Patrones de interfaz de usuario para la exploración orientada a objetos

Pedro J. Molina[1], Ismael Torres[1], Oscar Pastor[2]

[1] CARE Technologies S.A.
Ctra. Las Marinas, Km 2.
Urb. Retiro III
03700 Denia, España.
{pjmolina|itorres}@care-t.com

[2] Dpto. de Sistemas
Informáticos y Computación
Universidad Politécnica de Valencia
Valencia, España.
opastor@dsic.upv.es

Resumen

Los patrones conceptuales aplicados a toma de requisitos de interfaces de usuario pueden proporcionar un lenguaje común a los miembros del equipo de desarrollo de interfaces de usuario. El presente trabajo propone una serie de patrones y ejemplifica su uso y aplicaciones a través de un sencillo caso de estudio desde la especificación hasta la implementación.

Palabras clave

Patrones conceptuales, interfaces de usuario, interacción persona-ordenador, modelado conceptual, orientación a objetos.

1 Introducción

Los patrones se han revelado como un campo con muchas aplicaciones en el mundo de la Ingeniería del Software. Se han beneficiado de su uso: el campo de diseño de software [Gamma94], la arquitectura de aplicaciones [Coplein99] o los ambientes Web por citar algunos ejemplos. En particular, en el ámbito de las interfaces de usuario, también existen trabajos como [Weile01] donde se tipifican patrones de diseño para el desarrollo de interfaces de usuario. Sin embargo, desde el punto de vista del Modelado Conceptual, la aplicación de patrones a la especificación de interfaces de usuario ha sido un campo poco explorado y que pretendemos abordar.

El presente trabajo muestra una serie de patrones conceptuales identificados en el proceso de análisis de interfaces de usuario para aplicaciones de gestión. Su uso en el ámbito conceptual como primitivas de construcción facilitará la fabricación de tales sistemas.

Por si mismos, los patrones aportan un lenguaje común de referencia para las personas implicadas en un desarrollo. El uso de patrones como ideas bien documentadas permiten salvar ambigüedades o mal interpretaciones frecuentes en el lenguaje natural. Los patrones mejoran la comunicación y compresión entre los componentes del equipo de desarrollo. Al mismo tiempo, los patrones conceptuales al estar expresados en términos del dominio del problema permiten validar los requisitos con el usuario final.

El trabajo comienza presentando un pequeño caso de estudio que servirá para ilustrar la aplicación de los patrones. Seguidamente se describirán los conceptos y los patrones empleados para a continuación presentar la especificación e implementación de la interfaz de usuario para el ejemplo ilustrado. Finalmente se proporcionarán las conclusiones y bibliografía empleada.

2 Caso de estudio

En esta sección presentaremos un pequeño caso de estudio para ilustrar los patrones propuestos. Comenzaremos presentando una breve especificación de requisitos seguida del modelo conceptual orientado a objetos construido para dar cuenta de la funcionalidad del sistema. El modelo conceptual propuesto es un modelo clásico donde no se recogen características propias de la interfaz de usuario. Dicho modelo será completado posteriormente.

2.1 Requisitos

Se pretende informatizar la gestión de un camping. El negocio principal del camping es el alquiler de bungalows y parcelas. Existen veinte bungalows de varios precios en función de sus características. Por otra parte se dispone de cien parcelas de precio diverso, dependiendo de la superficie de la parcela.

Las instalaciones del camping cuentan con servicios adicionales para mejorar la estancia de los clientes: aparcamiento, televisión, conexión eléctrica para las parcelas, servicios de limpieza de bungalows, lavadoras, etc. Además, para aumentar la oferta lúdica del camping, los clientes disponen de instalaciones deportivas que pueden ser usadas por los clientes de manera gratuita o pagando una pequeña cantidad de dinero. Las actividades deportivas que se pueden desarrollar en dichas instalaciones son: musculación, aeróbic, natación, fútbol, tenis y golf. Los precios de todos los servicios varían en función de la temporada y la categoría del cliente. Al finalizar una estancia se emite su factura correspondiente que podrá pagarse en efectivo o mediante tarjeta de crédito.

Se necesita disponer de una herramienta capaz de gestionar el camping: controlando las estancias de los clientes, así como la facturación y la disponibilidad de los recursos del camping.

2.2 Modelo Conceptual

A partir de la especificación inicial de requisitos, los analistas pueden construir un modelo conceptual clásico orientado a objetos para recoger los requisitos funcionales del sistema. La Figura 1 muestra las clases de análisis identificadas en el problema, así como sus relaciones.

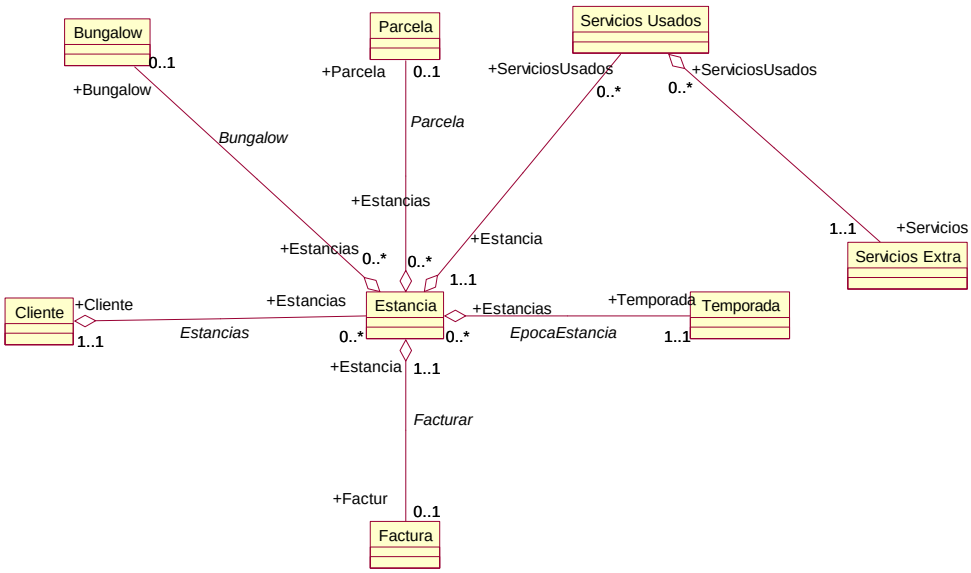

Figura 1. Esquema de clases de análisis construido para el caso de estudio.

La clase central del diagrama (`Estancia`) se encarga de almacenar la información de las estancias de los diferentes clientes en el camping. Relacionadas con la `Estancia` se encuentran las clases `Cliente` y `Factura` cuya misión es la típica en las aplicaciones de gestión. La clase `Temporada` representa temporadas a lo largo del año y fija los precios de las estancias. La clase `Parcela` y `Bungalow` mantienen recursos asignados a la `Estancia`. Finalmente la clase `ServiciosExtra` recoge los servicios adicionales del camping y las actividades deportivas, mientras que la clase `ServiciosUsados` mantiene la lista de servicios extra usados en una determinada estancia por un cliente.

Con un poco más de detalle, podemos observar en la Figura 2 los atributos y servicios (métodos) identificados en cada clase del dominio.

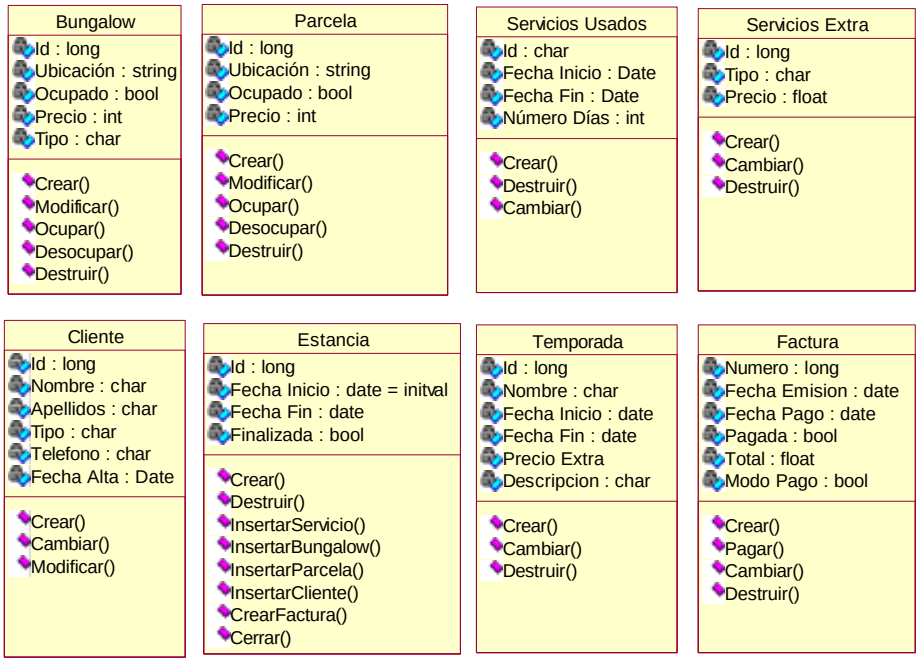

Figura 2. Atributos y servicios identificados para cada clase del caso de estudio.

El modelo ilustrado permite dar cuenta de los requisitos funcionales planteados en el caso de estudio. Su introducción en este punto facilitará la comprensión de los conceptos posteriores para completar la especificación con información de interfaz de usuario ya que los correspondientes ejemplos estarán siempre referidos a este caso de estudio.

3 Conceptos y patrones conceptuales

Antes de abordar la especificación de la Interfaz de Usuario para el caso de estudio, se presentará el conjunto de conceptos necesarios para abordarla. Se describirán las primitivas y los patrones conceptuales basados sobre estas primitivas.

3.1 Primitivas

En el desarrollo de interfaces de usuario orientadas a objetos, hemos identificado cinco primitivas recurrentes en el tratamiento de los objetos en la interfaz de usuario:

- Filtro o criterio de selección (*cómo buscar*)
- Criterio de ordenación (*cómo ordenar*)
- Conjunto de visualización (*qué propiedades mostrar*)
- Navegación ofertada (*qué información adicional puede ser consultada*)
- Acciones ofertadas (*qué acciones pueden llevarse a cabo sobre los objetos*)

3.1.1 Filtro

Un filtro o criterio de selección expresa una condición de búsqueda que el usuario emplea para localizar información que necesita sobre una población de objetos pertenecientes a una determinada clase.

Un filtro se define como una expresión lógica abierta, es decir, puede contener variables libres. En ejecución, el usuario debe dar valor a dichas variables antes de invocar un filtro. Esta asignación de valor nos permite alcanzar una fórmula cerrada que es evaluable a un valor lógico. La fórmula cerrada de filtro es evaluada entonces para cada objeto perteneciente a la población de la clase. Los objetos que satisfacen la condición de filtro constituyen el subconjunto de objetos que serán presentados al usuario.

Ejemplo:

Un filtro puede responder a la siguiente necesidad el usuario: "*Necesito conocer los bungalows disponibles de un determinado tipo.*" El ejemplo expresado en lenguaje formal OASIS [Pastor95] tendría la siguiente fórmula de filtro para la clase `Bungalow`:

$$\texttt{Ocupada=False AND Tipo = } v_{\texttt{Tipo}}$$

Donde se asume que:

- `Ocupada` es un atributo de la clase `Bungalow` que indica si el bungalow está ocupado o no en estos momentos.
- `Tipo` es un atributo de la clase `Bungalow` que indica el tipo del Bungalow.
- $v_{\texttt{Tipo}}$ es una variable libre que el usuario da valor durante la interacción con el filtro.

Se prefiere el uso de filtros específicos frente a técnicas de tipo *Query By Example* [Zloof77] debido a que, en general, el usuario no necesita herramientas genéricas de búsqueda. Al contrario, en una unidad de interacción con unas tareas bien definidas, el analista puede determinar los criterios más frecuentes de búsqueda para las tareas a realizar y ofertar solo estos. Este modo de proceder, redunda en una mayor usabilidad. Un mecanismo de consulta de tipo QBE requiere de un aprendizaje por parte del usuario y solo los usuarios avanzados sacarán realmente partido de este tipo de característica. Los mecanismos basados en QBE pueden añadirse como características avanzadas para los usuarios expertos.

Otra ventaja adicional de la identificación temprana de los filtros como parte de los requisitos consiste en que ya desde el diseño de la aplicación pueden sugerirse optimizaciones para mejorar la eficiencia de estos procesos de búsqueda.

3.1.2 Criterio de ordenación

Un criterio de ordenación proporciona un mecanismo de ordenación de objetos basándose en las propiedades de estos. El criterio de ordenación consiste en una lista ordenada de pares `<expresión, sentido>`. Donde expresión es un atributo visible por la clase (propio o accesible por relaciones de agregación) y sentido indica ASC (ascendente) o DES (descendente). El orden de los elementos en la lista indica la prioridad de ordenación: el primer criterio es el más prioritario.

Ejemplo:

Podríamos ordenar los clientes por apellido y nombre y lo expresaríamos del siguiente modo:

`<Apellido, ASC>, <Nombre, ASC>`.

Donde se asume que:

- `Apellido` y `Nombre` son atributos de la clase `Cliente`.

3.1.3 Conjunto de visualización

Recoge las propiedades de los objetos de una clase que el usuario necesita observar. El conjunto de visualización se especifica como una lista ordenada de expresiones de propiedades (atributos de la propia clase o atributos alcanzables mediante relaciones de agregación alcanzables desde la clase).

Ejemplo:

Para obtener información de las facturas, se pretende observar el número de factura, el importe total, así como el nombre y apellidos del cliente asociado a la estancia que representa la factura. El conjunto queda expresado del siguiente modo[1]:

```
Numero, Total, Estancia.Cliente.Nombre, Estancia.Cliente.Apellidos
```

3.1.4 Navegación ofertada

La información normalmente no está aislada. Si no que, al contrario, está relacionada entre sí por medio de relaciones semánticas. La navegación ofertada se define como un subconjunto de las relaciones semánticas definidas entre clases (por ejemplo: relaciones de agregación y herencia). La navegación ofertada permite, de este modo, cruzar desde un objeto hasta los objetos relacionados con él por medio de una relación semántica.

Este tipo de navegación es de carácter hipermedial, permite alcanzar nuevos escenarios mediante saltos a través de relaciones entre objetos. La utilidad percibida por el usuario es un eficaz mecanismo de exploración de los datos para localizar objetos a partir de sus relaciones con otros.

Ejemplo:

Dada una estancia, desearemos poder consultar los bungalows (`UII_BungaloW`), parcelas (`UII_Parcela`), la temporada (`UII_Parcela`) y el cliente (`UII_Cliente`) asociado a la estancia.

3.1.5 Acciones ofertadas

Existe un segundo tipo de navegación con una semántica bien diferente. Una vez el usuario ha alcanzado un objeto y lo ha seleccionado, es posible que desee actuar sobre él para alterar su estado. En un mundo orientado a objetos, el mecanismo designado para tal fin son los servicios (o métodos) definidos en la signatura de clase.

Por tanto, las acciones ofertadas constituyen una lista ordenada de servicios pertenecientes a una clase. Determinando, de este modo, los servicios que serán ofertados al usuario en un determinado escenario para alterar el estado de los objetos.

Ejemplo:

Sobre una estancia se puede crear (`UIS_Crear`) y posteriormente añadir un bungalow (`UIS_InsertarBungalow`), añadir una parcela (`UIS_InsertarParcela`), añadir un servicio extra (`UIS_InsertarServicioExtra`) y finalmente crear una factura(`UIS_CrearFactura`).

3.2 Patrones de exploración

Una vez definidas las primitivas de trabajo sobre objetos, emplearemos esas primitivas como *ladrillos* o piezas base para construir escenarios recurrentes en los sistemas de información.

La *unidad de interacción* es el gránulo de la interfaz de usuario con la cual el usuario interacciona con el sistema. En esta línea Bodart [Bodart96] definió el término PU (*Presentation Unit*) como el objeto abstracto de interfaz de más alto nivel encargado de dirigir la interacción con el usuario.

Mediante el estudio de aplicaciones de gestión y los desarrollos llevados a cabo en el mundo de la industria, hemos identificado cuatro subtipos de unidades de interacción con características recurrentes de dominio en dominio [Molina02]. Dichos patrones o subtipos de unidades de interacción tratan de recoger construcciones muy frecuentes en las interfaces de usuario y que pueden ser especificados de un modo riguroso con muy poco esfuerzo por parte del analista.

[1] La notación `a.b` es similar a la empleada en las fórmulas en OCL. Donde `a` expresa el cruce de una asociación o agregación usando el rol `a`.

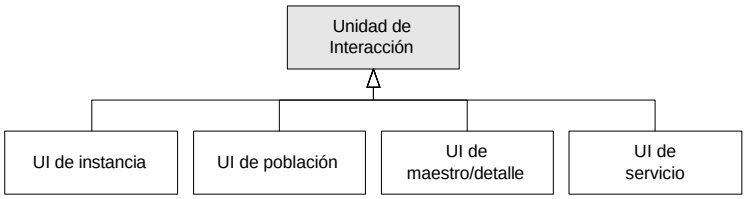

Figura 3. Unidades de interacción especializadas.

Los patrones identificados son los siguientes (véase Figura 3): patrón de instancia (*muestra un objeto*), patrón de población (*muestra un conjunto de objetos de una misma clase*), patrón de maestro/detalle (*muestra información relacionada en una sola unidad*) y patrón de servicio (*encaminado al lanzamiento de servicios*). A continuación describiremos cada uno de estos patrones.

3.2.1 Patrón de instancia

Un patrón de instancia es una unidad de interacción donde se muestra información relativa a un objeto (una instancia de una clase). La función de este tipo de unidades de interacción consiste en permitir consultar el estado de un objeto. Opcionalmente, suele ser un buen sitio para ofertar servicios que permitan alterar el estado del objeto así como ofertar enlaces a información relacionada semánticamente.

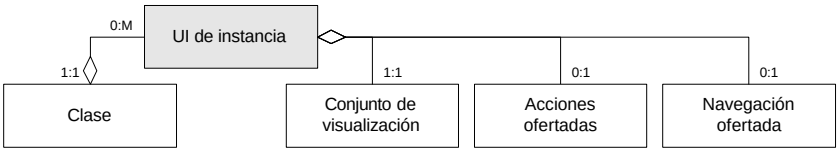

Figura 4. Metamodelo de la unidad de interacción de instancia.

Un patrón de instancia se define como composición de (véase Figura 4):

- un conjunto de visualización (*qué vemos del objeto*),
- cero o un conjunto de acciones ofertadas (*cómo podemos modificar* los objetos) y
- cero o un conjunto de navegaciones ofertadas (*qué enlaces a otra información relacionada disponemos*).

Ejemplo:

En el caso de estudio será de utilidad definir una unidad de interacción de instancia para la clase `Estancia`. Esta unidad de interacción permitirá observar la información relacionada con una estancia en particular.

Nombre: `UII_Estancia`

Conjunto de visualización: `Cliente.Nombre, Cliente.Apellidos, FechaInicio, FechaFin`

Acciones ofertadas: `InsertarServicio, InsertarParcela, InsertarBungalow`

Navegación ofertada: `Cliente, Bungalows, Parcelas, Servicios`

3.2.2 Patrón de población

Un patrón de población es una unidad de interacción destinada a mostrar conjuntos de objetos pertenecientes a la población de una misma clase. Dentro de estos escenarios, el usuario puede requerir de facilidades de búsqueda, ordenación, selección, y como en el caso anterior puede ser necesario proporcionar acciones y navegación.

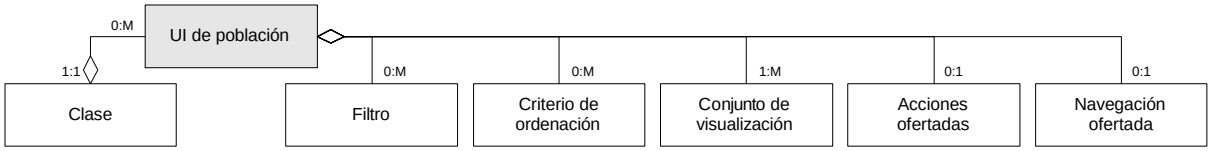

Figura 5. Metamodelo de la unidad de interacción de población.

Un patrón de población se define como composición de (véase Figura 5):

- Uno o más conjuntos de visualización (*qué vemos de los objetos*)
- Cero o más filtros (*cómo buscamos los objetos*)
- Cero o más criterios de ordenación (*cómo ordenamos los objetos*)
- Cero o un conjunto de acciones ofertadas (*cómo podemos modificar los objetos*)
- Cero o un conjunto de navegación ofertadas (*qué enlaces a otra información relacionada disponemos*)

Ejemplo:

Para el caso de estudio será interesante tener una unidad de interacción de población para la clase `Bungalow` que nos permitirá observar los Bungalows.

Nombre: `UIP_Bungalow`
Filtros: `Ocupado =` $V_{ocupado}$ `AND Tipo=` V_{tipo}
Criterio de ordenación: `<Id, Asc>`
Conjunto de visualización: `Id, Ocupado, Precio, Tipo, Ubicación`
Acciones ofertadas: `Crear, Modificar, Destruir`
Navegación ofertada: `Estancias, Bungalow`

3.2.3 Patrón de maestro/detalle

El patrón de maestro/detalle es un patrón compuesto por otras unidades de interacción. Su semántica es la de mostrar información relacionada de modo que el detalle mostrado está relacionado de algún modo con el objeto que actúa como maestro.

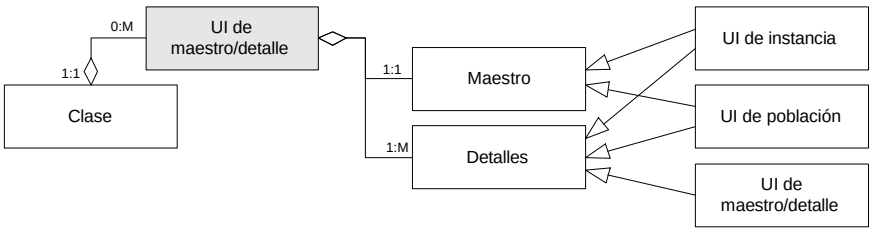

Figura 6. Metamodelo de la unidad de interacción de maestro/detalle.

Un patrón de maestro/detalle se define mediante (véase Figura 6):

- Un maestro: una unidad de interacción que juega el rol de maestro. Su misión es la de presentar un objeto que determinará el contenido de las unidades de detalle. Como maestro, puede usarse cualquier unidad de interacción que fije un objeto, es decir, un patrón de instancia o un patrón de población (donde puede seleccionarse un objeto).
- Uno o más detalles: son unidades de interacción que muestran información relacionada con el objeto maestro. Cada detalle queda ligado al maestro mediante una expresión de camino que indica la relación semántica entre el componente maestro y el detalle. Por ejemplo: si el componente maestro pertenece a la clase `Factura` y el componente detalle pertenece a la clase `Línea`, la expresión de camino podría ser: `LineasDeFactura` donde la expresión representa el camino de roles atravesado en una relación

de agregación entre las clases mencionadas. El rol de detalle puede ser tomado por las siguientes unidades de interacción: de instancia, de población y, recursivamente, de maestro/detalle.

Ejemplo:

Para el caso de estudio en cuestión será muy interesante tener una unidad de interacción Maestro/Detalle que nos permitiera observar para una determinada `Estancia` todos los `ServiciosUsados`.

Nombre: `UIMD_Estancia`
Maestro: `UII_Estancia`
Detalle: `UIP_ServiciosUsados`, Rol: `ServiciosUsados`

3.2.4 Patrón de servicio

Por último, el patrón de servicio modela aquellas unidades de interacción encaminadas exclusivamente al lanzamiento de un servicio. Su misión consiste en solicitar al usuario un conjunto de datos para dar valor a los parámetros de un servicio. Una vez completados los datos, el usuario puede solicitar la invocación del servicio confirmándolo o bien puede cancelarlo.

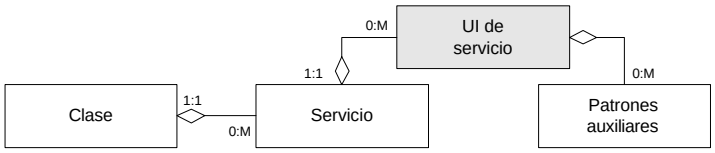

Figura 7. Metamodelo de la unidad de interacción de servicio.

Un servicio definido en una clase puede reificarse o corresponderse con una o más unidades de interacción de servicio (véase Figura 7). De este modo, el usuario puede usar esta unidad de interacción como interfaz hacia el servicio de interés. La unidad de interacción de servicio puede a su vez, puede tener aplicados otros patrones que se salen del alcance de este trabajo. Para una consulta detallada de dichos patrones auxiliares véase [Pastor00] y [Molina01].

4 Especificación de la Interfaz de Usuario

Una vez presentados los conceptos de trabajo, estamos en disposición de poder construir la especificación para el caso de estudio. Introduciremos una notación grafica (mostrada en la Figura 8) que representa las unidades de interacción como cajas y la navegación entre unidades (acciones o navegación ofertada) como flechas dirigidas. De este modo podemos construir diagramas isomorfos a grafos dirigidos donde las unidades de interacción representarían nodos y las navegaciones los correspondientes arcos dirigidos.

Figura 8. Notación gráfica para las unidades de interacción.

Usaremos está notación para ilustrar la especificación del caso de estudio. La notación gráfica proporciona un modo más cómodo de trabajar facilitando la comprensión de las especificaciones por parte de los miembros del equipo e incluso facilitando la validación por parte del usuario.

Retomando el problema, podemos partir de la especificación de casos de usos para realizar un análisis de tareas [Constantine99]. La Tabla 1 ejemplifica un caso de uso identificado en el dominio del problema.

Caso de uso:	`Crear_Estancia`
Actores:	Cliente (iniciador), Recepcionista
Tipo:	Primario
Descripción:	Un cliente se comunica con el recepcionista, indicándole que desea hospedarse en el camping. El Recepcionista le pregunta al cliente los datos necesarios para realizar la creación de la estancia (fecha, bungalow o parcela deseados). Al terminar esta acción el cliente puede hospedarse en el camping en el periodo de tiempo indicado.

Tabla 1. Ejemplo de caso de uso para la tarea: Crear Estancia.

A partir de dicho análisis de tareas ordenamos por similitud (funcionalmente, etc. a criterio del analista) las principales tareas que los usuarios van a desarrollar en el sistema. Esta información puede recogerse en el Árbol de Jerarquía de Acciones[Insfran01] (véase Figura 9) para construir un mecanismo de acceso a la aplicación.

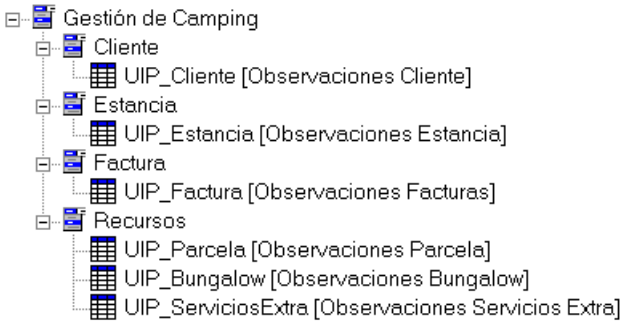

Figura 9. Árbol de Jerarquía de Acciones para el caso de estudio.

Partiendo del caso de uso ilustrado (véase Tabla 1) y componiendo los patrones de interfaz de usuario presentados, el analista puede especificar el mapa navegacional ilustrado en la Figura 10.

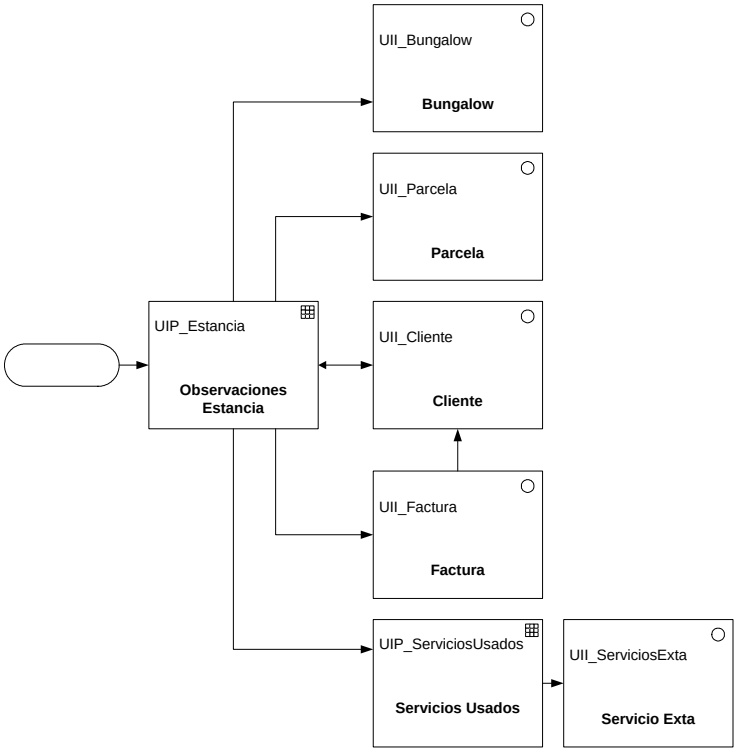

Figura 10. Escenario de estancia.

La Figura 11 muestra otro escenario especificado a partir del caso de uso para la tarea: *gestionar el cobro* en la cual se requiere poder localizar al cliente, consultar la estancia, verificar sus datos y emitir la correspondiente factura.

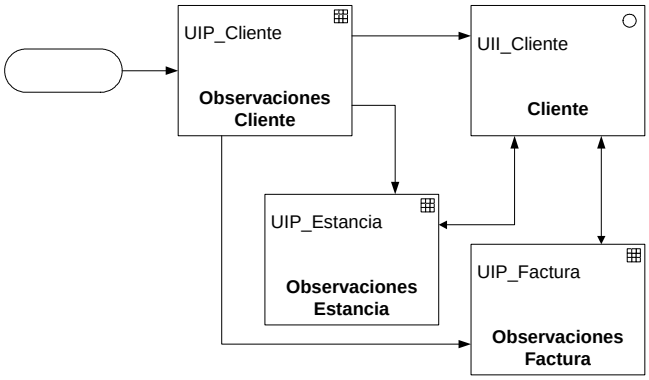

Figura 11. Escenario de clientes.

Los ejemplos ilustrados muestran como obtener un diagrama de navegación a partir de una tarea en particular. Para obtener una aplicación, debemos integrar los diferentes escenarios detectados. Siguiendo a Lausen [Lausen01] pueden aplicarse técnicas de reuso de partes de la interfaz de usuario para conseguir una aplicación más compacta y usable para el usuario final.

5 Implementación

Los conceptos definidos y empleados en el análisis de la interfaz de usuario siguen siendo de utilidad en la fase de implementación. Mediante refinamientos sobre la especificación original podemos obtener componentes de implementación en un lenguaje dado que satisfagan los requisitos iniciales. Está traducción de AIO (*Abstract Interface Object* [Bodart96]) a CIO (*Concrete interface Object*) puede incluso llevarse a cabo mediante la

aplicación de técnicas de producción automática de código. Esta estrategia, en particular, es la seguida en los productos de CARE Technologies.

A continuación describiremos una posible implementación que ha sido derivada automáticamente por medio de técnicas de traducción de modelos conceptuales. La implementación se ha realizado en Windows, en particular, para el entorno de programación Visual Basic 6.0. El componente generado contiene todo lo necesario para comunicarse con un componente de lógica de negocio (que también puede obtenerse por medio de técnicas de producción automática) que facilita el acceso a los datos y permite alterar el sistema por medio del lanzamiento de los servicios.

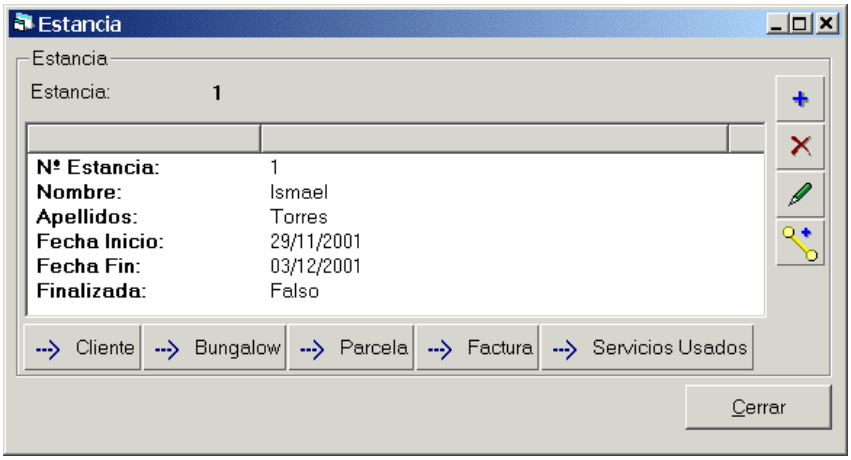

Figura 12. Ejemplo de implementación de una UI de Instancia.

En la Figura 12 se puede observar un formulario obtenido para una unidad de interacción de instancia (representa a `UII_Estancia`). El formulario que representa está delimitado por tres zonas bien diferenciadas. Una zona central donde se visualizan las propiedades de la Estancia a observar, la zona de la derecha está compuesta por una botonera (*toolbar*) que permiten acceder a la funcionalidad de creación, destrucción, modificación y inserción de Servicios Usados a la estancia seleccionada. Por ultimo, la zona inferior muestra una segunda botonera que permite alcanzar información relacionada con la Estancia como son el Cliente, Bungalow, Parcela, Factura y los Servicios Usados.

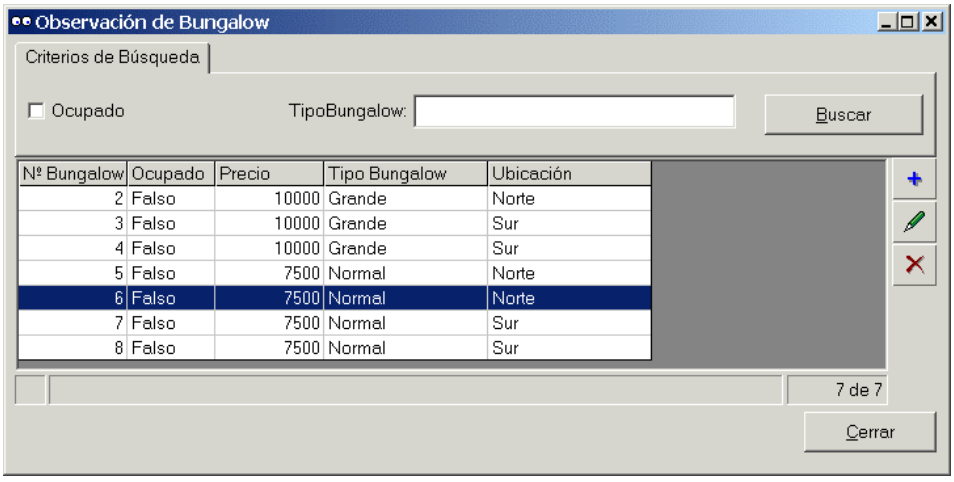

Figura 13. Ejemplo de implementación de una UI de Población.

En la Figura 13 se muestra un formulario derivado de una unidad de interacción de población (en particular, representa a `UIP_Bungalow`). El formulario, al igual que el anterior, está delimitado por tres zonas. Una zona central donde se visualiza en formato tabular las propiedades de un conjunto de Bungalows, La zona superior se

utiliza para realizar búsquedas y restringir la población de bungalows, la zona de la derecha, de nuevo, está compuesta por una botonera para acceder a la funcionalidad de creación, destrucción o modificación sobre Bungalows. Al igual que el formulario anterior, este tipo de formulario puede disponer además de una cuarta zona, situada en la parte inferior que se utiliza para alcanzar información relacionada con el bungalow seleccionado.

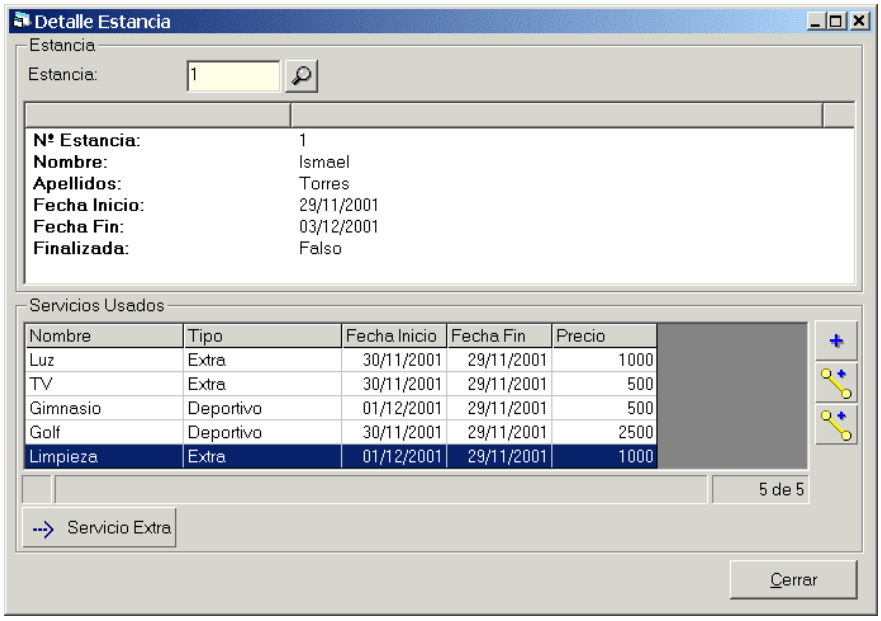

Figura 14. Ejemplo de implementación de una UI de Maestro/Detalle.

Por ultimo, la Figura 14 muestra un formulario derivado de una unidad de interacción maestro/detalle (representa a `UIMD_Estancia`). Este formulario esta compuesto por dos componentes bien diferenciados:

- Un componente maestro (unidad de interacción de instancia) en la parte superior que muestra los datos asociados a una estancia en particular.
- Un componente detalle (unidad de interacción de población de clase) en la parte inferior que presenta los servicios relacionados con la instancia seleccionada en el componente maestro. En este componente, la botonera de la derecha posibilita el cambio de la información de los servicios usados.

6 Trabajos relacionados

TRIDENT [Bodart95, Vander99] obtiene unidades de presentación a partir de un grafo de encadenamiento de tareas. A partir de las tareas del usuario y con la ayuda del grafo de encadenamiento de tareas se logran identificar unidades de interacción. Sin embargo, TRIDENT está basado en un modelo entidad-relación que solo define la estática del sistema.

OVID [Roberts98] es un trabajo de investigación desarrollado en IBM para la especificación de interfaces de usuario. Es un primer intento serio de conjugar los modelos orientados a objetos para la especificación de sistemas a la UML con especificaciones sobre interfaces de usuario. En OVID se presenta de noción de *vista* como estereotipo de una clase. Este concepto de vista permite definir el equivalente a lo que será una unidad de interacción. Sin embargo, el contenido de la vista queda totalmente libre a criterio del diseñador.

La aproximación presentada en este trabajo comienza a enriquecer la descripción conceptual de las unidades de interacción justo en el punto donde TRIDENT y OVID ceden el control al diseño. En esta etapa, como se ha demostrado, todavía pueden capturarme muchos requisitos de usuario acerca de la interfaz a construir. Esta colección de requisitos puede llevarse a cabo gracias a la especialización de las unidades de interacción con tareas bien definidas y al empleo de patrones para su construcción.

7 Conclusiones

Se ha presentado una serie de conceptos y patrones que ayudan a fijar una nomenclatura común para interfaces de usuario orientadas a objetos y que pueden ser empleados a lo largo del ciclo de vida de un sistema así como para su validación conjuntamente con el usuario final.

Los patrones presentados refinan el concepto de unidad de interacción en subtipos con cometidos bien diferenciados.

El lenguaje de patrones es independiente de la tecnología subyacente. Pudiendo, de este modo, refinar un análisis desarrollado sobre diferentes plataformas. Por ejemplo: implementando las unidades de interacción como ventanas en Windows o como páginas HTML en la Web.

El uso de una especificación precisa y no ambigua abre la puerta al empleo de generadores automáticos de código capaces de producir prototipos de las interfaces de usuario sobre diferentes plataformas.

En CARE Technologies S.A. estamos desarrollando esta tecnología soportada en herramientas de especificación y técnicas de producción automática e aplicaciones a partir de modelos conceptuales para obtener las interfaces de usuario de un modo muy ágil. Este modo de proceder, nos ha permitido obtener prototipos para mostrar al usuario en estadios muy tempranos del proyecto, y una vez validada la especificación, la interfaz de usuario final de la aplicación. De este modo se reduce considerablemente el tiempo total de desarrollo de aplicaciones.

Como trabajos futuros quedan planteados los siguientes:

- Continuar con el estudio y detección de nuevos patrones en el ámbito de aplicaciones de gestión.
- Proporcionar refinamientos a nivel de diseño del modelo propuesto para capturar propiedades de diseño dependientes de la plataforma destino.
- Ampliar el número de plataformas soportadas como destino para la producción automática de aplicaciones como, por ejemplo, dispositivos móviles: WAP, UMTS, etc.

8 Agradecimientos

Queremos agradecer la colaboración prestada para la elaboración de este trabajo al Dpto. de Investigación y Desarrollo de CARE Technologies S.A. y al Departamento de Sistemas y Computación de la Universidad Politécnica de Valencia.

9 Bibliografía

Bodart95 François Bodart, Jean Vanderdonckt. Towards a Systematic Building of Software Architectures: The Trident Methodological Guide. *Design, Specification and Verification of Interactive Systems*, páginas 262–278, Junio de 1995.

Bodart96 Fançois Bodart, Jean Vanderdonckt. *Widget Standardisation Through Abstract Interaction Objects*. Informe técnico, Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, Bélgica, 1996.

Constantine99 Larry Constantine, Lucy Lockwood. *Software for use: A practical guide to the Models and Methods of Usage-Centered Desing.* Addison Wesley, 1999.

Coplein99 J. Coplein. *Software Patterns.* `http://www.enteract.com/~bradapp/docs/patterns-intro.html`, 1999.

Gamma94 Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addyson Wesley, 1994**.**

Insfran01 Emilio Insfrán, Pedro J. Molina, Sofía Martí, Vicente Pelechano. *Ingenieria de Requisitos aplicada al modelado conceptual de interfaz de usuario.*, 4tas Jornadas Iberoamericanas

de Ingeniería de Requisitos y Ambientes Software, IDEAS 2001, Santo Domingo, Heredia, Costa Rica. Ed. CIT, Abril 2001, pags. 181-192.

Lausen01 Soren Lausen, Morten Borup Harning. *Virtual Windows: Linking User Tasks, Data Models & Interface Design.* IEEE Software, páginas 67-75, Julio-Agosto de 2001.

Molina01 Pedro J. Molina, Oscar Pastor, Sofía Martí, Juan J. Fons, Emilio Insfran. *Specifying Conceptual Interface Patterns in an Object-Oriented Method with Code Generation.* Proceedings of User Interfaces for Data Intensive Systems, UIDIS '2001, Zurich, Suiza, pages 72-79, IEEE Computer Society, mayo de 2001.

Molina02 Pedro J. Molina, Santiago Meliá, Oscar Pastor. JUST-UI: A User Interface Specification Model. *Proceedings of Computer Aided Design of User Interfaces, CADUI'2002*, Les Valenciens, Francia, páginas 323-334, Kluwer Academic Publishers, mayo de 2002. (*Pendiente de publicación***.**)

Pastor95 Oscar Pastor, Isidro Ramos. *OASIS 2.1.1: A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*, Febrero 94 (1ª ed.) Marzo 95 (2ª ed.), Octubre 95 (3ª ed.).

Pastor00 Pastor O., Molina P.J., Aparicio A. *Specifying Interface Properties in Object Oriented Conceptual Models.* ISBN 1-58113-252-2, In Proceedings of Working Conference on Advanced Visual Interfaces, AVI 2000, Palermo, Italy. Ed. ACM, mayo de 2000, págs. 302-304.

Roberts98 D. Roberts, D. Berry, S. Isensee y J. Mullaly. *Designing for the User with OVID: Bridging User Interface Design and Software Engineering*. MacMillan, 1998.

Vander99 Jean Vanderdonckt. *Assisting Designers in Developing Interactive Business Oriented Applications.* In Proc. of HCI'99, 1999.

Weile01 Martijn van Welie. *The Amsterdam Collection of Patterns.* `http://www.weile.com/patterns/`, 2001.

Zloof77 M. Zloof, Query-By-Exaple: A Data Base Language. *IBM System Journal. Vol 4*, páginas 324-343, diciembre de 1977.