que le permite adquirir los productos deseados
Prioridad	Esencial
Frecuencia	Frecuente
Precondiciones	El usuario debe haber seleccionado una serie de productos, esta selección se traduce en la incorporación de dichos productos a un carrito de compra que es la metáfora habitual utilizada en este tipo de aplicaciones El usuario no necesita haber hecho un login sobre el sistema para seleccionar los productos deseados, aunque sí que deberá identificarse llegado el momento de la formalización de la compra
Escenario principal	- El cliente puede recorrer el catálogo o catálogos de productos ofrecidos. - El cliente puede obtener información adicional relacionada con los productos. - El cliente puede adquirir cualquier producto que sea de su interés. - El cliente deberá proporcionar aquella información necesaria para formalizar la compra de los productos deseados y seleccionados. Dicha información no tendrá que proporcionarse siempre si el usuario está registrado y ya ha facilitado la información necesaria. - La información que debe proporcionar el cliente estará relacionada con información propia (nombre completo, dirección, información de contacto, etc.), con información relacionada con la forma de envío y con la forma de pago.
Escenario alternativos	El proceso de suministro de información relacionada con el cliente, así como sus preferencias en cuanto a modalidad de envío y pago pueden saltarse si el usuario ya está registrado, recuerda y utiliza su identificador. Si el usuario no recuerda su identificador será informado de que existe un error en el proceso de identificación y se proporcionará aquellos mecanismos que se consideren necesarios para poder obtener dicha información cuando el usuario no la recuerde
Notas y cuestiones	Al tratarse con información sensible al usuario (Cliente) debe informarse y cumplirse con la normativa legal vigente que regule el tratamiento que ha de darse a este tipo de información

Como ya hemos comentado y como puede refrendarse de la lectura del caso de uso recogido en la tabla anterior, en la documentación y en la identificación de los distintos casos de uso juega un papel fundamental la experiencia del ingeniero y, por ello, consideramos realmente necesaria la inclusión de características que redunden en la calidad (a los que hemos venido definiendo como *requisitos no explícitamente funcionales*) en su elaboración. Del caso de estudio inicial nos centraremos en lo sucesivo en el caso de uso "Realizar pedido" por tratarse de un caso de uso significati-

vo para seguir haciendo un recorrido de la propuesta metodológica. Dicho caso de uso está compuesto por diferentes fases en su realización y esta misma depende de que el usuario ya esté registrado en el sistema, y ya haya proporcionado la información necesaria para formalizar la compra o no lo haya hecho. De estas consideraciones se desprende que a los diagramas de casos de uso se les debe someter a un estudio mayor o refinamiento adicional que permita avanzar al ingeniero por las diferentes fases de desarrollo consideradas de una forma menos dependiente de la experiencia que posea y donde pueda, a su vez, tenerse presente la calidad del producto que pretende elaborar.

7.2.2 Requisitos no explícitamente funcionales

En esta fase de análisis en la que estamos se pretende principalmente identificar aquellos requisitos funcionales que deben ofrecerse al usuario cuando haga uso de la aplicación. En el ejemplo que estamos utilizando dichos requisitos funcionales son básicamente los recogidos en los diagramas de casos de uso elaborados y mostrados en las Figs. 7-3 y 7-4.

En la identificación de requisitos funcionales pueden contribuir los patrones de interacción (véase el icono que simboliza la aparición en escena y el uso de patrones, cualquiera que sea su ámbito, pero concretamente en esta fase se hace referencia a las colecciones de patrones de interacción). Paralelamente con la identificación de dichos requisitos funcionales, también se consideran los requisitos no explícitamente funcionales ligados a los funcionales. Esta doble labor de identificación puede abordarse gracias a la incorporación de características de calidad y experiencia de la que hacen gala los patrones, especialmente los de interacción.

En el capítulo quinto se recogían sendas tablas (Tabla 5-6 y 5-7) en las que aparecen cruzados dos ingredientes deseables en lo que respecta a elaboración de productos software, la experiencia y la calidad (Fig. 7-5). Los patrones recogidos en dichas tablas son los propuestos en (Tidwell, 1999) y los criterios de calidad se derivan del modelo de calidad propuesto. Sin embargo, en la referida colección de patrones de interacción no aparecen reflejados explícitamente características de calidad que se logren utilizando los patrones. En esta tesis doctoral asociamos el modelo de calidad propuesto en el capítulo tercero con la calidad disponible identificada en el capítulo cuarto. Fruto de dicha asociación, en el capítulo quinto aparecieron recogidas características de calidad que pueden potenciarse al utilizar la experiencia disponible en forma de patrones de interacción. Por ello, debido a la selección y utilización de determinados

patrones de dicha colección puede elaborarse un modelo de calidad deseable por el usuario y disponer de otra experiencia que también redunda en dicho modelo de calidad y que podría no haber sido considerada inicialmente debido a la experiencia de la que dispone el equipo de desarrollo.

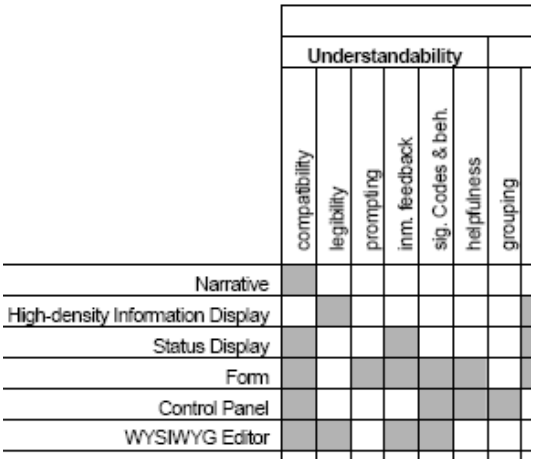


Figura 7-5. Tabla cruzada de experiencia y calidad

De este modo, en el ejemplo de venta electrónica y concretamente llegado el momento de considerar el caso de uso "Realizar pedido", identificaremos que el mismo se puede abordar de diferente forma en función de que el cliente esté registrado (y por lo tanto no tenga que proporcionar otra información que no sea su identificación) o no lo esté (en cuyo caso deberá proporcionar información relacionada con su identificación, con la forma de envío y con la forma de pago).

Tomando las consideraciones anteriores como premisa, si se revisan los patrones de interacción recopilados en la colección *Common Ground* (Tidwell, 1999) puede identificarse un sublenguaje de patrones orientado al tratamiento específico de situaciones en las que se desea realizar una secuencia de operaciones (*Step-by-Step Instructions*). Fruto de las relaciones que se documentan en los propios patrones puede, a partir del patrón mencionado, identificarse una serie de patrones relacionados: *Go Back One Step*, *Go Back to a Safe Place*, *Progress Indicator*, *Map of Navigable Spaces*, *Interaction History*, *Optional Detail On Demand*, *Disabled Irrelevant Things*, *Convenient Environment Actions* y *Good Defaults*. A su vez, en esta tesis hemos identificado que la aplicación de cada patrón redunda positivamente sobre una o varias características de calidad presentes en un modelo de calidad global elaborado utilizando criterios ergonómicos, concretamente, con la utilización de los patrones anteriormente citados se influye en diferentes criterios recogidos en la Tabla 7-2, que

muestra información extraída de la que aparecía en las tablas 5-6 y 5-7 en el capítulo quinto.

De la información tabulada y mostrada en la Fig. 7-2 puede extraerse, por un lado, un modelo de calidad asociado al contexto de uso identificado y extrapolable al propio producto software que se está elaborando y, por otro, un conjunto de características que podrán ser comprobadas en una fase de evaluación posterior. Además, fruto de disponer de la misma asociación entre experiencia y calidad, si en fases posteriores se identifican limitaciones en el logro de algún criterio en un contexto de uso determinado podremos buscar en sentido inverso (respecto al considerado hasta ahora), es decir, buscando experiencia que conduzca al logro de dicho criterio de calidad deseado por el usuario o deseable a criterio del ingeniero.

Tabla 7-2. Modelo de calidad resultante de considerar distintos patrones

	Usability																					
	Understandability						Learnability						Operability									
	compatibility	legibility	prompting	inm. feedback	sig. Codes & beh.	helpfulness	grouping	Physical workload	Minimal actions	conciseness	Information density	consistency	Explicit user action	User control	User experience's	flexibility	Error protection	Quality error msg.	Error correction	Privacy policies	Accessibility
Step-by-step Instructions	■		■	■	■	■	■		■		■	■	■	■			■		■		■
Go Back One Step						■							■	■					■		■
Go Back to a Safe Place						■		■					■	■					■		■
Convenient Environment Actions	■		■	■								■	■				■				■
Progress Indicator				■				■					■	■							■
Interaction History						■		■	■			■					■				■
Map of Navigable Spaces	■		■		■		■	■				■					■				■
Optional Detail On Demand		■					■	■	■	■	■		■	■							■
Disabled Irrelevant Things	■			■			■	■	■				■								■
Good Defaults	■			■		■		■				■					■				■

En la propuesta metodológica utilizada se estaría considerando todo lo comentado, es decir, las diferentes vistas que puede ofrecer un patrón en función de a quién pertenezcan los ojos que lo miren (usuario e ingeniero) y de a qué secciones se preste más atención.

7.2.3 Resultados de la fase de análisis de requisitos

Se abandona esta fase inicial de análisis de requisitos con una colección de patrones que aportan un modelo de calidad y que han permitido complementar otras técnicas con las que lograr identificar requisitos funcionales y no explícitamente funcionales. En esta fase se han utilizado técnicas procedentes de la Ingeniería del Software como son los diagramas

de casos de uso, pero también experiencia, una experiencia fundamentalmente proveniente del campo de la Interacción Persona-Ordenador.

Debido a las distintas vistas que pueden recogerse en la documentación de los patrones, en esta fase se dispone de información textual, de información relacionada con la calidad y de especificaciones de la solución aportadas por el patrón en forma de modelos, lo que permitirá no partir de cero en la fase de diseño abstracto posterior a esta fase. Dichas especificaciones estarán, según esta tesis doctoral, orientadas a la tarea y, por tanto, lo que se logra con los patrones de interacción documentados es disponer de unos modelos que indican qué requisitos funcionales y no explícitamente funcionales están asociados a la solución ligada a cada patrón. En este sentido, en la Fig. 7-6 se muestra el modelo que se ha asociado al patrón principal (*Step-by-Step Instruction*) del que se ha partido para abordar la solución al caso de uso "Realizar pedido" (asociado con la formalización de la compra realizada por el cliente).

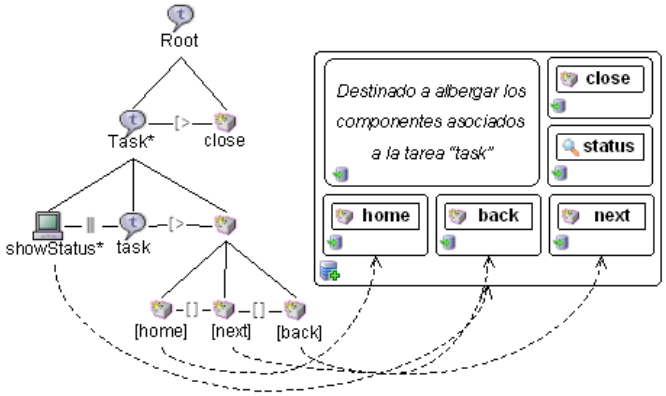


Figura 7-6. Patrón de interacción *Step-by-Step Instruction*

En la Figura 7-6 se han considerado especialmente las actividades colaterales asociadas a la realización secuencial de una serie de tareas que persiguen un objetivo. La especificación de dichas tareas estaría localizada en el lugar marcado por la tarea abstracta *task* y se acompañarían de otras tareas colaterales que redundarían en la calidad de la interacción que el usuario puede experimentar al hacer uso del producto software, y que permiten, en primera instancia, volver a un lugar seguro (*home*) o avanzar (*next*) y retroceder (*back*) respecto del estado en el que se encuentre. Se representan inicialmente, aunque se abundará más adelante en los mapping que pueden identificarse entre el análisis de tareas propuesto y el modelo de presentación especificado a nivel abstracto.

7.3 Fase de diseño abstracto

Fruto de la información disponible, una vez se ha alcanzado esta fase de diseño abstracto, se dispone de un conjunto de requisitos funcionales y, junto con ellos, de un conjunto de requisitos no explícitamente funcionales que deben tenerse en cuenta ya que redundan en el uso que el usuario hará del producto software. Los iconos que representan esta información, que se obtiene a medio camino entre la fase anterior y ésta, son los recogidos a lo largo de la sección anterior y se resumen seguidamente:

Análisis de requisitos				Inicio de la fase de diseño
				
Requisitos funcionales especificados textualmente	Patrones de interacción (textual, gráfico y modelo)	Mapping establecido empíricamente entre experiencia y calidad	Requisitos no explícitamente funcionales relacionados con la calidad	Modelado del dominio y de la presentación

De igual forma cabe reseñar, antes de empezar con el desarrollo de esta otra fase, que en la anterior no solamente se ha identificado el patrón *Step-by-Step Instructions*, sino que también se considerarán en este ejemplo los patrones *Form*, ligado al ofrecimiento de formularios al usuario, y el patrón *High-density Information Display*. Ambos patrones, como ocurría con el patrón considerado hasta el momento, mantienen relaciones con otros patrones formando sublenguajes, concretamente éstos son lenguajes específicos orientados a la elaboración de formularios y a la visualización de información respectivamente. Dichos lenguajes están recogidos en la documentación de los patrones (Tidwell, 1999) y se han resumido en la Tabla 7-3.

Con los patrones mencionados se podrán elaborar los distintos contextos de uso que se contemplan en el caso de estudio que se está utilizando y en el caso de uso concreto "Realizar pedido" que se ha elegido para acotar la complejidad del desarrollo, hacer el ejemplo manejable y poder mostrar la viabilidad de aplicación de la propuesta metodológica asociada con esta tesis doctoral. De esta forma, la realización de un pedido conllevará seleccionar una serie de artículos de entre un conjunto disponible (ofertados en un contexto de uso caracterizado por el patrón *High-Density Information Display*) y la formalización de la compra consistirá en proporcionar información que permita identificar al cliente, su forma de pago y la dirección a la que enviar el pedido realizado.

Tabla 7-3. Patrones considerados en el ejemplo y relaciones con otros patrones

Patrón principal	Patrones Relacionados
Form	Choice from a Small Set, Choice from a Large Set, Editable Collection, Sliding Scale, Forgiving Text Entry, Structured Text Entry, Good Defaults, Remembered State, Step-by-Step Instructions, Small Groups of Related Things, Disabled Irrelevant Things, Pointer Shows Affordance, Optional Detail On Demand
High-density Information Display	Series of Small Multiples, Hierarchical Set, Tabular Set, Chart or Graph, Navigable Spaces, Small Groups of Related Things, Optional Detail On Demand, Disabled Irrelevant Things, Short Description, User's Annotations

En esta fase de diseño abstracto, con la información disponible de la fase anterior, se realiza el diseño conceptual y dentro de esa tarea se incluye el modelado de la presentación. Las transformaciones a nivel de diseño llevadas a cabo entre especificaciones ligadas a la fase de análisis y los modelos de la fase de diseño se apoyan en la experiencia disponible en los patrones, tanto de colaboración (Nicola et al., 2001) como de diseño (Gamma et al., 1995). La experiencia disponible en dichos patrones es la utilizada como herramienta para llevar a cabo el modelado de dominio en sus vertientes estática y dinámica. En lo que respecta a la especificación de las tareas seguirá teniéndose presente la experiencia recopilada en los patrones de interacción considerados (Tidwell, 1999).

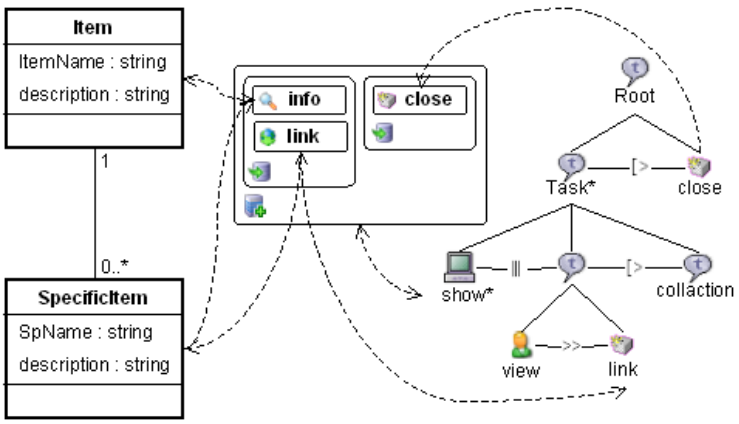


Figura 7-7. Especificación del patrón de interacción *Form*

Hasta ahora el enfoque principal ha estado orientado a las tareas más que al dominio, sin embargo, muchos de los patrones de interacción referidos hasta el momento consideran, en su especificación, entidades pertenecientes al dominio, especificaciones de tareas y de la presentación, como

ejemplo de ello considérese el ejemplo el patrón *Form* (Tidwell, 1999) mostrado en la Fig. 7-7.

Las especificaciones asociadas a los patrones de interacción sirven en esta fase para obtener una especificación conceptual lo más fiel posible al problema que se desea resolver. Pero no todos los problemas y situaciones están recogidos en los patrones de interacción, se presentan habitualmente casos en los que la utilización de otros patrones puede resultar interesante, esto se verá seguidamente en las distintas secciones dedicadas al modelado del dominio, las tareas y la presentación.

7.3.1 El modelo de dominio

La propuesta metodológica que se presenta en esta tesis doctoral está orientada a la tarea, pero no se puede prescindir o menospreciar la labor que supone modelar el dominio que se manipula e involucra la realización de las tareas identificadas. La tarea principal que hemos seleccionado en el caso de estudio considerado en este capítulo está relacionada con la formalización de un pedido. Dicha tarea involucra trabajar con una serie de entidades que permiten gestionar pedidos y manipular la información involucrada en ellos. Para la identificación de las entidades, el modelado de las mismas y de sus relaciones se recurre en esta tesis doctoral a la experiencia, concretamente a los patrones de colaboración que se recogen en el Apéndice A. Los patrones de colaboración ya han sido comentados en los capítulos anteriores cabe prestar atención a la columna, en el Apéndice A, en la que se reseñan diferentes ejemplos relacionados con los patrones de colaboración.

Los mencionados ejemplos ayudan al ingeniero sin experiencia a identificar entidades y relaciones entre ellas que son similares a las que él necesita para modelar su problema, o a extrapolar a partir de las entidades y relaciones identificadas otras similares que mantienen otras entidades implicadas en el modelado de la solución a un problema diferente al que se le presenta al ingeniero. Los patrones de colaboración consideran personas, cosas, lugares y eventos y estas entidades generales serán las que deberemos identificar en nuestro caso de estudio, en el caso de uso y en el contexto que nos sirve de excusa para poner en práctica la propuesta metodológica asociada con esta tesis doctoral. Las personas que identificamos desempeñarán el papel de clientes y las tareas que realizarán con el sistema estarán determinadas por transacciones que persiguen la adquisición de un pedido constituido por diferentes artículos ofertados.

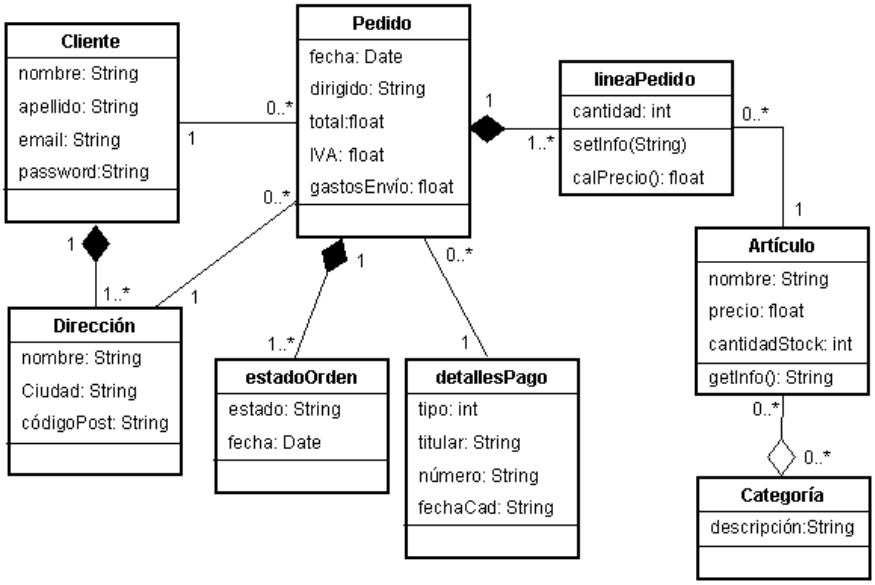


Figura 7-8 Diagrama de clases asociado a un sistema de compra electrónica

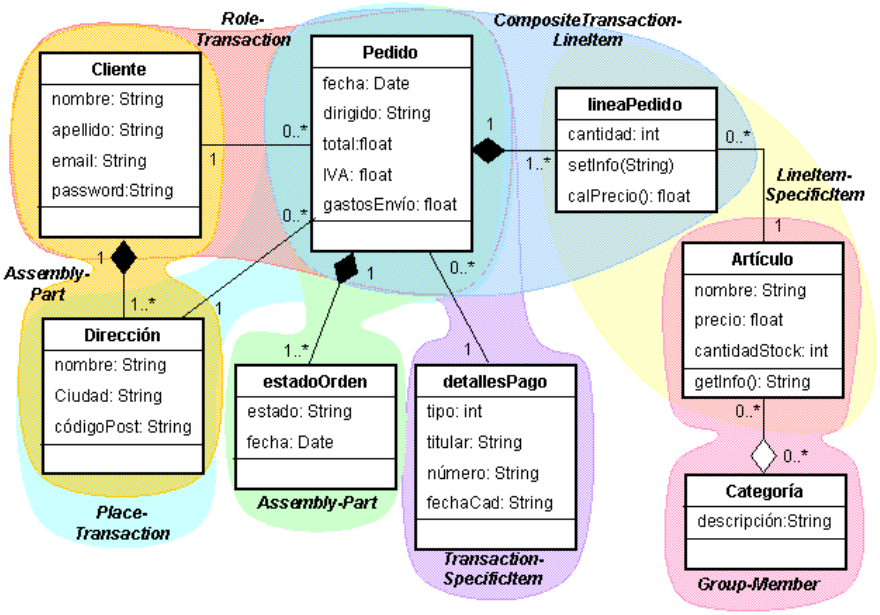


Figura 7-9. Patrones de colaboración utilizados para la elaboración del diagrama de clases asociado al caso de estudio

Con los patrones de colaboración mostrados en el Apéndice A pueden identificarse las entidades necesarias para abordar tareas de carácter transaccional y las relaciones que éstas mantienen entre ellas. Además, como se comentó anteriormente, el ingeniero sin experiencia puede extrapolar, utilizando los ejemplos que acompañan a los patrones de colaboración, la información que esos patrones documentan, adaptándola a la situación particular que se le presente. Con todo ello, el ingeniero puede obtener un modelo, plasmado en forma de diagrama de clases, donde quedan recogidas las entidades, las relaciones entre ellas y las características estáticas que caracterizan a dichas entidades.

Como ejemplo de uso de los patrones de colaboración, en la Fig. 7-8 se muestra el diagrama de clases asociado con las entidades necesarias para gestionar la realización de un pedido o compra electrónica. Posteriormente, en la Fig. 7-9 se muestra ese mismo diagrama de clases y se resaltan, utilizando áreas coloreadas, las instancias de patrones de colaboración utilizadas para sugerir las entidades utilizadas y las relaciones entre ellas.

7.3.2 El modelo de tareas y el de presentación

El hecho de tratar conjuntamente dos modelos como son el de tareas y el de presentación está justificado por la relación tan estrecha que mantienen y que pasamos seguidamente a justificar. Un vistazo rápido del recorrido seguido al poner en práctica el proceso metodológico propuesto hasta el punto en el que nos encontramos desvela una clara orientación a la tarea, fruto de la cual, en la fase de análisis, se optaba por la elaboración de los diagramas de casos de uso asociados al sistema y por la realización de un primer nivel de refinamiento basado en la aplicación del concepto de contexto de uso y la utilización del patrón fundamental para realizar un estudio más detallado de cada uno de los casos de uso identificados. La elaboración y especificación de los contextos de uso viene facilitada por la disponibilidad de patrones de interacción documentados utilizando para ello aquellas notaciones que se han identificado más adecuadas. En el caso de estudio que estamos manejando fundamentalmente se parte de tres patrones primarios ya mencionados como son el patrón *Step-By-Step Instructions*, el patrón *Form* y el patrón *High-Density Information Display*. Con ellos se pretende dar solución a las diferentes tareas de interacción que de manera general se han identificado en un producto software que quiere ofrecer facilidades para llevar a cabo labores de compra y venta electrónica. De esta manera el patrón *Step-By-Step Instructions* permitiría la especificación de aquellos procesos en los que, de forma guiada, el usuario aborda tareas rutinarias como puede ser la formalización de la compra de un

pedido de productos previamente seleccionados o tareas relacionadas con el mantenimiento del *stock* de productos, añadiendo nuevos productos al catálogo de productos ofrecidos. El patrón *Form* (Fig. 7-10) es útil para proporcionar formularios que el usuario debe utilizar para facilitar información al sistema, por ejemplo, el usuario tendrá la necesidad de identificarse y de proporcionar aquella información necesaria para formalizar sus compras y establecer la dirección de envío, la forma de pago o los datos del cliente.

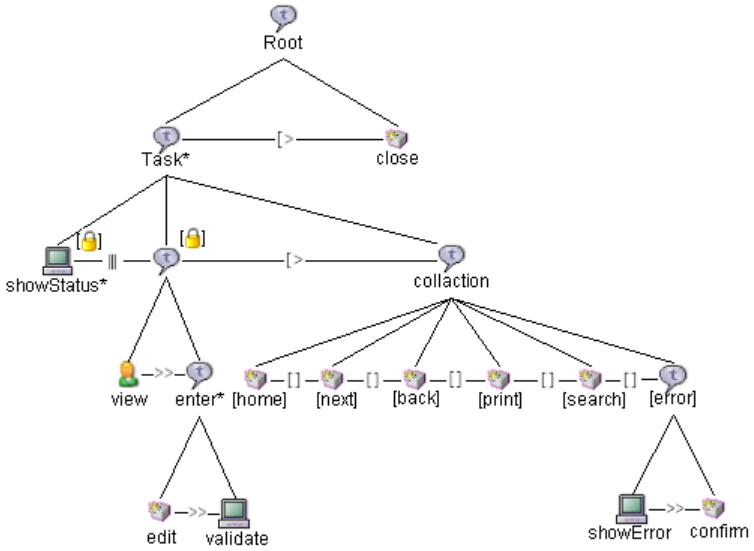


Figura 7-10. Análisis de tareas asociada al patrón *Form*

El último patrón mencionado, el *High-Density Information Display* (Fig. 7-11), está ideado para documentar la necesidad de tener que ofrecer al usuario información de diferente contenido y en la que hay relaciones de navegación que permiten obtener, por ejemplo, información adicional bajo demanda.

Las especificaciones mostradas en las figuras 7-10 y 7-11 son parciales ya que muestran únicamente la vista orientada a la tarea que puede ofrecer un patrón de interacción. Junto a ellas también tienen cabida consideraciones en el ámbito de la presentación y del dominio asociada a la tarea que se pretende realizar. Ambas dimensiones adicionales se documentan de forma general utilizando patrones allí donde es posible, por ejemplo en la especificación de las consideraciones relacionadas con el modelo de dominio se utilizan patrones de colaboración (Nicola et al., 2001).

Cabe también reseñar, a la vista de las especificaciones mostradas, que la bondad que ofrece la documentación asociada a cada patrón de interacción no está en la propia disponibilidad de las especificaciones incluidas,

aunque éstas suponen una asistencia digna de consideración para el ingeniero y una ventaja en sí misma, sino en la identificación que, paralelamente con la especificación de la tarea, se hace de las consideraciones relacionadas con la calidad (*collaction*; acciones colaterales) asociados con la solución que el patrón, a través de la puesta en práctica de la solución asociada, aporta. Dichas consideraciones de calidad están relacionadas y son coherentes con el modelo de calidad que se propuso en el capítulo tercero.

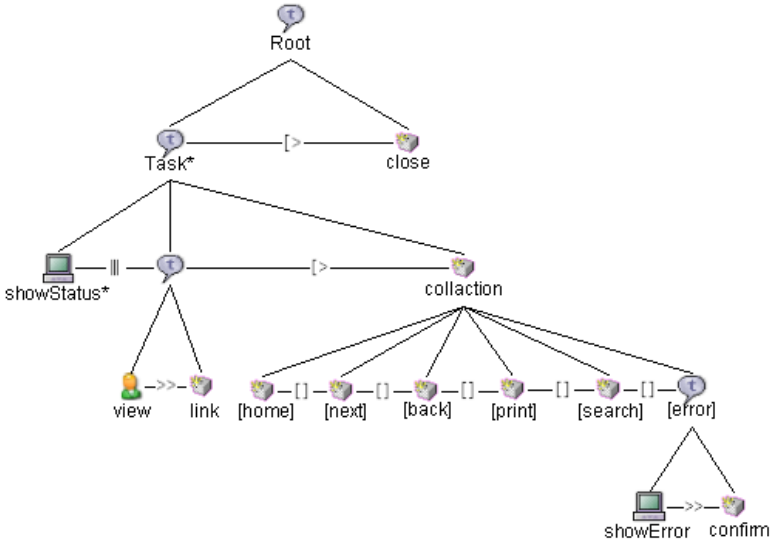


Figura 7-11. Análisis de tareas del patrón *High-Density Information Display*

En cualquier caso, la posibilidad de especificar características relacionadas con la calidad no queda restringida al apartado de las acciones colaterales, también puede recogerse a lo largo de la especificación. Como ejemplo de ello puede verse el patrón *Form* (Fig. 7-10), en él la información que el usuario introduce es validada y además se especifica el carácter sensible de la misma, la necesidad de un tratamiento seguro de la misma, de acuerdo con la legislación vigente en cada país.

Retomando el punto en el que nos encontramos correspondiente al tratamiento conjunto de tareas y presentación vemos que, con el uso del patrón fundamental como esqueleto para especificar un caso de uso o parte del mismo, siempre aparece una tarea relacionada con la presentación de la interfaz (showStatus), es por ello que, al menos, un prototipo de la misma deberá estar disponible, siendo está la causa por la que lo trataremos conjuntamente. Para la elaboración de los prototipos o especificaciones a nivel abstracto de la interfaz recurrimos, como se mencionó en el capítulo anterior, a la utilización del lenguaje de especificación abstracto propuesto en

usiXML (Limbourg et al., 2004), a su vez, basado en la propuesta de (Constantine, 2003) y al que en esta tesis doctoral se le ha asociado una notación gráfica.

Previamente a la elaboración de prototipos de interfaz, el caso de estudio merece una consideración más detallada, realizada a la luz de los contextos de uso y de los patrones de interacción considerados. Dichos contextos de uso están asociados con estados de ejecución utilizando otra experiencia, los patrones de diseño (Gamma et al., 1995). Estaríamos, con ello, ante el segundo nivel de refinamiento identificado en la propuesta metodológica, aquel por el cual se asocia uno o varios estados y con ellos se modela conceptualmente cada contexto de uso. Para llevar a cabo esta asociación se utilizan dos patrones de diseño, dichos patrones son el patrón *State* y el patrón *Composite*. El primero es un patrón de comportamiento que permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Ese mismo comportamiento es el que deseamos otorgar a la interfaz de usuario de un producto software o al propio producto software que modificará su aspecto en función de la tarea que este realizando y de las realizadas hasta ese momento. El otro patrón de diseño mencionado, el *Composite*, es un patrón estructural con el cual es posible organizar objetos en estructuras de árbol para representar jerarquías de parte-todo, algo que ocurrirá habitualmente con las interfaces de usuario ofrecidas a lo largo de un asistente, las cuales contando con un estado propio que las representa, también pertenecerán a un conjunto de estados que determinan el estado que las aglutina.

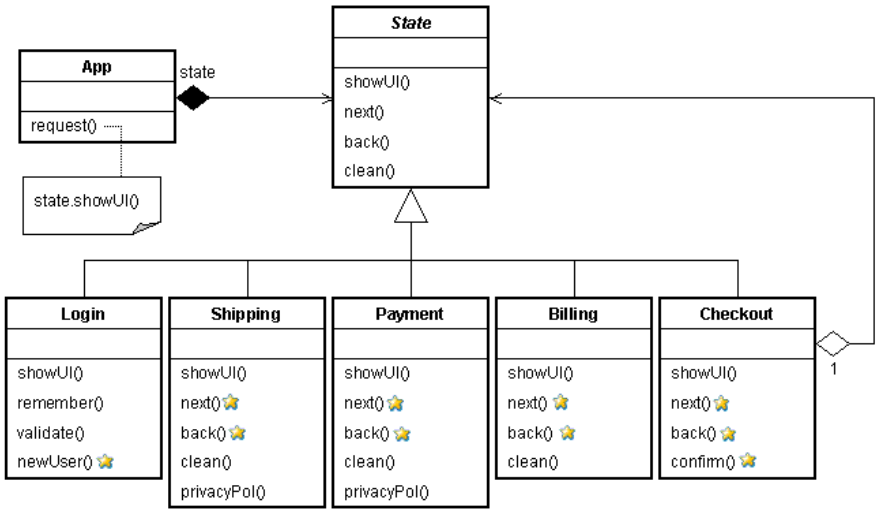


Figura 7-12. Segundo nivel de refinamiento (de contextos de uso a estados)

La Fig. 7-12 muestra gráficamente el resultado de aplicar los patrones de diseño *State* y *Composite* a la representación de los diferentes contextos de uso identificados para abordar el caso de estudio relacionado con la formalización de una compra (*checkout*) que estamos tratando.

La mencionada tarea se puede llevar a cabo de dos formas dependiendo de que el usuario esté registrado en el sistema o no lo esté. Si el usuario está registrado sólo tendrá que identificarse, introduciendo el identificador y la contraseña proporcionada, y la compra estará formalizada, ya que los datos del cliente están disponibles en la base de datos destinada al efecto. Si, por el contrario, el cliente no ha facilitado información alguna sobre él previamente, fruto de no haber hecho una compra anterior, y no se encuentra registrado, deberá facilitar aquella información que permita el pago y el envío del producto a la dirección elegida. Este proceso puede ofrecerse a través de una serie de tareas en las que el usuario facilitará secuencialmente aquella información que sea necesaria. Por ejemplo, y en primer lugar, el cliente deberá introducir aquella información que lo identifique (nombre, apellido, dirección completa, teléfono de contacto, etcétera), después, el cliente proporcionará información relacionada con la forma de pago (tipo de tarjeta de crédito, número de la misma, fecha de caducidad, etcétera), y finalmente, el cliente introducirá la información relacionada con la dirección de envío en caso de que no sea la dirección habitual en la que reside el cliente y que ya fue proporcionada al identificarse. Cada una de estas tareas, consistentes en esencia en proporcionar información, se abordan a través del uso de formularios asociados y compuestos de aquellos campos necesarios para editar la información requerida.

Así, asociado a cada caso de uso tendremos una serie de contextos de uso posibles y cada uno de los contextos de uso, por un segundo nivel de refinamiento estará ligado a un estado que conceptualiza su comportamiento. Cada estado ofrecerá una vista al usuario y dicha vista será especificada a través de prototipos. Dichos prototipos mostrarán aquellos objetos de interacción a nivel abstracto que permitan facilitar la realización de las actividades de interacción asociadas a cada estado. En dichas actividades de interacción están también incluidas aquellas no explícitamente funcionales que redundan en la calidad y son una contribución destacable de los patrones de interacción, para ellas también se dispondrán objetos de interacción que permitan su invocación. Fruto de las relaciones de herencia entre estados y de la posibilidad de disponer de clases abstractas asociadas a esos mismos estados las presentaciones ofrecidas podrán beneficiarse de ese tipo de relaciones establecidas a nivel de entidades. De esta forma, una presentación establecida en un nivel determinado de la jerarquía de clases especificada podrá heredarse por aquellos estados situados por debajo de aquel y completar, de este modelo la especificación realizada a partir de

ese momento. Es decir, a nivel más alto en la jerarquía se establecerán *Containers* que irán recopilando en su interior otros *Container* con *Components* o directamente *Components* que ofrecerán facilidades de interacción a través de las facetas definidas en su interior.

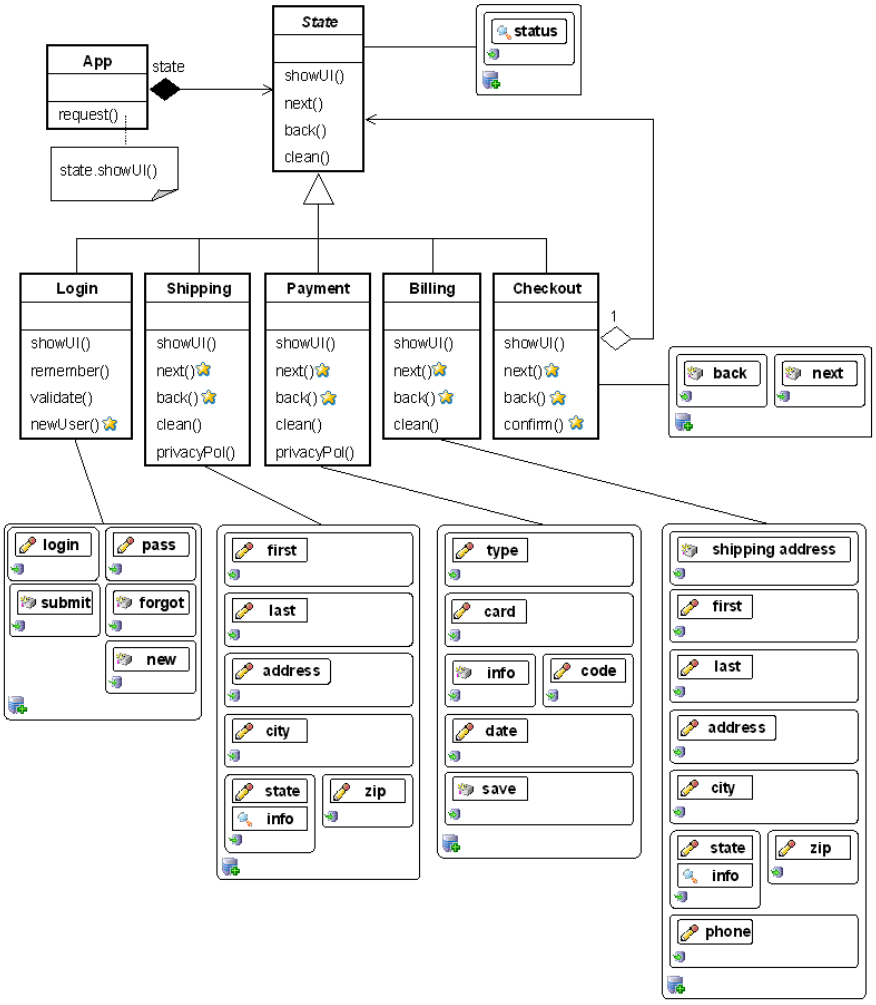


Figura 7-13. Asociación entre estado y presentación abstracta

En la Fig. 7-13 se muestra la asociación no caracterizada[1] entre los estados, derivados de los contextos de uso identificados, y la presentación de-

[1] (la caracterización de las asociaciones entre especificaciones será tratada en un tercer nivel de refinamiento adicional al que se dedicará un apartado posterior)

finida para cada uno de ellos. El icono en el diagrama de clases asociado a los estados mostrado simboliza que una invocación al método asociado conllevará una *conmutación de contexto*, y el paso, por tanto, a la realización de una tarea diferente.

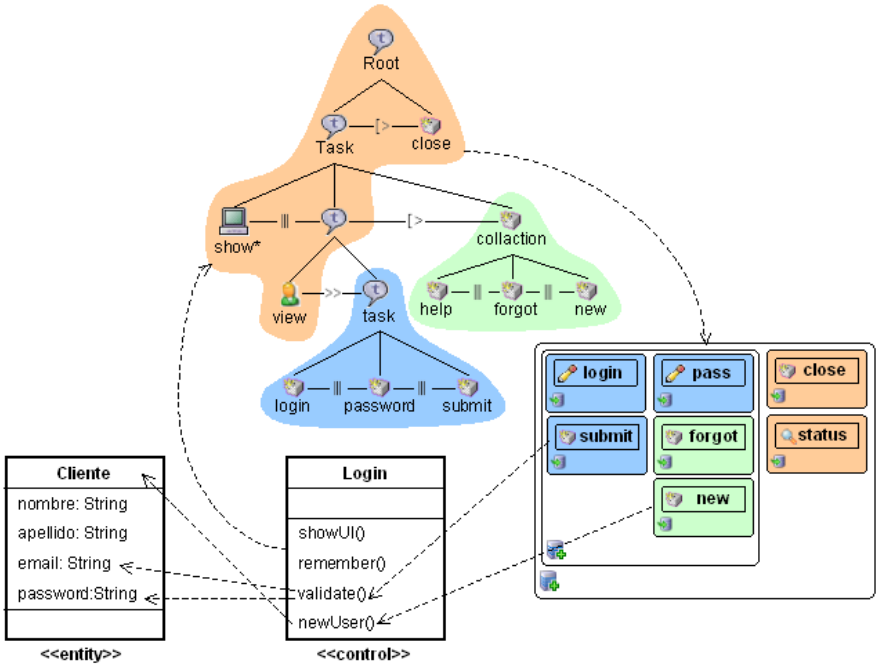


Figura 7-14. Presentación derivada de tareas y dominio para la tarea *login*

Si nos centramos en un contexto de uso concreto como es el estado *login* mostrado en la Fig.7-14, podremos ver el resultado de realizar el segundo refinamiento que se viene comentado. Inicialmente, partimos de un análisis de tareas elaborado utilizando la experiencia recopilada en forma de patrones. El usuario debe identificarse y para ello debe proporcionar un identificador y una contraseña. Esta información, en este caso se corresponde con su correo electrónico y con una contraseña facilitada previamente y conocida por el usuario y almacenada en el modelo de dominio. El patrón de interacción *Form* se utiliza para analizar la tarea *login* especificándose, mediante el uso de la notación CTT (Paternò, 1999), la información involucrada en la identificación y las acciones disponibles en ese contexto de uso para el usuario. Acciones y datos relacionados con el contexto pasan a formar parte de una entidad encargada de controlar el contexto de uso y de determinar su presentación. La presentación se elabora en base a la tarea de facilitar la invocación a las tareas identificadas para ese contexto de uso. Muchas de las tareas identificadas para cada contexto de uso o

estado están directamente relacionadas con la calidad que la interfaz ofrece, por ejemplo y en el caso del *login*, ofrecer mecanismos para que el sistema recuerde la contraseña al usuario en caso de que éste la haya olvidado, o bien que dándose la situación de que el usuario no se encuentre registrado den la posibilidad de abandonar el contexto de uso actual y pasar a crear una nueva cuenta en el sistema. Esta funcionalidad adicional está documentada en los patrones de interacción y es accesible desde el patrón *Form* haciendo uso de las relaciones definidas entre patrones y explorando el sublenguaje de patrones accesible desde cada patrón seleccionado. En el caso que nos ocupa los patrones que aconsejan el ofrecimiento de la funcionalidad comentada son los patrones de interacción *Good Defaults*, *Optional Detail on Demand* o *Forgiving Text Entry*.

7.3.3 El modelo de mapping

Alcanzado este punto de la fase de diseño, asociado a la propuesta metodológica presentada en esta tesis doctoral, disponemos de diferentes modelos y especificaciones. Tenemos, por una lado, un modelo de dominio independiente de la plataforma y en el que quedan recogidas aquellas entidades involucradas en el problema objeto de estudio, al que se le quiere ofrecer una solución en forma de producto software. Al nivel de tareas, atrás quedaron los diagramas de casos de uso elaborados inicialmente, y a los que se sometió a dos fases de refinamiento con ayuda de la experiencia disponible. Fruto de dicho refinamiento se identificaron, en la fase de análisis, los así llamados, contextos de uso y, más tarde (en la fase de diseño abstracto), a éstos se les asoció un estado para modelarlos, utilizando patrones de diseño. Además, cada uno de estos estados ofrecía una vista que también ha sido modelada utilizando el lenguaje de especificación de interfaces a nivel abstracto definido en usiXML. Pero, las asociaciones que pueden establecerse no se limitan a estos estados ligados a contextos de uso y a sus correspondientes presentaciones, existen otros mappings entre elementos de especificación y entre las propias especificaciones.

Las relaciones de asociación que consideramos fueron recogidas en el capítulo anterior y gracias a su establecimiento se facilita en la fase de implementación la realización de transformaciones entre especificaciones y se redunda en la trazabilidad del proceso de desarrollo seguido.

Un patrón arquitectónico como es el Modelo-Vista-Control determina las principales relaciones entre los elementos que constituyen la especificación llevada a cabo hasta el momento. La parte dedicada a la interacción viene determinada fundamentalmente por las entidades dedicadas a la vista y al control. En función de esto, las relaciones que pueden definirse entre

las tres entidades son las definidas en el patrón MVC, que se muestran en la Fig. 7-15.

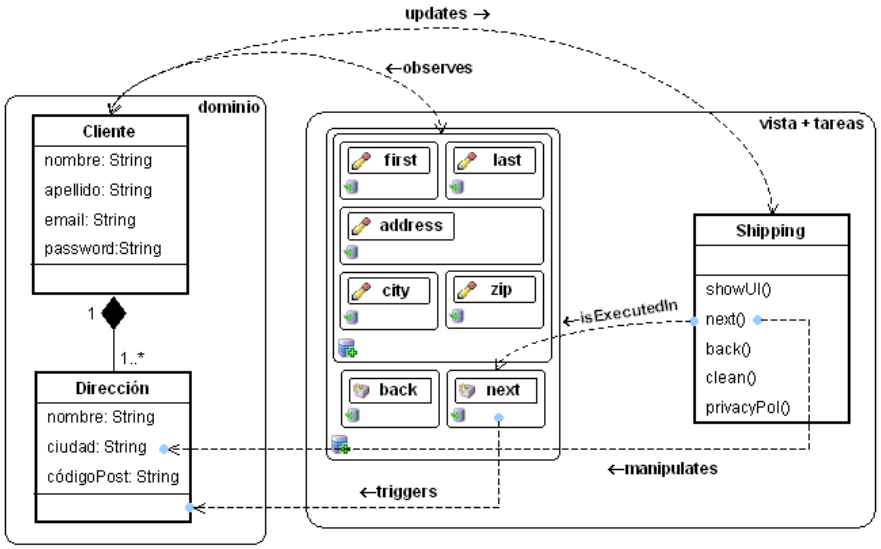


Figura 7-15. Asociaciones identificadas entre los distintos modelos considerados

Las relaciones existentes entre las especificaciones elaboradas hasta este momento se establecen, principalmente, entre el modelo de dominio y los modelos de vista y control, como muestra la Fig. 7-15. Entre ellas, cabe destacar las relaciones *updates*, *observes*, *triggers*, *manipulates* y *isExecutedIn*. Las relaciones updates y observes garantizan la coherencia de los datos almacenados en el modelo de dominio, mostrados en la vista y manipulados en el control, de tal forma que la vista y el control observan al dominio y el modelo de dominio notifica las modificaciones producidas en su estado a las anteriores, propiciando la actualización de aquellos objetos de interacción que muestran los datos almacenados en el dominio. Este par de relaciones responde al patrón de diseño *Observer* (Gamma et al., 1995) y será con su utilización con el que se diseñará la respuesta al establecimiento de estas relaciones.

El usuario a través de la interacción con los objetos de interacción ofrecidos podrá *disparar* (*triggers*) la ejecución de determinadas tareas, accediendo a los métodos almacenados en el dominio. Dichas tareas serán *ejecutadas* (*isExecutedIn*) por el control a través de la interacción que el usuario mantiene con los objetos de interacción facilitados. Dichos procedimientos del control manipularán (*manipulates*), a su vez, el estado de los objetos que conforman el dominio.

La identificación de estas relaciones facilita la trazabilidad de la especificación y la realización de transformaciones en la fase de implementación

automática basada en reglas que se realiza posteriormente. IDEALXML permite la especificación de los mappings establecidos entre las especificaciones realizadas.

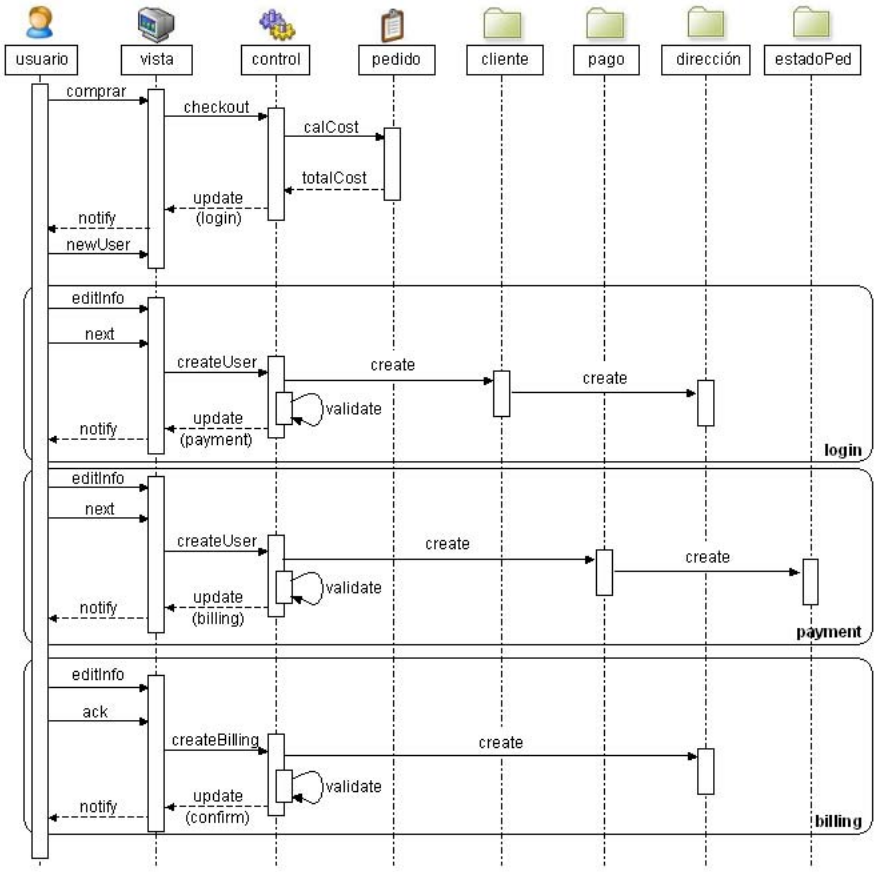


Figura 7-16. Diagrama de secuencia asociado al proceso de *checkout*

Con la información asociada a estos mappings se podrá abordar el tercer nivel de refinamiento que centra la atención del ingeniero en los métodos identificados. Para ello se elaborarán diagramas de secuencia, colaboración y de estados, en función de la complejidad de los métodos identificados, que permitirán verificar y completar la especificación conceptual. La Fig. 7-16 muestra el diagrama de secuencia asociado al proceso de *checkout* llevado a cabo por el cliente una vez desea formalizar la compra de su pedido.

7.3.4 Resultados de la fase de diseño abstracto

De la realización de la fase de diseño abstracto se obtiene una especificación conceptual completa de las entidades de control y de dominio, una especificación abstracta de la interfaz, elaborada utilizando objetos de interacción abstractos que podrán adecuarse a diferentes modos de interacción y plataformas con las que opere el usuario, y una identificación de las relaciones identificadas entre los elementos que conforman todas estas especificaciones (Tabla 7-4).

En esta fase se ha utilizado IDEALXML para realizar aquellas especificaciones en las que ha tenido cabida la consideración de la experiencia y de la calidad.

Tabla 7-4. Iconografía que representa los resultados de la fase de diseño abstracto

Fase de diseño abstracto			
			
Modelado del dominio y de la presentación teniendo en cuenta la calidad	Experiencia disponible en distintos ámbitos (dominio, diseño e interacción)	Especificación conceptual completa (dominio, tareas y presentación abstracta)	Nueva experiencia identificada y documentada

Fruto de la disponibilidad del entorno IDEALXML es posible documentar aquella experiencia de diseño que se descubra o recopile desarrollando un producto software (Tabla 7-4). De esta forma, dicha experiencia estará disponible para futuros diseños.

Una característica que no se ha tenido en cuenta, al estar trabajando con especificaciones abstractas de la interfaz de usuario, es la relacionada con el diseño y la disposición final de los objetos de interacción. Dicha disposición influye también en la calidad y la usabilidad del producto software finalmente producido. Estas consideraciones de diseño deberán tenerse presentes en la fase de implementación cuando se aborde la transformación de los objetos de interacción abstractos y se obtengan los objetos de interacción concretos para una plataforma y modo de interacción determinados.

7.4 Fase de implementación

La tarea que fundamentalmente se aborda en la fase de implementación, teniendo en cuenta la consideración de la calidad en la interacción y la re-

utilización de la experiencia presentes en esta tesis doctoral, pasa por la obtención de la interfaz de usuario concreta que ofrecerá el producto software desarrollado. La misma se obtendrá a partir de la especificación abstracta de la presentación previamente elaborada.

En esta fase tendrán cabida diferentes transformaciones de modelos, unas conllevarán transformaciones realizadas entre modelos y otras serán meras transformaciones de modelo a texto. Los iconos que reflejarán estas transformaciones y otros elementos característicos de esta fase están recogidos en la Tabla 7-5.

Tabla 7-5. Iconografía que representa los resultados de la fase de implementación

Fase de Implementación			
			
Transformación de modelos (modelo-modelo)	Transformación de modelos (modelo-texto)	Especificación concreta de la interfaz	Producto software finalmente obtenido

A esta fase de implementación se llegará con una especificación conceptual completa y una especificación abstracta de la presentación donde dos elementos (*containers* y *components*) están presentes. A su vez, los *components* podrán ofrecer diferente funcionalidad y modo de interacción en función de las facetas definidas en su interior (*input*, *output*, *control* y *navigation*). Si la especificación abstracta es completa, y la misma se puede revisar utilizando y conociendo los patrones de interacción disponibles, los pasos a seguir pasarán por identificar la plataforma y el modo de interacción deseado.

En la Fig. 7-17 se recogen diferentes transformaciones entre las especificaciones abstractas y concretas de una presentación. Cada una de las presentaciones reflejadas se asocian con diferentes contextos de uso identificados en la anterior fase de diseño abstracto. Concretamente dichos contextos de uso son los asociados a las tareas de identificación (*login*), determinación del lugar y forma de entrega, especificación de la forma de pago y, finalmente, la identificación del cliente. Las especificaciones abstractas de la presentación se transforman utilizando equivalencias similares a las recogidas en el capítulo sexto (véanse las figuras 6-18 y 6-19).

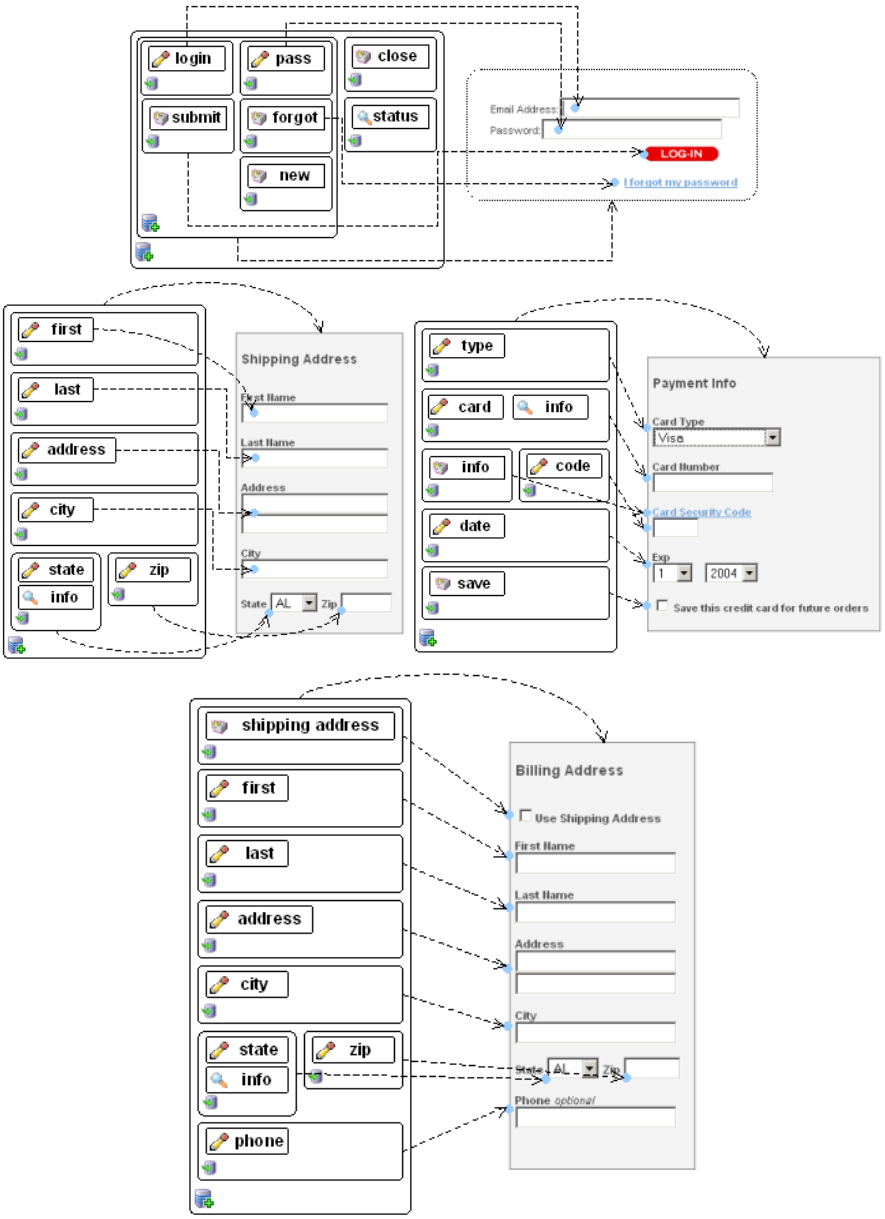


Figura 7-17. Obtención de la presentación concreta a partir de la especificación abstracta para (arriba) la tarea login, (centro) las tareas de introducción de la dirección de envío y forma de pago y (abajo) para la identificación del cliente

7.5 Posibilidad de integración con otras propuestas

La incorporación de la calidad y de la experiencia en el proceso de desarrollo de interfaces de usuario es la principal aportación que está detrás de esta tesis doctoral. Para ello, se ha presentado una propuesta metodológica que se ha puesto en práctica en este capítulo y que está soportada por la herramienta IDEALXML. Dicha propuesta conduce a una especificación conceptual y abstracta de la interfaz que no supone un producto software final directamente, pero que si que puede beneficiarse de otros trabajos, unos desarrollados y otros en desarrollo, en los que la propuesta presentada está integrada. En este sentido, esas otras propuestas pasan fundamentalmente por plasmarse en una serie de herramientas que están recogidas en el sitio web dedicado a usiXML (http://www.usixml.org) y donde puede consultarse información adicional sobre ellas. TransformiXML, KnowiXML, GrafiXML, VisiXML, SketchiXML, FormiXML, VisualiXML y ReversiXML son las denominaciones que reciben estas propuestas mencionadas y con las que puede integrarse IdealXML ya que comparten el mismo lenguaje que no es otro que el facilitado con usiXML (Limbourg et al., 2004).

Seguidamente haremos un repaso breve por cada una de las herramientas presentadas destacando su utilidad más relevante.

TransformiXML (Limbourg et al., 2004) es una aplicación con la que es posible definir, almacenar, manipular y ejecutar especificaciones realizadas utilizando gramáticas de grafos y con la que es posible realizar transformaciones de modelo a modelo.

KnowiXML (Furtado et al., 2004) consta de un sistema experto basado en Protégé (Noy et al., 2001) que automáticamente produce diferentes especificaciones abstractas a partir de modelos de dominio y de tareas elaborados considerando diferentes contextos.

GrafiXML (Limbourg et al., 2004) es una aplicación que permite editar especificaciones concretas de interfaz de usuario y del modelo de contexto, y también es capaz de generar automáticamente el código equivalente a la especificación realizada de la interfaz en HTML, XHTML, XUL y Java mediante una serie de plug-ins.

VisiXML es un plug-in elaborado para MS Visio con el que es posible especificar interfaces de usuario a nivel concreto y almacenarlos utilizando el lenguaje usiXML.

SketchiXML (Coyette et al., 2004) es una herramienta que permite hacer prototipos de interfaz de usuario considerando múltiples usuarios, plataformas y contextos de uso. Esta herramienta está implementada sobre un sistema de agentes como es Jack (Howden, et al., 2001).

FormiXML es un editor dedicado a la elaboración de formularios interactivos donde se dan facilidades para la reutilización de componentes. Los formularios generados pueden almacenarse en código Java.

VisualiXML (Schlee et al., 2004) permite la personalización y manipulación de interfaces de usuario producidos utilizando técnicas de programación generativas.

ReversiXML (Bouillon et al., 2004) obtiene las especificaciones concretas y abstractas de interfaces de usuario en usiXML relacionadas aquellas interfaces que aparecen en páginas web elaboradas utilizando HTML.

Como queda reflejado por las relaciones con las herramientas mencionadas, el hecho de compartir un mismo lenguaje ofrece muchas posibilidades de éxito y desarrollo futuro a la propuesta recogida en esta tesis doctoral. Una de los retos más reseñables pasa por la automatización de un proceso que en este capítulo se ha mostrado en una faceta inicial bastante manual.

7.6 Análisis y conclusiones

Una parte muy importante de la especificación y documentación que se utiliza habitualmente en la elaboración de muchos proyectos de desarrollo software responde a patrones identificables y, en muchos casos, establecidos. Dichos patrones pueden estar disponibles al ingeniero, y éste, sobre todo el ingeniero sin la suficiente experiencia, puede utilizarlos para especificar los modelos que precisan ser elaborados a la hora de desarrollar un producto software.

Junto a la experiencia va asociada una característica deseable y dependiente de ella, la calidad en sus diferentes dimensiones. Concretamente, al nivel de interacción, especialmente desde hace algunos años, hay disponible experiencia recopilada aunque su documentación, como ha quedado patente con el ejemplo recogido en este capítulo, puede mejorarse en distintas dimensiones. Por un lado puede potenciarse la relación entre experiencia y calidad, para dejar constancia de la relación que dicha experiencia tiene con el logro de determinadas características de calidad y, por otro, puede incrementarse la potencia generativa de la experiencia documentada incluyendo especificaciones de la solución para facilitar la labor de especificación que debe realizar el ingeniero y hacer esa labor utilizando técnicas de modelado.

Fruto de la consideración de la calidad y de la experiencia desde las fases iniciales, especialmente en la fase de análisis de requisitos, al alcanzar-

se las fases posteriores éstas podrán abordarse con mayores garantías de éxito.

7.7 Contribuciones relacionadas con este capítulo

Las contribuciones relacionadas con los contenidos recogidos en este capítulo han sido las siguientes:

- **Montero, F**., López-Jaquero, V., Molina, J.P., Lozano, M. Improving e-shops environments by usign usability patterns. 2nd workshop on software and usability cross-polination. The role of usability patterns. September, 1-2, 2003, Zürich, Switzerland. 2003
- **Montero, F.**, Lozano, M., González, P. IDEALXML: an Experience-Based Environment for User Interface Design and pattern manipulation. Technical Report DIAB-05-01-4. Universidad de Castilla-La Mancha, Albacete (2005).
- García, F. J., Lozano, M., **Montero, F**., Gallud, J.A., González, P. A Controlled Experiment for Measuring the usability of webapps using patterns. 7th International Conference on Enterprise Information Systems (ICEIS, 2005). 24-28 May, Miami, USA. 2005.
- García, F.J., Lozano, M., **Montero, F.**, Gallud, J.A., Ruiz, V. Survey on Quality Models to Measure the Quality of Web Sites and Applications. 11th International Conference on Human-Computer Interaction (HCI, 2005). 22-27 July, 2005, Las Vegas, Nevada, USA. 2005.

Se incluyen aquellas publicaciones que están surgiendo en el terreno de aplicación práctica de la propuesta asociada a esta tesis doctoral y que se constituirán en previsibles nuevos casos de estudio, que se encuadran dentro de un trabajo por desarrollar y futuro de la línea de investigación asociada con esta tesis doctoral.

- Molina, J.P., Vanderdonckt, J., **Montero, F.**, González, P. Towards virtualization of user interfaces based on UsiXML. Web3D 2005: 169-178. 10th International Conference on 3D Web Technology, March 29 - April 1 2005.
- Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B. and **Montero, F**. A Transformational Approach for Multimodal Web User Interfaces based on UsiXML.The Seventh International Conference on Multimodal Interfaces (ICMI 2005). Trento, Italy, October 3-7, 2005 (to appear)

Referencias bibliográficas

Bouillon, L., Vanderdonckt, J., Chow, K.C.: Flexible Re-engineering of Web Sites; In: Proc. of 8th ACM Int. Conf. on Intelligent User Interfaces IUI'2004 (Funchal, 13-16 January 2004

Constantine, L.L. (2003): Canonical Abstract Prototypes for Abstract Visual and Interaction Design, Proc. of DSV-IS'2003, Springer-Verlag.

Furtado, E., Furtado, V., Soares Sousa, K., Vanderdonckt, J., Limbourg, Q., KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in UsiXML, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004),), Ph. Palanque, P. Slavik, M. Winckler (eds.), ACM Press, New York, 2004

Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns - Elements of Reusable Object-Oriented Software. Addison Wesley, 1995

Howden, N., Ronnquist, R., Hodgson, A. and Lucas, A.: JACK Intelligent Agents - Summary of an Agent Infrastructure. In Proc. 5th ACM Int. Conf. on Autonomous Agents, 2001

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-Jaquero, V., UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004)

Nicola, J., Mayfield, M., Abney, M., Abney, M. Streamlined Object Modelling: Patterns, Rules, and Implementation. Prentice Hall PTR; 1st edition (September 21, 2001)

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., Musen, M. A.: Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems 16(2):60-71, 2001

Schlee, M., Vanderdonckt, J.: Generative Programming of Graphical User Interfaces. In: Proc. of 7th Int. Working Conference on Advanced Visual Interfaces AVI'2004 (Gallipoli, May 25-28, 2004). ACM Press, New York (2004)

Paternò, F. Model-based Design and Evaluation of Interactive Applications. Springer Verlag, November 1999, ISBN 1-85233-155-0.

Tidwell, J. Interaction Design Patterns (aka. Common Ground). http://www.mit.edu/%7Ejtidwell/interaction_patterns.html. 1999.

Vanderdonckt, J., A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005), O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31

Capítulo 8 Conclusiones y trabajo futuro

La vida es el arte de sacar conclusiones suficientes
a partir de datos insuficientes.
Samuel Butler (Poeta ingles)

Me gustan más los sueños del futuro que la historia del pasado.
Thomas Jefferson (Político estadounidense)

8.1 Introducción

Finalizada la presentación de la propuesta asociada a esta tesis doctoral y de un caso de estudio donde se ha aplicado la misma llega el momento de recopilar las conclusiones alcanzadas fruto del trabajo desarrollado, así como de esbozar posibles caminos que se han ido identificando durante su puesta en práctica y que dan lugar a nuevas direcciones de investigación y desarrollo ligadas a la tesis doctoral presentada.

8.2 Reflexiones y conclusiones

En esta tesis dos conceptos han centrado el interés desde prácticamente el principio de este documento. Estos conceptos han sido la usabilidad y la experiencia. Ambos conceptos, a su vez, se han considerado desde dos disciplinas, como son la Ingeniería del Software y la Interacción Persona-Ordenador (véase el capítulo dos).

La **usabilidad** está incluida dentro de otro concepto más general y envolvente como es la calidad, pero dicha cualidad no se encuentra caracterizada plenamente y esto supone un serio problema a la hora de su consecución. La comunidad relacionada con la Ingeniería del Software está, en estos momentos, implicada en el reto que supone *industrializar* el desarrollo de productos software de calidad, pero la calidad que fundamentalmente considera es la que tiene que ver con el mantenimiento. En Interacción Persona-Ordenador, sin embargo, la usabilidad es un concepto clave y al que se presta especial atención proponiéndose metodologías y técnicas diversas para su consecución.

En esta tesis doctoral se identifica un punto en común en las tendencias de consideración de la usabilidad en una y otra disciplina. Este punto en común pasa por la **búsqueda de características de calidad relacionadas**

con la usabilidad en la fase de requisitos, y no considera la calidad un elemento, directa y exclusivamente, ligado a la presentación de acciones o de información que necesariamente se realiza en todo producto software.

La **usabilidad no es sólo presentación**, y así se deriva de las múltiples definiciones disponibles (véase el capítulo tercero) y de las propuestas elaboradas para su logro. Una componente fundamental es la satisfacción del usuario, pero ésta ha quedado fuera de un tratamiento central en esta tesis doctoral. En esta tesis se ha preferido apostar por la identificación y caracterización de la usabilidad considerando dos dimensiones. La primera de ellas y desde el punto de vista del usuario, considera la usabilidad como la disponibilidad de facilidades adicionales con las que contribuir al desarrollo de sus objetivos funcionales principales y que redudan en un valor añadido respecto de la actividad que el usuario puede llevar a cabo y de cómo esa actividad se lleva a cabo. En segundo lugar, y considerando el punto de vista del ingeniero, en esta tesis doctoral se establece que la usabilidad debe considerarse especialmente en las primeras fases de elaboración de un producto software, es decir, debe identificarse conjuntamente con los requisitos que el usuario desee realizar, y se traduce en el ofrecimiento al usuario de funcionalidad adicional que facilita la ejecución de los requisitos funcionales.

Para conseguir los propósitos anteriores hemos recurrido a la utilización de la **experiencia** (véase el capítulo cuarto). La misma, especialmente la disponible y documentada utilizando **patrones de interacción**, puede organizarse atendiendo a criterios de calidad relacionados con la usabilidad. De esta forma experiencia y calidad se consideran conjuntamente dentro de un **modelo de calidad**, que está elaborado utilizando experiencia y criterios de calidad de componente ergonómica (véase el capítulo quinto). Fruto de dicha organización y **documentando los patrones de forma generativa**, mediante el uso de más experiencia (patrones de diseño, patrones de colaboración, etc.), y apoyándonos en la actual tendencia de desarrollo de software; el **desarrollo dirigido por modelos,** es posible disponer de repositorios de experiencia reutilizable, caracterizada con modelos y disponible en su versión tradicional, es decir documentada utilizando lenguaje natural. Esa doble disponibilidad de la experiencia permite que sea utilizada tanto por el usuario final como por el equipo de ingenieros, con ello se facilita el proceso de comunicación que se orienta al patrón. De esta forma, se actúa sobre las fases iniciales del desarrollo del software permitiendo: (1) identificar requisitos de calidad, y (2) dar soporte al diseño y, en las fases posteriores, (3) dar soporte a la evaluación y al mantenimiento de la propia aplicación. En este sentido, la experiencia sería el verdadero factor que permitiría disminuir el gap semántico entre requisitos y diseño, autén-

tico caballo de batalla, en estos momentos, a la hora de desarrollar un producto software que considere la calidad como la entiende el usuario.

Las **aportaciones principales** de esta tesis doctoral están directamente relacionadas con los objetivos y resultados recogidos en el capítulo primero. Las mismas se resumen en el establecimiento y caracterización de la usabilidad a través de la recopilación de aquella experiencia que nos permite tener presente qué espera el usuario y cómo puede ofrecérselo el ingeniero. En función de ello se ha conseguido:

- **Caracterizar la calidad y definición de un modelo de calidad.** Dicha caracterización y modelo están elaborados, en primera instancia, en base a criterios de calidad, y, después, utilizando la experiencia disponible. De esta forma **experiencia y calidad están integradas a través del modelo de calidad** en el proceso de desarrollo y además lo hacen desde el primer momento. Una experiencia que deberá ser completada, como es menester en todo modelo de calidad que se precie, con métricas que permitan ponderár cuán bueno o malo es utilizar en cada momento la experiencia recopilada y asociada al criterio de calidad de que se trate. La disponibilidad de experiencia frente a métricas permite, en las primeras fases, detectar qué falla y cómo tratar el fallo. La métrica permite identificar que algo falla, pero no tanto qué o, especialmente, cómo mejorarlo.

- La caracterización de la calidad mediante experiencia permite **potenciar las metodologías que defienden un proceso centrado en el usuario**, disponibles desde la Interacción Persona-Ordenador. Esto se debe a que a todos los implicados en el proceso de desarrollo se les ofrece una herramienta, como es un **lenguaje común**, con el que es posible facilitar la comunicación entre ellos y permitir la evaluación de los productos finalmente elaborados. Además, hay que resaltar que ese lenguaje común presenta implicaciones directas relacionadas con la calidad.

- Se concluye que **la experiencia** organizada atendiendo a criterios de calidad y documentada utilizando modelos **es aplicable, reutilizable y transferible**. Permitiendo, además, ofrecer soluciones a problemas relacionados directamente con los que el usuario encuentra cuando utiliza un producto software.

- **La experiencia y la calidad se entienden orientadas al servicio**. Fruto de ello la calidad puede caracterizarse mediante funcionalidad im-

plícita recogida y documentada como un valor añadido que debe tenerse en cuenta desde el mismo momento en el que se elicita la funcionalidad explicita con la que aparecerá ligada. Dicha funcionalidad implicita es recurrente, surge en múltiples contextos y se asocia a diferente funcionalidad explícita. Estas características son las que permiten identificarla, modelarla y utilizarla de forma sistemática.

- **El concepto de patrón es útil para representar la funcionalidad implícita ligada a la calidad**. Con él puede documentarse la experiencia utilizando modelos y seguir aportándose su descripción utilizando lenguaje natural. Ambas vertientes redundan de forma positiva en su posterior utilización por sendos colectivos, es decir, ingenieros y usuarios respectivamente.

- **El concepto de patrón es útil para potenciar la trazabilidad del proceso de desarrollo**. Los modelos asociados a las características de calidad no suelen presentarse de forma aislada sino que mantienen relaciones entre ellos pudiéndose determinar la necesidad de considerar unos elementos en función de la existencia de otros.

- **El gap semántico entre Ingeniería del Software e Interacción Persona-Ordenador se nos ha ofrecido su verdadera dimensión,** que no es otra que el análisis de requisitos, es decir, el adecuado equilibrio que se desea alcanzar entre lo que el usuario espera y lo que el ingeniero no debe pasar por alto.

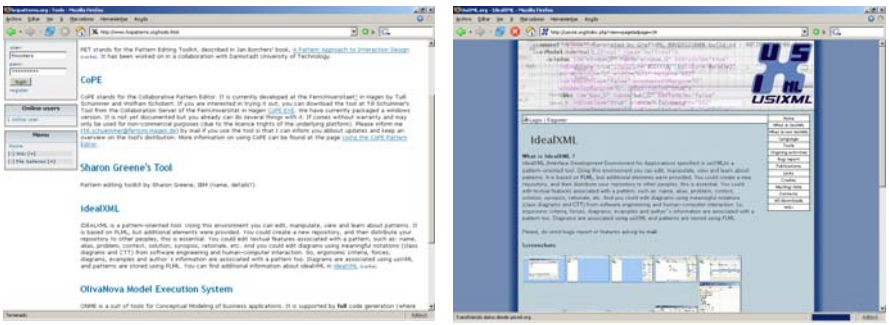
- **Se ha ofrecido una herramienta**, IDEALXML, que permite documentar y gestionar la experiencia, ya se ésta propia o disponible de diferentes fuentes, utilizando para ello concepto de patrón. La herramienta permite consultar, guardar, editar, añadir, eliminar patrones relacionados con diferentes ámbitos y no exclusivamente con interacción. De esta forma el concepto de patrón se convierte en pieza clave en el proceso de desarrollo dirigido por modelos. La experiencia está estrechamente ligada a la calidad a través de un modelo de calidad y de la aportación que se realiza a través de la consideración de los patrones de interacción disponibles.

- **Se ha contribuido a la puesta en práctica de un marco de trabajo** ideado para el desarrollo de interfaces de usuario como es usiXML. Aportándose al mismo la consideración de calidad y experiencia.

8.3 Repercusión del trabajo de investigación realizado

La aplicación y presentación de las conclusiones alcanzadas con la realización de esta tesis doctoral han sido publicadas y defendidas en muy diversas conferencias, congresos y talleres, a nivel internacional y nacional, los mismos están relacionados, fundamentalmente, tanto con la Ingeniería del Software como con la Interacción Persona-Ordenador.

En paralelo con la elaboración de la tesis se ha desarrollado una herramienta que permite la gestión, manipulación y transferencia de experiencia modelada en forma de patrones. Esta herramienta fue bautizada con el nombre de IDEALXML y está disponible para su descarga en un sitio web propio (http://www.info-ab.uclm.es/personal/fmontero/idealxml) y, además, existen referencias en otros sitios web relacionados con el estudio de los patrones de diseño (www.hcipatterns.org - Fig. 8-1a). La principal aportación de la mencionada herramienta es que convierte a los patrones (interacción, colaboración, diseño), en elementos abiertos y no en cajas negras propias y únicamente manipulables internamente por su propietario.



(a) hcipatterns.org (b) usiXML.org

Figura 8-1. Diferentes sitios web, de carácter internacional, en los que es posible encontrar información sobre IDEALXML. En (a) se recalca su uso como gestor de patrones y en (b) como entorno de soporte al desarrollo y a usiXML

Otro sitio web donde tiene especial cabida la herramienta IDEALXML es en usiXML (Fig. 8-1b). Además de incluir una referencia al entorno en dicho sitio web, también es considerada plenamente integrada en el marco propuesto con usiXML debido a la funcionalidad ofrecida por ella para editar cualquiera de los modelos relacionados con el dominio, las tareas, la presentación abstracta y el mapping entre ellos. Fruto de ello son las publicaciones de reciente aparición en las que la herramienta IDEALXML viene referenciada, y que ahora paso a recoger:

- Vanderdonckt, J., and Limbourg, Q. Developing user interfaces according to model driven architecture. Tutorial sponsored by SIMILAR, the European research task force creating human-machine communication and E-Mode ITEA 7 research project. Tenth IFIP TC13 International Conference on Human-Computer Interaction. 12-16 September 2005, Rome, Italy (to appear)

- Sousa, K., Furtado, E. A Unified Process Supported by a Framework for the Semi-Automatic Generation of Multi-Context UIs. 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'2005), Newcastle upon Tyne, England, July 13-15, 2005. Springer-Verlag, Berlin, 2005 (to appear)

- Vanderdonckt, J., A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005), O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31

- Furtado, E., Furtado, V., Soares Sousa, K., Vanderdonckt, J., Limbourg, Q., KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in UsiXML, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004),), Ph. Palanque, P. Slavik, M. Winckler (eds.), ACM Press, New York, 2004, pp. 121-128

- Limbourg, Q., Vanderdonckt, J., Addressing the Mapping Problem in User Interface Design with UsiXML, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004), Ph. Palanque, P. Slavik, M. Winckler (eds.), ACM Press, New York, 2004, pp. 155-163

- Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence, in Proceedings of Workshop on Device Independent Web Engineering DIWE'04 (Munich, 26-27 July 2004), M. Lauff (Ed.), Munich, 2004

- Limbourg, Q. Multi-path development of user interfaces. Thèse de docteur en sciences de gestion. Faculté des sciences économiques, sociales et politiques. Université catholique de Louvain. 2004

En función de estas referencias, de la consideración de la herramienta IDEALXML dentro del marco de trabajo que se define con usiXML (véase Fig. 8-2), de la inclusión del grupo de investigación LoUISE como partner en la *European taskforce creating human-machine interfaces SIMILAR to human-human communication* y de contar, en el grupo de investigación al que pertenece el doctorando, con la disponibilidad de diferentes proyectos concedidos recientemente (como son el proyecto CICYT TIN2004-08000-C03-01 y el proyecto PCB-03-003 concedido por la Junta de Comunidades de Castilla-La Mancha a través de su Consejería de Educación) está asegurada la continuidad de realización de actividades de I+D+i, así como la factibilidad de abordar los trabajos futuros que más tarde pasaremos a detallar.

En la Fig. 8-2 se ha resaltado el papel que IDEALXML está jugando en la propuesta usiXML. El mismo se traduce en las facilidades que proporciona IDEALXML y que están relacionadas con la edición de los diferentes modelos definidos en usiXML, específicamente los de dominio, tareas y especificación de interfaz abstracta. Además, y gracias a las aportaciones que puede ofrecer IDEALXML y que se traducen en facilidades para la documentación y utilización de la experiencia disponible en forma de patrones, su acogida dentro de la propuesta usiXML es plena.

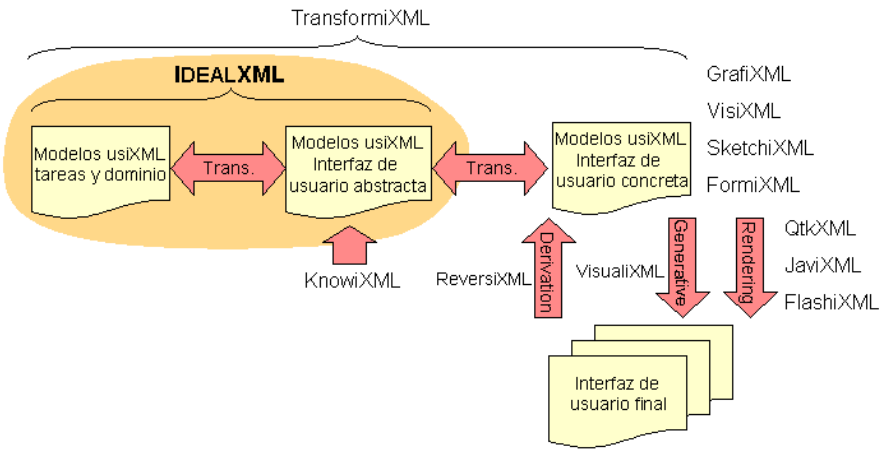


Figura 8-2. Situación de IDEALXML dentro del marco arquitectónico y de herramientas que se contempla en usiXML

Respecto a la documentación de la experiencia, que también ha tenido un papel destacado en esta tesis doctoral, mediante la elaboración y propuesta de lenguajes de patrones de interacción, cabe destacar que las colecciones de patrones presentadas que mayor difusión han tenido han sido

las publicadas en sendas ediciones de las conferencias PloP (*Pattern Languages of Programming*) celebradas en Brasil y en Dinamarca en el año 2002:

- **Montero, F**., Lozano, M., González, P., Ramos, I. Designing web sites by using design patterns. The second latin american conference on pattern languages of Programming. Itaipava- Rio de Janeiro, Brasil. 5 a 7 de agosto de 2002. (ISBN: 85-87837-07-9)
- **Montero, F**., Lozano, M., González, P., Ramos, I. A first approach for design web sites by using patterns. First Nordic conference on Pattern Languages of Program. Hojstrupgard, Dinamarca. 20 a 22 de septiembre de 2002. (ISBN: 87-7849-769-8)

Dichas publicaciones han tenido calado en la comunidad afín al tema de los patrones de interacción, y se han utilizado en la organización de workshops. Por ejemplo, el Departamento de Computación de la Universidad Federal de San Carlos organizó en julio de 2004 un Workshop sobre patrones de interacción en el contexto de la Web. En dicho workshop los patrones recogidos en las publicaciones antes mencionadas fueron utilizados como caso de estudio y ejemplo. Información adicional puede encontrarse en la página web de la profesora Júnia Coutinho Anacleto Silva (www.dc.ufscar.br/~junia).

Los artículos antes citados también han sido referenciados en la *Second International Workshop From Software Requirements to Architectures* (STRAW'03), celebrado en la *International Conference on Software Engineering* 2003 (ICSE'03) (http://se.uwaterloo.ca/~straw03/) en el artículo:

- Chandra, S., Bhattaram, S. Patterns approach to building software systems. International Workshop From Software Requirements to Architectures (STRAW'03), celebrado en la International Conference on Software Engineering 2003 (ICSE'03).

Finalmente, en el terreno de la usabilidad conviene citar también que la puesta en práctica de esta tesis doctoral en un entorno industrial está pendiente de la aprobación y puesta en marcha de una Agencia de acreditación de la Usabilidad que se pretende crear a nivel regional y en la que se contaría con el apoyo del Instituto de Investigación en Informática de Albacete (I^3A). En este proyecto el estudio y caracterización de la experiencia realizado en esta tesis doctoral constituiría el pilar básico con el que la Agencia podría llevar a cabo procesos de evaluación de la usabilidad a nivel cualita-

tivo y, con ellos, recomendar acciones en la dirección de mejorar los defectos detectados.

Además, en febrero de este año empezaron las actividades de una Acción Europea (COST 294) denominada *Towards the Maturation of IT Usability Evaluation* (**MAUSE**). Como grupo de investigación, nuestro objetivo será estar involucrados en, al menos, uno de sus grupos de trabajo y compartir con el resto de involucrados en esta acción los conocimientos disponibles y relacionados con la usabilidad. En este sentido se ha enviado un artículo al primer workshop que se ha organizado ligado a dicha iniciativa:

- Montero, F., Vanderdonckt, J., Lozano, M., González, P. Quality Models for Automated Evaluation of Web Sites Usability and Accessibility. International COST 294 Workshop on User Interface. Quality Models. Tenth IFIP TC13 International Conference on Human-Computer Interaction. 12-16 September 2005, Rome, Italy (to appear)

8.4 Nuevos retos y desafíos

En el apartado de nuevos retos y desafíos que se han identificado durante la realización de la tesis doctoral que se ha presentado y que han quedado, finalmente, postergados para fases posteriores de innovación y desarrollo merece la pena destacar los siguientes:

- La experiencia documentada y disponible puede ser caracterizada de forma genérica con éxito, como ha quedado de manifiesto en esta tesis doctoral, pero se puede profundizar en la dirección de tratar de dar soporte al ingeniero para que pueda adecuar la descripción general de la solución documentada en los patrones a cada problema particular. Para ello conviene considerar la aportación que puede ofrecer otra disciplina, como es la Inteligencia Artificial en la que se describen y presentan mecanismos como, por ejemplo, el Razonamiento Basado en Casos (Kolodner, 1992) muy interesante por las semejanzas que pueden establecerse entre un caso y un modelo.

- Considerar en mayor detalle la obtención de la interfaz de usuario concreta a partir de la especificación abstracta de la interfaz producida con la propuesta asociada a esta tesis doctoral. Para ello deben caracterizarse las interfaces de usuario concretas, incorporando y considerándose, entre otras, las métricas propuestas en (Constantine et al., 1999), así

como las guías de estilo disponibles. En esta línea, los patrones aparecerían en escena en la fase de especificación y las guías de estilo surgen para asistir de forma automática o semiautomática a la generación de interfaces de usuario dependientes de la plataforma.

- En esta tesis doctoral se ha trabajado en el ámbito de sistema y de modelo, capas M0 y M1 en la pila de metamodelos establecida por OMG. Sería interesante estudiar qué relación o implicaciones supone la consideración de patrones considerados como modelos en el resto de capas de nivel superior: metamodelo (M2) y de meta-metamodelo (M3).

- Profundizar en las relaciones que pueden establecerse entre los distintos modelos, especialmente en aquellas relacionadas con la inversión de la trayectoria habitual (*Reingeniería* de IUs), es decir, aquella que iría de la presentación concreta a la presentación abstracta y de ella al modelado.

- Consideración de características del usuario y adaptatividad de la interfaz. En función de ello establecer un mayor grado de integración con otras líneas de investigación llevadas a cabo en el propio grupo de investigación al que pertenezco, integrando las propuestas de Víctor López-Jaquero. Hay un trabajo ya realizado que se traduce en la disponibilidad de distintas publicaciones (véase la sección Compendio de publicaciones representativas) en esta dirección de las que cabe destacar: López-Jaquero, V., **Montero, F.**, Molina, J. P., González, P. A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties. The 9th IFIP Working Conference on Engineering for Human-Computer Interaction *Jointly with The 11th International Workshop on* Design, Specification and Verification of Interactive Systems (EHCI-DSVIS, 2004). Tremsbüttel Castle, Hamburg, Germany, July 11-13, 2004.

- Estudiar la validez de las conclusiones y resultados obtenidos cuando deban considerarse otros modos de interacción (por ejemplo, móviles o 3D). Específicamente en esas direcciones también existen publicaciones (Stanciulescu et al., 2005; Molina et al., 2005) que avalan la viabilidad de esta dirección, fruto de las colaboraciones con Adrian Stanciulescu (miembro del grupo BCHI de la Universidad de Louvain) y de José Pascual Molina (miembro del grupo LoUISE).

- Desarrollar e integrar completamente, y dentro de un mismo entorno, todos los diagramas y notaciones necesarios para poner en práctica la propuesta presentada en esta tesis doctoral (secuencia, estado, colaboración, etc.).

- Considerar la inclusión en ese mismo entorno mencionado en el punto anterior la posibilidad de realizar labores de verificación y animación de las diferentes notaciones manejadas.

8.5 Compendio de publicaciones representativas

Se recogen seguidamente y de forma cronológica aquellas referencias a publicaciones representativas en las que ha participado el doctorando en el proceso de elaboración de esta tesis doctoral. Se comentarán de forma breve aquellas publicaciones que, por su contenido, guardan una relación directa con la metodología de trabajo y el tratamiento de conceptos principales tratados en esta tesis doctoral.

Montero, F., Lozano, M., González, P. User Interfaces Development by using Patterns. VIIP 2001: 39-43

Se establecen la propuesta de esta tesis doctoral. Las dos palabras clave iniciales son la experiencia en forma de patrones y el desarrollo de interfaces de usuario

Montero, F., López-Jaquero, V., Lozano, M., González, P. Interfaces de usuario. El gran reto de los Sistemas de Información Geográfica. Editado por Fundación Dintel. (ISBN: 84-931933-1-3)

Montero, F., Lozano, M., González, P. Patrones de Interfaz para entornos de trabajo en grupo. II Jornadas de trabajo Dolmen. Universidad Politécnica de Valencia. 12 y 13 de marzo de 2002.

Montero, F., Lozano, M., González, P. Patrones de interacción: Taxonomía y otros problemas. Congreso Internacional de Interacción 2002. Universidad Carlos III, Madrid, 8 a 10 de mayo de 2002.

Montero, F., Lozano, M., González, P., Ramos, I. Designing web sites by using design patterns. The second latin american conference on pattern languages of Programming. Itaipava- Rio de Janeiro, Brasil. 5 a 7 de agosto de 2002. (ISBN: 85-87837-07-9)

Montero, F., Lozano, M., González, P., Ramos, I. A first approach for design web sites by using patterns. First Nordic conference on Pattern Languages of Program. Hojstrupgard, Dinamarca. 20 a 22 de septiembre de 2002. (ISBN: 87-7849-769-8)

Los patrones de interacción centraron el interés en la primera fase del desarrollo de la tesis doctoral. Se hicieron diferentes contribuciones en las

que se identificaron limitaciones en la forma de documentarlos y organizarlos. También se propusieron diferentes lenguajes de patrones.

Sendin, M., Lores, J., **Montero, F.**, López-Jaquero, V. Using Reflection on Dynamic Adaptation of User Interfaces. 4th International Workshop on Mobile Computing. IMC Workshop. Assistance, Mobility, Applications. Rostock, Germany, 2003.

Fernández-Caballero, A., López-Jaquero, V., **Montero, F**. Métricas de usabilidad y sistemas multiagente en hipermedia adaptativa. IV Congreso Internacional de Interacción Persona-Ordenador. Interacción 2003. Vigo. España. (ISBN: 84-932887-4-8)

Sendin, M., **Montero, F**., Lozano, M., Lorés, J. El prototipo interactivo del Montsec. Hacia la automatización de diversos aspectos adaptativos en el desarrollo de la interfaz de usuario. IV Congreso Internacional de Interacción Persona-Ordenador. Interacción 2003. Vigo. España. (ISBN: 84-932887-4-8)

Sendín, M., Lorés, J., **Montero, F.**, López-Jaquero, V. Towards a Framework to Develop Plastic User Interfaces. Mobile HCI 2003: 428-433

López-Jaquero, V., **Montero, F.**, Fernández-Caballero, A., Lozano, M. Usability metrics in Adaptive Agent-Based Tutoring Systems. 10th International Conference on Human - Computer Interaction (HCI, 2003). 22 a 27 de Julio, Creta, Grecia. 2003.

Montero, F., Lozano, M., López-Jaquero, V., González, P. A Quality Model For Testing The Usability Of The Web Sites. 10th International Conference on Human - Computer Interaction (HCI, 2003). 22 a 27 de Julio, Creta, Grecia. 2003. Human-Computer Interaction: Theory and Practice (part 1). J. Jacko, C. Stephanidis (Eds.). Lawrence Erlbaum Associates. Londrés, Reino Unido, 2003. (ISBN: 0-8058-4931-9)

La calidad, tercero de los conceptos considerados en esta tesis, entra en escena. Se presenta una primera aproximación al posterior modelo de calidad. Las preguntas que se nos plantean son dos: ¿qué es la calidad? y ¿cómo podemos caracterizarla?.

Montero, F., López-Jaquero, V., Molina, J.P., González, P. An Approach to Develop User Interfaces with Plasticity. DSV-IS 2003: 420-423

Molina, J.P., González, P., Lozano, M., **Montero, F.**, Víctor López-Jaquero: Bridging the Gap: Developing 2D and 3D User Interfaces with the IDEAS Methodology. DSV-IS 2003: 303-315

Existen algunas colaboraciones que han sido destacadas en la sección de trabajos futuros.

López-Jaquero, V., **Montero, F.**, Molina, J.P., Fernández-Caballero, A., González, P. Model-Based Design of Adaptive User Interfaces through Connectors. DSV-IS 2003: 245-257

Montero, F., López-Jaquero, V., Lozano, M., González, P. Usability and Web Site Evaluation: Quality Models and User Testing Evaluations. ICEIS (1) 2003: 525-528

Fernández-Caballero, A., López-Jaquero, V., **Montero, F.**, González, P.: Adaptive Interaction Multi-agent Systems in E-learning/E-teaching on the Web. ICWE 2003: 144-153

Sendín, M., Lorés, J., **Montero, F.**, López-Jaquero, V., González, P. User Interfaces: A Proposal for Automatic Adaptation. ICWE 2003: 263-266

Lozano, M., **Montero, F.**, González, P. A usability and accessibility oriented development process. 8th ERCIM Workshop User Interfaces for all. Viena, 28-29 de junio de 2004.

Montero, F., López-Jaquero, V., Molina, J.P., Lozano, M. Improving e-shops environments by using usability patterns. 2nd workshop on software and usability cross-polination. The role of usability patterns. September, 1-2, 2003, Zürich, Switzerland. 2003

En este período de tiempo la experiencia y la calidad aparecen tratadas de forma conjunta, pero la experiencia adolece de capacidad para la generación, lo que impide ponerla en práctica de forma natural, es decir, como si fuese propia.

Montero, F., Lozano, M., González, P. Calidad en interfaces de usuario. Capítulo 5 del libro Calidad en el desarrollo y mantenimiento del software. Coordinadores: Mario G. Piattini, Felix O. García. Editorial Ra-Ma. 2003. (ISBN: 84-7897-544-6).

Montero, F., López-Jaquero, V., Lozano, M., González, P. De Platón al desarrollo de Interfaces de Usuario. V Congreso Interacción Persona Ordenador. 3 - 7 de mayo de 2004 Universitat de Lleida. 2004.

López-Jaquero, V., **Montero, F.**, Molina, J. P., González, P. A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties. The 9th IFIP Working Conference on Engineering for Human-Computer Interaction *Jointly with The 11th International Workshop on* Design, Specification and Verification of Interactive Systems (EHCI-DSVIS, 2004). Tremsbüttel Castle, Hamburg, Germany, July 11-13, 2004.

Montero, F., Lozano, M., González, P.: IDEALXML: an Experience-Based Environment for User Interface Design and pattern manipulation. Technical Report DIAB-05-01-4. Universidad de Castilla-La Mancha, Albacete (2005).

Se realiza la estancia en la Universidad católica de Louvain y bajo la supervisión del profesor D. Jean Vanderdonckt se considera y se termina incorporando el lenguaje usiXML en el marco integrador de calidad y experiencia previamente disponible. En la parte final de la estancia comienza el desarrollo de la herramienta IDEALXML. Esta herramienta da soporte a la recopilación y documentación de la experiencia en forma de patrones con dimensión generativa y, debido a esa última característica, sirve como entorno de desarrollo dirigido por modelos al estar integrada con otras herramientas disponibles y desarrolladas en el grupo de investigación del profesor D. Jean Vanderdonckt.

Montero, F., López-Jaquero, V., Lozano, M., González, P. A user interfaces development and abstraction mechanisms. Artículo seleccionado en el V Congreso Interacción Persona Ordenador para su publicación en Springer-Verlag, Berlin, 2005. (to appear)

Lozano, M., **Montero, F.**, García, F.J., Gonzalez, P. Calidad en el desarrollo de aplicaciones web: modelos, métodos de evaluación y métricas de calidad. Capítulo 8 del libro Ingeniería de la web y patrones de diseño. Coordinadores de la obra: Paloma Díaz, Susana Montero e Ignacio Aedo. Pearson Prentice Hall. 2005. (ISBN: 84-205-4609-7)

García, F. J., Lozano, M., **Montero, F**., Gallud, J.A., González, P. A Controlled Experiment for Measuring the usability of webapps using patterns. 7th International Conference on Enterprise Information Systems (ICEIS, 2005). 24-28 May, Miami, USA. 2005.

Este grupo de publicaciones pone en práctica el modelo de calidad centrado en la usabilidad y ligado a la experiencia recogida en esta tesis doctoral.

Montero, F., López-Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M.D., Solving the Mapping Problem in User Interface Design by Seamless Integration in IDEALXML. 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'2005), Newcastle upon Tyne, England, July 13-15, 2005. Springer-Verlag, Berlin, 2005 (to appear)

García, F.J., Lozano, M., **Montero, F.**, Gallud, J.A., Ruiz, V. Survey on Quality Models to Measure the Quality of Web Sites and Applications. 11th International Conference on Human-Computer Interaction (HCI, 2005). 22-27 July, 2005, Las Vegas, Nevada, USA. 2005.

Montero, F., López-Jaquero, V., Ramírez, Y., Lozano, M., González. P. Patrones de Interacción: para usuarios y para diseñadores. VI Congreso de Interacción Persona-Ordenador. 13-16 de septiembre, Granada, España. 2005 (por aparecer).

Molina, J.P., Vanderdonckt, J., **Montero, F.**, González, P. Towards virtualization of user interfaces based on UsiXML. Web3D 2005: 169-178. 10th International Conference on 3D Web Technology, ACM Press. Bangor, Wales. March 29 - April 1 2005.

López-Jaquero, V, Montero, F., Molina, J.P., González, P., Fernández-Caballero, A. A Multi-Agent System Architecture for the Adaptation of User Interfaces. 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005). 15-17 September 2005, Budapest, Hungary. Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin. (to appear).

Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B. and **Montero, F**. A Transformational Approach for Multimodal Web User Interfaces based on UsiXML.The Seventh International Conference on Multimodal Interfaces (ICMI 2005). Trento, Italy, October 3-7, 2005 (to appear)

En estos últimos artículos se reflejan algunas de las líneas de trabajo futuro recogidas en el apartado anterior y que son posibles gracias a la colaboración con otros miembros de investigación del grupo LoUISE así como por el mantenimiento de las relaciones establecidas con la realización de la estancia predoctoral llevada a cabo.

Apéndices

En está sección se recoge información adicional relacionada, en primer término, con la experiencia documentada y disponible que, en forma de patrones, puede encontrase y aplicarse para el modelado en distintos aspectos considerados en el desarrollo de un producto software (dominio, diseño, interacción).

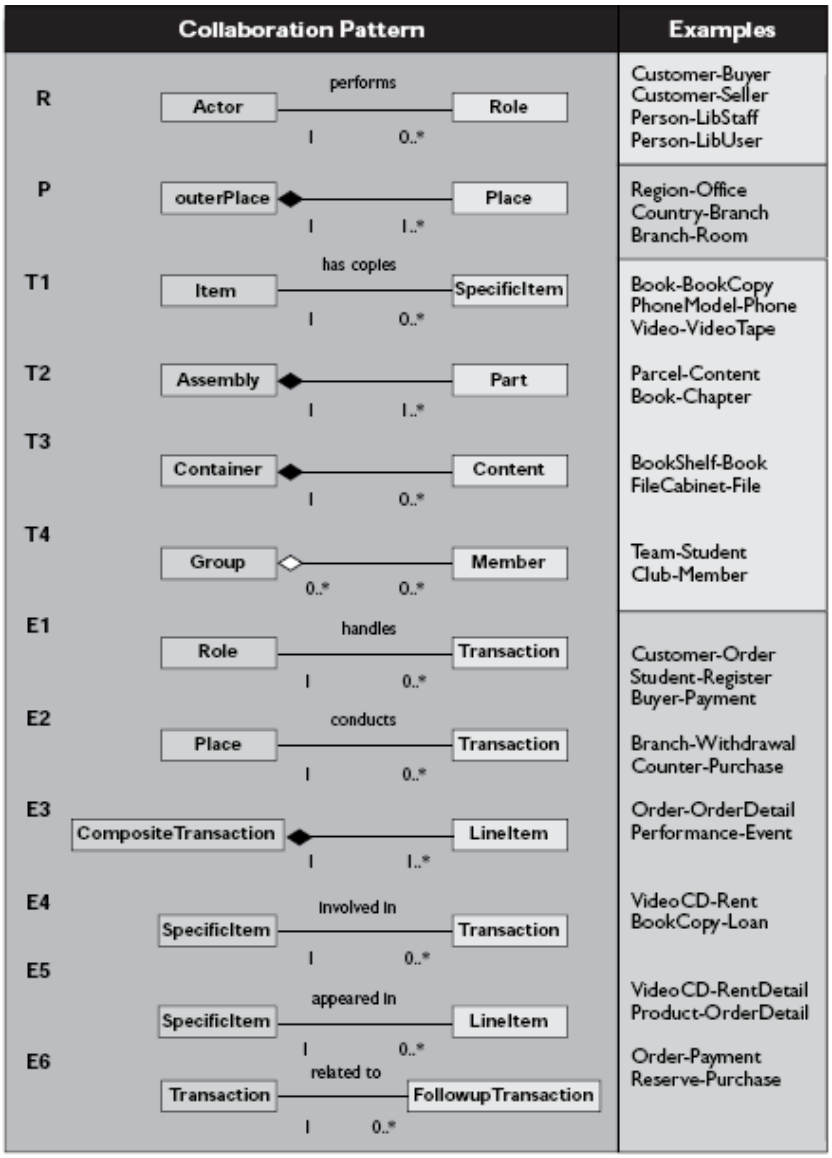
Posteriormente, en el Apéndice D se recogerán una serie de definiciones relacionadas con la calidad considerada desde el punto de vista que ofrecen los criterios ergonómicos.

Otros apéndices que tratan propuestas que han tenido influencia en la propuesta metodológica presentada con la elaboración de esta tesis doctoral son los dedicados a los patrones WebML (Apéndice E), a la metodología de desarrollo de interfaces de usuario basada en modelos IDEAS (Apéndice F) y al lenguaje y marco de especificación de interfaces de usuario usiXML (Apéndice G).

Finalmente, el Apéndice H está dedicado a presentar de forma breve las características que ofrece IDEALXML.

Apéndice A. Patrones de colaboración

Para llevar a cabo la especificación del modelo de dominio, elaborado en la fase de diseño de la propuesta metodológica presentada en esta tesis doctoral, se utiliza el catálogo de patrones propuesto en (Nicola et al., 2001). Dicho catálogo se recoge en la siguiente figura.



En la figura anterior se muestran los doce patrones que constituyen la colección de patrones de colaboración. En ellos se describen relaciones habituales que pueden identificarse entre *personas*, *lugares*, *cosas* y *eventos*. En la columna de la derecha aparecen ejemplos significativos relacionados con la aparición de cada patrón en situaciones reales. A partir de dichas situaciones, el ingeniero sin la suficiente experiencia podrá extrapolar esas mismas relaciones, identificando así las entidades involucradas, en el problema concreto que el ingeniero debe considerar.

Una relación especialmente interesante entre las recopiladas en el catálogo de patrones mostrado es la que se define entre un *role* y una *transaction*. Dicha relación es equiparable con cualquier operación de interacción mantenida entre una persona y un ordenador, independientemente del modo de interacción.

Más información sobre los patrones de colaboración puede encontrase en la siguiente referencia: *Nicola, J., Mayfield, M., and Abney, M. Streamlined Object Modeling. Prentice Hall, Upper Saddle River, NJ, 2001. ISBN: 0130668397*.

Apéndice B. Patrones de diseño

Los patrones de diseño son descripciones de clases y objetos relacionados que están particularizados para resolver un problema de diseño general en un determinado contexto (Gamma et al., 1995).

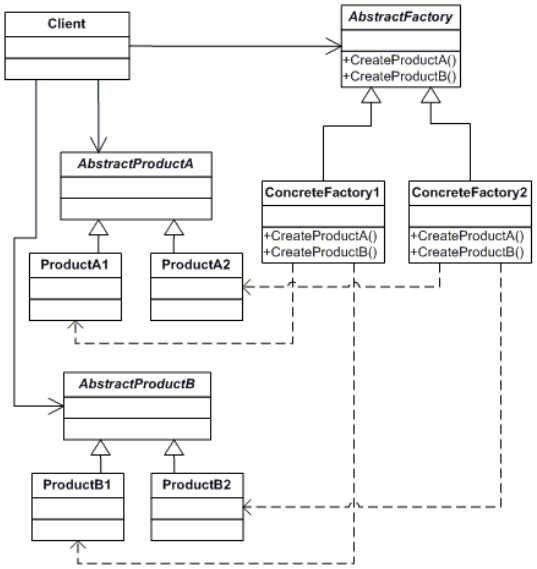
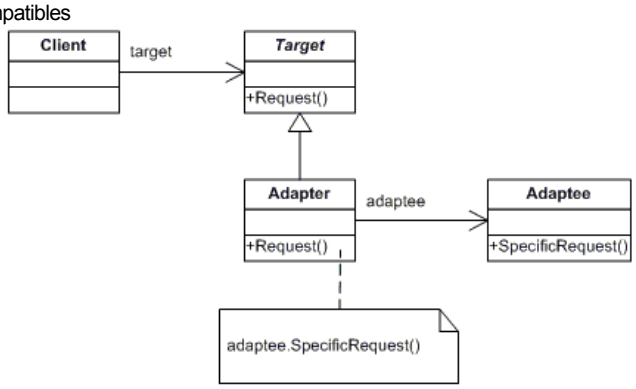
En esta tesis doctoral los patrones de diseño se utilizan en una segunda fase de refinamiento a la que se someten los casos de uso iniciales con los que analizan los requisitos funcionales que debe cumplir un sistema. Posteriormente, los casos de uso son sometidos a un análisis más detallado utilizando la notación CTT, propuesta en (Paternò, 1999), y a partir de ella es cuando surgen los patrones de diseño para gestionar los comportamientos asociados a los requisitos funcionales identificados y a los no explícitamente funcionales deseados.

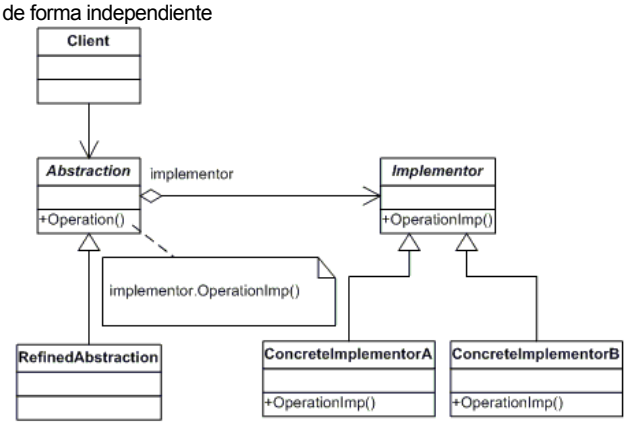
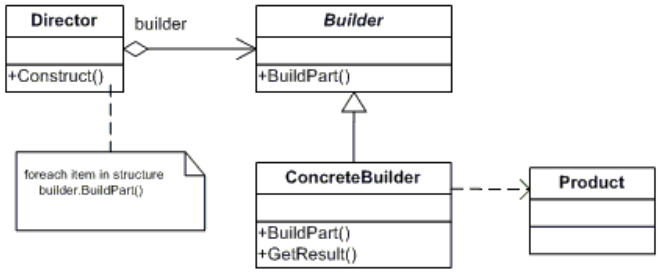
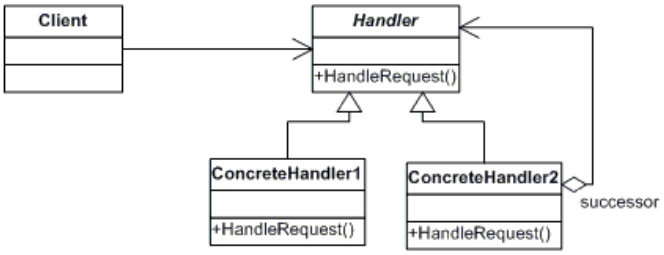
Los patrones pueden tener un propósito de creación, estructural o de comportamiento. Los patrones de creación tienen que ver con el proceso de creación de objetos. Los patrones estructurales tratan con la composición de clases u objetos. Los de comportamiento caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad.

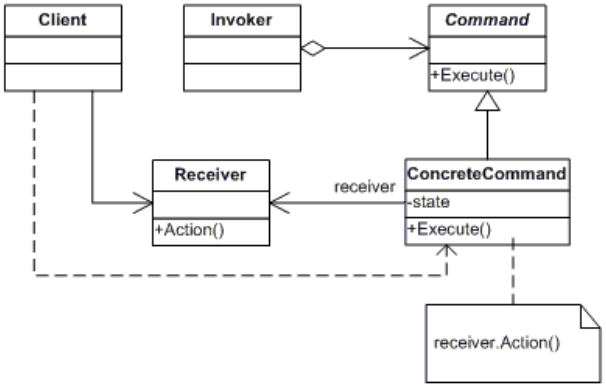
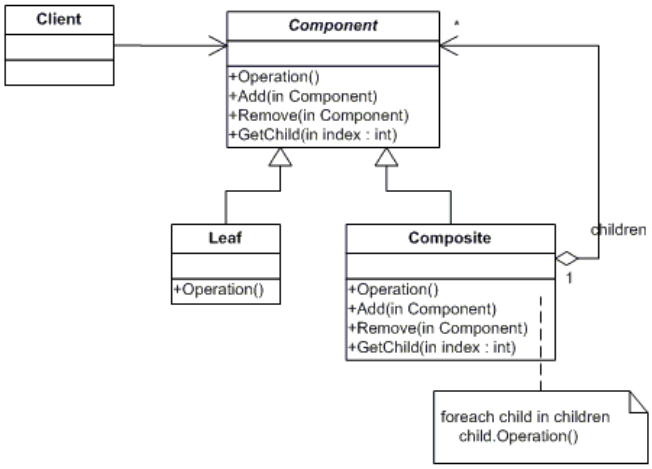
Seguidamente se recogen en este apéndice los 23 patrones de diseño propuestos en un libro referencia dentro de la Ingeniería del Software y de la Programación Orientada a Objetos. Cada patrón será descrito introduciendo su nombre y una pequeña descripción del propósito que lleva asociado, junto con una descripción gráfica utilizando diagramas de clases. Una descripción más detallada de los patrones recogidos en este apéndice puede encontrarse en Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, 1995. A lo largo del libro se usan diagramas para ilustrar las ideas importantes. Los patrones de diseño usan notaciones formales para denotar relaciones e interacciones entre clases y objetos. Estas notaciones son gráficas y se corresponden con:

- Un **diagrama de clases**, que representa clases, su estructura y las relaciones estáticas entre ellas.
- Un **diagrama de objetos**, que muestra una determinada estructura de objetos en tiempo de ejecución.

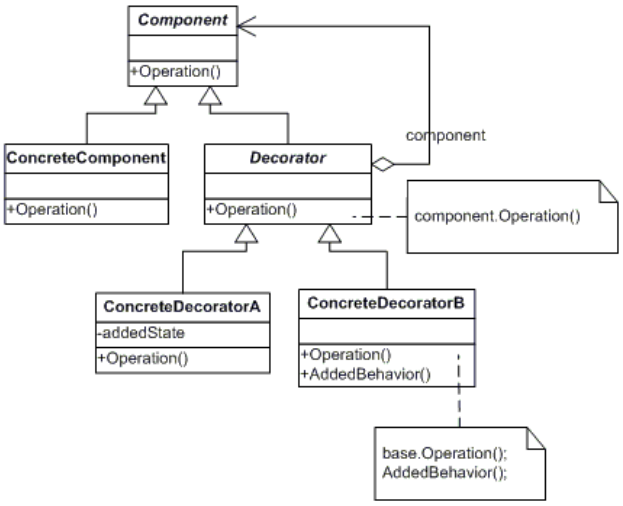
Un **diagrama de interacción**, que muestra el flujo de peticiones entre objetos.

Nombre	Propósito
Abstract Factory	Proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas 
Adapter	Convierte la interfaz de una clase en otra distinta que es la que esperan los clients. Permite que cooperen clases que de otra manera no podrían por tener interfaces incompatibles 

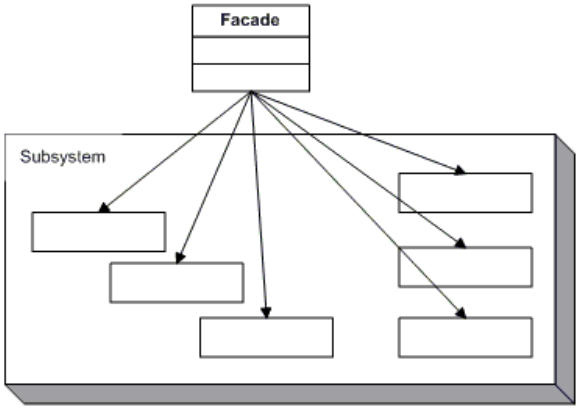
Bridge	Desacopla una abstracción de su implementación, de manera que ambas puedan variar de forma independiente 
Builder	Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones 
Chain of Responsability	Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que ésta sea tratada por algún objeto 

Command	Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones 
Composite	Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos 

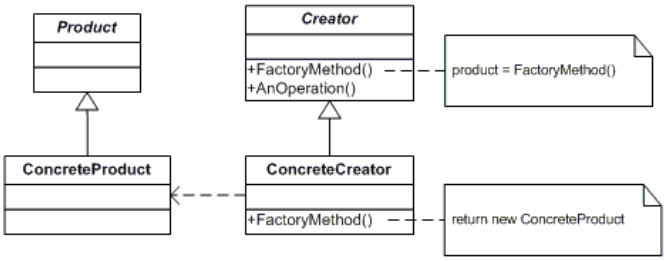
Decorator	Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad



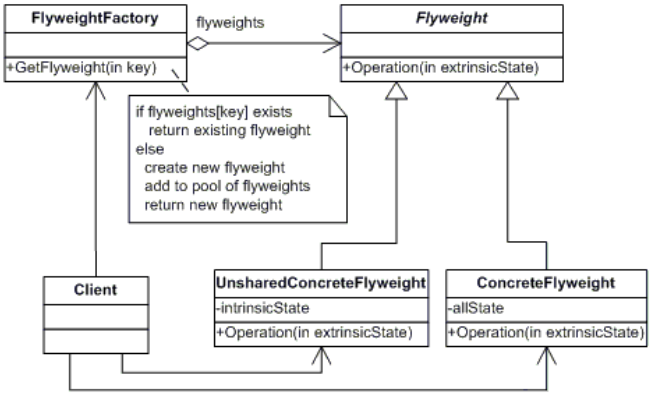
Façade	Proporciona una interfaz unificada para un conjunto de interfaces de un sistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar



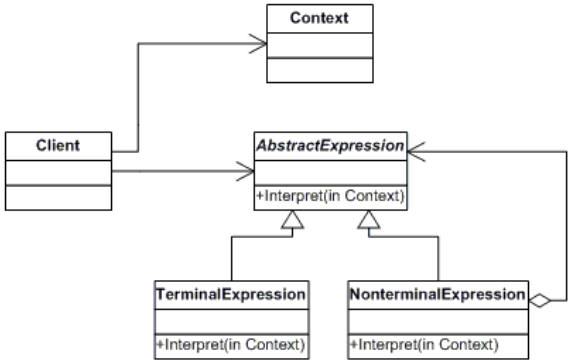
Factory Method	Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos



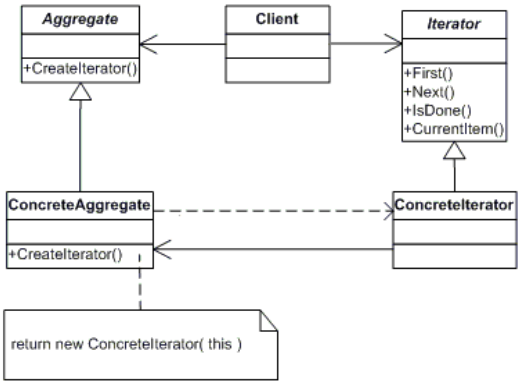
Flyweight	Usa el comportamiento para permitir un gran número de objetos de grano fino de forma eficiente

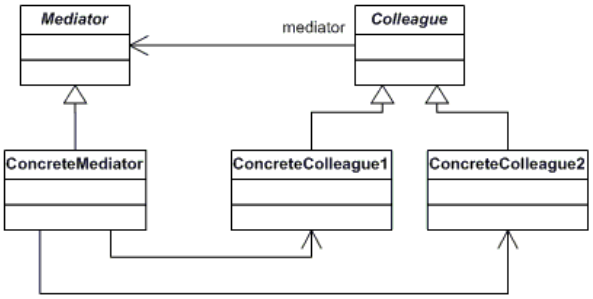
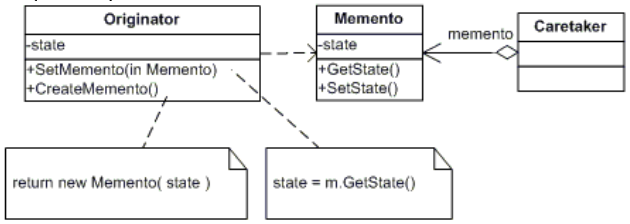
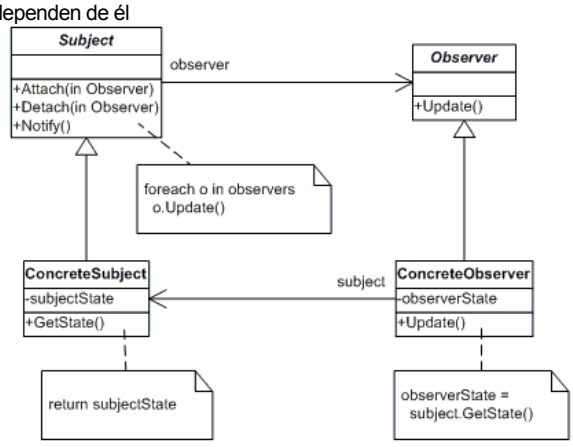


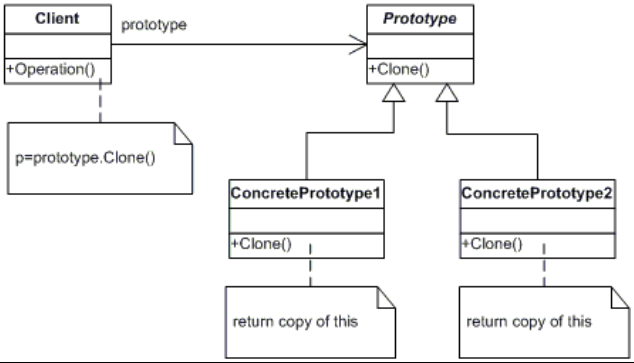
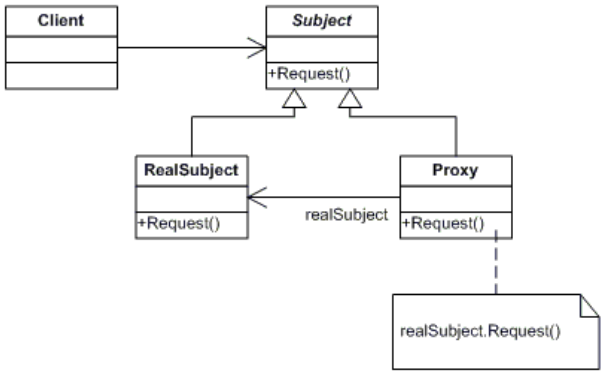
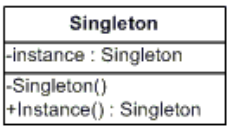
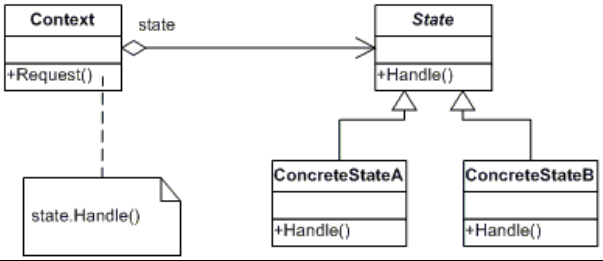
Interpreter	Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar sentencias del lenguaje

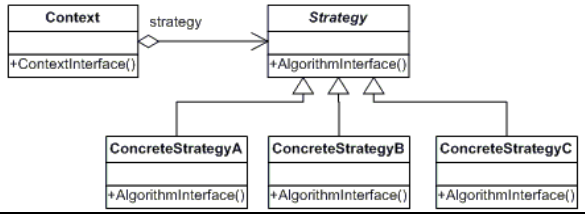
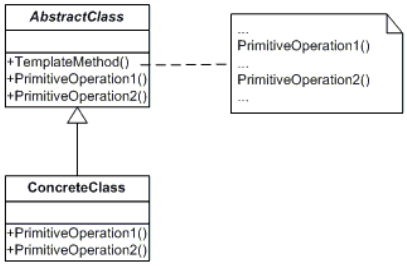
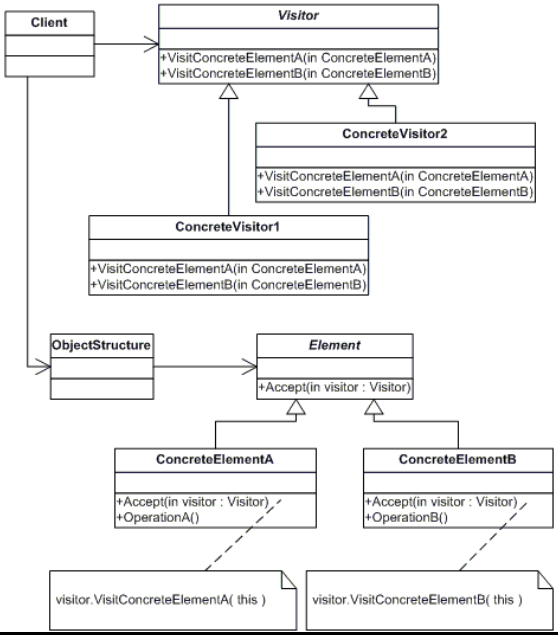


Iterator	Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna



Mediator	Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente 
Memento	Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste pueda volver a dicho estado más tarde 
Observer	Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifica y se actualizan automáticamente todos los objetos que dependen de él 

Prototype	Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crea nuevos objetos copiando de este prototipo 
Proxy	Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste 
Singleton	Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella 
State	Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Parecerá que cambia la clase del objeto 

Strategy	Define una familia de algoritmos, encapsula cada uno de los y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan 
Template Method	Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura 
Visitor	Representa una operación sobre los elementos de una estructura de objetos. Permite definir una operación sin cambiar las clases de los elementos sobre los que opera 

Apéndice C. Patrones de interacción

El catálogo de patrones al que más se ha hecho referencia en esta tesis doctoral es el elaborado por Jenifer Tidwell. Este catálogo de patrones está constituido por cerca de sesenta patrones y está disponible en la referencia web http://www.mit.edu/~jtidwell/common_ground.html.

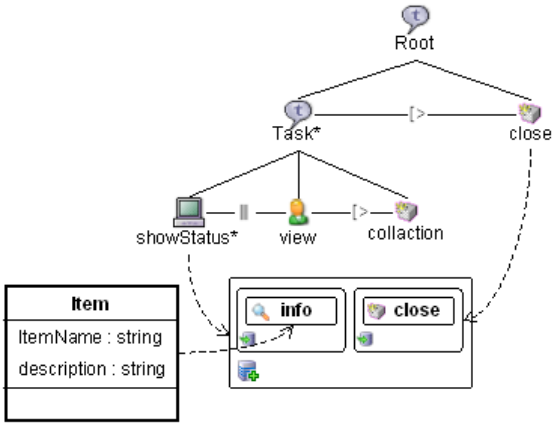
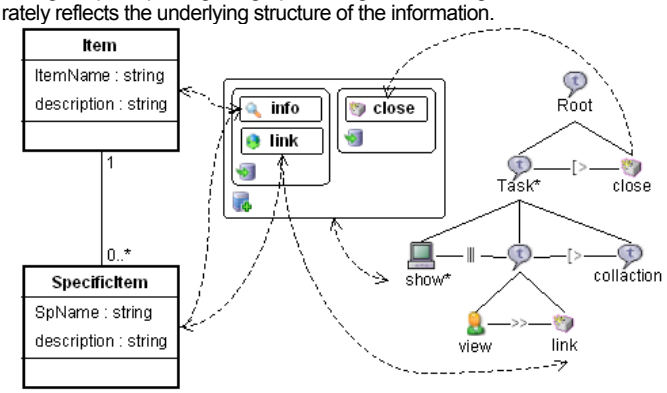
La colección de patrones referida ofrece una serie de características que la hacen digna de consideración frente a otras disponibles. La principal característica pasa por que es una colección independiente del dispositivo y del modo de interacción, es decir, no está elaborada pensando en el desarrollo de una clase concreta de interfaces de usuario y, por ello, los patrones que recopila puede aplicarse al desarrollo de prácticamente cualquier dispositivo o situación. Se trata, además, de un referente dentro del ámbito de las colecciones de patrones de interacción elaboradas en años sucesivos.

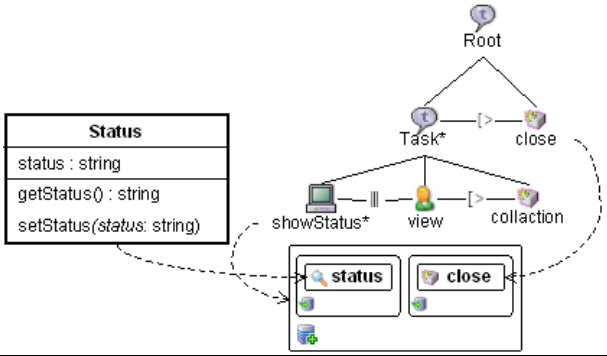
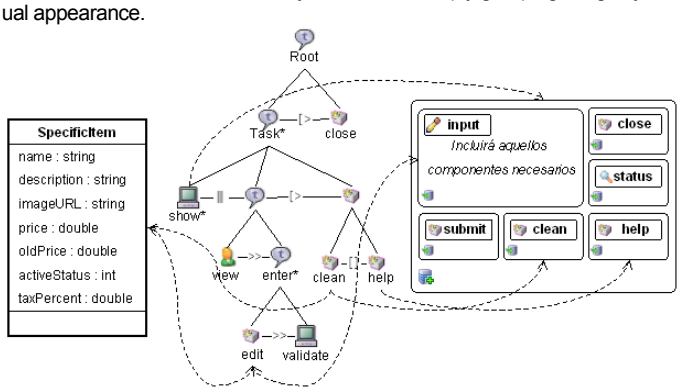
La organización de la colección atiende sin pretenderlo (aparentemente) a una serie de características de calidad, a las que no se hace referencia explícitamente. Dichas características son significativas para el ingeniero, pero tienen repercusión en el usuario. La colección de patrones está organizada atendiendo a una serie de características relacionadas con la forma de presentar los contenidos y las acciones, con cómo se anticipan las acciones al usuario, con cómo se utiliza el espacio y la atención del usuario, con cómo se organizan contenidos y acciones en las superficies de trabajo, con cómo el usuario puede navegar a través del producto software, con las acciones específicas que el usuario puede llevar a cabo, con cómo el usuario puede modificar el producto software, con cómo el producto software se hace atractivo y visualmente claro, y con cómo el producto software da soporte activo al usuario.

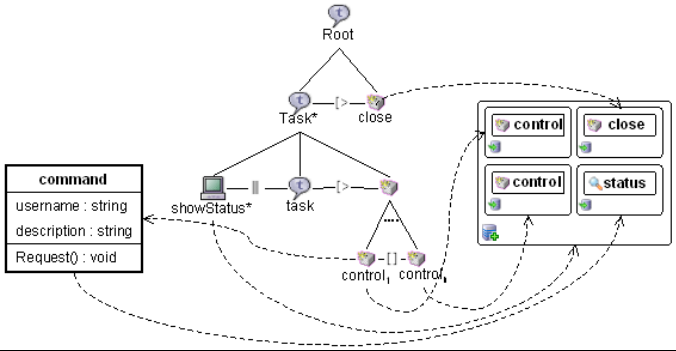
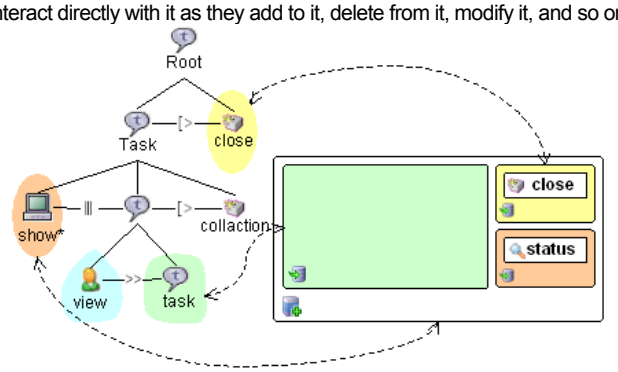
Veíamos en el apéndice anterior que cada patrón de diseño incluía al menos un diagrama de clases para su descripción. Esta misma filosofía es la que se ha extrapolado, en esta tesis doctoral, al ámbito de los patrones de interacción; determinándose y utilizándose aquellas notaciones más adecuadas para especificar la estructura de cada patrón de interacción. Para el caso de los patrones de interacción se usan tres notaciones:

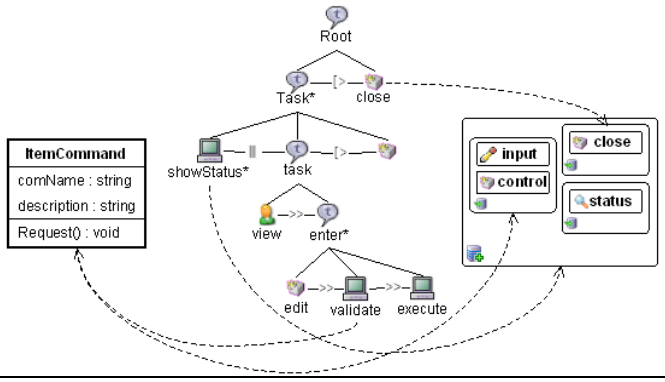
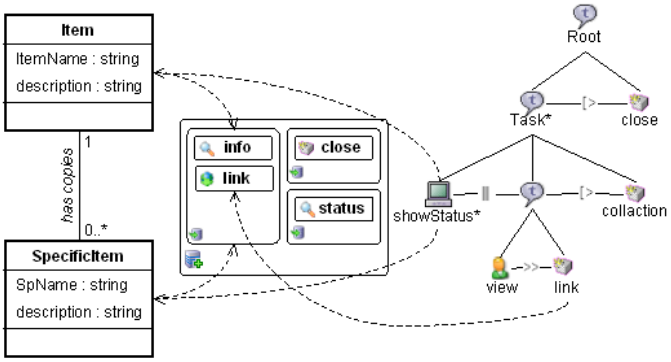
- **Diagramas de clases**, con los que se pretende representar las clases necesarias para modelar el dominio asociado a la puesta en práctica del patrón de interacción.
- **Análisis de tareas**, que sirve para describir la situación en la que el patrón de interacción es útil, poniendo de manifiesto que tareas se abordar y las tareas colaterales que aparecen asociadas a esa situación. Para su elaboración se utiliza la notación de análisis de tareas ConcurTaskTree (Paternò, 1999).

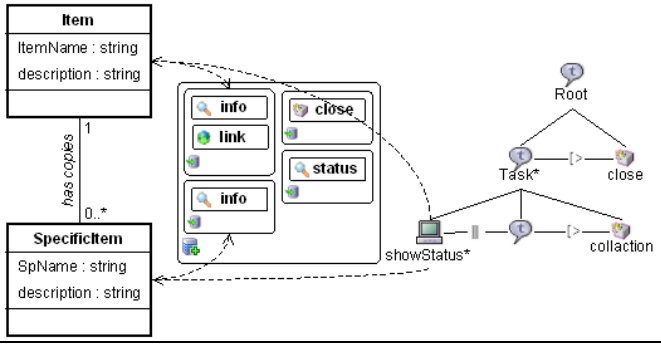
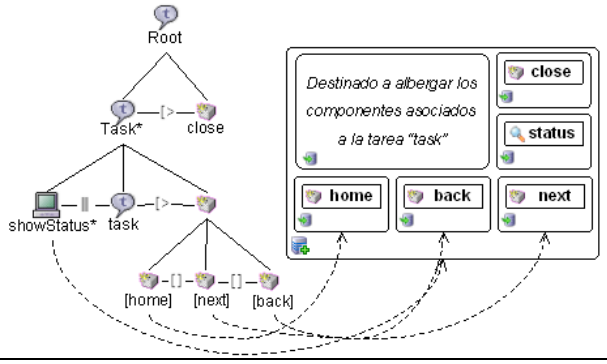
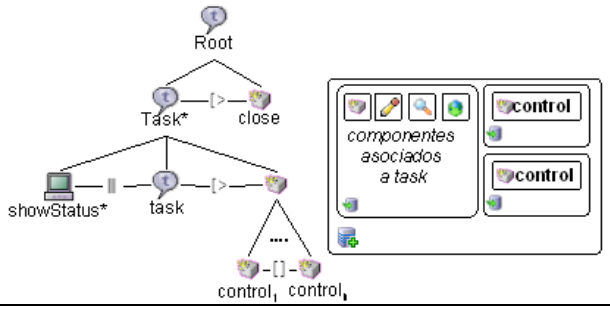
- **Diagramas de presentación a nivel abstracto**, donde se describe, utilizando objetos abstractos de interacción, la presentación asociada a cada situación que debe contemplar la interfaz de usuario del producto software. Para su elaboración se utilizan los objetos de interacción abstractos definidos en el lenguaje de especificación usiXML (Limbourg, 2004).

Nombre	Problema / Solución
Narrative	*Problem:* In what form should the information be displayed to the user? *Solution:* Convey the information via natural language. 
High-density information display	*Problem:* In what form should the information be displayed to the user? *Solution:* Pack as much information into one working surface as possible, following the precepts of good graphic design, with an organization that accurately reflects the underlying structure of the information. 

Status display	Problem: How can the artifact best show the state information to the user? *Solution:* Choose well-designed displays for the information to be shown. Put them together in a way that emphasizes the important things, deemphasizes the trivial, doesn't hide or obscure anything, and prevents confusing one piece of information with another. 
Form	*Problem:* How should the artifact indicate what kind of information should be supplied, and the extent of it? *Solution:* Provide appropriate "blanks" to be filled in, which clearly and correctly indicate what information should be provided. Visually indicate those editable blanks consistently, such as with subtle changes in background color, so that a user can see at a glance what needs to be filled in. Label them with clear, short labels that use terminology familiar to the user; place the labels as close to the blanks as is reasonable. Arrange them all in an order that makes sense semantically, rather than simply grouping things by visual appearance. 

Control Panel	*Problem:* How can the artifact best present the actions that the user may take? *Solution:* For each function or state variable that is part of the user's mental model, choose one well-designed control that performs the function or displays the variable's value; put them all together such that the most commonly-used controls are the most prominent. 
WYSIWYG Editor	*Problem:* How can the artifact best present what is being created, and what the user needs to do to create or change it? *Solution:* Always show the user an accurate and up-to-date representation of the artifact they are creating ("what you see is what you get"); allow the user to interact directly with it as they add to it, delete from it, modify it, and so on. 

Composed Command	*Problem:* How can the artifact best present the actions that the user may take? *Solution:* Provide a way for the user to directly enter the command, such as by speech or by typing it in. 
Navigable Spaces	*Problem:* How can you present the content so that a user can explore it at their own pace, in a way which is comprehensible and engaging to the user? *Solution:* Create the illusion that the working surfaces are spaces, or places the user can "go" into and out of. 

Overview Beside Detail	*Problem:* How can you present this large amount of content so that a user can explore it at their own pace, in a way which is comprehensible and engaging to the user? *Solution:* Show the whole set of objects, or the whole undetailed data set, in one part of the display area, to act as an overview of the content 
Step-by-step instructions	*Problem:* How can the artifact unfold the possible actions to the user in a way that does not overwhelm or confuse them, but instead guides them to a successful task completion? *Solution:* Walk the user through the task one step at a time, giving very clear instructions at each step. 
Small groups of related things	*Problem:* How should the items or actions be organized? *Solution:* Group the closely-related things together, nesting them in a hierarchy of groups if needed. 

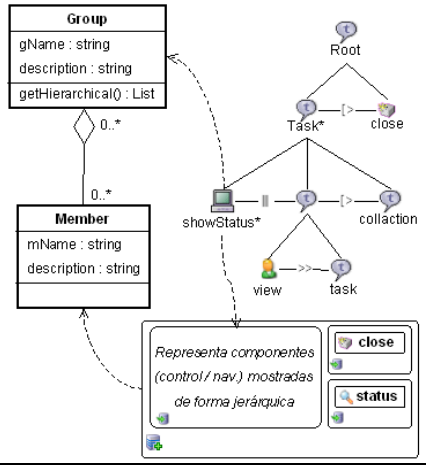
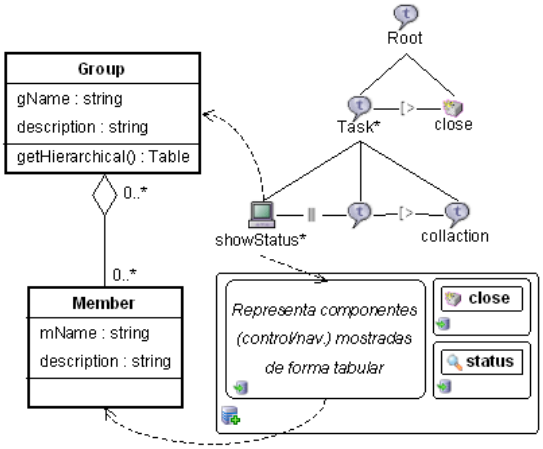
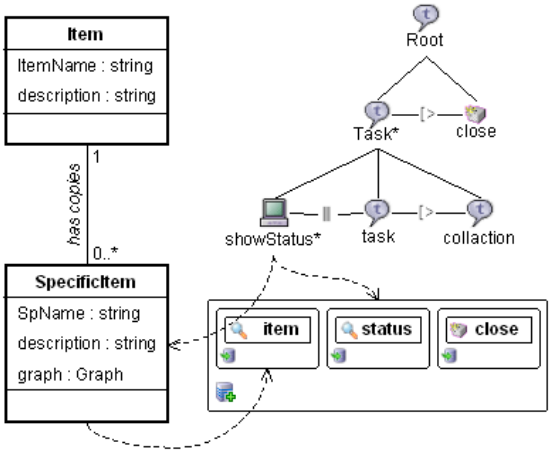
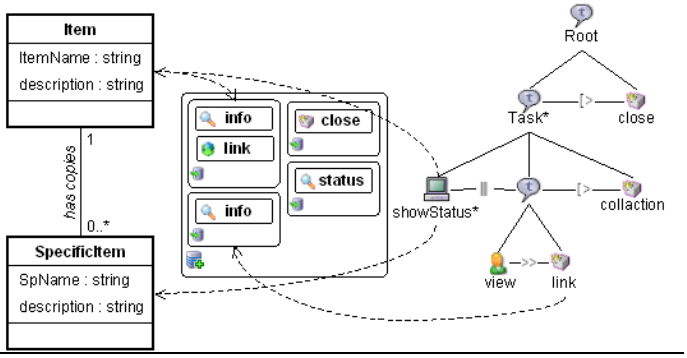
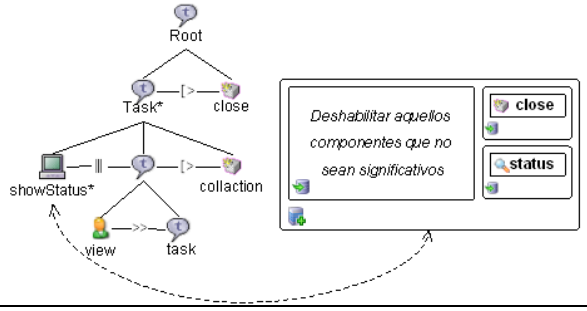
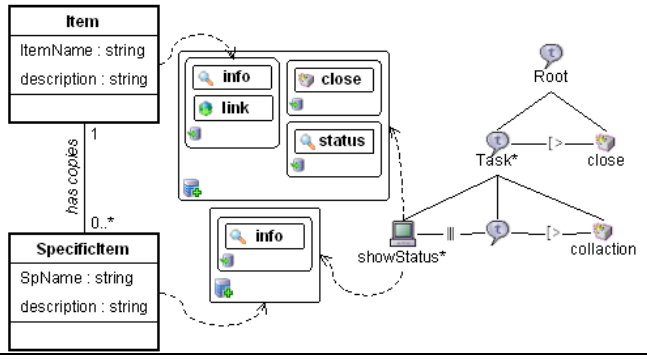
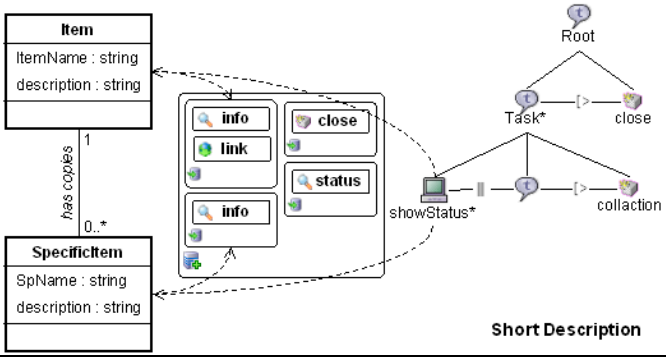
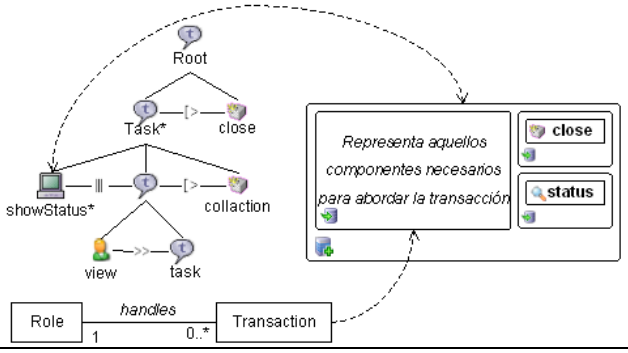
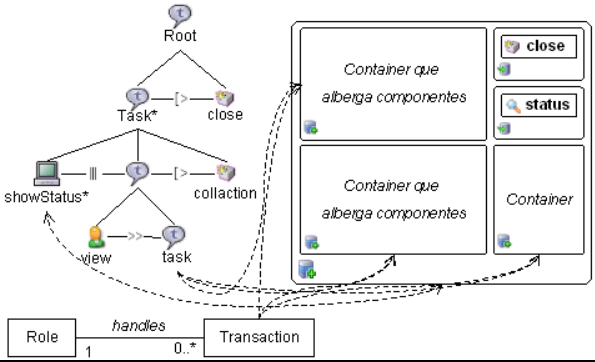
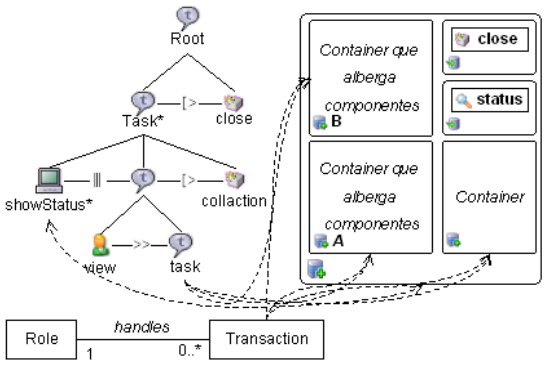
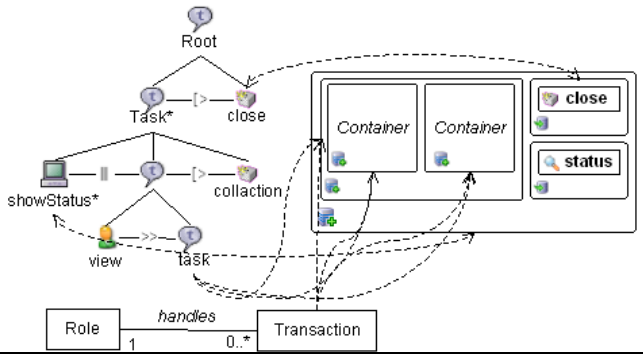
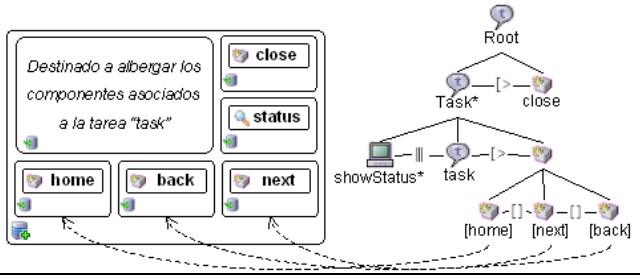
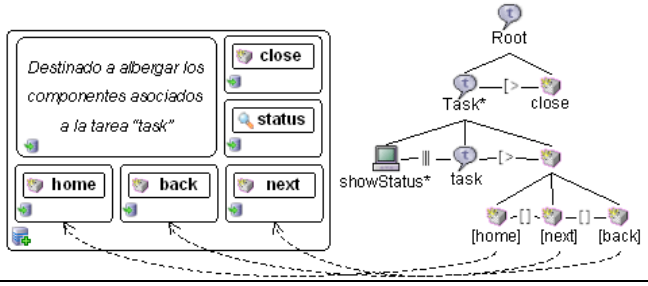
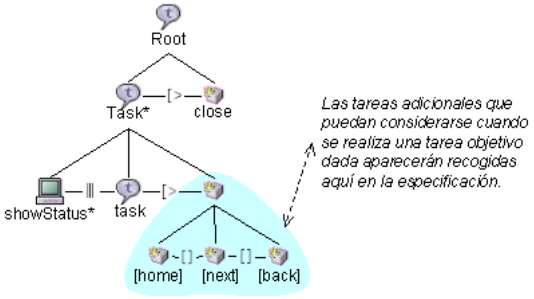
Hierarchical set	*Problem:* How should the information be organized? *Solution:* Show the data in a tree-like structure. 
Tabular set	*Problem:* How should the information be organized? Show the data in a table structure. 

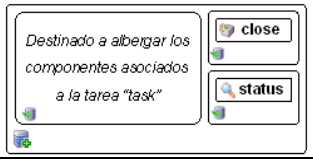
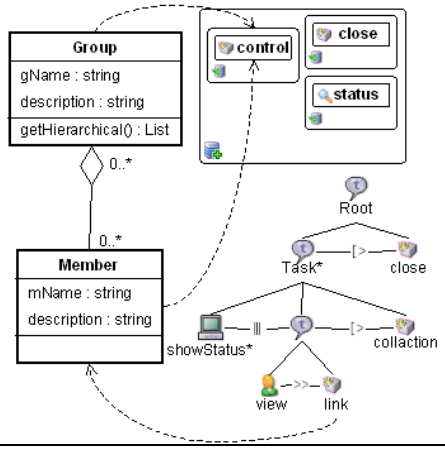
Chart or Graph	*Problem:* How should the information be organized? *Solution:* Show the data plotted against time or some other variable. Plot it together with other variables for further comparison. 
Optional Detail On demand	*Problem:* When should these usually-unneeded items be presented to the user, and how? *Solution:* Up front, show the user that which is most important and most likely to get used. Details and further options which won't be needed most of the time -- say 20% or less of expected uses -- can be hidden in a separate space or working surface (another dialog, another piece of paper, behind a blank panel). 

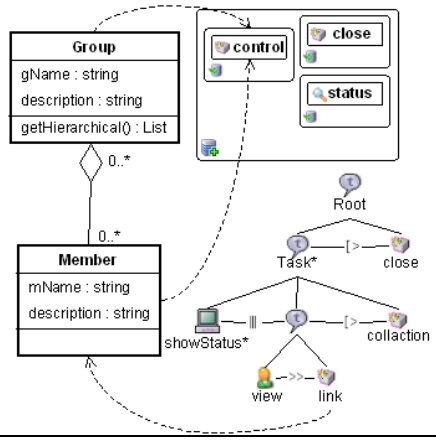
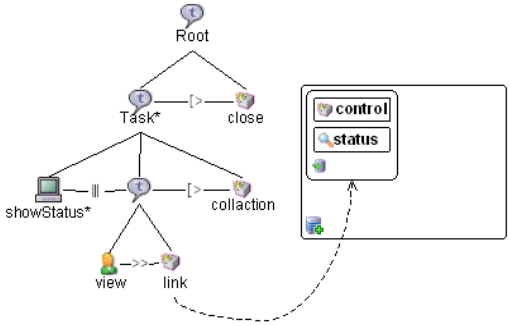
Disabled irrelevant things	*Problem:* How can the artifact steer the user away from actions that cannot or should not be taken, while still maintaining visual calm and stability? *Solution:* Disable the things which have become irrelevant. 
Pointer shows affordance	*Problem:* How can the artifact indicate that a visual entity represents an action that the user may take? *Solution:* Change the affordance of the thing as the pointer moves over it. 
Short description	*Problem:* How should the artifact present additional content, in the form of clarifying data or explanations of possible actions, to the users that need it? *Solution:* Show a short (one sentence or shorter) description of a thing, in close spatial and/or temporal proximity to the thing itself. 

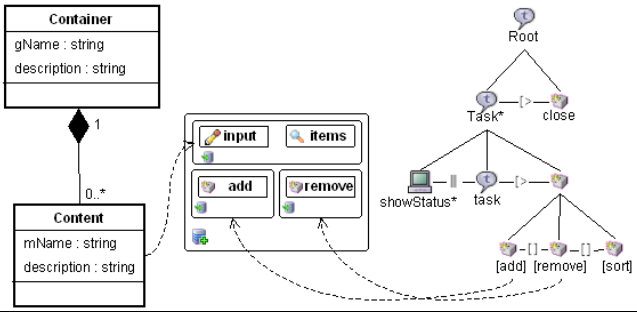
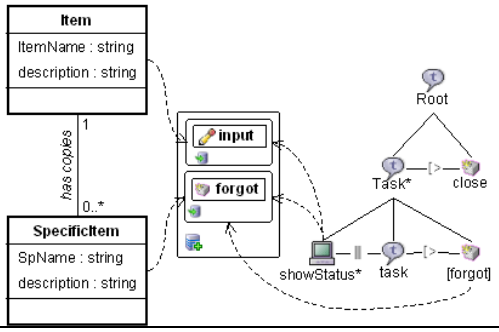
Sovereign posture	*Problem:* How should this artifact relate spatially to other artifacts that might share its space, and how can it best use the space it has? *Solution:* Allow the artifact to take up all the space it needs to get the job done efficiently and gracefully.
Helper posture	*Problem:* How should this artifact relate spatially to other artifacts that might share its space, and how can it best use the space it has? *Solution:* Use as much space as needed to make it comprehensible, but no more; focus tightly on the activity by excluding all but the commonest actions, information, etc. from the top level, but let those common ones take whatever space they need.
Background posture	*Problem:* How should this artifact relate spatially to other artifacts that might share its space, and how can it best use the space it has? *Solution:* Make the artifact small, relative to the other primary activities going on at the same time, and keep it inobtrusive.
Central working surface	*Problem:* How should the artifact's working surfaces be organized? *Solution:* Create one working surface where the artifact's major functions are collected together; if most of the work can actually be done there, so much the better. 
Tiled working surfaces	*Problem:* How should the artifact's working surfaces be organized? *Solution:* Place the working surfaces together in a plane, such that they do not obscure each other, and show the whole thing to the user. 

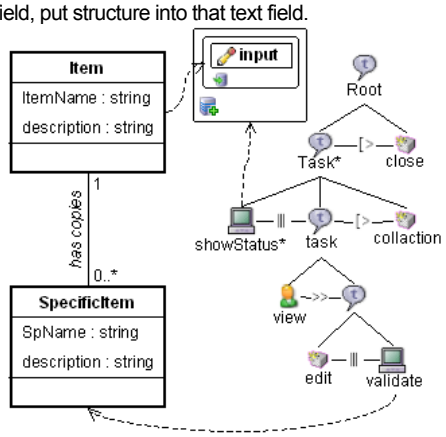
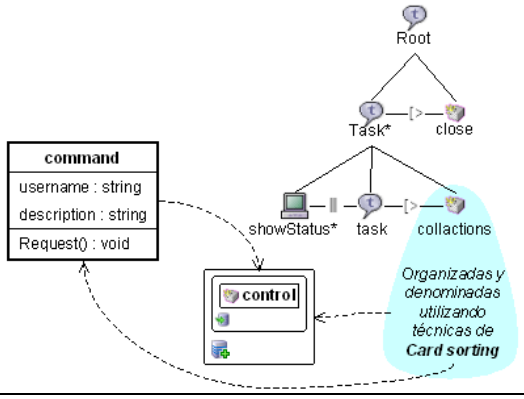
Stack of working surfaces	*Problem:* How should the artifact's working surfaces be organized? *Solution:* Stack the surfaces together. Label each surface with a unique and recognizable name or icon (or let the user pick the label), and visually cluster those labels together near the stack. 
Pile of working surfaces	*Problem:* How should the artifact's working surfaces be organized? *Solution:* Stack the surfaces loosely so that they obscure each other most of the time, but so that one or more surfaces of the user's choosing can be on top. 
Map of navigable spaces	*Problem:* How can the artifact help a user navigate effectively and remain oriented? *Solution:* Provide a map or diagram of the *Navigable Spaces* relevant to the artifact. (véase Navigable spaces)
Clear entry points	*Problem:* How does the user know where to start? *Solution:* Provide a small set of well-defined, clearly-named entry points to the network of Navigable Spaces. (implementado a nivel concreto, utilizar etiquetas significativas)
Color-Coded sections	*Problem:* How can an artifact both give users a sense of place, and also tell them where they are, within a large network of spaces? *Solution:* Use color to identify the major sections of the artifact. (implementado a nivel concreto o asignando colores a los OIA definidos)

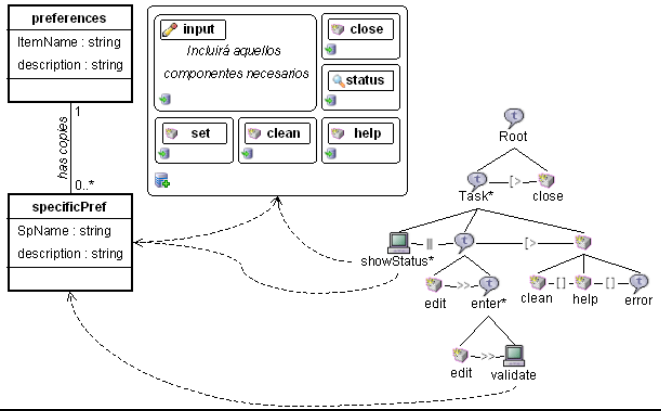
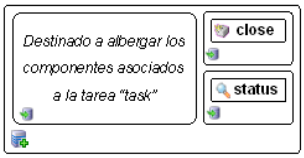
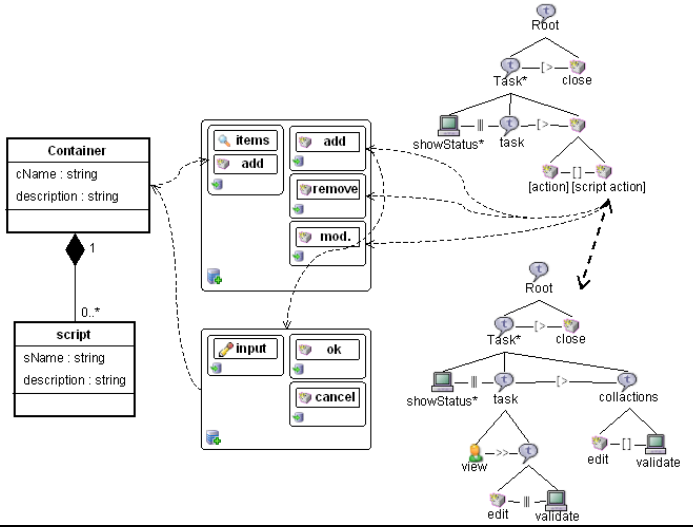
Go back one step	*Problem:* How can the artifact make navigation easy, convenient, and psychologically safe for the user? *Solution:* Provide a way to step backwards to the previous space or state. 
Go back to a safe place	*Problem:* How can the artifact make navigation easy, convenient, and psychologically safe for the user? *Solution:* Provide a way to go back to a checkpoint of the user's choice. 
Convenient environment actions	*Problem:* How should the artifact present these actions? *Solution:* Group these actions together, label them with words or pictures whose meanings are unmistakable, and put them where the user can easily find them regardless of the current state of the artifact. 

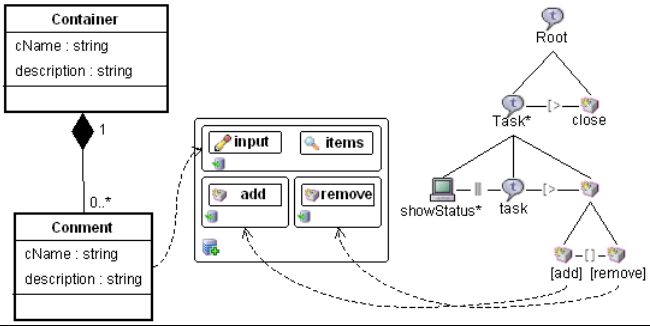
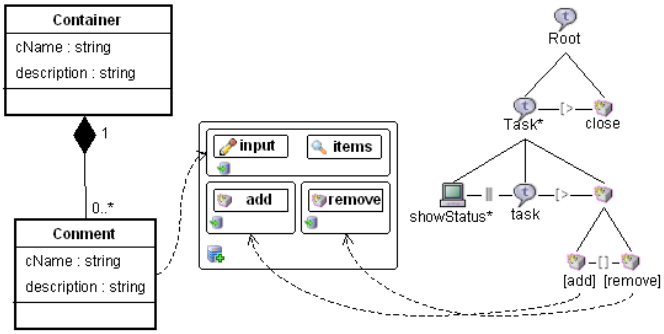
Localized Object Actions	*Problem:* How should artifact present the actions that may be taken on those objects? *Solution:* Group object actions together, even more so than for Convenient Environment Actions, and spatially localize them to the object. 
Actions for multiple objects	*Problem:* How can the artifact make repetitive tasks easier for the user? *Solution:* Allow the action to be performed "in parallel" across a set of user-selected objects. (utilizar el patrón Observer para implementar la notificación múltiple)
Choice from a small set	*Problem:* How should the artifact indicate what kind of information should be supplied? *Solution:* Show all the possible choices up front, show clearly which choice(s) have been made, and indicate unequivocally whether one or several values can be chosen. 

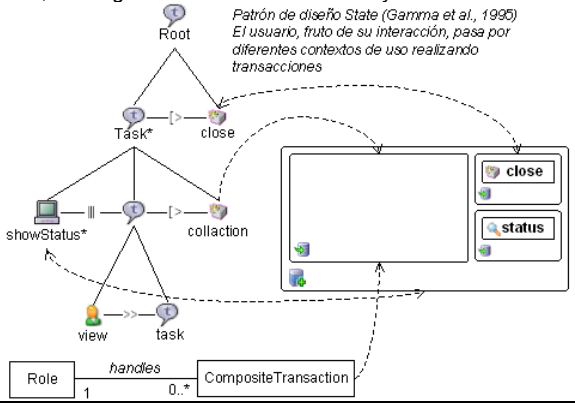
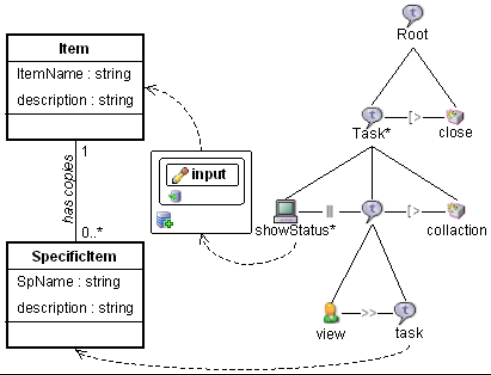
Choice from a large set	*Problem:* How should the artifact indicate what kind of information should be supplied? *Solution:* Clearly show the selected value up front; organize the set of possible values, but hide them nearby if they take up too much space. 
Sliding scale	*Problem:* How should the artifact indicate what kind of information should be supplied? *Solution:* Show the range of values visually, as with a line or arc; show the current value as a location in that line or arc, and if the value is settable by the user, make that location directly changeable by the user, as with a slider or knob -- or by simply touching or clicking on the desired value. 

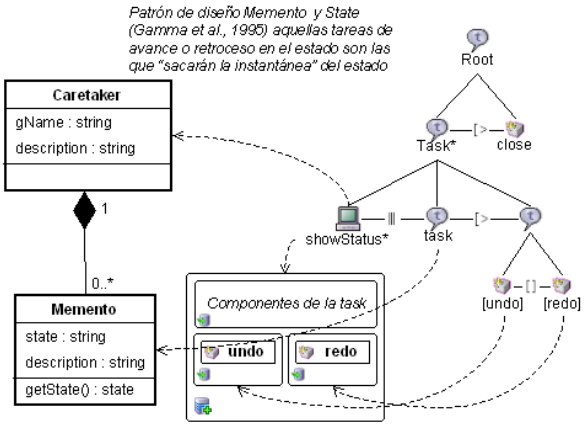
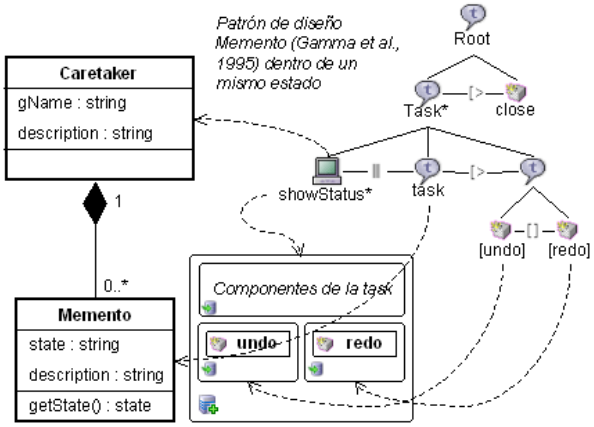
Editable collection	*Problem:* How should the artifact indicate what the user is supposed to do with that collection? *Solution:* Show the collection to the user, along with obvious ways to remove or change the position of each item. To add an item, make it eminently clear whether the user should obtain the item before or after the "add" command or gesture. 
Forgiving text entry	*Problem:* How does the artifact indicate what kind of information should be supplied? *Solution:* Allow the user to enter text in any recognizable format for that context. 

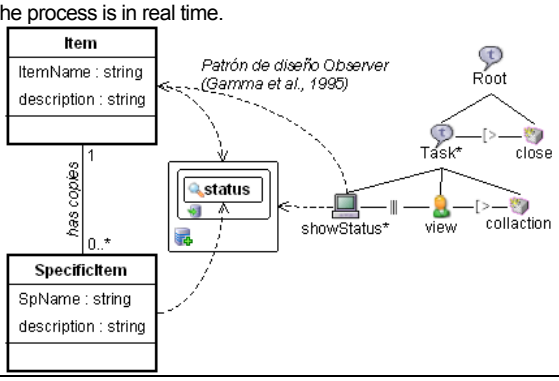
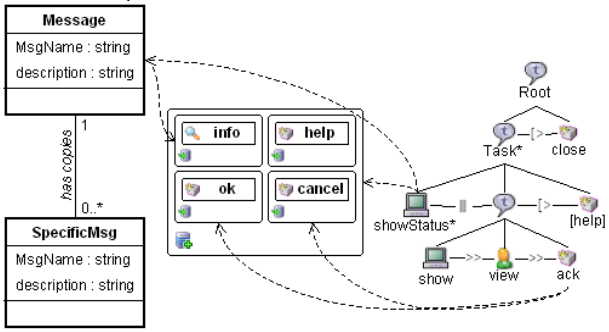
Structured text entry	*Problem:* How does the artifact indicate what kind of information should be supplied? *Solution:* Rather than letting a user enter information into a blank and featureless text field, put structure into that text field. 
Toolbox	*Problem:* How should the artifact present the actions that the user may take? *Solution:* Keep tools together, put them physically close to the user's working surface, and make sure they are distinct from other actions that the user may take. 

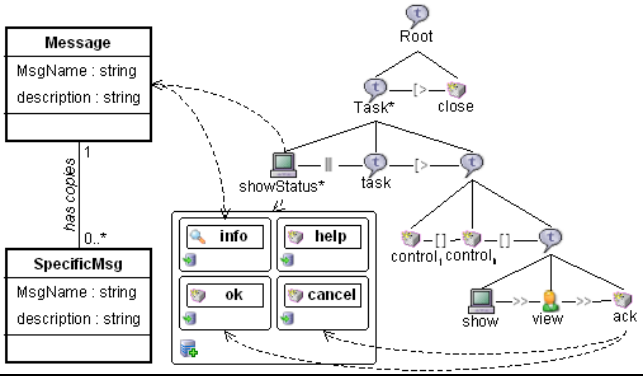
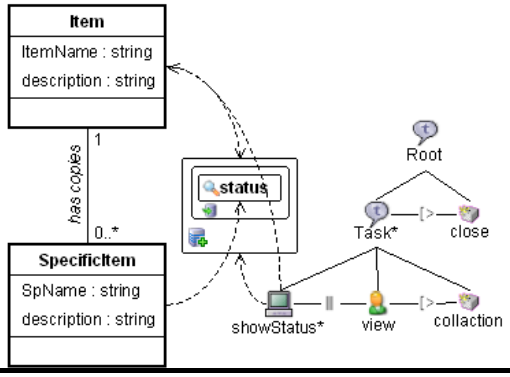
User preferences	*Problem:* How does the artifact present the actions that the user may take? *Solution:* Provide a place or working surface where users can pick their own settings for things like language, fonts, icons, color schemes, and use of sound. 
Personal object space	*Problem:* How should the items in question be organized? *Solution:* Allow users to place things where they want, at least in one dimension but preferably in two. 
Scripted action sequence	*Problem:* How can the artifact make repetitive tasks easier for the user? *Solution:* Provide a way for the user to "record" a sequence of actions of their choice, and a way to easily "play them back" at any time. 

User's annotations	*Problem:* How can the artifact help preserve the user's hard-won understanding from one use session to the next? *Solution:* Support ways for users to add their own comments and other annotations to the artifact. 
Bookmarks	*Problem:* How can the artifact support the user's need to navigate through it in ways not directly supported by the artifact's structure? *Solution:* Let the user make a record of their points of interest, so that they can easily go back to them later. 

Repeated framework	*Problem:* How should the content be presented in a unified and consistent way, so that the user can easily navigate through it and quickly become familiar with it? *Solution:* Design a simple, flexible visual framework for the content, then repeat it on every page or working surface; position the content within that framework, allowing the form of the content to vary as needed. 
Good defaults	*Problem:* How does the artifact indicate what kind of information should be supplied? *Solution:* Supply reasonable default values for the fields in question. 

Remembered state	*Problem:* How can the artifact help save the user time and effort? *Solution:* Design the artifact so that it can remember its state from session to session. 
Interaction history	*Problem:* Should the artifact keep track of what the user does with it? If so, how? *Solution:* Record the sequence of interactions as a "history." 

Progress indicator	*Problem:* How can the artifact show its current state to the user, so that the user can best understand what is going on and act on that knowledge? *Solution:* Show the user a status display of some kind, indicating how far along the process is in real time. 
Important Message	*Problem:* How should the artifact convey this information to the user? *Solution:* Interrupt whatever the user is doing with the message, using both sight and sound if possible. 

Reality check	*Problem:* How can the artifact protect itself and the user from these kinds of actions, while allowing the user to have the final say over whether or not an action is performed? *Solution:* Before the action is performed, tell the user what the side effects of the action will be, and ask the user to confirm that that's what they really want to do. 
Demonstration	*Problem:* How can the user learn how to use the artifact? *Solution:* Demonstrate how to do it. 

Apéndice D. Criterios ergonómicos

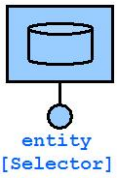
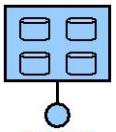
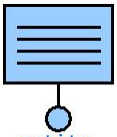
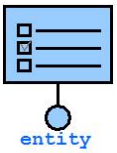
Los criterios ergonómicos utilizados en esta tesis doctoral son los recogidos en *Bastien, J. M. C., & Scapin, D. L. (1993). Ergonomic criteria for the evaluation of human-computer interfaces (Report No. 156). Rocquencourt, France: Institut National de Recherche en Informatique et en Automatique*.

Tabla. Descripción de los criterios ergonómicos propuestos

Criteria	Description
Guidance	User *Guidance* refers to the means available to advise, orient, inform, instruct, and guide the users throughout their interactions with a computer (messages, alarms, labels, etc.), including from a lexical point of view. The criterion *Guidance* is subdivided into four criteria: *Prompting*, *Grouping / Distinction of Items*, *Immediate Feedback*, and *Legibility*.
Workload	The criterion Workload concerns all interface elements that play a role in the reduction of the users' perceptual or cognitive load, and in the increase of the dialogue efficiency. The criterion Workload is subdivided into two criteria: Brevity (which includes Concision and Minimal Actions), and Information Density.
Explicit control	The criterion Explicit Control concerns both the system processing of explicit user actions, and the control users have on the processing of their actions by the system. The criterion Explicit Control is subdivided into two criteria: Explicit User Action, and User Control.
Adaptability	The adaptability of a system refers to its capacity to behave contextually and according to the users' needs and preferences. The criterion Adaptability is subdivided into two criteria: Flexibility and User Experience.
Error management	The criterion Error Management refers to the means available to prevent or reduce errors and to recover from them when they occur. Errors are defined in this context as invalid data entry, invalid format for data entry, incorrect command syntax, etc. The criterion Error Management is subdivided into three criteria: Error Protection, Quality of Error Messages, and Error Correction.
Consistency	The criterion Consistency refers to the way interface design choices (codes, naming, formats, procedures, etc.) are maintained in similar contexts, and are different when applied to different contexts.
Significance of codes	The criterion Significance of Codes qualifies the relationship between a term and/or a sign and its reference. Codes and names are significant to the users when there is a strong semantic relationship between such codes and the items or actions they refer to.
Compatibility	The criterion Compatibility refers to the match between users' characteristics (memory, perceptions, customs, skills, age, expectations, etc.) and task characteristics on the one hand, and the organisation of the output, input, and dialogue for a given application, on the other hand. The criterion Compatibility also concerns the coherence between environments and between applications.

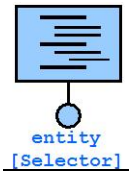
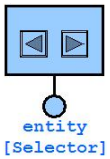
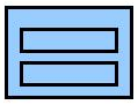
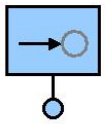
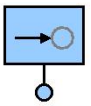
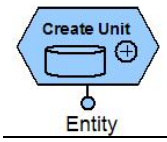
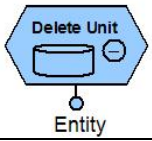
Apéndice E. Patrones en WebML

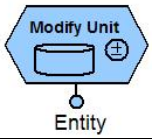
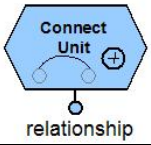
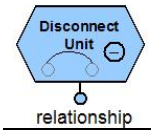
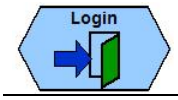
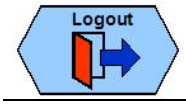
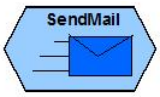
Se recogen en este apéndice los patrones definidos en el lenguaje WebML[1] (*Web Modeling Language*). WebML es una notación visual para el diseño de aplicaciones Web complejas que usan datos intensivamente. Provee especificaciones gráficas formales para un proceso de diseño completo que puede ser asistido por herramientas de diseño visuales, concretamente por la herramienta WebRatio[2].

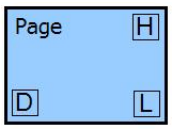
Elemento WebML	Descripción	Propiedades
Dato 	Sirve para mostrar un dato simple perteneciente a una entidad	Nombre Entidad fuente Selector (opcional) Atributos incluidos
Multidato 	Sirve para mostrar múltiples objetos de una entidad juntos	Nombre Entidad Selector (opcional) Atributos incluidos Cláusula de orden
Índice 	Presenta múltiples elementos de una entidad como una lista	Nombre Entidad Selector (opcional) Atributos incluidos Cláusula de orden
Multielección 	Es una variante de la unidad índice donde cada elemento de la lista está asociada con un checkbox permitiendo a los usuarios seleccionar múltiples elementos	Nombre Entidad Selector (opcional) Atributos incluidos Cláusula de orden

[1] Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a Modeling Language for Designing Web Sites". WWW9 Conference, Amsterdam, May 2000. http://www.webml.org

[2] WebRatio: Your CASE tool. http://www.webratio.com

Jerarquía  	Es una variante de la unidad índice en la que las entradas están organizadas en árbol	Nombre Para cada nivel Entidad Selector (opcional) Atributos incluidos Cláusula de orden
Scroller  	Proporciona los comandos para moverse a través de un conjunto de elementos	Nombre Entidad Selector (opcional) Factor de bloqueo Cláusula de orden
Formulario 	Permite la entrada de datos	Nombre Para cada campo Nombre Tipo Valor inicial Modificabilidad Predicado de validez
Set  	Permite la asignación de valores a los parámetros globales	Parámetro global
Get  	Permite recuperar los valores asignados a parámetros globales	Parámetro global
De creación  	Permite la creación de una nueva instancia	Nombre Entidad Conjunto de valores asignados
De borrado  	Permite borrar uno o más elementos de una entidad	Nombre Entidad Selector

De modificación 	Permite actualizar uno o más elementos de una entidad dada	Nombre Entidad Selector Conjunto de valores asignados
De conexión 	Sirve para crear nuevas instancias de una relación	Nombre Papel de la relación Selector de la entidad fuente Selector de la entidad destino
De desconexión 	Sirve para borrar instancias de una relación	Nombre Papel de la relación Selector de la entidad fuente Selector de la entidad destino
De login 	Sirve para verificar la identidad de un usuario para acceder a un sitio	Parámetros Usuario Contraseña
De logout 	Sirve para enviar al usuario a una página sin control de acceso	Ninguno
De envío de correo 	Permite el envío de un correo electrónico	Parámetros Emisor Receptor Asunto Cuerpo Adjuntos
De operación definida por el usuario 	Permite al usuario definir operaciones cuyos parámetros serán definidos por el diseñador	Definidos por el diseñador
Transacción	Permite la especificación de una secuencia de pasos que se ejecutarán automáticamente	Ninguno

Página 	Representa la interfaz visualizada por el usuario. Está constituida de unidades y subpáginas	Nombre Marca de visibilidad Contenido: unidad, páginas alternativas
Páginas alternativas	Especifica parte de la pantalla que puede contener elementos alternativos cada uno especificado como una página diferente	Páginas anidadas Páginas anidadas por defecto
Páginas simultáneas	Se utiliza para dividir la página contenida en una pantalla en porciones	Páginas anidadas
Área 	Es un container de páginas, también, y alternativamente, de otras áreas, sirve para dar una organización jerárquica al hipertexto	Nombre Marca de visibilidad Contenido: páginas y sub-áreas Página por defecto o sub-área
Sitio Web	Representa al sitio web	Nombre Contenido: páginas y áreas Homepage
Enlace	Representa una conexión entre dos unidades o páginas. Puede ser automático o de transporte, el primero no necesita la intervención del usuario a diferencia del segundo. También puede ser OK o KO para señalar operaciones de éxito o fracaso en la realización de cierta operación asociada al mismo	Carácter: automático o de transporte: Nombre Elemento fuente Elemento destino Tipo de enlace (normal, automático o de transporte) Parámetros del enlace Nombre Fuente En los enlaces OK y KO Nombre Elemento fuente Elemento destino Parámetros de enlace

Apéndice F. La metodología IDEAS

El principal objetivo de la propuesta IDEAS[1] pasa por proponer un modelo y un lenguaje para la especificación y desarrollo de interfaces de usuario, definiendo una aproximación metodológica propia y dándole un soporte tecnológico que facilita su integración en el proceso de desarrollo software de acuerdo a los principios del paradigma orientado a objetos.

La propuesta Ideas está soportada formalmente por el lenguaje de especificación OASIS[2], al que enriquece para dar soporte a la especificación de interfaces de usuario gráficas.

Fundamentalmente, los principales logros de IDEAS pasan por:

- Definir un modelo de interfaz de usuario completo que capta todos los aspectos relativos a la interfaz, así como la ontología asociada a dicho modelo con el que define de forma precisa la semánica asociada a cada uno de los modelos.
- Integra el proceso de desarrollo de GUIs dentro del entorno de desarrollo de software

En Ideas se contemplan criterios de calidad centrados en la usabilidad a través de la consideración de modelos que permiten conocer al usuario, tales como los de casos de uso, tareas y usuario. Los dos primeros permiten describir de forma precisa, la funcionalidad del sistema y las tareas que los usuarios pueden llevar a cabo. El último, permite describir las características individuales de los diferentes tipos de usuarios que interactuarán con el sistema.

Las principales características que presenta el entorno metodológico asociado a Ideas son:

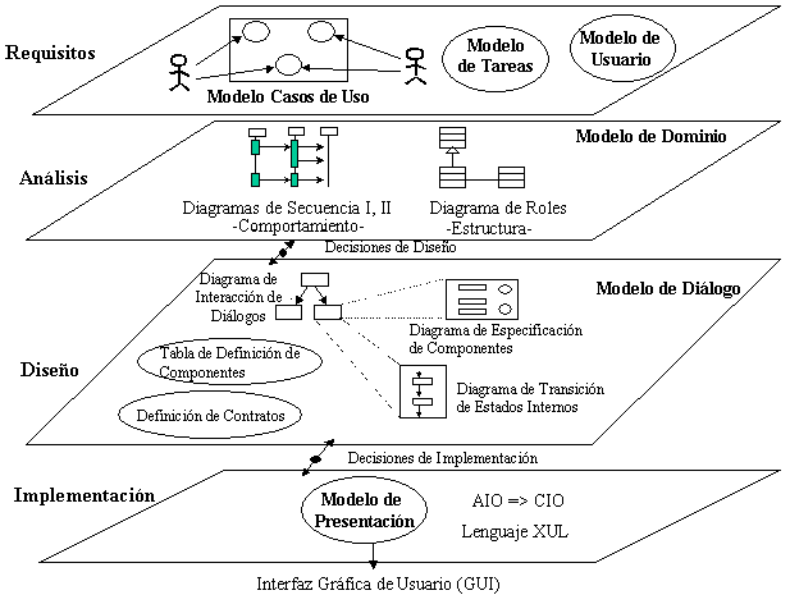
- Se trata de una aproximación centrada en el usuario.
- Está basada en modelos declarativos.
- Utiliza la notación UML, con algunas modificaciones en ciertos casos, para la construcción de algunos diagramas.
- Sigue la arquitectura de tres capas que separa la interfaz de usuario de la lógica del negocio.

[1] Lozano, M. (2001). Entorno metodológico orientado a objetos para la especificación y desarrollo de interfaces de usuario. Tesis doctoral realizada en la Universidad Politécnica de Valencia.

[2] Pastor, O. (1992). Diseño y desarrollo de un entorno de producción automática de software basdo en el modelo orientado a objetos. Tesis doctoral realizada en la Universidad Politénica de Valencia.

- Incorpora el uso de contratos para formalizar la relación entre la interfaz de usuario y los objetos de dominio.

La propuesta IDEAS puede resumirse de forma gráfica mediante la figura mostrada seguidamente:



En la figura se aprecia como la metodología abarca todas las fases del desarrollo de un producto software, desde los requisitos hasta la implementación ejecutable de la interfaz de usuario. A cada una de las fases de desarrollo se asocian una serie de diagramas basados, en unos casos, en los propuestos en UML y, en otros, en nuevos diagramas que se han estimado necesarios. Ejemplo de ellos son los diagramas de interacción de diálogos (asociados al comportamiento) y los diagramas de especificación de componentes (asociados al estado).

La especificación abstracta de una interfaz en Ideas se lleva a cabo utilizando una serie de objetos de interacción abstractos (AIOs). En Ideas se han definido tres tipos de AIOs: el tipo *componente*, el tipo *herramienta de visualización* y el de *herramienta de control*. Una componente es una colección de AIOs que se agrupan semánticamente en el contexto de una aplicación. Las herramientas de visualización se utilizan para mostrar información al usuario y para que el usuario pueda introducir información. Las herramientas de control sirven para controlar el funcionamiento de la aplicación, es decir, aceptar acciones, cancelar acciones y navegar entre los distintos diálogos.

Apéndice G. El marco de especificación de especificación y transformación usiXML[1]

Detrás de usiXML (acrónimo de *USer Interface eXtensible Markup Language*) se engloba algo más que un mero lenguaje de especificación de interfaces de usuario. La idea que subyace detrás de dicho lenguaje es dar soporte a un doble marco de trabajo que permita el desarrollo de interfaces de usuario. Este marco de trabajo por una parte es ontológico, ya que en el que se contemplan todos los conceptos relevantes para el desarrollo de interfaces de usuario, y por otro es metodológico, ya que pone en práctica el marco anterior mediante un paradigma de desarrollo de interfaces de usuario denominado *multi-path development of UIs*, caracterizado por:

- Estar dirigido por transformaciones. La metodología de desarrollo está constituida por una secuencia de fases, en cada una de ellas se realizan transformaciones para pasar de una fase a otra.
- La metodología propuesta, siendo sistemática, no responde a una trayectoria o traza definida, sino que es de *multiple-path*, es decir, el método puede aplicarse describiendo diferentes trazas o secuencias de pasos, consiguiendo igualmente alcanzar su objetivo con éxito.

Desde el punto de vista de la especificación de la interfaz de usuario, en usiXML, se establecen tres niveles de abstracción, así es posible trabajar a nivel de:

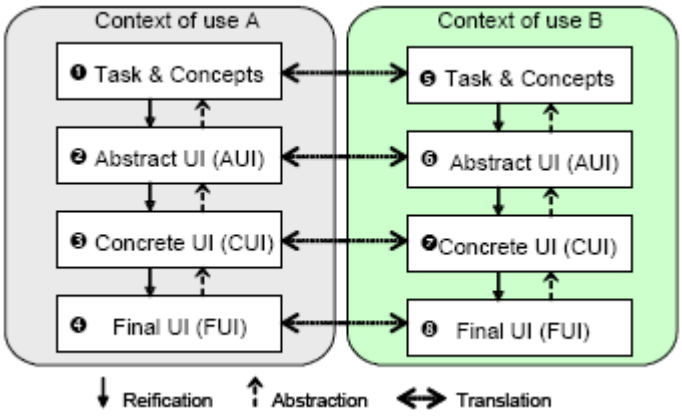
- interfaz de usuario abstracta (abstract UI) en la que la interfaz se define de forma independiente de la plataforma y de la implementación, de forma concreta,
- interfaz de usuario concreta (concrete UI) en la que la interfaz se describe de forma independiente de la posterior implementación,
- interfaz de usuario final (final UI) que describe la interfaz de usurio utilizando un lenguaje de programación concreto (java, html, etc.).

De forma similar a Ideas y aunque entrando en algo más de detalle, en usiXML (permítase la licencia de extender el nombre dado al lenguaje asociado a la metodología para hacer referencia a la propia propuesta metodológica) se abunda en la especificación del modelo de diálogo a través del uso de notaciones matemáticas basadas en grafos, aportándose, como

[1] Limbourg, Q. (2004). Multi-Path Development of User Interfaces. Thesis. University catholique de Louvain

novedad, los mecanismos necesarios para manipular la estructura de grafo asociada.

En definitiva, en usiXML se descompone cualquier actividad de desarrollo en una secuencia de pasos que se traducen en la realización de transformaciones del artefacto manteniéndose dentro de los límites de la correspondiente fase. En este contexto, un *camino de desarrollo* se define como una composición de pasos de desarrollo, identificándose tres caminos típicos: ingeniería directa, ingeniería inversa y adaptación de contexto.



Otro de los aspectos que hemos considerado interesantes de usiXML y que utilizados en la realización de la tesis doctoral presentada en este documento es su mecanismo de descripción de interfaces de a nivel abstracto. En usiXML, en su modelo de interfaz de usuario a nivel abstracto se consideran dos posibles objetos de interacción abstracta (AIOs): los *containers* y los *components*. A su vez, estos últimos pueden contener hasta cuatro tipos de facetas ligadas a acciones de entrada (*input*), de salida (*output*), de navegación (*navegation*) y de control (*control*). Para la realización de modelos de presentación abstracto se desarrolló, entre otras actividades llevadas a cabo en la estancia realizada en Louvain-La Neuve, un editor que permite realizar especificaciones de interfaz de usuario a nivel abstracto que pueden ser almacenadas utilizando usiXML. Dicho editor está incluido en la herramienta IDEALXML.

Apéndice H. IDEALXML: un entorno[1] para la documentación, gestión y uso de la experiencia disponible en forma de patrones

IDEALXML es un entorno desarrollado para facilitar la documentación de distintos tipos de patrones, en especial de los patrones de interacción. Tradicionalmente, dichos patrones se documentan utilizando descripciones en lenguaje natural donde quedan recogidos diferentes secciones significativas (véase el apartado del capítulo cuarto dedicado a los patrones de interacción en el que se trata PLML). Una parte importante de la descripción la constituyen los ejemplos asociados a cada patrón donde puede verse de forma gráfica una posible implementación de la solución que el patrón documenta.

Una de las propuestas realizadas en esta tesis doctoral ha pasado por enriquecer la propuesta PLML, sustituyendo cualquier sección de la misma que fuera indefinida (ANY), y añadiendo, entre otras consideraciones, una sección estructura donde se recogiesen aquellos modelos (dominio, tareas, presentación y mapping) significativos para la documentación del patrón. Para la elaboración de cada uno de los posibles modelos que se pueden asociar a un patrón se han identificado aquellas notaciones más representativas respecto a cada labor. Así, las descripciones de un modelo de dominio tienen asociado los diagramas de clases como elemento de documentación habitual. Los modelos de tareas se representan utilizando la notación ConcurTaskTrees[2] y para la especificación del modelo de presentación se utiliza la notación propuesta en usiXML (véase el apéndice B).

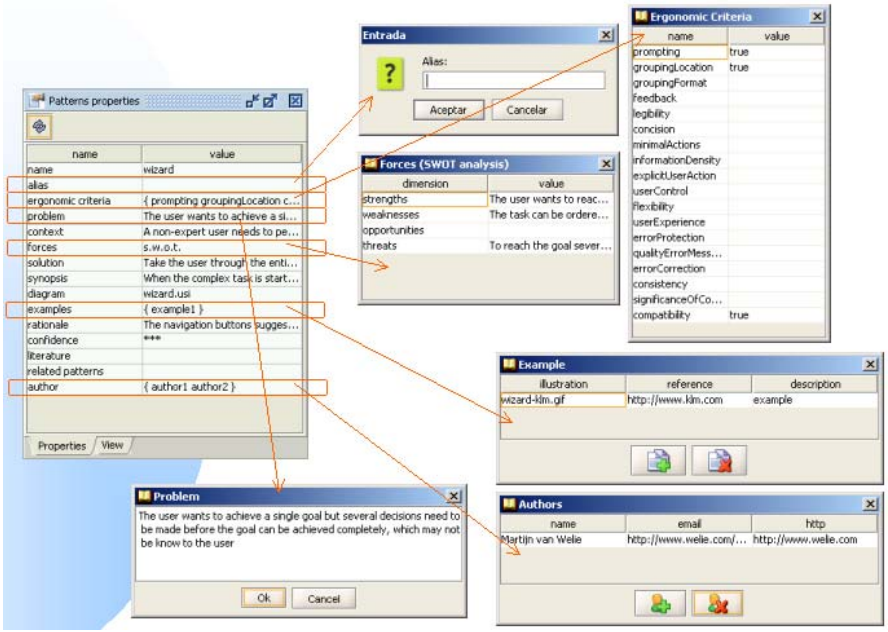
Con el entorno IDEALXML se pueden documentar, gestionar, recuperar, compartir y utilizar un catálogo de patrones, y también es posible asignar a cada patrón criterios de calidad que afectan a la usabilidad y se constituyen en modelo de calidad (véase el capítulo tercero).

El entorno IDEALXML sirve como entorno de especificación de un producto software en el que puede considerarse la experiencia modelada y asociada a los patrones disponibles en el repositorio asociado en cada momento al entorno. Los patrones se consideran modelos en IDEALXML, pudiendo ser enriquecidos y utilizados para la especificación de distintos productos software.

---

[1] IdealXML se encuentra disponible en la dirección http://www.info-ab.uclm.es/personal/fmontero/idealxml.

[2] Paternò, F., Model Based Design and Evaluation of Interactive Applications, Springer-Verlag, London, 1999.

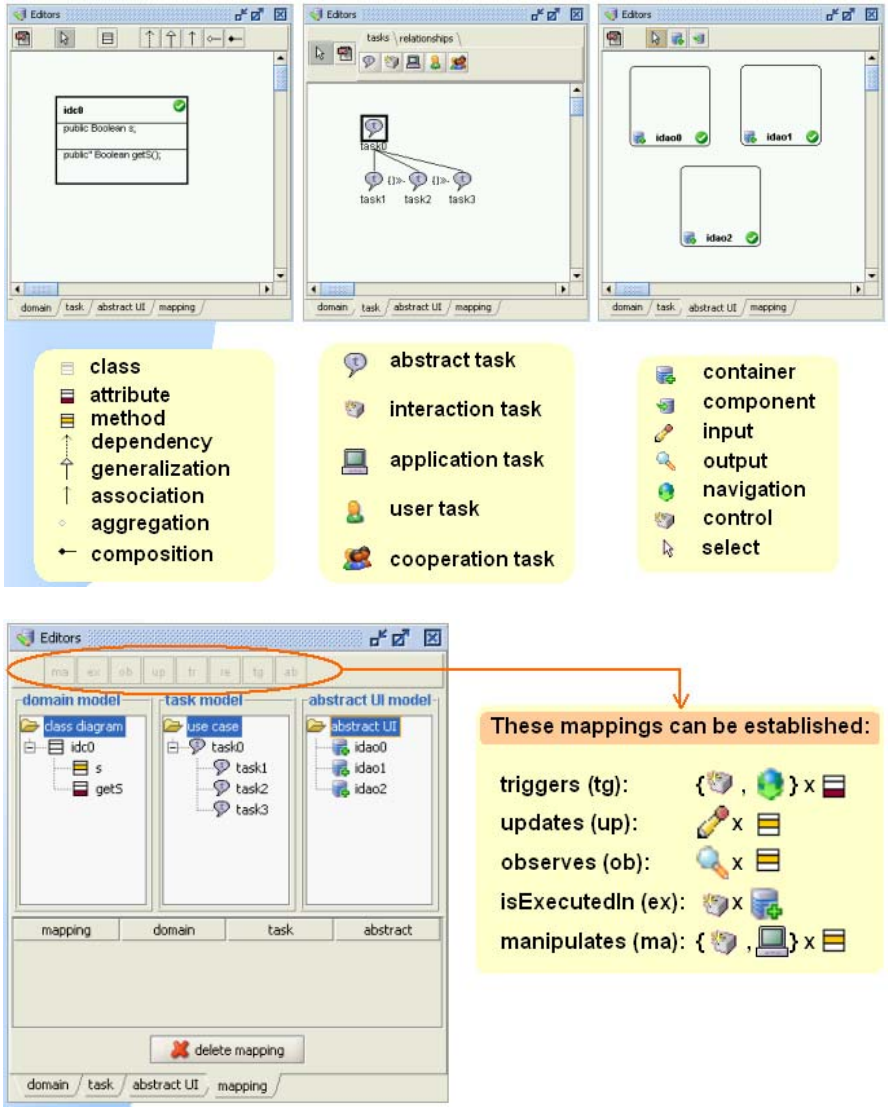


En la figura anterior se muestran las diferentes secciones que describen un patrón en IDEALXML, las mismas aparecen recogidas en la ventana titulada *Patterns properties*. Las diferentes flechas que parten de algunas de las secciones de dicha ventana muestran, a su vez, los diálogos asociados a la edición del correspondiente valor afín a cada una de ellas. Una de las secciones cuya descripción no aparece reflejada en la figura mostrada arriba es la sección *diagram*, con ella es posible asociar diferentes modelos, concretamente los modelos de dominio, tareas, presentación y mapping, a cada patrón.

Los diferentes editores implementados en IDEALXML para permitir la especificación de los modelos asociados se muestran en las figuras recogidas en la página siguiente, junto con cada editor aparecen reflejados los posibles elementos de modelado que pueden utilizarse en cada uno de ellos y su representación gráfica.

Los patrones editados y almacenados utilizando IDEALXML se guardan utilizando XML siguiendo el esquema recogido en el capítulo quinto de este documento. El diseño y posterior implementación de IDEALXML se ha hecho contemplando la posiblidad de que otras notaciones pudieran utili-

zarse para describir un patrón de interacción. Además, en su diseño e implementación se han utilizado diferentes patrones de diseño[1].



[1] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). Design patterns: Elements of reusable object-oriented software. Reading, MA: Addison Wesley.

Esta tesis doctoral identifica tendencias y puntos comunes que pueden establecerse entre Ingeniería del Software (IS) e Interacción Persona-Ordenador (IPO), con ello se consigue dar soporte al proceso de desarrollo de productos software de calidad.

Una de las principales hipótesis de la que se parte en esta tesis es la que establece que el *gap* semántico que se identifica entre ambas disciplinas (IS e IPO) no es tal, y que el verdadero *gap* surge cuando se consideran las fases de análisis de requisitos y de diseño en el proceso de elaboración de un producto software.

En esta tesis doctoral, las herramientas que se utilizan para dar soporte a las fases mencionadas pasan por:

- caracterizar y considerar la calidad de un producto software, entendiendo como tal aquella que directamente percibe el usuario (la usabilidad)
- utilizar la experiencia disponible, especialmente los patrones de interacción, para dar soporte a las mismas, así como a las fases de mantenimiento y evaluación.

Ambos elementos, calidad y experiencia, se consideran conjuntamente a través de la elaboración de un modelo de calidad centrado en la usabilidad. Los patrones de interacción, a su vez, se documentan de forma *generativa*, es decir, son modelos que se pueden aprender y utilizar una y otra vez.

LoUISE
Laboratorio de Interacción con el Usuario e Ingeniería del Software

TESIS DOCTORAL