OO-Sketch: una herramienta para la captura de requisitos de interacción[1]

José Ignacio Panach, Sergio España, Inés Pederiva, Óscar Pastor

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46071 Valencia, España
{jpanach, sergio.espana, ipederiva, opastor}@dsic.upv.es
+34 96 387 7000

Resumen. Actualmente no existe un método ampliamente aceptado para la captura de requisitos de interacción. El dibujo de bocetos de la interfaz de usuario va cobrando poco a poco un papel importante dentro de este ámbito. Es necesario el desarrollo de herramientas que permitan dibujar estos bocetos en formato digital. Además, siguiendo el paradigma MDA, el esfuerzo aplicado en la captura de requisitos debería verse reflejado en etapas posteriores de Modelado Conceptual. La herramienta encargada de soportar el dibujo de bocetos también debería ser capaz de realizar esta transformación desde los requisitos al Modelo Conceptual. El objetivo de este trabajo es abordar ambos aspectos. Por un lado, se presentan las primitivas para realizar los bocetos. Por otro lado, se presenta la estrategia abordada para su transformación automática hacia el Modelo Conceptual definido por OO-Method, un método de producción de software conforme con MDA. La herramienta de edición de bocetos, llamada OO-Sketch, alcanzaría su máxima potencia como parte de la tecnología ONME, la cual incluye un compilador de modelos que, a partir del Modelo Conceptual, es capaz de generar la correspondiente aplicación software totalmente funcional.

1 Introducción

El campo de la Ingeniería del Software (IS) ha mostrado una preocupación histórica por el modelado de los requisitos funcionales de los sistemas. Sin embargo, así como hay modelos ampliamente conocidos para la captura de requisitos funcionales, no existe un modelado comúnmente aceptado para los requisitos de interacción con el usuario. Para encontrar estudios sobre este tipo de requisitos, es necesario recurrir a la comunidad de Interacción Persona Ordenador (IPO).

Una de las técnicas usadas dentro del ámbito de la comunidad IPO para la captura de requisitos de interfaz de usuario es el uso de bocetos (*sketches*). Esta técnica se basa en el dibujo a mano alzada de las interfaces a través de las cuales el usuario quiere interactuar con el sistema de información. Esta forma de modelar tiene como principal

[1] Este trabajo ha sido desarrollado con el soporte del MEC bajo el proyecto DESTINO TIN2004-03534 y cofinanciado por FEDER. También ha participado el programa de becas FPI de la Generalitat Valenciana.

ventaja su simplicidad a la hora de ser entendida por el usuario final ya que, aunque sus conocimientos en informática sean escasos, le será fácil razonar con estos bocetos sobre el aspecto que presentará la aplicación software. Hoy en día son numerosas las aplicaciones que se han construido para soportar el dibujo de estos bocetos en formato digital, diseñándolos con variados dispositivos de interacción (p.e. lápices electrónicos). De esta manera, el usuario final de la aplicación puede involucrarse en la fase de captura de requisitos de interacción. Este hecho facilita que la aplicación final generada sea más usable para el usuario, ya que éste puede validar desde las primeras fases de desarrollo del software si los bocetos cumplen los requisitos de interacción que él mismo plantea. La propuesta del trabajo presentado en este artículo está basada en esta línea de soporte digital para los bocetos. La idea es engarzar esta técnica de diseño temprano de interfaz en un marco de desarrollo de software que sigue el paradigma Model Driven Architecture (MDA) [9], con el fin de reaprovechar las especificaciones realizadas en la etapa de Captura de Requisitos para derivar parte de los Modelos de Análisis. Como caso particular, se pretende introducir esta técnica dentro de OO-Method [13], un método de producción de software que permite la generación automática de código. Este método se basa en la separación de dos niveles de abstracción diferentes: el espacio del problema y del espacio de la solución. Esto sitúa a OO-Method como una metodología para implementar herramientas siguiendo las directrices MDA, ya que separa la especificación conceptual de las aplicaciones de sus posibles implementaciones software. OO-Method tiene su implementación industrial en la tecnología OlivaNova Model Execution[2] (ONME). En la Figura 1 se presenta el proceso de desarrollo de software OO-Method, desde la captura de requisitos a la cual proponemos incluir una actividad de Captura de Requisitos de Interacción, hasta la generación automática del código final mediante un compilador de modelos.

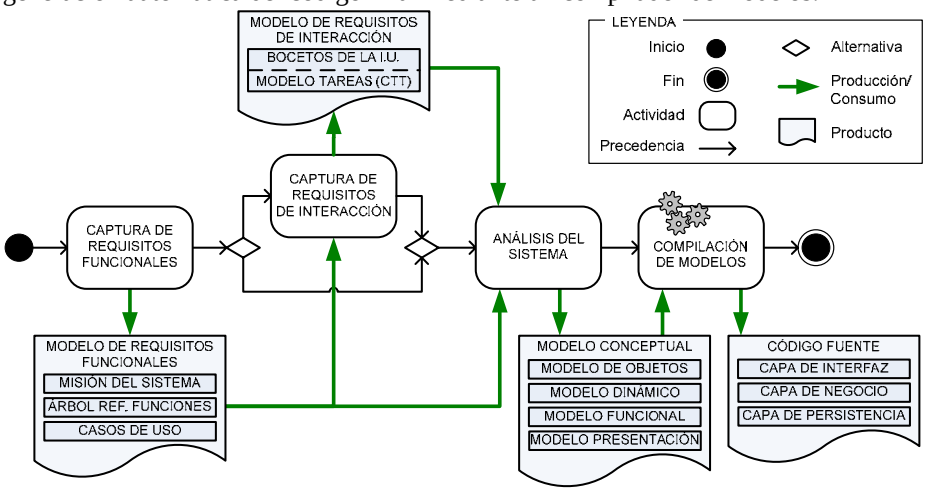

Fig. 1. Proceso de desarrollo de sistemas software con OO-Method

En este trabajo se presenta la estrategia adoptada para la especificación de los requisitos de interacción. Se propone un modelo combinado con dos vistas: bocetos de

[2] Care Technologies: http://www.care-t.com

interfaz y modelos de tareas en la notación ConcurTaskTrees (CTT) [14]. En este artículo se presentan las primitivas de los bocetos de interfaz; es decir, la manera en que se va a permitir construir los bocetos. Estas primitivas se presentan a nivel conceptual con el fin de implementarlas en la herramienta de construcción de bocetos OO-Sketch en trabajos futuros. Una interesante aportación al campo del modelado de interfaces es que estas primitivas gráficas se hacen corresponder con patrones estructurales de tareas, que son las piezas con las que proponemos componer los modelos de tareas. Así, los modelos de tareas se convierten en la representación estructurada y algebraica de los bocetos, de manera que constituyen un soporte formal para su repositorio. Ambas vistas se construyen de manera simultánea y el sustrato CTT resulta transparente para el ingeniero de requisitos.

Otra aportación de este trabajo es la posibilidad de encuadrar el diseño de los bocetos en un marco de producción automática de software. En trabajos anteriores [12] fueron definidos los patrones estructurales de tareas y su correspondencia con patrones del Modelo de Presentación OO-Method, a partir del cual es posible la generación del código de la aplicación software. El presente trabajo amplía esta estrategia mediante la inclusión de bocetos, poniéndola al alcance de los usuarios.

La estructura del documento es la siguiente: en la sección 2 se presenta el método de producción automática de software OO-Method, en la sección 3 se muestran las correspondencias entre los bocetos y los árboles estructurales de tareas, en la sección 4 se exponen las distintas variedades de interacción que debe ofrecer la herramienta presentada, en la sección 5 se aplican todos estos conceptos sobre un caso de estudio, en la sección 6 se realiza un estudio del estado del arte de otras herramientas ya existentes que soportan el dibujo de bocetos, identificando sus bondades y carencias y, finalmente, en la sección 7 se presentan las conclusiones del trabajo.

2 OO-Method: hacia un proceso de desarrollo holístico

OO-Method es un método para la construcción de sistemas software; abarca desde la fase de requisitos hasta la generación automática del código final. En la Figura 1 se muestra gráficamente el proceso de producción y los modelos que lo soportan.

En una primera fase se lleva a cabo la Captura de Requisitos. Consideremos un modelo de requisitos funcionales compuesto por la Misión del Sistema (una descripción del ámbito del sistema), el Árbol de Refinamiento de Funciones (una jerarquía de descomposición de funciones que debe ofrecer el sistema) y un modelo de Casos de Uso (cada hoja del árbol se considera un caso de uso, que es refinado mediante plantillas de escenario).

Estos artefactos facilitan la fase de Análisis, durante la cual se realiza el modelado conceptual del sistema. El Modelo Conceptual describe de manera prescriptiva los tres ejes de un sistema de información:

- El *Modelo de Objetos* permite especificar la vista estática, la estructura de las clases identificadas en el dominio del problema, así como las relaciones estructurales y las relaciones de agentes sobre los servicios de las clases.
- El Modelo Dinámico y el Modelo Funcional se centran en el comportamiento del sistema. El *Modelo Dinámico* determina las posibles secuencias de eventos que

pueden ocurrir en la vida de los objetos. El *Modelo Funcional* especifica el efecto que tienen los eventos sobre el estado de los objetos.

- Por último, el *Modelo de Presentación* ofrece una descripción abstracta de la interfaz del sistema. Este modelo está estructurado en tres niveles de patrones de interfaz [10]:

  Nivel 1. **Árbol de jerarquía de acciones**: siguiendo el principio de aproximación gradual [7] expresa cómo la funcionalidad será presentada al usuario que acceda al sistema. Es una abstracción del menú de la aplicación que da acceso a los patrones del siguiente nivel.

  Nivel 2. **Unidades de interacción** (UI): modela las unidades de interfaz abstracta que el usuario deberá emplear para llevar a cabo sus tareas. Cuatro patrones pueden combinarse para conformar la interfaz: la UI de Instancia modela la presentación de los datos de un objeto del dominio, la UI de Población permite mostrar una lista de instancias, la UI de Maestro-Detalle permite hacer combinaciones de las anteriores para presentaciones más complejas, la UI de Servicio modela un formulario de entrada o modificación de datos. Su estructura y comportamiento puede especificarse con más detalle mediante patrones de nivel 3.

  Nivel 3. **Patrones elementales**: Permiten restringir y precisar el comportamiento de las diferentes unidades de interacción. Con ellos se definen filtros, ordenaciones, acciones, navegaciones, etc.

A partir del Modelo Conceptual se aplica la Compilación de Modelos y se obtiene una aplicación software completamente funcional. El código final está organizado en una arquitectura de tres capas: la de interfaz, la de lógica y la de persistencia.

OO-Method define un proceso de desarrollo basado en el modelado conceptual. El Modelo Conceptual definido, artefacto estrella de esta aproximación, tiene en consideración tanto los aspectos estáticos del sistema, como su dinámica y la interacción con agentes externos; ofrece mecanismos para la descripción del sistema como un todo. Sin embargo, el Modelo de Requisitos Funcionales necesita ser complementado con un Modelo de Requisitos de Interacción que permita la captura de las necesidades interactivas de los usuarios. En trabajos anteriores [4] se ha resuelto esta carencia mediante el uso de modelos de tareas, utilizando la notación CTT. El trabajo que presentamos extiende esta propuesta superponiendo al árbol de tareas una capa de bocetos de interfaz. De esta manera es posible implicar al usuario más directamente en la captura de requisitos de interacción. La principal contribución del trabajo es la definición de correspondencias entre la capa de bocetos y la capa de tareas que permiten la construcción simultánea de ambos modelos. Al editar el boceto, es posible construir de manera automática el árbol de tareas subyacente.

3 Construcción sincrónica de bocetos y modelos de tareas

Los bocetos son un modelo temprano de la interfaz gráfica de usuario. Para definir un nuevo modelo, la primera cuestión es determinar su conjunto de constructores elementales. Se trata de definir las primitivas con las que se podrá construir un boceto de interfaz. Por las características eminentemente gráficas de un boceto, la concepción de estas primitivas está muy ligada a su representación visual. Conviene separar varios

aspectos: (1) la propia primitiva, el concepto que hay detrás del constructor elemental definido; (2) por otro lado aparece la interacción con la herramienta, la variedad de formas en que se puede añadir (y manipular) la primitiva a un boceto; (3) otro aspecto es la variada representación visual que dicha primitiva tiene en el boceto.

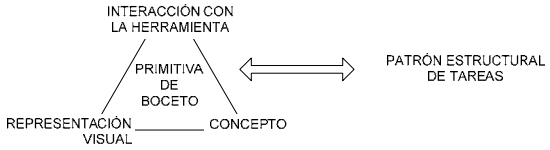

Fig. 2. Aspectos ligados a la definición de primitivas de boceto

A continuación se presentan los aspectos conceptuales, acompañándolos de la representación visual cuando sea conveniente. La estrategia que la herramienta OO-Sketch debe proporcionar para la construcción sincrónica de los bocetos y sus árboles CTT equivalentes, hace aconsejable una correspondencia biunívoca entre las primitivas de boceto y los patrones estructurales de tareas. De esta manera, ya queda definido el conjunto de conceptos: por cada patrón estructural de tareas debe definirse una primitiva de boceto. Por motivos de espacio, este trabajo está centrado en los patrones estructurales de tareas relacionados con el listado de instancias.

3.1. Población

El objetivo es presentar al usuario un listado con instancias de una clase de objetos de negocio (p.e. un listado con los clientes de la empresa). En [12] se presentaron los patrones estructurales de tareas vinculados con esta Unidad de Interacción.

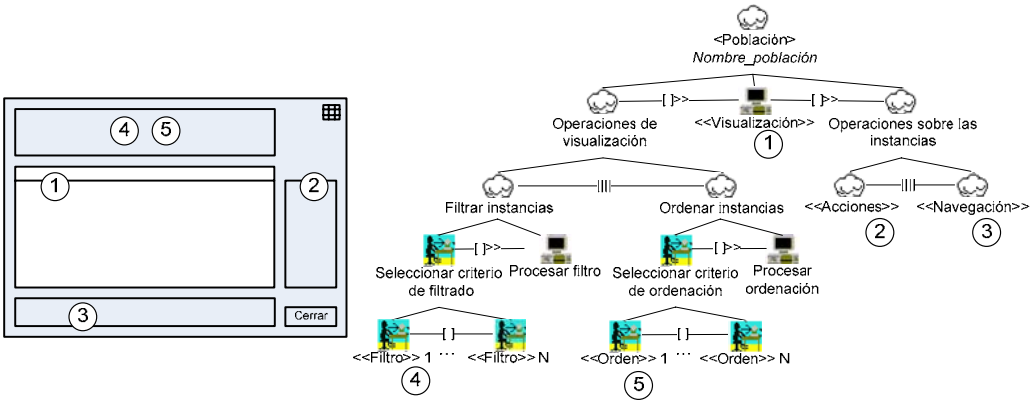

Fig. 3. Primitiva gráfica y patrón estructural de tareas para la Población.

Para dotar de flexibilidad gramatical a los modelos de tareas, se propuso un mecanismo de composición de patrones: los dobles corchetes << >> denotan puntos donde se enganchan otros patrones estructurales de tareas. Los círculos con números se incluyen en las figuras como ayuda para facilitar el entendimiento de los vínculos entre las primitivas.

Como se observa en la Figura 3, la primitiva de boceto para la Población de instancias consiste en un contenedor marcado con un icono de rejilla en la esquina superior derecha, según la notación propuesta en [10] para este patrón. En el interior del contenedor aparecen espacios delimitados para agregar otras primitivas de boceto.

3.2. Visualización

La primitiva de Visualización permite especificar gráficamente los campos que se mostrarán de las instancias de la población. Se trata de un conjunto de columnas a las que se puede asignar nombre. En la fase de análisis, cuando derivemos el Modelo de Objetos, los campos aquí definidos estarán vinculados con atributos de las clases.

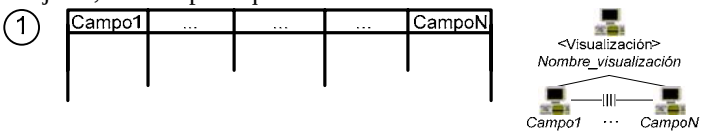

Fig. 4. Primitiva gráfica y patrón estructural de tareas para la Visualización.

3.3. Acciones

Las Acciones son operaciones que se pueden realizar sobre la instancia seleccionada entre las listadas por la Población. Algunas acciones son muy habituales (p.e. la creación de nuevas instancias, la modificación y el borrado), otras son más particulares del sistema que estamos bosquejando.

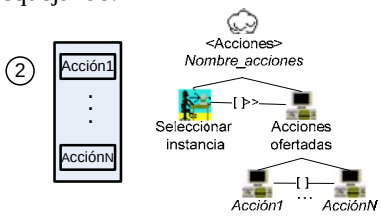

Fig. 5. Primitiva gráfica y patrón estructural de tareas para las Acciones.

3.4. Navegación

La Navegación permite acceder a otras pantallas de la aplicación. Si bien el menú de una aplicación constituye una navegación taxonómica (organiza el acceso a la funcionalidad del sistema), la primitiva de boceto que nos ocupa define una navegación estructural, a través de la cual se accede a partes de la aplicación que soportan la edición o consulta de objetos del negocio estructuralmente asociados con el objeto origen. Por ejemplo, desde una población de líneas de factura podríamos navegar a la ficha del cliente o al pedido del cual proceden.

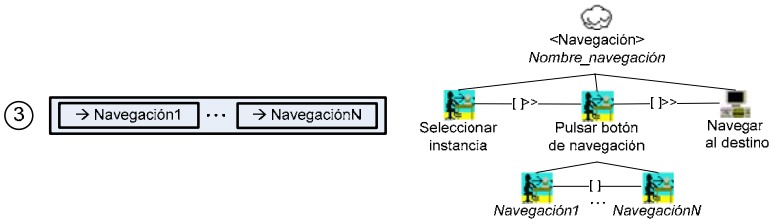

Fig. 6. Primitiva gráfica y patrón estructural de tareas para la Navegación.

3.5. Filtro

La primitiva de boceto del Filtro permite especificar un criterio de filtrado para las instancias listadas por la población. Para ello se realizan marcas sobre los campos por los cuales se desea filtrar. Estos campos marcados instancian los argumentos del patrón estructural de tareas correspondiente.

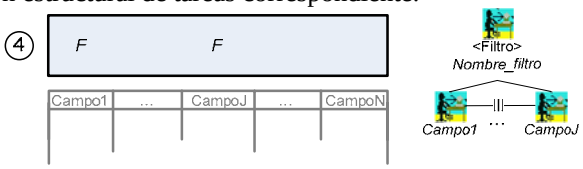

Fig. 7. Primitiva gráfica y patrón estructural de tareas para el Filtro.

3.6. Ordenación

La primitiva de Ordenación se define de manera similar a la de Filtro, realizando una serie de marcas sobre los campos por los cuales se desea una ordenación. Junto con la primitiva de Filtro, proporciona al usuario el tratamiento sencillo de poblaciones muy extensas.

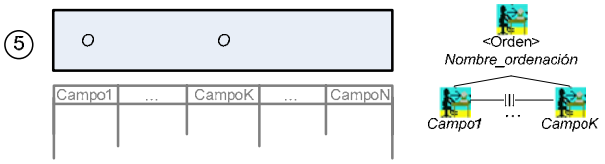

Fig. 8. Primitiva gráfica y patrón estructural de tareas para la Ordenación.

4 Consideraciones sobre la usabilidad de la herramienta

Las herramientas de boceto de interfaces se caracterizan por ofrecer una interacción ágil para construir el boceto de las ventanas. Los avances tecnológicos en materia de reconocimiento de formas y escritura, unidos al creciente interés por los desarrollos dirigidos por modelos, han hecho posible la aparición de herramientas que interpretan

las interacciones del diseñador para ir construyendo un modelo estructurado de la interfaz dibujada. Para que una herramienta de este tipo tenga aceptación entre los usuarios, es necesario que sea flexible en sus formas de interacción.

La herramienta que planeamos construir debe ofrecer gran variedad en la edición de las primitivas de boceto. A continuación, se desarrolla el vértice superior de la Figura 2 (la interacción con la herramienta), y se elaboran ciertos aspectos de la representación visual que completan las definiciones de la sección anterior.

Se conciben dos vertientes interactivas principales, que dependerán en gran medida de los dispositivos hardware de interacción utilizados por el diseñador.

- **Lápiz**: Permite trazar los bocetos con rapidez, pero la introducción de texto requiere de técnicas de reconocimiento de formas textuales. Especialmente apropiado para el uso con PDAs y tablet PCs, con portátiles y ordenadores de sobremesa se acompaña de tableta digitalizadora. Es indicado para reuniones con el usuario.
- **Ratón y teclado**: La flexibilidad a la hora de dibujar formas geométricas es menor; sin embargo la introducción de texto es más precisa. Es indicado para completar, en una segunda iteración, los bocetos realizados con el usuario.

Concebimos una herramienta de edición de bocetos que permita la inclusión de las primitivas de varias maneras, siempre buscando la mínima interacción necesaria para llevar esto a cabo. Por ejemplo, la primitiva de población es una estructura gráfica compleja, pero basta trazar una forma rectangular con un icono de rejilla para que la herramienta pueda interpretar que debe colocar la primitiva (interacción con lápiz, ver Figura 9). Otra opción es arrastrar la primitiva desde una librería hasta el modelo activo (interacción con ratón, ver Figura 10).

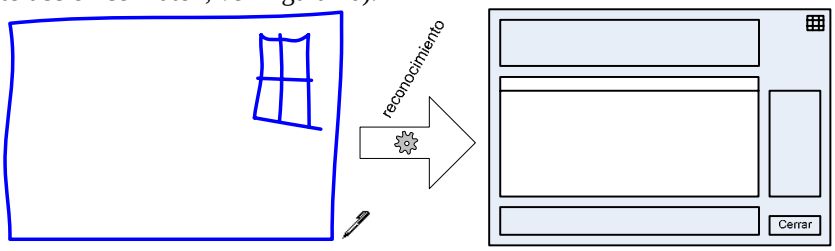

Fig. 9. Trazado de la primitiva y reconocimiento (modo de interacción con lápiz).

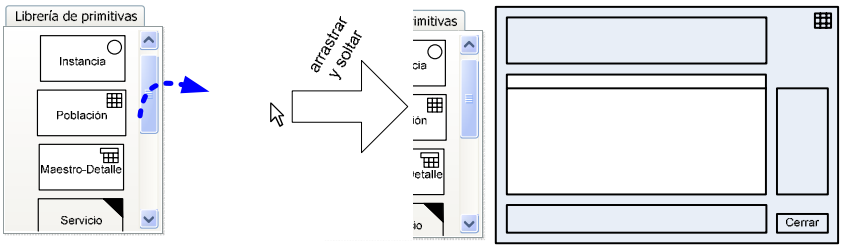

Fig. 10. Librería de primitivas (modo de interacción con ratón).

Nuestra propuesta para la interacción con lápiz supone, además, una concepción del sketching que mejora la calidad visual de los bocetos diseñados: al reconocer una

primitiva, se sustituye el trazo del diseñador por la representación gráfica correspondiente. De esta manera, el resultado es más homogéneo (véase la Figura 9).

5 Caso de estudio

Como caso de estudio se ha elegido un sistema de gestión de una empresa de suministro de agua. Esta aplicación tiene como finalidad la gestión de clientes, de materiales, registro de las lecturas en los contadores, emisión de facturas y un control sobre las tareas que cada operario realiza. Para simplificar el caso de estudio, se ha elegido la tarea **Listar Contadores,** que se utiliza para obtener un listado con todos los contadores del sistema. El boceto que dibujaría el analista para capturar los requisitos de interfaz sería similar al de la Figura 11.

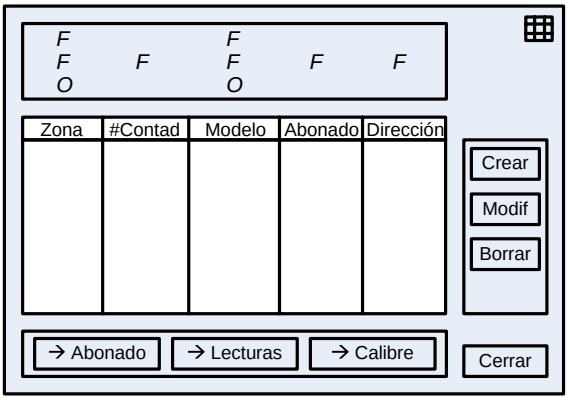

Fig. 11. Boceto para Listar Contadores

La Figura 11 muestra un boceto que se convertirá en una Unidad de Interacción de Población una vez aplicada la transformación de modelos. El listado de contadores puede ser filtrado y ordenado, utilizando para ello las primitivas Filtro (F) y Orden (O). En la Figura 11 se aprecian dos filtros, representados por dos filas de F's. Cada uno de ellos utiliza como filtrado los campos en los cuales aparece la marca F. En el boceto quedan reflejados los campos que se mostrarán de los contadores, las acciones que se puedan ejecutar sobra las instancias y las navegaciones que se pueden realizar.

Mientras el analista va dibujando los bocetos con el usuario, se van creando los árboles CTT que representan las interfaces dibujadas. El árbol de la tarea estudiada (ver Figura 12) se ha construido utilizando las transformaciones del apartado 4.

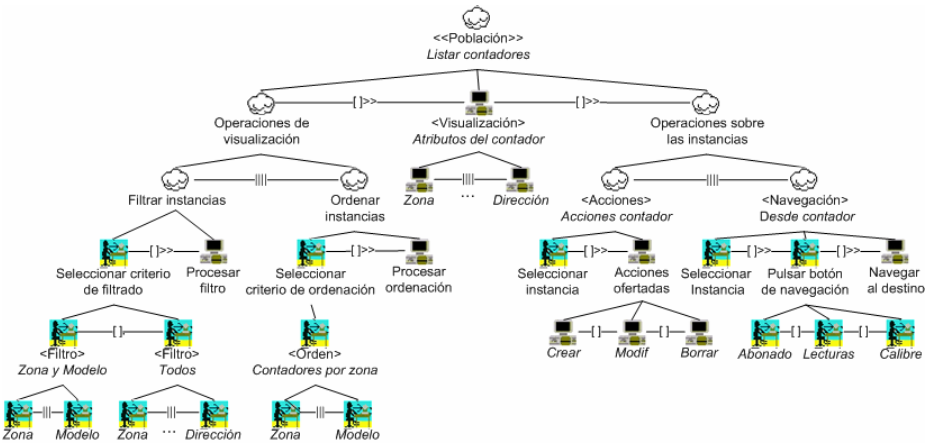

Fig. 12. Árbol CTT para la tarea Listar contadores

Tal y como se explica en trabajos anteriores [12], una vez se han construido los árboles CTT, el analista puede derivar automáticamente el Modelo de Presentación de OO-Method. Para ello se utilizan las correspondencias definidas entre los patrones estructurales de tareas y los patrones del Modelo de Presentación. Construido el Modelo de Presentación, la generación de la interfaz final es también totalmente automática, utilizando para ello el compilador de modelos de ONME.

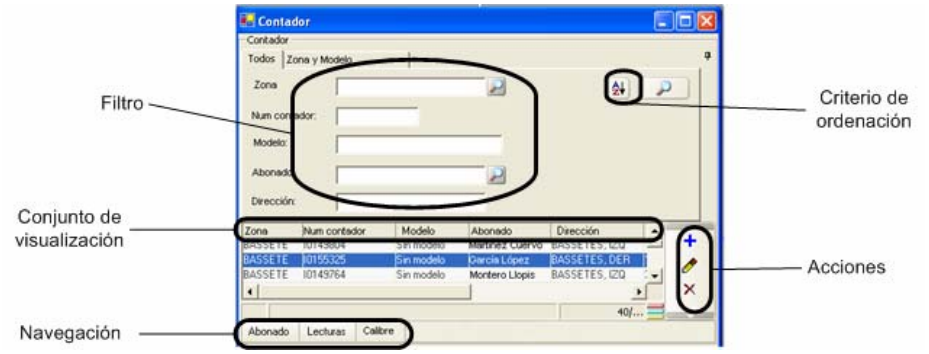

Fig. 13. Interfaz final para Listar contadores

La Figura 13 muestra la interfaz final para la tarea estudiada. En ella se pueden apreciar las características dibujadas en el boceto, es decir, los filtros, ordenaciones, navegaciones, campos que se visualizan y acciones.

6 Trabajos relacionados

Actualmente existe una cierta variedad de herramientas para realizar el dibujo de bocetos de interfaz y, a partir de estos, obtener una interfaz final de manera automática.

En esta sección se presentan algunas de las más relevantes. Por un lado están las herramientas específicas para el diseño de un determinado tipo de aplicaciones; así ocurre en el caso de DEMAIS [1], especialmente pensada para diseñar aplicaciones multimedia. Esta herramienta permite a un diseñador plasmar las ideas de interacción y de temporalidad y ver el resultado obtenido con la herramienta. Fue desarrollado para el uso sobre una pantalla electrónica que permite al diseñador dibujar directamente sobre ésta. Para crear el diseño, DEMAIS proporciona *storyboards*, scripts de voz y editores multivista.

Otra herramienta similar es DENIM [11], que ayuda a los diseñadores de sitios web durante las primeras fases de diseño a través de bocetos a diferentes niveles de refinamiento: el mapa del sitio, *storyboard* y páginas individuales. También unifica los niveles a través de vistas. Al igual que la herramienta anterior, los dibujos están pensados para ser realizados sobre la pantalla usando un lápiz electrónico.

Por otro lado se encuentran algunas herramientas utilizadas para diseñar aplicaciones de cualquier tipo. Una de estas herramientas es SILK [8], que permite dibujar los bocetos tanto con pantalla gráfica como con ratón. El usuario puede usar cuatro primitivas: rectángulo, línea libre (para el texto), línea recta y elipse. Combinando estas primitivas se forman los prototipos de interfaz, que serán transformados mediante la herramienta en ventanas reales en Visual Basic 5.0 o Common Lisp. El diseñador puede elegir el estilo de los componentes de la interfaz una vez la herramienta reconoce a cada uno de esos componentes (p.e. el tipo de botón). Para ilustrar la navegación entre las interfaces se utilizan *storyboards*.

Otra herramienta similar es JavaSketchIt [2] que, al igual que la anterior, permite el dibujo de bocetos sobre una pantalla usando un lápiz electrónico. Para ello, usa una combinación de figuras simples que representan cada uno de los componentes de interfaz posibles (*widgets*). La identificación de estas figuras se realiza utilizando la librería CALI [4]. Cuando no es posible la identificación inequívoca de una figura, el sistema de reconocimiento devuelve una lista de posibles candidatos. La aplicación puede entonces elegir el mejor candidato dependiendo de la información contextual. Identificados todos los componentes gráficos se genera la interfaz en código Java.

Freeform [15] es otra herramienta basada en el dibujo de bocetos sobre una pantalla electrónica para diseñar formularios de Visual Basic. La herramienta tiene un espacio para trazar los bocetos, donde se puede dibujar, escribir sobre ellos y editar. El texto escrito a mano es interpretado posteriormente usando el algoritmo de Rubine [16]. Antes de que se genere el formulario en Visual Basic, el usuario puede modificar las decisiones de reconocimiento que ha llevado a cabo la herramienta, eligiendo otro tipo de control a través de una lista. Para la navegación entre bocetos se utilizan también los *storyboards*. Dispone de un modo de ejecución en el que el usuario puede probar las navegaciones y escribir sobre los componentes dibujados con el editor. Esta herramienta también posee la funcionalidad de alinear los controles creados y determinar un tamaño estándar para esos controles.

Por último, existe una herramienta que no genera código específico para un determinado lenguaje de programación. SketchiXML [3] genera especificaciones de interfaces de usuario en UsiXML [17], un lenguaje de descripción de interfaces de usuario independiente de plataforma. A partir de UsiXML se puede generar código en HTML, XHTML y Java, mediante el uso de otras herramientas de su suite. El usuario de SketchiXML traza los dibujos que luego son reconocidos utilizando la librería CALI,

al igual que en JavaSketchIt. Esta herramienta es capaz de avisar al usuario sobre potenciales problemas de usabilidad en los bocetos dibujados. Una vez se ha completado la fase de diseño, SketchiXML analiza sintácticamente la información de diseño (*parsing*) para generar una especificación UsiXML. Existen reglas de transformación para adaptar un modelo UsiXML creado en un contexto concreto a otro distinto (p.e. pasar de un diseño para PDA a uno para ordenador de sobremesa). Además, las representaciones de las figuras que se pueden dibujar en el boceto son configurables por el usuario.

En cualquier caso, existe una limitación común a todas las herramientas descritas anteriormente: solo generan la interfaz del sistema software, y en algunos casos, la navegación entre interfaces. La propuesta de este trabajo es una aportación notable al área del diseño de bocetos: OO-Sketch está concebida para incorporarse a un proceso de generación automática de código completamente funcional. Como caso particular, describimos su inclusión en la tecnología ONME, donde no solo se genera la interfaz de la aplicación software, sino toda su funcionalidad del sistema de información.

OO-Sketch es, junto con SketchiXML, una de las pocas herramientas de bocetos totalmente independiente del lenguaje de implementación que se generará. La mayoría de herramientas están pensadas para lenguajes de implementación específicos, como es el caso de Freeform o JavaSketchIt. Sin embargo, a partir de los bocetos de OO-Sketch se pueden generar aplicaciones en Visual Basic, .NET, Java/Swing, ColdFusion, JSP, ASP y PocketPC.

Por último, cabe destacar que la propuesta OO-Sketch está pensada para ser ejecutada tanto en ordenadores de sobremesa como en dispositivos móviles. La inserción de texto puede realizarse bien mediante un lápiz electrónico cómo a través de un teclado y las figuras se pueden dibujar tanto con un lápiz electrónico, como con un ratón. La variedad interactiva depende del tipo de dispositivo sobre el que se dibuje el boceto. Las herramientas anteriormente citadas no proporcionan esta flexibilidad a la hora de realizar los bocetos; si bien SILK permite el dibujo de figuras mediante lápiz electrónico o ratón, el texto únicamente puede ser introducido mediante teclado.

7 Conclusiones y trabajos futuros

En este trabajo se ha presentado una técnica de generación automática de interfaces a partir de bocetos. Esta técnica, unida a la compilación de modelos ya existente en la tecnología ONME, permite que se generen sistemas software totalmente ejecutables. Realizando el Modelado Conceptual y los bocetos no es necesario escribir ni una sola línea de código.

Para materializar estas ideas, se ha definido una serie de transformaciones que permite construir sincrónicamente bocetos de interfaz de usuario y árboles estructurales de tareas (CTT). Una vez construidos estos árboles, se puede generar automáticamente el Modelo de Presentación que utiliza ONME. A partir de este Modelo de Presentación, junto con los otros modelos que habrá definido el analista (Modelo de Objetos, Modelo Dinámico, Modelo Funcional) se genera el sistema software de manera automática.

Como trabajo futuro contemplamos la implementación de una herramienta capaz de dibujar bocetos que generen árboles CTT de manera transparente para el analista. Para ello la herramienta debería disponer de un reconocedor de formas para identificar los componentes de los bocetos dibujados con lápices electrónicos. Por otro lado, utilizando las correspondencias entre primitivas de boceto y patrones estructurales de tareas descritas en este artículo, se implementaría la creación sincrónica de árboles de tareas a partir de los bocetos. Esta funcionalidad, unida a la transformación del modelo de tareas al Modelo de Presentación y del Modelo de Presentación a la interfaz final, proporcionará una herramienta capaz de generar interfaces de usuario de manera automática con una gran rapidez, lo que permitirá una rápida realimentación del usuario

Otra ventaja de usar un lenguaje formal como CTT es la posibilidad de validar los bocetos con los cuales están vinculados. Así pues, se pretende implementar un validador sintáctico de árboles de tareas de manera que solo se permitan árboles CTT (y por lo tanto bocetos) válidos.

Por último, se planea realizar un estudio empírico de la propuesta. Para ello se llevarán a cabo una serie de experimentos sobre la herramienta OO-Sketch con el objetivo de comprobar las mejoras metodológicas que conlleva su uso. Del mismo modo, se llevarán a cabo evaluaciones de la usabilidad tanto en el uso de la herramienta como de las aplicaciones que se generen. Además, se estudiará la posibilidad de incorporar a la herramienta OO-Sketch un evaluador temprano de usabilidad [5] que realice predicciones parciales sobre la usabilidad de la aplicación software que se esté modelando.

8 Referencias

[1] Bailey, B. P., Konstan, J.A. (2003). Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. En *Actas del Human Factors in Computing Systems CHI'2003*. New York: ACM Press.

[2] Caetano, A., Goulart, N., Fonseca, M., Jorge, J. (2002). JavaSketchIt: Issues in Sketching the Look of User Interfaces. En *Actas de AAAI Spring Symposium - Sketch understanding*.AAAI Press; pp. 9–14.

[3] Coyette, A., Vanderdonckt, J. (2005). A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. En *Actas de INTERACT 2005*, LNCS 3585; pp. 550-564.

[4] España, S., Panach, I., Pederiva, I., Pastor, O. (2006) Towards a Holistic Conceptual Modelling-Based Software Development Process. En *Actas de 25th International Conference on Conceptual Modeling (ER2006)*; Tucson, Arizona, USA; pp. 437-450

[5] España, S., Pederiva, I., Panach, J.I., Abrahão, S., Pastor, O. (2006). Evaluación de la Usabilidad en un Entorno de Arquitecturas Orientadas a Modelos. En *Actas del 9° Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS 2006)*. La Plata, Buenos Aires, Argentina; pp. 331-334.

[6] Hong, J. I., Li, F.C., Lin, J., Landay, J.A. (2001). End-User Perceptions of Formal and Informal Representations of Web Sites. En *Extended Abstracts of CHI'2001*; pp. 385–386.

[7] ISO/IEC TR 9126-4 (2004). *Software engineering - Product quality - Part 4: Quality in use metrics*.

[8] Landay, J., Myers, B.A. (2001). Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer 34*. pp. 56–64.

[9] MDA: http://www.omg.org/mda Última visita junio 2006.

[10] Molina, P. J. (2003). *Especificación de interfaz de usuario: de los requisitos a la generación automática*. PhD. Departamento de Sistemas Informáticos y Computación. Valencia, Universidad Politécnica de Valencia; 382 páginas.

[11] Newman, M. W., Lin, J., Hong, J.I., Landay, J.A. (2003). DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice." *Human-Comp. Interaction 18*; pp. 259–324.

[12] Panach, I., Pederiva, I., España, S., Pastor, O. (2006). Generación Automática de Interfaces a Partir de Patrones Estructurales de Tareas. En *Actas de Interacción 2006*, Puertollano, España.

[13] Pastor, O., Gómez, J., Insfrán, E. Pelechano, V. (2001) The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems, 26*(7) 507–534.

[14] Paternò, F., C. Mancini, et al. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. En *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*, Chapman & Hall, Ltd. pp. 362-369.

[15] Plimmer, B. E., Apperley, M. (2003). Software for Students to Sketch Interface Designs. En *Proc. of IFIP Conf. on Human-Computer Interaction INTERACT'2003*, IOS Press; pp. 73–80.

[16] Rubine, D., (1991) Specifying gestures by example. En actas de *18th annual conference on Computer graphics and interactive techniques*, ACM Press; pp. 329-337.

[17] Vanderdonckt, J., Q. Limbourg, B. Michotte, et al. (2004) USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. En actas de *W3C Workshop on Multimodal Interaction WMI'2004*. Sophia Antipolis, Grecia.