Ingeniería de Requisitos aplicada al modelado conceptual de interfaz de usuario

Emilio Insfrán[1], Pedro J. Molina[2], Sofía Martí[2], Vicente Pelechano[1]

[1] Dpto. de Sistemas
Informáticos y Computación
Universidad Politécnica de Valencia
Valencia, España.
{einsfran|pele}@dsic.upv.es

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

Resumen

Uno de los desafíos más interesantes en entornos de producción de automática de software es la obtención de interfaces de usuario útiles y ergonómicas. Algunas propuestas en este sentido intentan abstraer propiedades de interfaz de usuario a partir del modelo conceptual especificado. Otras intentan empezar el proceso de desarrollo por la especificación del aspecto externo (ventanas y diálogos) del sistema a desarrollar. En este trabajo presentamos un método para obtener las propiedades de interfaz de usuario a partir de un Modelo de Requisitos que captura propiedades en tres dimensiones complementarias: funciones, comunicación y comportamiento. El método propuesto se ha definido para una metodología de producción automática de código, OO-Method, y está siendo incorporado a una herramienta CASE que soporta la metodología, en el marco de un proyecto de I+D entre la Universidad Politécnica de Valencia y CARE Technologies S.A.

Palabras clave

Ingeniería de requisitos, interfaz de usuario, modelado conceptual, orientación a objetos, patrones conceptuales, generación automática de código.

1 Introducción

Una de las aspiraciones no satisfechas más frecuentes en la *generación automática de código basada en modelos* [Bell98] reside en la complejidad inherente a lograr que la interfaz producida sea ergonómica y útil para el usuario final [Janssen93]. Una estrategia seguida en este contexto consiste en determinar algoritmos que a partir de la información estructural de los modelos conceptuales determinen la creación de menús de acceso a los servicios ofertados por la aplicación. En la práctica, las interfaces basadas en esta estrategia no son lo suficientemente ergonómicas.

Para solventar esta y otras necesidades propias de las interfaces de usuario, el modelo conceptual de OO-Method (objetos, dinámico y funcional) [Pastor97] fue extendido con un Modelo de Presentación que captura los conceptos relativos a los requisitos de interfaz de usuario. Este Modelo de Presentación tiene como principal contribución que, haciendo uso exclusivamente de información proveniente del dominio del problema, captura de modo declarativo e independiente de la plataforma propiedades para la interacción entre el usuario y la aplicación. Esta información es fundamental para la posterior producción automática de software en plataformas de desarrollo de uso industrial como por ejemplo: Visual Basic 6.0, Java/Swing, interfaces web basadas en HTML 4.0, o interfaces WAP basadas en WML.

El Modelo de Requisitos de OO-Method está basado en el Marco Conceptual para Requisitos definido en [Wieringa98]. Este marco conceptual permite describir los requisitos en tres dimensiones interdependientes que son: funciones (interacciones externas), comunicación (existente entre las funciones y los actores involucrados) y comportamiento (descripción de las acciones atómicas en y entre los componentes internos del sistema). Este modelo, además de capturar los requisitos del usuario teniendo en cuenta únicamente su interacción con el sistema, permite obtener un modelo conceptual orientado a objetos (en términos de clases del sistema) como resultado de un Proceso de Análisis de Requisitos como se explica en [Insfrán99].

En este trabajo se propone un método específico basado en la información recogida en el Modelo de Requisitos para obtener una agrupación jerárquica de la funcionalidad ofertada por el sistema (interfaz de usuario) que será expresado en términos del patrón de *jerarquía de acciones* definido en el Modelo de Presentación de OO-Method.

La estructura del artículo es la siguiente. En primer lugar se presenta OO-Method como marco metodológico de trabajo describiendo con más detalle los modelos de Requisitos y de Presentación. La siguiente sección presenta el método propuesto y un ejemplo de su aplicación. Para finalizar se presentan las conclusiones y los trabajos futuros.

2 OO-Method

OO-Method [Pastor96] [Pastor97] es un método orientado a objetos de producción de software que utiliza técnicas gráficas convencionales de modelado y está basado en el lenguaje de especificación formal y orientado a objetos OASIS [Pastor95].

El Modelo Conceptual recoge de forma gráfica, utilizando una notación basada en UML [Booch99], las propiedades relevantes que definen el sistema a desarrollar sin tener en cuenta aspectos de implementación. En esta fase los modelos a especificar son:

- **Modelo de Objetos:** muestra la estructura de las clases identificadas en el dominio del problema así como sus relaciones. Gráficamente se representa con un Diagrama *de Clases* (DC).
- **Modelo Dinámico:** describe los aspectos relacionados con el control, vidas posibles e interacción entre objetos. Utiliza dos diagramas:

  a) *Diagrama de Transición de Estado (DTE):* determina la dinámica para cada clase del sistema, entendida como las vidas posibles (estados válidos) para las instancias de la clase.

  b) *Diagrama de Interacción de Objetos (DIO):* determina la comunicación entre objetos y describe las reglas de actividad interna (disparos) del sistema de una forma gráfica.

- **Modelo Funcional:** captura la semántica ligada al cambio de estado de los objetos en términos de la modificación de los valores de sus atributos como consecuencia de la ocurrencia de eventos.

Dos nuevos modelos se incorporan a OO-Method para capturar más información relevante del espacio del problema con técnicas y notaciones específicas. Estos modelos son:

- **Modelo de Requisitos:** captura la información relativa a los requisitos del usuario en términos de las interacciones externas del sistema a desarrollar. En este modelo se

consideran los aspectos funcionales, de comunicación y comportamiento a alto nivel. El Modelo de Requisitos representa una capa previa al Modelo Conceptual.

- **Modelo de Presentación:** captura las interacciones del usuario con la aplicación a través de abstracciones conceptuales basadas en patrones de interfaz de usuario. El Modelo de Presentación está incluido dentro del Modelo Conceptual.

En la Figura 1 se muestra el marco metodológico de trabajo. En las siguientes subsecciones se describen con más detalle cómo derivar la agrupación jerárquica de la funcionalidad del sistema (especificada en el Modelo de Presentación) a par tir de la información recogida en el Modelo de Requisitos.

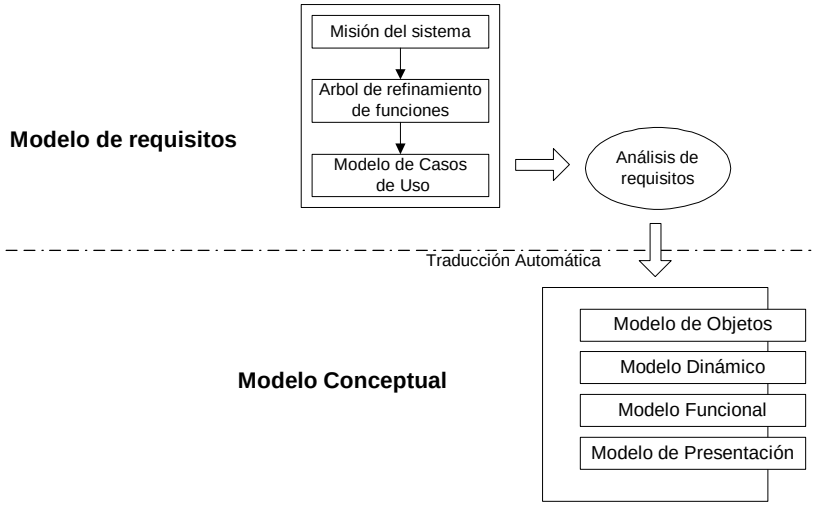


Figura 1. Nuevo marco arquitectónico de OO -Method.

2.1 Modelo de Requisitos

El modelo de requisitos contiene una descripción de los objetivos y com portamiento externo del sistema, es decir, *qué* debe hacer el sistema sin describir *cómo* debe hacerlo.

Las técnicas usadas en este modelo son:

- *Misión del Sistema:* describe el objetivo general del sistema.
- *Árbol de Refinamiento de Funciones:* particiona la funcionalidad del sistema en interacciones externas y lo muestra de forma jerárquica como el resultado de un refinamiento del objetivo del sistema.
- *Modelo de Casos de Uso:* incluye la *especificación de casos de uso* para especificar la descomposición de las interacciones externas y el *diagrama de casos de uso* para mostrar la comunicación entre el entorno (actores) y el sistema.

2.1.1 Misión del Sistema y Árbol de Refinamiento de Funciones

Las interacciones externas siempre se pueden representar como funciones en una jerarquía de refinamiento de modo que la raíz de la jerarquía sea la Misión del Sistema, y las hojas sean las interacciones externas (funciones elementales). Los nodos intermedios son grupos de funciones y normalmente representan un tipo de actividad o un área de negocio del sistema. No es una tarea trivial distinguir entre nodos intermedios y nodos hoja del árbol de

refinamiento funcional. *Una función se considera elemental si se activa por un evento enviado por un usuario del sistema (actor) o por la ocurrencia de un evento temporal* [Wieringa98].

De este modo, el Árbol de Refinamiento de Funciones se puede usar para representar una descomposición jerárquica de las funciones de negocio del sistema independiente de la estructura interna del sistema. En la Figura 2 se muestra un ejemplo de Árbol de Refinamiento de Funciones para una Terminal de Punto de Venta (TPV).

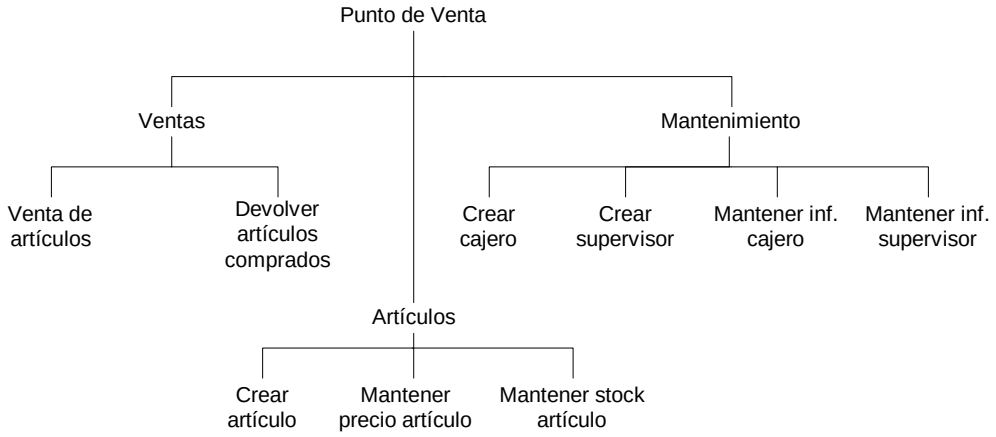


Figura 2. Misión del Sistema y Árbol de Refinamiento de Funcione s.

2.1.2 Casos de uso

Los diagramas de casos de uso se introdujeron en OOSE [1] [Jacobson92] para representar la funcionalidad externa y comunicación del sistema, y ha sido adoptado desde entonces por varios métodos y notaciones. Un caso de uso es una interacción entre el sistema y una entidad externa. Normalmente esta interacción se puede descomponer en un conjunto de actividades (especificación del caso de uso) definidas a este nivel como funciones atómicas.

Combinando los casos de uso con el Árbol de Refinamien to de Funciones (ARF), cada uno de los casos de uso se corresponderá con un nodo hoja de dicho árbol. De esta forma nos aseguramos que el nivel de abstracción para la especificación de casos de uso vendrá determinada por el nivel de refinamiento del ARF. E sta propuesta es similar a la construcción del modelo de casos de uso basado en eventos [2] pero con dos diferencias importantes:

- con el Árbol de Refinamiento de Funciones las interacciones externas se dividen y organizan en una jerarquía de refinamiento del objetivo del negocio, y

[1] Object-Oriented Software Engineering

[2] Primero se identifican los even tos externos a los que el sistema en desarrollo debe responder, y en segundo lugar, se relacionan estos eventos con los actores y los casos de uso.

- los aspectos de comunicación (actor/caso de uso) se abstraen hasta que se alcanza el paso de modelado de casos de uso (y una vez que éstos han sido totalmente identificados). De esta forma los analistas tienen una visión más clara de la funcionalidad externa del sistema (especificada totalmente en el ARF) y es más fácil modelar la comunicación entre actores y los casos de uso.

También es posible estructurar el conjunto de los casos de uso para hacer que el modelo sea más fácil de comprender y trabajar con él. Las relaciones para la estructuración de los casos de uso son: <<uses>>, <<extends>> e <<include>> y son explicadas con detalle en [Insfrán99].

2.1.3 Construcción del Modelo Conceptual a partir del Modelo de Requisitos

Dada una especificación detallada de interacciones externas (modelo de requisitos) se puede obtener una descomposición interna en términos de componentes del sistema (clases). Estas clases capturan todo el comportamiento y responsabilidades especificadas en el modelo de requisitos de forma independiente a la implementación. Para ello se realiza un Proceso de Análisis de Requisitos basado en la especificación de los casos de uso y utilizando diagramas de secuencia como se explica en [Insfrán99] [Insfrán00].

2.2 Modelo de Presentación

El Modelo de Presentación [Molina98] [Pastor00] permite especificar propiedades de interfaz de usuario presentes en el espacio del problema en base a *patrones conceptuales de interfaz de usuario*. Por tanto, se proporciona un lenguaje de patrones conceptuales que permite especificar la esencia de la interfaz de usuario, libre de consideraciones de implementación, de una manera declarativa y en términos del propio modelo conceptual subyacente.

Una descripción breve de los distintos patrones, a efectos de contextualizar al lector, se presentan a continuación:

- *Selección de acción*: recoge cómo el usuario podrá navegar en la aplicación, en cada contexto de interacción qué podrá ejecutar o alcanzar mediante navegación. Se compone de tres subpatrones:
  - *Jerarquía de Acciones:* organiza y estructura en forma de árbol la funcionalidad de la aplicación para cada actor o grupo de actores del sistema. En las hojas del árbol se encuentran patrones de presentación.
  - *Navegación Ofertada:* declara la información accesible mediante mecanismos de navegación desde el objeto seleccionado a otros relacionados semánticamente.
  - *Acciones Ofertadas:* define las acciones que puede sufrir un objeto dado.
- *Presentación*: los patrones de presentación abstraen el concepto de unidad de presentación, dentro de los cuales se recogen los siguientes subtipos:
  - *Presentación de Servicio*: establece una unidad de presentación para la ejecución de un servicio por parte del usuario.

- *Presentación de Instancia*: define qué información de un objeto puede s er mostrada al usuario, para ello utiliza los subpatrones: Conjunto de Visualización, Navegación Ofertada y Acciones Ofertadas.
- *Presentación de Población de clase*: define unidades de presentación para la búsqueda y exploración de los objetos de una clase. Se compone de una lista ordenada de patrones de Selección de Población.
- *Presentación Maestro/Detalle*: establece unidades de presentación donde un objeto (maestro) y una relación semántica establecida determina los objetos relacionados (detalle) a través de la relación semántica, por ejemplo: Una factura y sus líneas. Los patrones de Maestro/Detalle se construyen por composición a partir de otros patrones de presentación.

A su vez, en el contexto de un servicio los argumentos pueden verse afectados por los siguientes patrones:

- *Introducción*: determina como ha de ser introducido un argumento de un servicio.
- *Selección definida*: define un tipo enumerado para la introducción de valores a un argumento.
- *Selección de población*: define cómo un objeto puede ser localiz ado por el usuario indicando: filtros, criterios de ordenación, conjuntos de visualización, acciones ofertadas y navegación ofertada.
- *Información complementaria*: establece mecanismos de retroalimentación para confirmar al usuario la correcta identidad de u n objeto.
- *Dependencia*: mediante un lenguaje formal basado en reglas ECA (evento –acción-condición) declara la dinámica existente entre argumentos de un servicio.

2.2.1 Jerarquía de Acciones

En este trabajo nos centraremos en la obtención automática del patrón **Jerarquía de Acciones** a partir de la información recogida en el Modelo de Requisitos, por lo que en la presente sección lo presentaremos con más detalle.

El patrón de Jerarquía de Acciones es una abstracción útil para construir un árbol que exponga al usuario final la funcionalidad del sistema siguiendo el *paradigma verbo-nombre*[3] [Foley91]. Siguiendo el principio de aproximación gradual, esta estructuración evita al usuario verse desbordado por la gran cantidad de funcionalidad que un sistema de tamaño medio podría presentar al usuario.

El patrón consta de una estructura arborescente donde el analista tiene que construir un árbol del siguiente modo:

- Los nodos intermedios (nodos con hijos) están etiquetados con una cadena de texto a la que llamaremos *alias*.

[3] Paradigma verbo-nombre: Donde primero se selecciona la acción a realizar y después el objeto que sufrirá dicha acción. Contrapuesto por tanto, al paradigma complementario: nombre -verbo.

- Los nodos hoja están también etiquetados con un *alias* pero además referencian a un servicio ofertado por una clase del sistema especificado.

De este modo, el analista puede construir un menú abstracto donde agrupa por criterios arbitrarios (funcionalidad, alfabéticamente, frecuencia de uso, por pertenencia a clases, etc.) la funcionalidad del sistema que desea ofertar a los usuarios finales.

La implementación de este patrón sobre una herramienta CASE que de soporte al analista en la definición, podría ayudar realizando comprobaciones de balanceado del árbol (profundidades semejantes en cada rama), ergonómicas (como es *la regla del 5±2*, es decir, en cada nivel esta compuesto entre tres y siete elementos, cantidades adecuadas para una buena retentiva en la memoria humana basada en estudios psicológicos) o bien proporcionado asistentes que creen árboles por defecto a partir de las clases y los métodos de éstas en un esquema conceptual. Los asistentes de éste tipo pueden encargarse de construir árboles a partir de los criterios anteriormente citados: funcionalidad, alfabéticamente, frecuencia de uso, por pertenencia a clases, etc.

Proyección del patrón de jerarquía de acciones

Una vez completado el proceso de especificación de las propiedades de interfaz de usuario se puede proponer implementaciones del patrón en diversas plataformas. Como ejemplo, ilustraremos dos de ellas, una representación para un entono típico de ventanas (WIMP) y para un entorno web con características hipermediales.

A. Ejemplo de representación en un entorno de ventanas

La jerarquía de acciones de un esquema podría ser representada en un entorno de ventanas como la definición de un menú de aplicación que se muestra en una ventana MDI marco de la aplicación que se está tratando.

Para construir el menú podríamos seguir el siguiente modo de proceder:

- El alias del nodo raíz del patrón constituiría el título de la ventana.
- Un nodo intermedio (que contiene hijos) se representaría mediante un elemento de menú que será desplegable para mostrar como sube lementos a sus respectivos hijos. El alias del nodo se emplearía como etiqueta del menú.
- Un nodo hoja (que no contiene hijos) se representaría mediante un elemento de menú final que conduce a la presentación de una nueva ventana para permitir el lanzamiento del método asociado al nodo. Al igual que en resto de casos, el alias del nodo se emplearía como etiqueta del menú.

Estos sencillos pasos permiten construir una representación fiel al patrón de jerarquía de acciones especificada por el analista. El proce so es lo suficientemente genérico y sencillo de aplicar para construir menús en cualquier entorno de ventanas y con cualquier lenguaje que las soporte.

B. Ejemplo de representación en un entorno web

En entornos web, los lenguajes como HTML 4.0 (Web) o WML (para WAP) permiten describir la información que va a ser mostrada al usuario. En este sentido, el patrón de jerarquía de acciones también puede ser traducido a estos entornos.

Como ejemplo, podríamos obtener una representación siguiendo los pasos:

- El alias del nodo raíz del patrón constituiría el título de la página raíz.
- Un nodo intermedio (que contiene hijos) se representaría mediante una pagina donde aparece una lista mostrando cada uno de los alias de sus respectivos hijos como enlaces. Al seleccionar uno de ellos se navega a una nueva página. El alias del nodo se emplearía como titulo de la página.
- Un nodo hoja (que no contiene hijos) se representaría mediante un enlace etiquetado con el alias del nodo que conduce a la presentación de una nueva pagina para permitir el lanzamiento del método asociado al nodo.

Otros entornos como las interfaces telefónicas habladas automatizadas siguen el mismo esquema para el acceso a las funcionalidades ofertadas por el sistema.

3 Obtención del patrón Jerarquía de Acciones a partir del Modelo de Requisitos

El árbol de refinamiento de funciones (ARF) del Modelo de Requisitos proporciona un mecanismo de refinamiento del objetivo del sistema en subobjetivos y así sucesivamente hasta llegar a las interacciones externas del sistema (servicios ofertados). La disposición de las interacciones externas como hojas de un determinado nodo intermedio en el ARF responde a su naturaleza, es decir, **qué** interacciones externas se debe realizar para cumplir el subobjetivo u objetivo (nodo intermedio o raíz) del nivel superior.

El Modelo de Casos de Uso (Mcu) particiona funcionalmente el sistema en casos de uso, basado en el ARF, y además especifica la comunicación existente con actores (los actores que inician el caso de uso, además de los que proveen y reciben información). El Mcu también especifica la comunicación entre casos de uso: un caso de uso puede comunicarse con otro caso de uso que representa una interacción externa (relación <<uses>>).

3.1 Proyección

Una vista del modelo de casos de uso (Vcu, Figura 3) representa un subconjunto de casos de uso (Vcu $\subset$ Mcu) con sus respectivos actores. Esta vista se define restringiendo el modelo de casos de uso con algún criterio: por actor, por actor que inicia algún o algunos casos casos de uso, por objetivos o subojetivos del sistema, etc. En particular, el Mcu restringido a un actor que inicia casos de uso (ai) define una vista con el conjunto de interacciones externas del sistema con dicho actor [4] ($Mcu \downarrow ai = Vcu_{ai}$).

[4] Algunas propuestas como [Booch99] [Larman98] proponen crear el modelo de casos de uso identificando inicialmente los actores y luego determinando los casos de uso con los que participan. Esta aproximación, a nuestro criterio, conlleva al problema clásico de determinar de forma uniforme el nivel de abstracción de los caso de uso. Además, el concepto de interacciones externas ofertadas a actores es más cambiante que el concepto de refinamiento de objetivos abordado con el ARF.

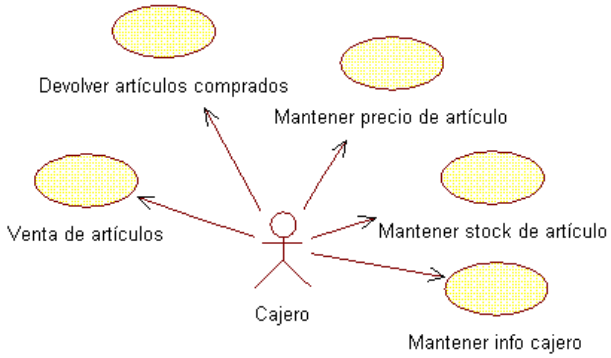


Figura 3. Vista de Casos de Uso restringida al actor Cajero (Vcu_{cajero}).

Si se proyecta nuevamente la Vcu_{ai} sobre el ARF y se restringe a las interacciones externas del actor ai (considerando sólo aquellas ramas en cuyas hojas se encuentren interacciones externas de ai), se obtiene una jerarquía de interacciones externas basada en el refinamiento de objetivos del sistema y agrupadas por el actor ai ($ARF \downarrow Vcu_{ai} = ARFai$).

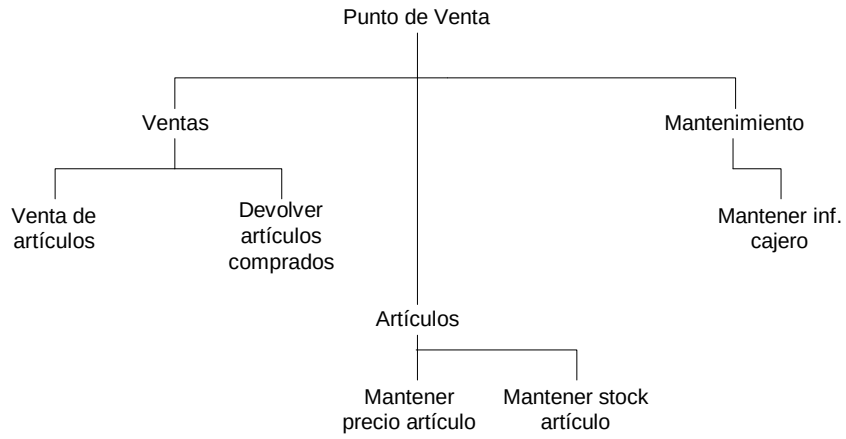


Figura 4. Vista de CU para el actor Cajero proyectado sobre el ARF (ARFai).

En la Figura 4 vemos el resultado de proyectar la vista de casos de uso restringida al actor Cajero (Vcu_{cajero}) sobre el ARF.

El árbol obtenido finalmente puede tomarse como Patrón de Jerarquía de Acciones de partida en el Modelo de Presentación para el sistema que esta siendo especificado. Posteriormente, el analista puede refinarlo y enriquecerlo empleando la expresividad proporcionada por el propio Modelo de Presentación.

3.2 Balanceado

La jerarquía de acciones obtenida por procedimiento anteriormente descrito, constituye un subárbol del Árbol de Refinamiento de Funciones del Modelo de Requisitos. Este nuevo árbol puede sufrir desequilibrios en cuanto a número de nodos por rama o estar desbalanceado en cuanto a profundidades del árbol. Para obtener una estructura uniforme que sirva para que los usuarios accedan a los servicios del sistema puede ser conveniente asegurar

el correcto balanceado del árbol. Para tal fin, se han desarrollado una serie de reglas heurísticas basados en el número de nodos por subrama y profundidad de ramas que permiten simplificar el árbol o al menos detectar el desbalanceo para informar al analista.

3.2.1 Detección de exceso/defecto de nodos en una rama

La regla heurística del 5±2 establece el número ideal de nodos hijos que debe contener cada nodo. Fijado un umbral mínimo (π_{min}) y máximo (π_{max}) parametrizables, podemos diseñar un algoritmo que recorra el árbol y detecte ramas donde se exceden por defecto o por exceso los valores límite. El algoritmo recursivo que se presenta en la Tabla 1 lleva a cabo este cometido:

```
Detectar(Nodo)
      si (Nodo.NumHijos() = 0) Salir
      si (Nodo.NumHijos() < π_min) entonces Avisar_PorDefecto(nodo, Nodo.NumHijos())
      si (Nodo.NumHijos() > π_max) entonces Avisar_PorExceso(nodo, Nodo.NumHijos())
      para cada i desde 1 hasta Nodo.NumHijos()
            Detectar(Nodo.Hijo_i)
```

Tabla 1. Algoritmo de detección de exceso/defecto de nodos.

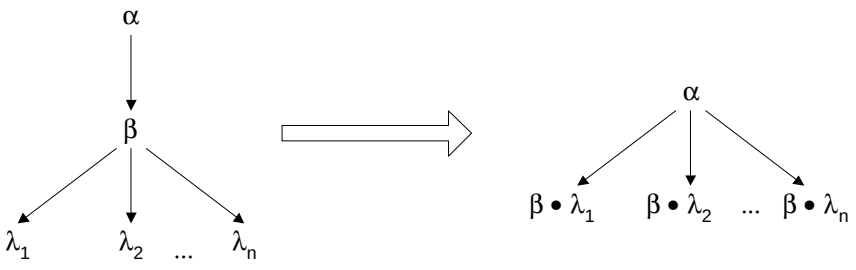
3.2.2 Detección de exceso de profundidad en las ramas del árbol

Cuantos más niveles tiene una Jerarquía de Acceso, más complicado es para el usuario encontrar lo que busca. Tanto la complejidad de la aplicación, como el número de operaciones realizables en el sistema influyen directamente en el tamaño de la Jerarquía de Acciones.

Aun así, la detección de árboles demasiado profundos teniendo en cuenta al número de nodos hoja puede dar indicios al analista de que debería rediseñar la jerarquía (sólo a efectos de presentación) para hacerla más ergonómica.

3.2.3 Simplificación de nodos intermedios

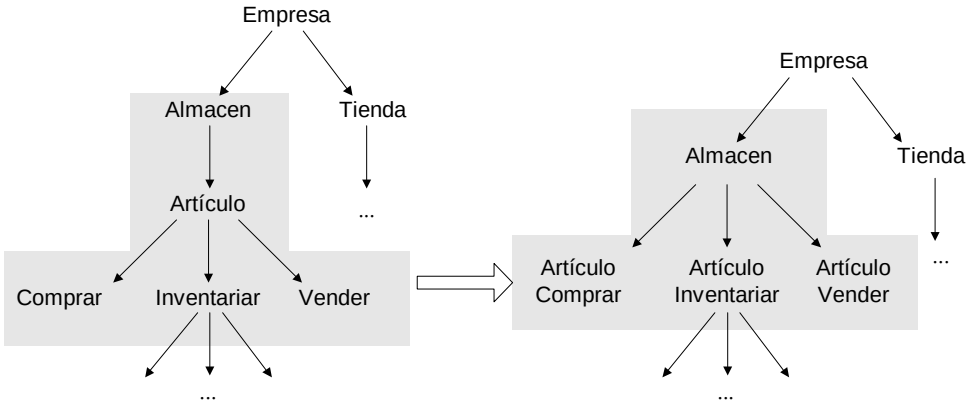
Dado un árbol perteneciente a un patrón de Jerarquía de Acciones donde se haya detectado un desbalanceo por falta de nodos y se aprecie una construcción en una subrama isomorfa a la configuración A (Figura 5): (un nodo raiz α con un **solo hijo** β que tiene como nodos hijos a: λ_1, λ_2 ... λ_n), podemos establecer una transformación (configuración A', Figura 5) donde se elimina un nodo (β) sin perdida de información al transferir su información a los nodos hijos. Los nodos hijos sufren la transformación: ($\beta \bullet \lambda_i$), donde el operador ($\bullet$) denota la concatenación de los alias de cada nodo separados por un espacio para mantener la legibilidad.



Configuración A Configuración A'

Figura 5. Transformación de simplificación de nodos intermedios.

Ejemplo: El árbol B (presentado en la Figura 6) un árbol perteneciente a un patrón de Jerarquía de Acciones, es transformado a un árbol B' más simple.



Configuración B Configuración B'

Figura 6. Ejemplo de simplificación.

El heurístico latente tras este algoritmo es el de evitar la existencia de nodos intermedios (meros agrupadores) excesivos que hacen innecesariamente profunda la jerarquía.

Este algoritmo puede aplicarse iterativamente con el beneplácito del analista hasta que se llega a un árbol donde no puede volver a ser aplicada ninguna simplificación más o bien el analista da por buena la Jerarquía de Acciones refinada y detiene el proceso.

Como ejemplo podemos observar las figuras 7 y 8 donde se muestran las configuraciones para dos jerarquías de acciones y dos ejemplos de menús de aplicación que podrían ser generados sobre plataformas Windows (figuras 9 y 10).

Figura 7. Jerarquía de acciones C.

Figura 8. Jerarquía de acciones C'.

Figura 9. Menú generado para C.

Figura 10. Menú generado para C'.

4 Conclusiones y trabajos en curso

La definición de métodos y técnicas que permitan sistematizar y automatizar la producción de softwa re es una tarea compleja. En este contexto, nuestro trabajo aporta la posibilidad de sistematizar la obtención de propiedades de interfaz de usuario a partir de requisitos de una forma automática, controlada e independiente de la plataforma de desarrollo. La disposición de los servicios (acciones que el usuario podrá llevar a cabo en el sistema) en una jerarquía facilita el acceso, la exploración y el aprendizaje de la interfaz por parte de los usuarios finales [Constantine99]. El soporte de estas técnicas mediante herramientas CASE facilita además al analista la manipulación y construcción basada en las abstracciones presentadas.

Así mismo, disponer de un esquema conceptual extendido con información de interfaz de usuario (y obtenida a partir de un Modelo d e Requisitos) permite disponer de modelos del problema más completos y coherentes. En este sentido, la obtención de un boceto de interfaz de usuario de manera rápida a partir de la propia especificación de requisitos favorece la rápida prototipación de la interfaz del sistema que esta siendo especificado. Por otro lado, la generación automática aplicaciones incluyendo interfaces de usuario puede llevarse a cabo con mayores cotas de calidad que las que podrían ser alcanzadas sin dicha información extra.

Actualmente se está trabajando en la definición de nuevos mecanismos para extraer mayor información del Modelo de Requisitos (y su posible extensión) de forma a completar la obtención automática del resto de los patrones del Modelo de Presentación. En particul ar los aspectos relacionados a la navegación de patrones de presentación en base a la relación existente entre nodos del Árbol de Refinamiento de Funciones y de casos de uso. Estas ideas están siendo desarrolladas en la empresa CARE Technologies S.A conjun tamente con la Universidad Politécnica de Valencia.

5 Bibliografía

Bell98 Bell R. *Code Generation from Object Models*. Embedded Programming Systems Journal. March 1998.

Booch99 Booch G., Rumbaugh J., Jacobson I. *Unified Modeling Language Notation Guide Version 1.3* 1999. Rational Software Corporation.

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

Foley91 Foley J., Kim W., Kovacevic S., Murray K.. *Inteligent User Interfaces. Chapter 15. UIDE an Inteligent User Interface Design Environment.* pags. 339-384, ACM Press, Addison Wesley, 1991.

Insfrán99 Insfrán E., Wieringa R., Pastor O. *Using TRADE to improve an Object-Oriented method.* Technical report, University of Twente. Computer Science Department, Enschede, Holanda, Julio 1999.

Insfrán00 Insfrán E., Wieringa R., Pastor O. *Requirementes Engineering-Based Conceptual Modeling.* Submitted to the Requirements Engineering Journal. (Available at http://www.dsic.upv.e s/~einsfran/publicaciones.html).

Jacobson92 Jacobson I., Christerson M., Jonsson P., Overgaard G. *Object Oriented Software Engineering, a Use Case Driven Approach* Addison -Wesley. Reading, Massachusetts 1992.

Janssen93 Janssen C., Weibsbecker A., Ziegler J. *Generating User Interfaces from Data Models and Dialogue Net Specification.* INTERCHI'93 Conference Proceedings. ACM Press, 1993.

Larman98 Larman C. *Applying UML and Patterns.* Prentice-Hall 1998.

Molina98 Molina, P.J. *Specification of User-Interface in OO-Method.* 1998 Master of Science Thesis, Technical University of Valencia.

Pastor95 Pastor O., Ramos I. *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.), Octubr e 95 (3ª ed.).

Pastor96 Pastor O., Pelechano V., Bonet B., Ramos I. *An OO Methodological Approach for Making Automated Prototyping Feasible.* Proceedings of DEXA96, Springer-Verlang, September 1996.

Pastor97 Pastor O., Insfrán E., Pelechano V., Romero J., M erseguer J. *OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods.* LNCS 1250, ISBN 3-540-63107-0, In Proceedings of 9th International Conference, CAISE -97, Barcelona, Spain. Ed. Springer-Verlag, June 1997, pages 145-159

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, May 2000, págs. 3 02-304.

Wieringa98 Wieringa R.J. *Postmodern Software Design with NYAM: Not Yet Another Method*. Requirements Targeting Software and Systems Engineering, 1998. Pag. 69-94. Springer.