Integración entre Rainbow y OO-Method

Vanessa Del Río Boix

Tesina del Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

Dirigida por **Ignacio Panach Navarrete (Universitat de València)**

Nathalie Aquino Salvioni (Universitat Politècnica de València)

Óscar Pastor López (Universitat Politècnica de València)

ÍNDICE

1. INTRODUCCIÓN

1.1. Propósito del documento

Uno de los problemas habituales en el desarrollo de software es que la solución proporcionada por los desarrolladores no da soporte a todas las necesidades de los usuarios finales. Esto es debido a que los usuarios finales no participan activamente en el desarrollo del sistema, lo que provoca que no puedan plasmar todas sus necesidades.

El propósito de este trabajo es, basándose en *OO-Method* [1] mejorar la metodología empleada por este durante la fase de especificación, utilizando las características y herramientas que *RAINBOW [2]* proporciona para la especificación de los aspectos de la futura aplicación.

1.2. Objetivo del proyecto

Para el desarrollo de un sistema software, el modelado de datos juega un papel primordial ya que define el núcleo semántico de la aplicación en el futuro.
El modelado de datos precisa de la obtención y validación de los requisitos por parte de los usuarios. Este es un aspecto fundamental para construir un sistema de información fiable así como para generar una documentación consistente del dominio de la aplicación.

En el desarrollo de un sistema la primera fase consiste en la elicitación o captura de requerimientos.
Existen diversas técnicas para proceder a la captura de requerimientos. Las técnicas más habituales en la elicitación de requerimientos son las entrevistas, el Joint Application Development (JAD) o Desarrollo Conjunto de Aplicaciones, el brainstorming o tormenta de ideas y la utilización de escenarios, más conocidos como casos de uso.

Estas técnicas, se suelen apoyar con otras técnicas complementarias como la observación in situ, el estudio de documentación, los cuestionarios, la inmersión en el negocio del cliente o haciendo que los ingenieros de requisitos sean aprendices del cliente…

Tras la captura de los requerimientos se procede al análisis de los mismos y posteriormente a su validación por parte de los usuarios antes de trasladarlos al modelo conceptual.
A pesar de que tras realizar el análisis de los requerimientos estos son validados por los usuarios, en muchas ocasiones los requisitos no se precisan con exactitud o se expresan de forma ambigua y esto conlleva que sean interpretados de manera diferente por los desarrolladores y por los usuarios provocando que el producto final no satisfaga las expectativas de los usuarios finales.

La *Ingeniería de Requisitos* podemos definirla como la determinación de las necesidades o de las condiciones a satisfacer para un software nuevo o modificado, tomando en cuenta los diversos requisitos de los usuarios, que pueden entrar en conflicto entre ellos. En otras palabras, la ingeniería de requisitos se define como la aplicación de la resolución de problemas de forma racional al análisis de las necesidades del usuario. Asumimos que la salida de esta tarea es un conjunto de propiedades que el producto ha de poseer, llamada especificación de requisitos del producto.

Dada una lista de propiedades de un producto, la tarea del modelado conceptual es encontrar una descripción explícita del comportamiento externo de un sistema.
Un modelo conceptual puede especificarse informalmente (lenguaje natural puro), semi-formalmente (lenguaje natural restringido o con la ayuda de diagramas) o formalmente (por ejemplo, con formalismos lógicos y/o algebraicos).

La razón que nos lleva a separar la *Ingeniería de Requisitos* del *modelado conceptual* es que la *Ingeniería de Requisitos* se centra en la *utilidad* y el *modelado conceptual* se centra en la *validez*. En la *Ingeniería de Requisitos* estamos especificando una respuesta a una necesidad del usuario, y en el *modelado conceptual* intentamos especificar dicha respuesta correctamente. El modelado conceptual es pues, una subactividad descriptiva que forma parte de la *Ingeniería de Requisitos* y que está orientada a encontrar una descripción correcta del comportamiento del producto software.

En el modelado conceptual, el ciclo es el siguiente: analizar las propiedades que ha de tener el producto, inducir un modelo conceptual del comportamiento, probar el modelo y evaluar los resultados de la prueba para comprobar si el modelo describe correctamente el comportamiento que ha de tener el sistema. Prácticamente, casi todos los métodos conocidos de modelado conceptual siguen este proceso que nos va a permitir construir modelos válidos. Estos métodos solamente difieren en las estructuras que modelan y en las heurísticas utilizadas para encontrar y validar sus modelos.

OO-Method es una metodología que cubre todas las fases del proceso de desarrollo de software, desde la recogida de requerimientos pasando por el desarrollo y la correspondencia del esquema conceptual hasta la generación del producto software final.
Esta metodología se explica más detalladamente en la sección 3.

El Modelo de Requisitos de OO-Method está basado en el Marco Conceptual para Requisitos definido en [21]. 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 [22].

Rainbow mantiene la misma filosofía pero se centra en la especificación de los requisitos como parte principal del proceso de la *Ingeniería de Requisitos*.
Uno de los objetivos de Rainbow es el de implicar a los usuarios finales de una manera sencilla e interactiva en el proceso de captura de requerimientos al tiempo que proporciona a los analistas un conjunto de herramientas semiautomáticas para la obtención de un conjunto de documentos de especificación y herramientas que apoyarán el desarrollo de futuras aplicaciones.
La implicación de los usuarios se consigue plasmando los requerimientos de la aplicación en interfaces basadas en formularios dibujados por ellos.
Posteriormente, se aplican técnicas de Ingeniería Inversa a estos interfaces para obtener el modelo conceptual de la aplicación.
De esta forma se consiguen evitar posibles ambigüedades en la definición de los requerimientos.

En este trabajo, proponemos ampliar y adaptar los principios del enfoque RAINBOW para que las interfaces basadas en formularios dibujadas por el usuario sean un medio para reunir las especificaciones necesarias para definir el esquema conceptual de OO-Method.

1.3. Estructura del documento

1.3.1. **Organización**

- La sección 2 presenta un estado del arte de otros autores que han definido diferentes lenguajes para la definición de interfaces.
- La sección 3 presenta la metodología OO-Method y la metodología Rainbow.
- La sección 4 muestra la correspondencia existente entre los distintos elementos de OO-Method y RAINBOW.
- La sección 5 presenta las carencias de Rainbow frente a OO-Method.
- La sección 6 donde se observan las correspondencias obtenidas en la sección 4.
- Por último la sección 7 recoge las conclusiones del documento.

1.3.2. **Convenciones tipográficas y notación empleada**

Para facilitar la lectura y la comprensión de este documento, se han adoptado las siguientes convenciones:

- Se utiliza la **Negrita** para remarcar aspectos importantes.

- Al definir un término, éste irá en cursiva y empezando por mayúsculas, por ejemplo: *Modelo de Objetos.*
- Palabras extranjeras en *cursiva*.

2. ESTADO DEL ARTE

Un lenguaje de descripción de interfaz de usuario (UIDL) consta de un lenguaje de programación de alto nivel para describir las características relevantes de una interfaz de usuario con respecto al resto de la aplicación interactiva con el fin de ser utilizado durante algunas etapas del ciclo de vida de desarrollo. Este lenguaje implica la definición de una sintaxis (¿cómo se pueden expresar estas características en términos de la lengua?) y de una semántica (¿qué significan estas características en el mundo real?).

Se puede considerar como una forma común para especificar una interfaz de usuario independiente de cualquier lenguaje de destino que servirán para implementar esta interfaz de usuario.

La necesidad de un lenguaje de descripción de interfaces de usuario (UIDL) surgió por primera vez cuando se planteo desarrollar una interfaz de usuario como un módulo de una aplicación interactiva en lugar de limitarse a una serie de códigos de líneas.

Esta idea tomo fuerza cuando se quiso modelar una interfaz de usuario con un conjunto de especificaciones y poder compartir estas especificaciones a través de grupos de interés, o generar automáticamente el código de la interfaz de usuario.

Otra de las causas de esta necesidad en cuando una interfaz de usuario requiere que se ejecute simultáneamente en plataformas informáticas diferentes.

En esta sección, se presentan algunos de los lenguajes de definición de interfaces.

2.1. Dialog and Interface Specification Language (DISL)

DISL *(Dialog y Lenguaje Interface Specification*) (DISL) se describe en Schaefer, Steffen, y Wolfgang [5]. Extiende el lenguaje UIML con el fin de permitir descripciones genéricas independientes y la modalidad de diálogo.

Las modificaciones incorporadas al lenguaje UIML principalmente consisten en la descripción de *widgets* genéricos y en mejoras en los aspectos del comportamiento. Los *widgets* genéricos se introducen con el fin de separar la presentación de la estructura y del comportamiento, es decir, principalmente para separar las propiedades de usuario y específicas de los dispositivos y de la modalidad de presentación. El uso de un atributo genérico *widget* permite asignar cada *widget* a un determinado tipo de funcionalidad (por ejemplo, comando, campo variable, campo de texto, etc.) Además, DISL puede utilizar esta información para crear componentes de interfaz asignados a la modalidad de interacción (es decir, gráfica, vocal) en el que el *widget* funcionará.

La estructura DISL global consiste en un elemento de encabezado opcional para obtener información y una colección de plantillas e interfaces de la que es considerada una interfaz para estar activos al mismo tiempo. Las interfaces se utilizan para describir la estructura de

diálogo, el estilo y comportamiento, mientras que las plantillas sólo describen la estructura y el estilo con el fin de ser reutilizables por los componentes de diálogo.

Las implementaciones actuales del lenguaje DISL incluyen la aplicación de los reproductores multimedia para archivos mp3 en dispositivos móviles con recursos limitados o los juegos de PC controlados de forma remota desde teléfonos móviles.

2.2. Generalized Interface Markup Language (GIML)

Generalized Markup Language Interface (GIML) es un lenguaje utilizado por la *Generalized Toolkit Interface* (gitk), ambos introducidos por Kost [6] en 2004. GIML se utiliza en este contexto como un descriptor de interfaz.

Siguiendo los principios de separación de los OMG, GIML divide funcionalidad y presentación. Mientras que la funcionalidad se conserva en GIML la interfaz de usuario se deriva de archivos XSL (figura 6), que provienen del usuario y perfiles del sistema.

Esta información se combina con las descripciones funcionales mediante XSLT para formar una descripción de la interfaz final.

Los datos del perfil podrían provenir directamente de un sistema de archivos o desde un servidor de perfiles remoto [6].

GIML evita el uso de conceptos tales como "*push-button*", "*scrollbar*", mientras que GIML utiliza términos tales como "*action*", "*data-entry/value-choice/single/limited*". El objetivo es utilizar los patrones de interfaz en el futuro. Estos identificadores medios neutros, son la base de una jerarquía de objetos de interfaz.

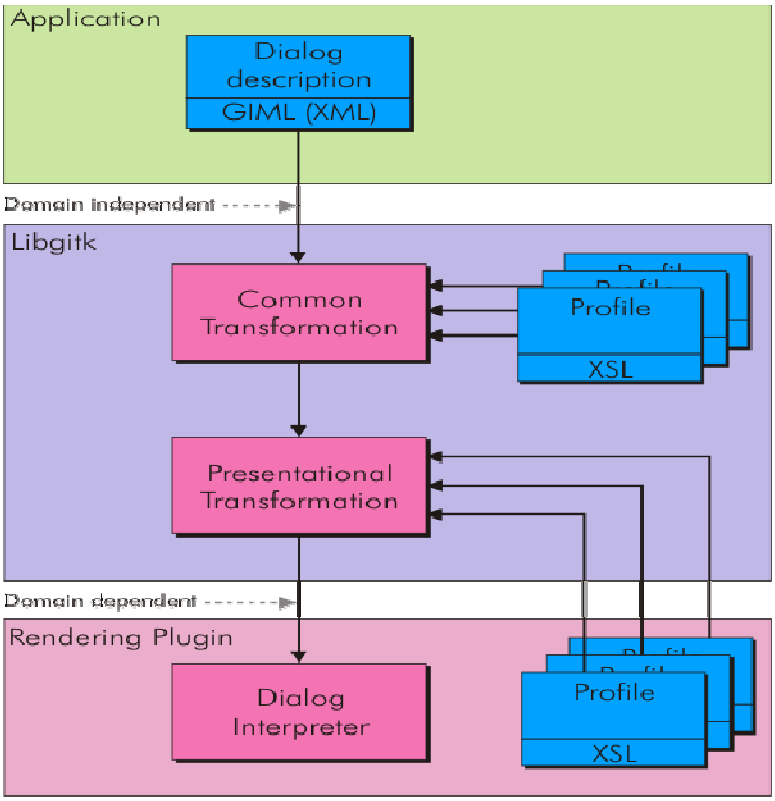

Figura 1. Perfiles XSL

En la Figura 1 se representa el enfoque de transformación basado en XSLT. En primer lugar, a una propuesta XSL se le aplica la descripción de diálogo. Las imágenes muestran que la descripción de diálogo XML es proporcionado por la aplicación. Además se puede observar, que la información adicional proviene de "perfiles" en forma de hojas de estilo XSL. Estas son proporcionadas por el sistema central y por los módulos de representación. Los conceptos compatibles con este lenguaje son:

- **Aplicación:** ofrece la descripción de diálogo funcional y la gestión de eventos codificados en C, C + +, Java o Pearl.

- **Envoltura:** estos componentes permiten al desarrollador de aplicaciones "libremente" elegir el idioma preferido para el desarrollo, C + +, Java o Pearl.

- **Núcleo:** este componente constituye la infraestructura básica. Gestiona todo lo demás (plugins, tuberías de transformación, etc.)

- **Transformador de plug-ins:** estos módulos proporcionan la transformación de medios neutrales (independientes del dominio de destino).

- **Renderer plug-ins:** estos son los intérpretes que generan y ejecutan la interfaz gráfica, basada en la Web y para GTK, y un texto para la entrada de teclado.

2.3. Multiple Device Markup Language (MDML)

Multiple Device Markup Language (MDML) elaborado por Johnson *y* Parekh [7] está basado en XUL. Los conceptos que especifica son: la navegación, el diseño y los componentes para una interfaz gráfica de usuario. Define un sistema de reglas como el mapa de especificaciones de los dispositivos: memoria, capacidades de visualización y la representación interna de los datos. Aunque la representación general se puede utilizar para todos los dispositivos, la dirección del marco de la interfaz de usuario puede producir un conjunto de reglas diferentes reglas para diferentes dispositivos o producir un conjunto de reglas diferentes para el mismo dispositivo.

La arquitectura (figura 2), utiliza el motor de transformación de Java. Se consideró que el uso de XSLT era demasiado complicado de mantener y aún más para volver a utilizar como reglas de transformación en casos de estudio dependientes.

La arquitectura se divide en cuatro partes: motor de reglas, *handler,* motor de visualización y generación de código. El motor de reglas analiza el archivo de reglas. El motor de visualización analiza el archivo MDML y manipula el diseño. El motor generará un archivo XML basado en la clase generalizada - class.dtd. La generación de código generará un archivo de idioma ejecutable. Las tres partes son aplicaciones basadas en Java. El archivo de propiedades contiene piezas específicas de información que necesita el motor de reglas y el de visualización [7].

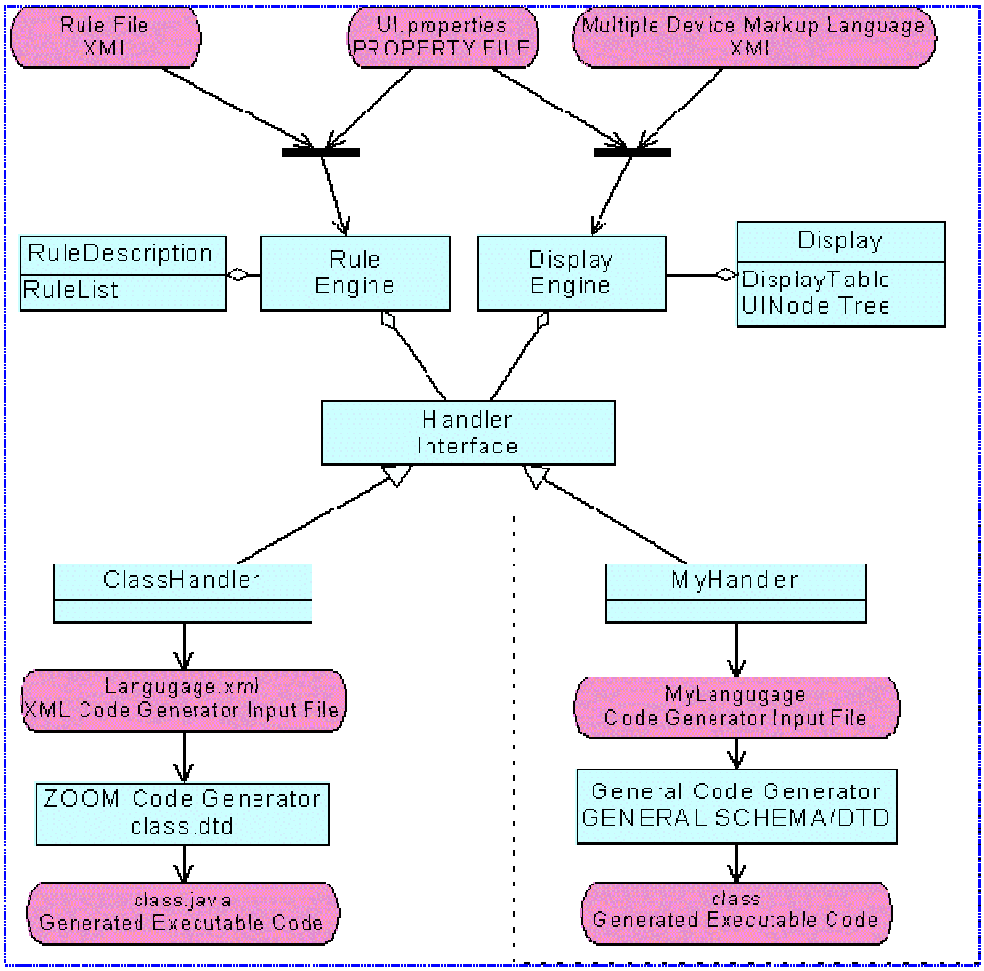

Figura 2. Arquitectura MDML

Hay cuatro perfiles: *desktop*, web, móvil y de voz. Para cada perfil existen diferentes idiomas soportados o limitaciones que se deben considerar como consecuencia de los componentes de la arquitectura. Todos los perfiles pueden responder de manera diferente para las distintas capas que componen la arquitectura:

- Motor de Regla. El motor de reglas es responsable de leer el archivo de reglas - rule.XML. El archivo de reglas tiene tres secciones <perfil>, *<tag>* y *<event>*. La etiqueta puede contener estos *tags* <perfil>, *<import>, <toolkit>, <accessor>* y *<topfunction>*. La etiqueta *<import>* se utiliza para cualquier importación o incluye que se debe especificar en el inicio de cualquier archivo de idioma.

- Visualización del motor. El motor de visualización es responsable de leer el archivo MDML inicial en la memoria y cambiar el diseño según lo deseado por las reglas. El motor de visualización utiliza *Document Object Model* (DOM) para capturar el archivo MDML inicialmente en la memoria.

- Manipulador / Generación de código. El controlador es la interfaz general que contiene el acceso al conjunto de reglas y el objeto de visualización MDML. Para generar el código ejecutable de este proyecto se utiliza el generador de código ZOOM.

2.4. Software Engineering for Embedded Systems using a Component-Oriented Approach (SeescoaXML)

Seescoa se describe en Luyten, K., Abrams, M., Vanderdonckt, J. *y* Limbourg, P. [8] consiste en un conjunto de modelos y un mecanismo para producir automáticamente diferentes interfaces gráficas de usuario en tiempo de ejecución para plataformas informáticas diferentes, equipadas con diferentes dispositivos de entrada / salida que ofrecen diversas modalidades (por ejemplo, un *joystick)*.

Este sistema es sensible al contexto se expresa, primero en una modalidad independiente, y luego se une a una especialización para cada plataforma específica. La sensibilidad al contexto de la interfaz de usuario se centra las variaciones de las distintas plataformas informáticas. La AUI contiene las especificaciones de los mecanismos de rendición de los diferentes aspectos (presentación) y su comportamiento relacionado (aspectos de diálogo). Estas especificaciones están escritas en un lenguaje de descripción XML compatible con la interfaz de usuario (UIDL) que luego se transforman en especificaciones específicas de la plataforma usando transformaciones XSLT. Estas especificaciones se unen a una descripción de alto nivel de los dispositivos de entrada / salida.

Para generar UIs sensibles al contexto la traducción se lleva a cabo en el nivel abstracto antes de ir hacia abajo en el marco para cada configuración específica.

En esta versión no se utilizan explícitamente conceptos o modelos de tarea. El punto de entrada de este enfoque de ingeniería directa está por lo tanto situado en el nivel de interfaces abstractas.

Dygimes es una versión ampliada de Seescoa presentada por Luyten et al [8] que adopta el mismo enfoque, excepto que la AUI se obtiene a partir de un modelo de tarea CTT (Paterno 2000) que progresivamente se transforma en un árbol de prioridad como punto de partida para la obtención de la AUI.

2.5. Renderer-Independent Markup Language (RIML)

RIML es un lenguaje basado en estándares de la W3C que permite la creación de documentos de forma independiente al dispositivo. En 2003 Demler, Wasmund, Grassel, Spriestersbach, y Ziegert presentaron el lenguaje, pero lamentablemente los principales resultados del proyecto, incluyendo más detalles sobre el lenguaje ya no están disponibles. Incluimos esta lengua como el resultado del Proyecto Consenso europeo y fue la fuente de inspiración de un vendedor D3ML idioma existente.

RIML se basaba en estándares tales como: XHMTL 2,0 y XForms. RIML utiliza las estructuras fila y columna para especificar la adaptación del contenido. Su semántica se ha mejorado para cubrir directivas de paginación y el diseño en el caso de paginación [9]. Debido al uso de XForms RIML es independiente del dispositivo y puede ser asignada a una especificación XHTML de acuerdo con el dispositivo de destino. Se introdujo por primera vez el mecanismo para manejar la paginación. En este sentido se pudo especificar cómo mostrar una secuencia de elementos de la interfaz de usuario. Tradicionalmente aparece en una fila, se pudo representada en las diferentes ventanas y crear botones de navegación para ir de una ventana a otra, hacia delante y hacia atrás. La superficie de la pantalla disponible se asumió que era demasiado pequeña para mostrar la lista completa de una vez. La paginación resultante se aplicó en varias páginas. Además, para cambiar entre las páginas están incluidos los hipervínculos.

Un objetivo principal de RIML era simplificar la creación con respecto al conocimiento del dispositivo [9]. La presentación del diseño estructurado es en gran medida independiente de las características del dispositivo.

2.6. Simple Unified Natural Markup Language (SunML)

SunML se describe en Picard, Fierstone, Pinna-Dery, y Riveill [10]. Es un lenguaje XML para especificar interfaces de usuario concretas que se pueden representar gráficamente para diferentes dispositivos (PC, PDA, voz). La innovación de este lenguaje es la capacidad de especificar los componentes de forma dinámica. SunML se compone de una serie de conceptos básicos para describir la interfaz de usuario que se detallan a continuación:

- Elemento: distribuye la información del usuario al sistema. Tiene un tipo de datos (*string, integer, float, Boolean*). Este nodo es fundamental, como hoja de la descomposición. El mapeo de un elemento se corresponde con la representación en modo grafico de las etiquetas y campos de texto. Usando el canal de voz podría ser una acción del sistema para sintetizar el texto en voz.

- Campo: se utiliza para recuperar información del usuario. Su funcionalidad está relacionada con el *widget* que representa.

- Enlace: Un enlace se refiere a la ejecución de algo en particular. Tradicionalmente referidos como controles, los enlaces pueden estar asociados a botones, elementos de menú, gramática de reconocimiento de voz… En cualquier caso, el uso inicia la interacción.

- Lista: un grupo de varios elementos con el objetivo de presentar una lista de valores o acciones. El elemento es más complejo que los anteriores, ya que puede estar compuesto de otros elementos.

- Dialogo: es una lista que agrupa los *widgets* con el fin de crear un diálogo entre ellos. El cuadro de diálogo se puede asignar a un cuadro de diálogo completo de *widgets* o a un diálogo interactivo vocal.

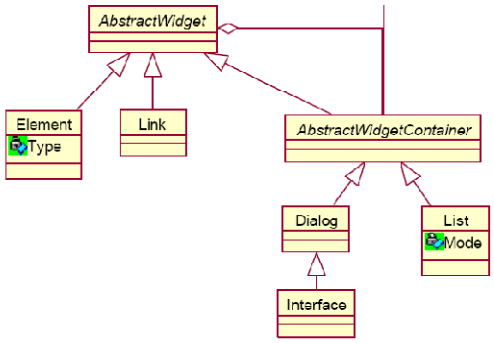

Figura 3. SunML Meta-Model

Este conjunto reducido de elementos parece no ser suficiente, pero la composición de *widgets* se utiliza para especificar los *widgets* más complejos. Por ejemplo, una lista de elementos se puede asignar a un cuadro combinado o una lista de enlaces a un menú.

En SunML también es posible encapsular el estilo y el contenido de cada *widget* independiente de los demás. Para este fin se utilizan dos tipos de archivos:

.

- El estilo se puede definir directamente mediante la propiedad *tag* o almacenarlo en una hoja de estilos en cascada (CSS).

- El contenido también se almacena en un archivo almacenado en el cliente, para evitar el exceso de datos al transformar desde el cliente y el servidor.

```
<sunml>
 <interface name="CustomerForm">
  <structure>
   <dialog name="Main" sequence="true"> ...
    <field name="Label1" mode="read">
     <element type="string">Nom  :</element>
    </field>
    <field name="Nom" mode="read-write">
     <element type="string">Toto</element>
    </field>
    <field name="Label2" mode="read">
     <element type="string">CA :</element>
    </field>
    <field name="CA" mode="read-write">
     <element type="double">1000</element>
    </field>
   </dialog>
   <dialog name="Main" sequence="true"> ...
    <link name="ActionSave">
     <element type="string">Save</element>
    </field>
    <link name="ActionQuit">
     <element type="string">Quit</element>
    </field>
   </dialog>
  </structure>
 </interface>
</sunml>
```

Figura 4. Ejemplo SunML

Otra característica que ofrece el SunML es la composición de *widgets* (figura 5). Tienen algunos operadores en la interfaz de usuario para fusionar dos interfaces: la unión, intersección, sustracción, sustitución, inclusión. Widgets Merging Language (WML) es la extensión utilizada para ese fin.

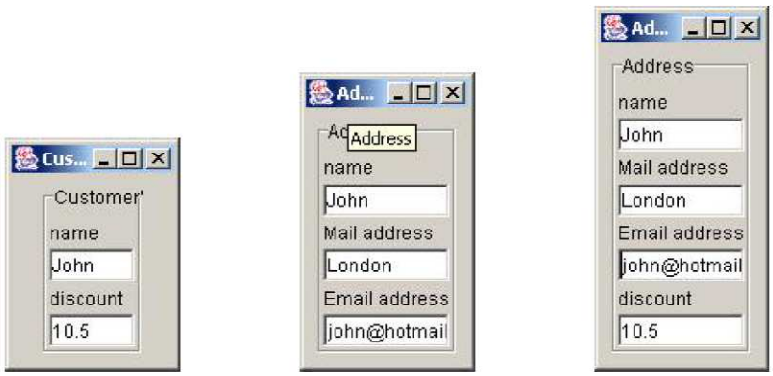

Figura 5. Unión entre dos interfaces

2.7. Teresa (TeresaXML)

Paterno y Santoro [11] en 2003 introdujeron un método para producir FUIs para múltiples plataformas de computación en tiempo de diseño. A partir del modelo de tarea del sistema propone identificar las especificaciones AUI en términos de su estructura estática (el modelo de presentación) y el comportamiento dinámico (el modelo de diálogo); estas especificaciones

abstractas son explotadas para impulsar la implementación. La traducción de un contexto de uso a otro se realiza al más alto nivel: tarea y conceptos. Esto permite una flexibilidad máxima, para apoyar múltiples variaciones de la tarea en función de las limitaciones impuestas por el contexto de uso. El contexto de uso está limitado a plataformas de programación.

El proceso se define en tiempo de diseño y no en tiempo de ejecución.

A nivel de la AUI, la herramienta proporciona a los diseñadores un poco de ayuda en el perfeccionamiento de las especificaciones para las plataformas informáticas diferentes. La AUI se describe en términos de objetos de interacción abstractos (AIO) [12] que son a su vez transformados en objetos de interacción concretos (CIO)[12].

2.8. User Interface Markup Language (lenguaje de marcado UIML)

UIML se describe en Helms, Schaefer, Luyten, Vanderdonckt, Vermeulen y Abrams [13]. Es un lenguaje basado en XML que proporciona: (1) un método independiente de dispositivo para describir una interfaz de usuario, (2) una modalidad independiente del método para especificar una interfaz de usuario.

UIML permite describir el aspecto, la interacción y la unión de la interfaz de usuario con la lógica de la aplicación. Los siguientes cuatro conceptos son clave en UIML:

1. UIML es un metalenguaje: UIML define un pequeño conjunto de etiquetas (por ejemplo, se usa para describir una parte de una UI) que son de independientes de la plataforma de destino (por ejemplo, PC teléfono,) y el objetivo del lenguaje independiente (por ejemplo, Java, VoiceXML). La especificación de una interfaz de usuario se realiza a través de un conjunto de herramientas de vocabulario que especifican un conjunto de clases de componentes y propiedades de las clases. Diferentes grupos de personas pueden definir vocabularios diferentes: un grupo puede definir un vocabulario cuyas clases tienen una correspondencia 1-a-1 a los *widgets* de interfaz de usuario en un idioma determinado (por ejemplo, Java Swing API), mientras que otro grupo podría definir un vocabulario cuyas clases coinciden con las abstracciones utilizadas por un diseñador de interfaces.

2. UIML separa los elementos de una interfaz de usuario e identifica: (a) qué partes se compone la interfaz de usuario y el estilo de presentación, (b) el contenido de cada parte (por ejemplo, texto, sonidos, imágenes) y la unión de los contenidos y los recursos externos, (c) la conducta expresada como un conjunto de reglas con condiciones y acciones, y (d) la definición del vocabulario de los componentes de las clases.

3. UIML define la interfaz de usuario en un árbol de elementos de UI que cambia con el tiempo de vida de la interfaz. Durante la vida de una interfaz de usuario los componentes del árbol inicial pueden cambiar dinámicamente la forma por adición o

eliminación de partes. UIML proporciona elementos para describir la estructura de árbol inicial y para modificar dinámicamente la estructura.

4. UIML permite empaquetar en plantillas los componentes de interfaz de usuario y la los componentes de los árboles: estas plantillas pueden ser reutilizadas en diseños de interfaz.

2.9. USer Interface eXtensible Markup Language (UsiXML)

Las principales características de UsiXML son:

- UsiXML está estructurado de acuerdo a los diferentes niveles de abstracción definidos por el marco de referencia Cameleon (Calvary, Coutaz, Thevenin, Limbourg, Bouillon, y Vanderdonckt, 2003). El marco representa una referencia para la clasificación de interfaces de usuario que soportan una plataforma de destino, AHMI, y un contexto de uso. Además permite estructurar el ciclo de vida de desarrollo en cuatro niveles de abstracción: las tareas y conceptos de interfaz de usuario abstracta (AUI), interfaz de usuario concreta (CUI) y IU final (FUI). El desarrollo AHMI sólo considera CUI y FUI para el ciclo de desarrollo. Así, el nivel de tareas y conceptos es computacionalmente independiente, el nivel AUI es independiente de la modalidad (GUI vocal, táctil) y el nivel CUI es independiente de las herramientas.

- UsiXML se basa en un enfoque de transformación que progresivamente se traslada desde el nivel de la UIF, al funcionamiento AHMI.

- La metodología de transformación de UsiXML permite la modificación de la sub-etapas de desarrollo, garantizando así varias alternativas para la existencia de sub-pasos para ser explorados y / o ampliados con nuevos sub-pasos. Lo que ayuda en la exploración de soluciones diferentes para la representación final de la AHMI y su evaluación en el modelo cognitivo.

- UsiXML tiene un único formalismo abstracto subyacente representado mediante un gráfico basado en sintaxis.

- UsiXML permite reutilizar partes especificadas en AHMI con el fin de desarrollar nuevas aplicaciones. Este servicio es proporcionado por la sintaxis XML subyacente de UsiXML que permite el intercambio de cualquier especificación. Además, la capacidad de transformar estas especificaciones se debe a un conjunto de reglas de transformación que aumenta su reutilización.

- El desarrollo progresivo de los niveles de UsiXML se basa en un enfoque de transformación mediante un gráfico basado en sintaxis. Está demostrado que la sintaxis es eficaz para la especificación de reglas de transformación y un formalismo apropiado para la comprensión humana.

- UsiXML garantiza la independencia de la modalidad (capacidad para modelar una interfaz de usuario independiente de cualquier modalidad) gracias al nivel AUI, que permite la especificación de interfaces de usuario independientes de cualquier modalidad de interacción, tales como física, interacción gráfica, vocal o 3D. Esto juega un papel muy importante en el contexto de humano porque permite el diseño de interfaces de usuario que no dependen de la modalidad de interacción. Por ejemplo, actualmente el piloto automático se activa al pulsar botones físicos, y si por cualquier motivo un botón en una pantalla táctil se utiliza para activar el piloto automático en lugar de los botones físicos entonces habría que concretar los cambios en la AUI. Sin embargo, la AUI seguiría siendo la misma.

- UsiXML apoya la incorporación de nuevas modalidades de interacción gracias a la modularidad del marco donde se define cada modelo de forma independiente y al carácter estructurado de los modelos garantizados por el formalismo gráfico subyacente, por lo tanto, tiene la propiedad de ser extensible a nuevas modalidades. La interacción en el AHMI es a través de la manipulación directa de una ruta de contacto, si en las futuras modalidades de interacción se añaden, entonces se pueden añadir también a UsiXML sin ningún problema.

- UsiXML se apoya en un conjunto de herramientas que permiten el procesamiento de su formato, tiene la propiedad de ser procesable por la máquina de los modelos involucrados.

- UsiXML permite el cruce de herramientas de desarrollo de aplicaciones interactivas gracias a su formato común para la descripción de la UI.

2.10. Web Service eXperience Language (WSXL)

Web Service eXperience Language (WSXL) de IBM [14] fue diseñado para representar los datos, la presentación y el control. También, como un lenguaje para las aplicaciones web que consideren la adaptación contexto que se puede ejecutar en modalidades diferentes:

- Directamente utilizando una aplicación de fondo existente. Esta aplicación se puede adaptar y se usa para el acceso directo del cliente o incrustada dentro de una aplicación remota.

- Portal Local. Dos niveles de distribuidores de aplicaciones desde el punto final del software. El generador proporciona los mismos datos que en el primer caso y un portal local utiliza esta entrada, y podría añadir datos adicionales.

WSXL se basa en los estándares (figura 6) basados en XML como XPath, XML Events, DOM, XForms y XLink así como servicios web estándares como SOAP, WSDL y WSFL. Define conceptos tales como:

- *WSXLServiceDescription*: Proporciona operaciones básicas de consulta que permiten a un cliente solicitar el servicio WSDL de documento de descripción) y preguntar si un *"portType"* concreto es compatible.

- *WSXLLifecycle*: Ciclo de Vida (indirectamente se refiere a los casos particulares en las llamadas posteriores) Gestor de elementos y colecciones.

- *WSXLPropertie.* Un componente WSXL debe implementar las operaciones de gestión de las propiedades por el cual los clientes pueden modificar las propiedades en otros momentos de la inicialización.

- *WSXLOutput:* Operaciones relacionadas con el marcado por el servicio.

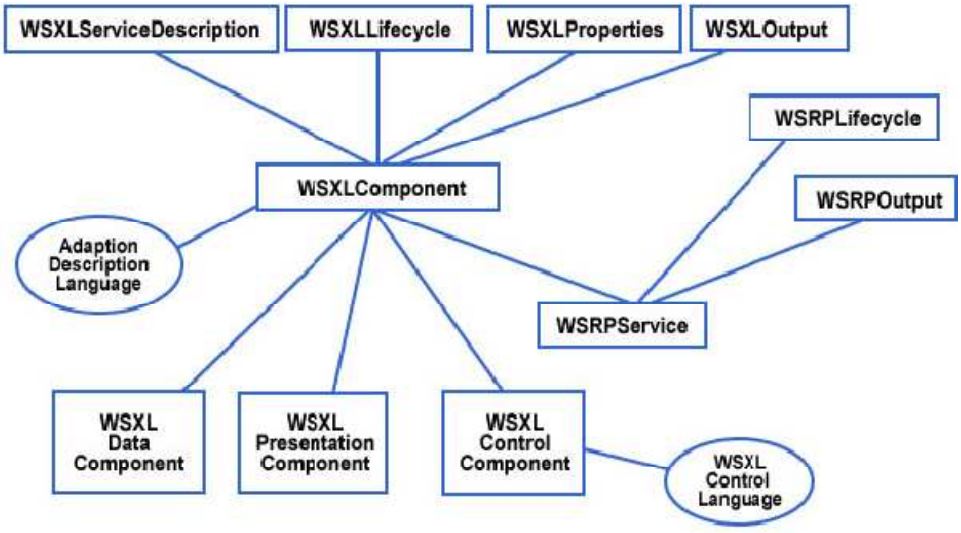

Figura 6. Meta Modelo WSXL

- Componente de Datos: El componente de datos WSXL se basa en la funcionalidad de los datos en XForms, e incluye tanto el modelo XForms y funciones de instancia. El

componente de datos WSXL puede estar unido a fuentes de datos externas a las aplicaciones WSXL.

- El componente de presentación: Los componentes de presentación WSXL puede generar un marcado de salida en cualquier lenguaje XML de destino y debe indicar en qué idiomas puede ser solicitada.

- Componente de control: El componente de control WSXL implementa *portTypes* utilizados para administrar los componentes de enlace de datos a los componentes de la presentación, para analizar e interpretar la especificación de control así para implementar un modelo de proceso que controla la propagación de las notificaciones de eventos en ambos sentidos entre los datos y la presentación.

WSXL incluye una extensión del *Adaptation Description Language* para especificar los puntos de adaptación, las operaciones permitidas sobre los puntos de adaptación (por ejemplo, insertar, eliminar, modificar), y las limitaciones sobre el contenido de la adaptación (por ejemplo, mediante un esquema XML) puede. El lenguaje puede ser utilizado durante la etapa de post-procesamiento donde se encuentra la salida de un componente WSXL adaptado de forma independiente sin necesidad de invocar el componente. Por último, una colección WSXL proporciona un entorno de gestión y ejecución de los componentes WSXL. Se llama a las operaciones del ciclo de vida de los componentes WSXL crea una instancia, y pone en práctica un conjunto de interfaces y un modelo de procesamiento para el uso de componentes WSXL y objetos externos a la colección. El objeto que implementa la interfaz colección WSXL no tiene que ser un componente WSXL.

2.11. eXtensible user-Interface Markup Language (XICL)

eXtensible user-Interface Markup Language (XICL) se presenta en Sousa y Leite [15]. Es una manera fácil de desarrollar componentes de interfaz de usuario basada en el navegador de software. Los nuevos componentes de interfaz de usuario se crean a partir de componentes HTML y otros componentes XICL. La descripción XICL se traduce en código DHTML.

Un documento XICL está compuesto por una descripción de la interfaz de usuario compuesta por elementos HTML o XICL y varios componentes (estructura, propiedades, eventos y métodos. Las principales contribuciones que los autores atribuyen a la lengua son:

- Reutilización: fácil de reutilizar los componentes de interfaz de usuario.
- Extensibilidad: la creación de nuevos componentes se extienden a los ya existentes.
- Abstracción: los componentes más potentes.

- Portabilidad: ejecución en la mayoría de los navegadores.
- Normalización: un lenguaje común para reutilizar y extender a los componentes.
- Productividad: se debe mejorar la productividad de desarrollo.

2.12. eXstensible user-Interface Markup Language (XIML)

El modelo de presentación está escrito en *eXstensible user-Interface Markup Language* (XIML) presentado por en Eisenstein, Vanderdonckt y Puerta [16], un lenguaje desarrollado por *Software Redwhale*, derivado de XML y capaz de almacenar los modelos desarrollados en MIMIC [17]. MIMIC es un meta-lenguaje que estructura y organiza los modelos de interfaz. Se divide la interfaz a los componentes del modelo: usuario-tarea, presentación, dominio de diálogo, el usuario y modelos de diseño. El modelo de diseño contiene todas las asignaciones entre los elementos pertenecientes a los otros modelos. El XIML es, pues, la versión actualizada de este lenguaje XML anterior.

El lenguaje XIML se compone principalmente de cuatro tipos de componentes: modelos, elementos, atributos y las relaciones entre los elementos.

- Modelo: podemos distinguir dos tipos de modelos, el modelo de interfaz y los componentes del modelo. La primera es la raíz de cualquier documento XIML y contiene los diversos sub-modelos (componentes del modelo) disponible en XIML. Todos los tipos de modelos no tienen que estar presentes y el mismo tipo de modelo-componente pueden existir varias veces bajo el modelo de la interfaz misma. Los componentes del modelo (tareas, dominio, usuario, la presentación, el diálogo, la plataforma, las preferencias y el modelo general) contienen información específica a una dimensión de la interfaz.

- Elemento: es la información que describe un modelo de componente. En el caso de elementos de presentación, que es una unidad de información que describe el aspecto visual de una interfaz de usuario. Cada elemento de presentación puede contener otros elementos de presentación (hasta que la CIO es sencilla). Por ejemplo, una ventana es un elemento de presentación que puede contener la tabla de elemento de presentación que contiene en sí otros elementos de presentación, etc. Un elemento de presentación puede referirse a un objeto externo al código XIML, por ejemplo un control ActiveX, una imagen, etc. En este caso, la ubicación atributo se puede utilizar para especificar una URL donde podría ser el objeto encontrado.

- Características: representa una unidad simple de información declarativa en relación con un modelo de interfaz, un modelo de componente o un elemento. Siempre es necesario definir un atributo para luego ser capaz de declarar en un

componente. La definición del atributo está compuesto por la lista de sus valores permitidos, el valor por defecto del atributo, su forma canónica, documentación (información sobre lo que representa) y de su tipo.

- Relación: define una relación entre los elementos y / o modelos. El elemento al que se refiere es el enlace gracias especificados en el atributo de referencia, que contiene el identificador al que se refiere el objeto. Toda relación debe ser definida antes de su uso (de forma similar a los atributos). Esta definición contiene las clases permitidas en la relación.

El modelo de presentación se compone de varios elementos incrustados que corresponden a los *widgets* de la interfaz de usuario, y los atributos de estos elementos que representan sus características (color, tamaño...). Las relaciones a nivel de presentación son principalmente los vínculos entre las etiquetas y los *widgets* que estas etiquetas describen.

2.13. Conclusiones del estado del arte

Los trabajos estudiados representan la interfaz a través de lenguajes basados en XML o a través de primitivas conceptuales. Estas técnicas son muy potentes para el desarrollo de aplicaciones, ya que permiten al analista especificar todas las características de interacción solicitadas por el usuario. Sin embrago, son técnicas difíciles de entender por parte del usuario. Esta dificultad hace que el usuario no pueda participar en el desarrollo de interfaces más allá de la captura de requisitos. Además, hasta que las interfaces no han sido totalmente desarrolladas, el usuario tampoco puede evaluarlas.

Otro de los inconvenientes que presentan los trabajos estudiados es que están centrados únicamente en representar características de interacción, dejando de lado la funcionalidad del sistema. Por tanto, las técnicas como tales no pueden desarrollar aplicaciones totalmente funcionales, sino que deben conjuntarse con otros métodos o técnicas capaces de producir el código que implemente su funcionalidad.

En este trabajo, se pretende suplir ambas carencias. Por un lado, se pretende utilizar Rainbow, que es un sistema que genera la persistencia de los sistemas a partir de bocetos de formularios. Este método es lo suficientemente intuitivo para que los usuarios puedan participar en el desarrollo del sistema. Rainbow se integrará con OO-Method, que es un método que no sólo representa interacción, sino también funcionalidad y persistencia. La integración de ambos métodos permitirá tanto la participación del usuario como la generación de aplicaciones totalmente funcionales a partir de modelos.

3. OO-METHOD Y RAINBOW

3.1. OO-METHOD

OO-Method [1] es un metodología de producción automática de Software que utiliza técnicas gráficas convencionales de modelado y está basado en el lenguaje de especificación formal y orientado a objetos OASIS [18].
Esta propuesta, proporciona un marco formal bien definido que permite construir modelos conceptuales de gran expresividad y desarrollar aplicaciones robustas, aprovechando las buenas propiedades de ambas aproximaciones (formales y convencionales).
En OO-Method, la información recogida del sistema de información, durante la fase de modelado conceptual, se realiza por medio de una serie de modelos gráficos que están orientados a completar una especificación formal en OASIS, pero ocultando los aspectos sintácticos que requiere una especificación de esta naturaleza.

OO-Method está basada en una clara separación del Espacio del Problema (el 'qué') del Espacio de la Solución (el 'cómo'). La definición de un problema (la descripción abstracta de una aplicación, representada en el correspondiente Modelo Conceptual), puede suceder con independencia de cualquier solución concreta.

Esto posiciona a OO-Method como una metodología para implementar herramientas que sigan las directrices MDA [19] de separación de la lógica de las aplicaciones de las posibles implementaciones de las mismas.
El formalismo que subyace a OO-Method es OASIS, un lenguaje formal y orientado a objetos para la especificación de sistemas de información. Este marco formal proporciona una caracterización de los elementos conceptuales requeridos para especificar un sistema de información.

Sus dos componentes principales son el ***Modelo Conceptual*** y el ***Modelo de Ejecución***.

3.1.1. Modelo Conceptual

El *Esquema Conceptual* recoge de forma gráfica las propiedades relevantes que definen el sistema a desarrollar, utilizando una notación basada en UML [20], 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:
        - **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.

        - **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.

- **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.

Estos cuatro modelos permiten que todos los aspectos funcionales de una aplicación puedan ser descritos de forma abstracta por medio de una serie de elementos conceptuales (a los que también nos referimos como primitivas conceptuales o patrones conceptuales) que tienen una semántica precisa. Cualquier Modelo Conceptual de OO-Method es una instancia del Metamodelo OO-Method.

3.1.2. Modelo de Ejecución

El *Modelo de Ejecución* establece las reglas de transformación de un Modelo Conceptual (sistema especificado) a su representación software correspondiente en una plataforma tecnológica concreta, de manera que se preserve la semántica de los elementos del Modelo Conceptual y se asegure que la aplicación resultante es funcionalmente equivalente a dicho modelo.

El modelo de ejecución tiene tres fases esenciales:

1. **Control de acceso**: en primer lugar, el objeto que desee conectarse al sistema deberá identificarse como miembro de la sociedad de objetos.

2. **Vista del sistema**: una vez se ha conectado un usuario (u otro objeto del sistema), éste tendrá una visión clara de la sociedad de objetos (qué clases de objetos puede ver, servicios que puede activar y atributos que puede consultar).

3. **Activación de servicios**: por último después de las fases anteriores, el objeto deberá ser capaz de activar cualquier servicio disponible en su visión del mundo, y de realizar las observaciones pertinentes.

Cualquier ***petición de servicio*** se caracterizará por la siguiente secuencia de acciones:

1. ***Identificación del objeto***: en primer lugar, se identificará el objeto que ha de servir la petición. La existencia de este objeto es una condición implícita para poder ejecutar el servicio, excepto si se trata del evento de *creación*. Si el objeto existe se recuperarán los valores de los atributos que caracterizan su estado.

2. ***Introducción de los argumentos del evento***: se introducen el resto de argumentos del evento a activar.

3. ***Corrección de la transición entre estados:*** verificaremos en el estado actual y para el servicio en cuestión que existe en el proceso una transición entre estados válida.

4. ***Satisfacción de precondiciones***: Comprobaremos que se cumple la precondición asociada al servicio que va a ejecutarse. Si no se cumple se elevará una excepción informando que el servicio no puede activarse porque se ha violado su precondición.

5. ***Realización de las evaluaciones***: una vez verificada la precondición, se hacen efectivas en el sistema de almacenamiento las modificaciones en el estado del objeto inducidas por el evento.

6. ***Comprobación de las restricciones de integridad en el nuevo estado***: para asegurar que la activación del servicio deja al objeto en un estado válido, se debe comprobar que las restricciones de integridad (estáticas y dinámicas) se cumplen en el estado final resultante.

7. ***Comprobación de las relaciones de disparo***: después de un cambio de estado válido, y como acción final, se deben verificar el conjunto de reglas *condición-acción* que representan la actividad interna del sistema. Si alguna de ellas se cumple, se disparará la correspondiente activación de servicio. Será responsabilidad del analista garantizar la terminación y confluencia de los disparos.

Los pasos anteriores guiarán la implementación de cualquier aplicación para asegurar la equivalencia funcional entre la descripción del sistema recogida en el modelo conceptual y su implementación en un entorno de programación de acuerdo con el modelo de ejecución.

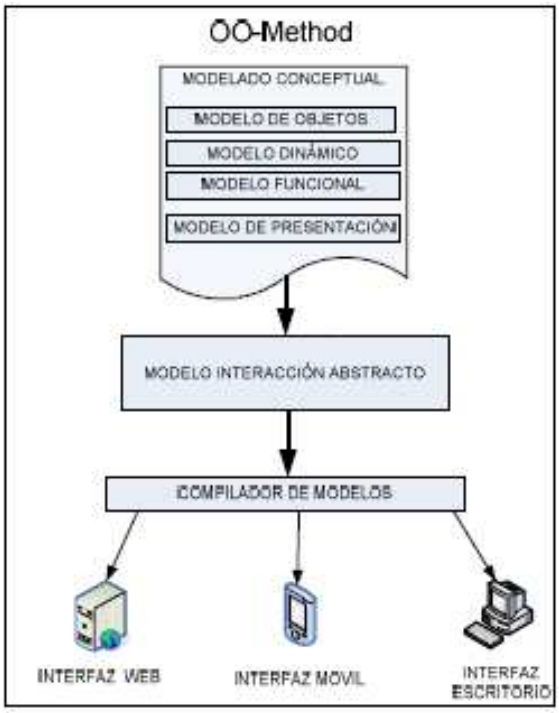

Figura 7. Esquema OO-Method

3.1.3. OASIS

OASIS (*Open Active Specification Of Information Systems*) [18] es un lenguaje de especificación formal orientado a objetos que sirve de base formal a la metodología OO-Method. Entre las características del lenguaje se puede destacar:

- Una especificación OASIS se formaliza como teorías de primer orden que evolucionan con el tiempo por la ocurrencia de eventos y que tienen una base formal natural, correcta y completa, dentro del contexto de la *Lógica Dinámica*.

- Se define también un álgebra de procesos para objetos (APO) con el objetivo de caracterizar con propiedad y entender los objetos como procesos observables.

La siguiente tabla (figura 8) muestra la correspondencia entre OO-Method y OASIS.

	Semi-formal (OO-Method)	Formal (OASIS)
Comportamiento	Diagramas de Transición entre Estados para representar las vidas posibles de los objetos y los Diagramas de Interacción entre Objetos para representar la comunicación y las transacciones	Conjunto de ecuaciones con variables del género proceso resueltas en un álgebra de procesos
	Dialogo interactivo para especificar el efecto de los eventos sobre los objetos	Axiomas de la lógica dinámica
Estructura de Datos	Diagramas de Configuración de Clases	Géneros ordenados parcialmente, funciones, atributos y eventos

Figura 8. Correspondencia de elementos

3.1.4. Primitivas del Modelo de Objetos

A continuación se presentan las primitivas de OO-Method que se utilizarán en la sección 4 para realizar la integración entre OO-Method y Rainbow.

Clase

Una clase es un elemento que denota una estructura y un comportamiento que es compartido por un conjunto de objetos.
Las propiedades de la clase se organizan en tres grupos que se detallarán a continuación: atributos, servicios y restricciones de integridad.

Atributo

Son las propiedades estructurales de la clase.

Servicio

Cuando se ejecuta un servicio, se modifica el estado de un objeto. Los servicios están asociados con la especificación del comportamiento de la clase.
Puede ser definido como una unidad de proceso que puede ser atómica (un evento) o molecular (una transacción local o una operación local).

Relaciones entre clases

Las relaciones entre clases determinan qué objetos puede ser vistos o en caso de que actúen como agentes, establecen la forma en que dos clases están asociadas. También determinan

las propiedades de una clase que pueden ser reutilizadas en la definición de una nueva clase. Para hacer frente a estos tres aspectos, OO-Method define tres tipos diferentes de relaciones entre las clases:

- Agente de relación: Representa que objetos están autorizados para activar los servicios.

- Asociación, agregación, composición: Relación semántica bidireccional entre clases (asociación), estricto caso de agregación (todo/parte semántica), y la composición (tipo fuerte de agregación).

- Especialización: Representa la herencia de las relaciones.

Evento

Los eventos son unidades atómicas de proceso que representan la abstracción de un cambio de estado.
Existen diferentes tipos de eventos:

- New: Eventos de creación.
- Destroy: Eventos de destrucción.
- Own: Eventos específicos de la clase.
- Shared: Eventos compartidos entre clases.

Restricciones de Integridad

La especificación de atributos y servicios junto con el conjunto de acciones que pueden modifica la vida útil de los objetos de la clase, permite la caracterización de las propiedades más relevantes del modelo.

Las restricciones de un objeto deben ser satisfechas en todos los estados de un objeto.
Una restricción de integridad es la expresión de una condición semántica que debe preservar el estado válido de un objeto.

Existen dos tipos de restricciones de integridad:

- Estáticas
- Dinámicas

3.1.5. Elementos del Modelo de Presentación

A continuación se presentan los elementos del modelo de presentación de OO-Method que se utilizarán en la sección 4 para realizar la integración entre OO-Method y Rainbow.

Entrada

Este patrón recoge los aspectos relevantes de los datos que deben ser introducidos por el usuario, por ejemplo, al proporcionar los valores para los argumentos de los servicios.

Los aspectos de interacción que se pueden especificar incluyen máscaras de edición, rangos de valores válidos, ayuda y mensajes de validación.

Unidad de Interacción de servicio

El patrón unidad de interacción de servicio es crucial dado que representa la interacción entre el usuario y el sistema software.
Una unidad de interacción de servicio representa un escenario particular de una interfaz de usuario a través del cual los usuarios pueden llevar a cabo tareas específicas (como la ejecución de los servicios o la búsqueda de objetos).

La interfaz de usuario del sistema está definida por una colección de unidades de interacción. Existen 3 tipos básicos de escenarios de interacción: ejecución de un servicio, manipulación de un objeto y manipulación de una colección de objetos. Para cada uno de estos escenarios, OO-Method propone una UI.

- **UI de Servicio:** Habilita un escenario que se define en el que el usuario interactúa con el sistema a fin de ejecutar un servicio. El usuario debe proporcionar los argumentos e iniciar el servicio.

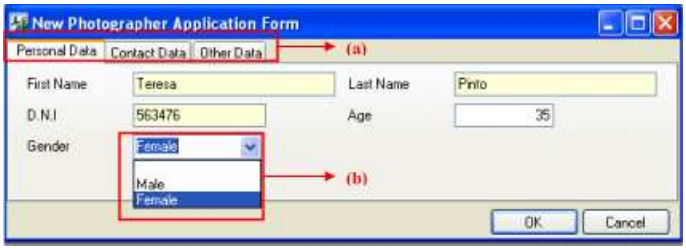

Figura 9. Ejemplo UI de Servicio

- **UI de Instancia:** Representa un escenario en el que se muestra la información de un solo objeto, incluyendo la lista de servicios que se puede ejecutar sobre el mismo, así como los escenarios por los que el usuario puede navegar.

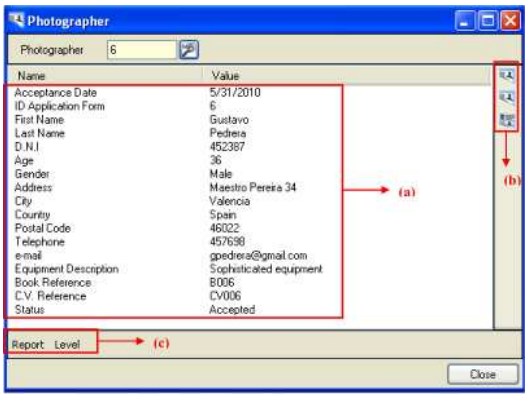

Figura 10. Ejemplo UI de Instancia

- **UI de Población:** Representa un escenario de interacción donde se presentan varios objetos. Este escenario incluye los mecanismos adecuados para realizar las siguientes acciones: seleccionar objetos y clasificar, seleccionar la información y los servicios disponibles para ser mostrados, y mostrar una lista de otros escenarios que se pueden alcanzar.

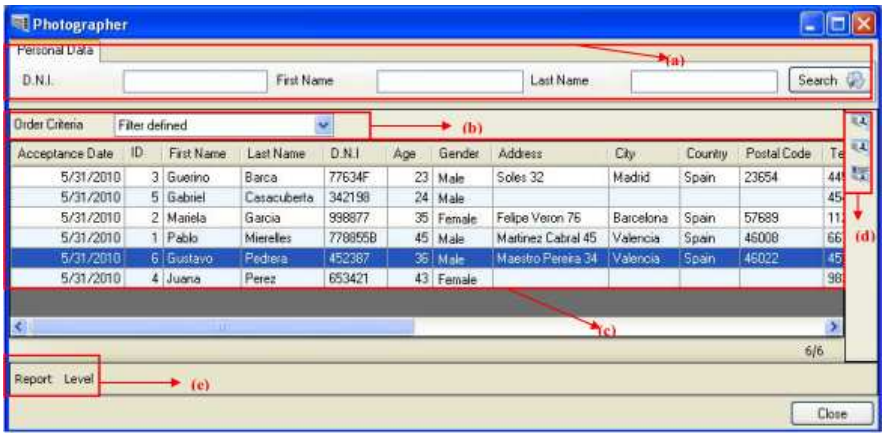

Figura 11. Ejemplo UI de Población

Criterio de Agrupación

El patrón de criterio de agrupación define la forma en la que se muestran los argumentos de entrada al usuario para un determinado servicio.

Selección definida

La selección definida permite definir un conjunto válido de valores asociados a un elemento. El conjunto definido se comporta como un tipo enumerado.
Los enumerados corresponden a un tipo de atributos que únicamente pueden adquirir un número cerrado de valores.

Conjunto de visualización

El patrón conjunto de visualización determina que propiedades de una clase se presentan al usuario y en qué orden. La visualización del conjunto de propiedades es proporcionada en función del tipo de interacción del usuario.

En muchas ocasiones, en escenarios convencionales no suelen mostrarse todas las propiedades de los objetos. Por este motivo, a una determinada colección de objetos se le aplican criterios de búsqueda y de ordenación y a continuación se decide que propiedades de estos objetos se mostrarán al usuario y en qué orden.
Consecuentemente, es importante especificar qué conjunto de propiedades se consideran relevantes en el contexto de interacción.

3.2. RAINBOW

En el ámbito de la Ingeniería de Requisitos el modelado de datos juega un papel decisivo, ya que define el núcleo semántico de la aplicación en el futuro.
El modelado de datos precisa de la obtención y validación de los requisitos por parte de los usuarios. Este es un aspecto fundamental para construir un sistema de información fiable así como para generar una documentación consistente del dominio de la aplicación.

La Ingeniería de Bases de Datos se centra en el modelado de datos. Los requisitos se expresan habitualmente por medio de un esquema conceptual aportando una visión abstracta del dominio de aplicación.

Existen diversas técnicas para la obtención de los requerimientos como el análisis de documentos de la empresa y/o entrevistas con las partes interesadas. Pero por lo general estas técnicas de obtención de requerimientos de datos no involucran activamente y de forma interactiva a los usuarios finales.
La participación de los usuarios finales, permitiéndoles plasmar sus necesidades, en la definición del sistema puede permitir evitar la resistencia al cambio de infraestructura así como estimular la productividad. Ellos conocen las cualidades y defectos de los Sistemas de Información utilizados en la actualidad, y por lo tanto tienen la capacidad de mejorar el nuevo sistema.

Dado que para la validación de los datos, la representación gráfica es a menudo difícil de comprender por los usuarios finales se presenta un enfoque apoyado en una herramienta para la elicitación y validación de los requisitos de base de datos basado en la participación interactiva de los usuarios finales a través de prototipos.

Los prototipos poseen la ventaja de la expresividad y comprensibilidad. Por este motivo se propone el uso de interfaces de usuario basadas en formularios como canal para la expresión eficiente, para capturar y validar los requisitos de datos con los usuarios finales.

Este enfoque aprovecha el poder de las técnicas de transformación de ingeniería inversa. El objetivo de estas técnicas es la extracción, a partir de las especificaciones existentes de artefactos como el código DLL de la base de datos, el código fuente de los programas que componen la aplicación, instancias de datos y así sucesivamente.

Rainbow [2] mantiene la misma filosofía pero se centra en la especificación de los requisitos como parte principal del proceso de la Ingeniería de Requerimientos.

Como ya se ha comentado anteriormente, Rainbow utiliza las interfaces basadas en formularios para la definición y validación de los requerimientos de la aplicación. Las interfaces son creadas por los usuarios finales.

A estos formularios los denomina *Modelo RAINBOW Simplificado de Formularios* (RSFM).

Para dibujar estas interfaces proporciona a los usuarios un conjunto limitado de *widgets* clasificados en:

- Contenedores: formularios*, fieldsets* y tablas.
- *Widgets* simples: datos de entrada, selecciones (*radiobuttons*, *checkboxes*, listas) y botones.

Uno de los objetivos de Rainbo es el de implicar a los usuarios finales de una manera sencilla e interactiva al tiempo que proporciona a los analistas un conjunto de herramientas semiautomáticas, con las cuales, a partir de las interfaces creadas por los usuarios y tras aplicarles a estos formularios las técnicas de Ingeniería Inversa, proporcionarles un diagrama entidad-relación extendido para la validación de los requerimientos.

El objetivo de Rainbow es proporcionar un conjunto de documentos de especificación y herramientas cuya finalidad es apoyar el desarrollo de futuras aplicaciones.

La siguiente imagen muestra en conjunto los distintos pasos de los que se compone Rainbow.

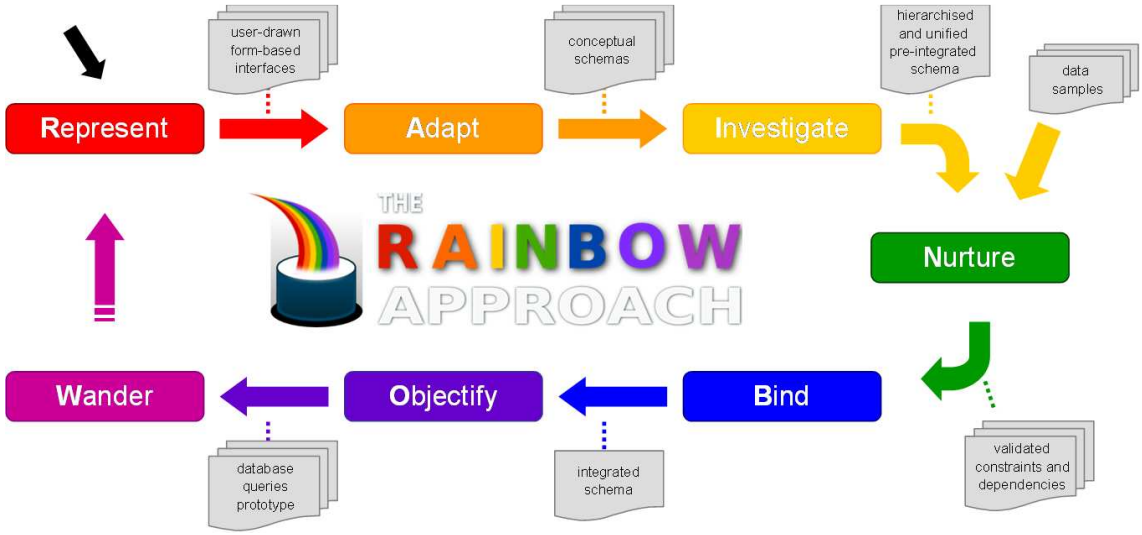

Figura 12. Fases de RAINBOW

El proceso se compone de siete pasos que se detallan a continuación.

- ***Represent:*** Los usuarios finales son invitados a crean las interfaces basadas en formularios para describir cada concepto clave del dominio de la aplicación.

- ***Adapt:*** Una vez las interfaces han sido generadas, se aplican técnicas de Ingeniería Inversa de Bases de Datos para obtener el esquema conceptual subyacente del dominio de la aplicación.

- ***Investigate:*** Posteriormente se analiza cada esquema individualmente para detectar ambigüedades y sacar a la luz información redundante contenida en las interfaces.

- ***Nurture:*** Una vez que las interfaces son validadas tanto por los analistas como por los usuarios finales se analizan las muestras de datos facilitadas por los usuarios para resaltar dependencias funcionales.

- ***Bind:*** Las redundancias y restricciones validadas se procesan para integrar los esquemas individuales en un esquema conceptual que representa los requisitos de datos.

- ***Objectify:*** Se genera un prototipo de la aplicación con su base de datos subyacente. Este prototipo es un gestor de bases de datos simples que utiliza las interfaces elaboradas por los usuarios y permite la navegación entre los conceptos expresados.

- ***Wander:*** Finalmente, los usuarios finales son invitados a utilizar el prototipo para validar en última instancia el esquema conceptual integrado.

3.2.1. *Especificidades metodológicas*

3.2.1.1 Ingeniería Inversa

La ingeniería inversa es un aspecto fundamental de la Ingeniería de Base de Datos. Consiste, entre otras cosas, en la recuperación o la reconstrucción de las especificaciones funcionales a partir de un elemento de software, sobretodo del código fuente de los programas, por lo general, cuando una base de datos existente tiene que ser reestructurada o migrado hacia una tecnología diferente. La Ingeniería Inversa por lo tanto, tiene por objeto recuperar un esquema conceptual lo más fiel al original, trabajando con múltiples artefactos del sistema, tales como: documentación (si está disponible), el código DDL de la base de datos, las instancias de datos, pantallas, informes y formularios, código fuente de los programas de la aplicación.

Sin embargo, en Rainbow el objetivo no es recuperar un esquema conceptual como en las situaciones tradicionales de Ingeniería Inversa, si no obtenerlo.
En Rainbow, las interfaces se utilizan como un lenguaje de especificación. Un formulario contiene estructuras de datos que se pueden ver como una particular vista del esquema conceptual. Por lo tanto, dado que un formulario es una implementación concreta de una parte de la base de datos, si se aplican las técnicas de ingeniería inversa, se puede obtener parte del esquema conceptual.

La Ingeniería Inversa explota el vínculo existente entre interfaces gráficas y modelos de datos.

En la figura 13 se muestra la correspondencia de los elementos de interfaz que proporciona Rainbow con las reglas de asignación que utiliza para la conversión a los elementos que compondrán el diagrama E-R extendido.

Widget	Visual Representation	ER Counterpart
Interface		Entity Type
Group box		Single-valued Compound Attribute
Table		Multivalued Compound Attribute

Widget	Visual Representation	ER Counterpart
Input Field		Single-valued Simple Attribute
Selection Field		Single-valued Simple Attribute with Value Domain
Button Panel		Single-valued Role of a Relation Type
Button Panel		Multivalued Role of a Relation Type

Figura 13. Elementos de interfaz disponibles en RAINBOW con las reglas de asignación en la estructura de datos.

Una vez que se han obtenido los requisitos y se ha elaborado el esquema conceptual de la base de datos las técnicas de transformación permiten automatizar la producción de esquemas lógicos y físicos, e incluso producir los artefactos de última aplicación: interfaces, programas, código de base de datos…

El siguiente ejemplo (figura 14) muestra como partiendo de las interfaces de usuario y aplicando las diversas técnicas de Ingeniería Inversa, Rainbow obtiene las entidades que compondrán el esquema y realiza la conversión de estas entidades en esquemas independientes.

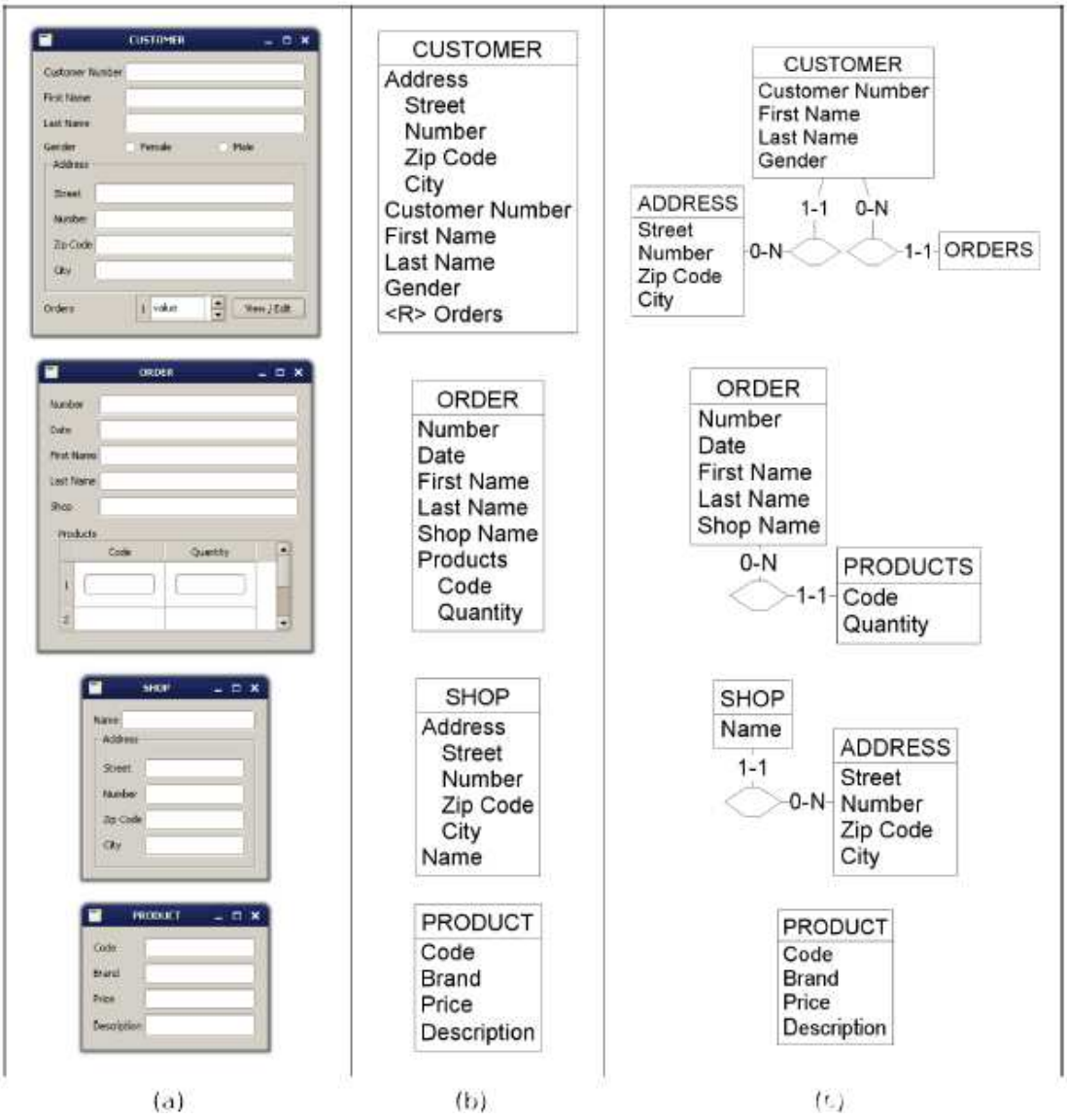

Figura 14. (a)Ejemplo de interfaces de usuario (b) Conversión de las interfaces en entidades (c) Conversión de los tipos de entidades en esquemas independientes.

3.2.1.2 Problemas

A continuación se detallan algunos de los problemas que surgen.
El primer tipo de problemas que aparecen es la redundancia semántica y la ambigüedad.

Este problema surge debido a las limitaciones del lenguaje escrito natural (similitud, polisemia / homografía, sinonimia), que conducen a etiquetas poco claras en las interfaces. Cuando se añade la posibilidad de errores de ortografía, la necesidad de clarificar las etiquetas de la interfaz se vuelve indiscutible.

Otra cuestión se refiere a la redundancia estructural. La redundancia estructural se produce cuando dos conjuntos de atributos son similares y a su vez son compartidos por interfaces diferentes.

Existen diversas técnicas para ayudar a identificar estos problemas e intentar solucionarlos. Pero estas técnicas no siempre facilitan la solución, siempre que sea necesario, los usuarios finales son invitados a elegir qué atributos de los objetos son relevantes y por lo tanto se deben mantener durante el proceso de fusión.

Otro de los problemas de Rainbow es que al generar únicamente el modelo E-R asociado al sistema, solo permite representar la persistencia de datos no permite representar la funcionalidad del sistema.

3.2.2 Software de programación

La herramienta Rainbow *Kit* es un entorno de desarrollo orientado al usuario. Está destinado a ayudar a los usuarios finales en la definición y validación de requisitos de bases de datos a través de prototipos. Ofrece a los usuarios un conjunto de *widgets* listos para usar y reglas de asignación con el modelo entidad-relación.

El kit de herramientas interactúa con el repositorio de DB-Main, que es una herramienta CASE de Ingeniería de Bases de Datos que proporciona todas las funcionalidades necesarias para apoyar un proceso completo de diseño de bases de datos.
También dispone de herramientas de transformación que se apoyan en las técnicas de Ingeniería Inversa de Bases de Datos.

La interacción entre las distintas herramientas que componen el kit permite cubrir todo el proceso de Ingeniería de Bases de Datos ya sea desde la perspectiva de los usuarios finales o desde el punto de vista de los analistas.

3.2.3. Elementos de Interfaz

A continuación se presentan los elementos de Rainbow que se utilizarán en la sección 4 para realizar la integración entre OO-Method y Rainbow.

El elemento *"FORM"* (figura2) representa una ventana o un cuadro de diálogo que constituye la interfaz de usuario de una aplicación.

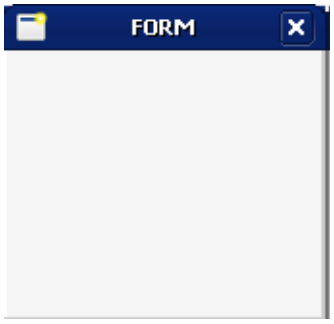

Figura 15. Elemento FORM

El elemento "*FIELDSET"* (figura 16) agrupa varios elementos del formulario relacionados semánticamente.

Figura 16. Elemento FIELDSET

Existen distintas posibilidades para representar el elemento "*TABLE*". El elemento "*TABLE*" consiste en una cuadrícula que contiene filas y columnas. Las columnas contienen encabezados.
Existen distintas formas de representación para este elemento. La representación dependerá de la finalidad para la que se vaya a utilizar el elemento dado que una de las opciones de representación es utilizada habitualmente para la visualización de datos (figura 17) y la otra forma de representación (figura 18) es utilizada para la inserción de datos.

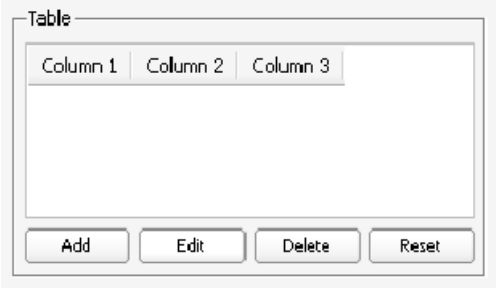

Figura 17. Elemento TABLA 1

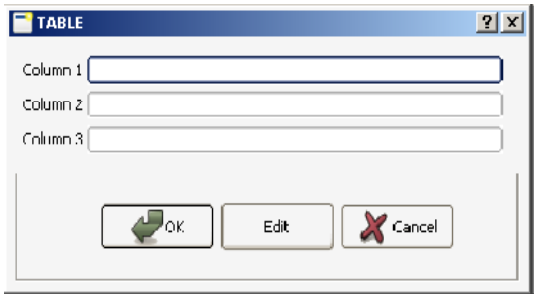

Figura 18. Elemento TABLA 2

El elemento *INPUT* (Figura 19) permite al usuario la entrada de información textual para ser usada por el sistema.
Las cajas de texto no editables pueden servir al propósito simplemente de exhibir texto.

Figura 19. Elemento INPUT

El elemento "*Radio Button*" (Figura 20) permite elegir sólo una opción de un conjunto predefinido de opciones.
Un botón de opción no seleccionado se mantiene en blanco, en tanto uno seleccionado se rellena. Al lado de cada botón de opción suele existir una etiqueta (*label*) que describe la opción que ese botón representa.
Cuando un usuario selecciona un botón de opción, cualquier otro botón de opción previamente seleccionado se deselecciona (siempre que esté dentro del mismo grupo de botones de opción).

El elemento "*Check Box*" permite al usuario (Figura 20) permite al usuario marcar múltiples selecciones de un número de opciones. Generalmente son mostrados en pantalla como cuadraditos que pueden estar vacíos (para falso) o tildados o rellenos (para verdadero). Por lo general al lado de los cuadrados hay un texto que explica el significado de que el casillero esté o no chequeado.

El elemento "*Selection List*" (Figura 20) permite al usuario seleccionar una o más opciones de una lista estática.
Un elemento "*Selection List*" se diferencia de un elemento "*Drop-down list"* porque son estáticas (no se despliegan).

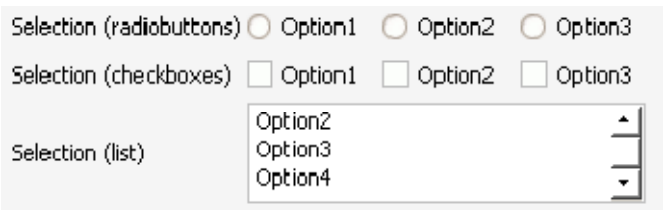

Figura 20. Elementos *Radiobuttons, Checkboxes, Selection List*

El elemento "*Drop-down list*" (Figura 21) permite al usuario seleccionar de una lista desplegable una opción. Se distingue de una simple lista en el hecho de que el usuario debe presionar sobre el menú para que se muestren las opciones disponibles. En otras palabras, el listado no está desplegado como en un elemento "*Selection List*".

Cuando el usuario selecciona una opción, generalmente el menú vuelve a cerrarse y la opción queda seleccionada.

Figura 21. Elemento *Drop-down list*

Los elementos "*BUTTON*" (Figura 22) permiten al usuario comenzar un evento, como buscar, aceptar una tarea, etc.

Figura 22. Elemento BUTTON

4. INTEGRACIÓN ENTRE RAINBOW Y OO-METHOD

4.1. Necesidad de la integración

Como se ha comentado anteriormente, uno de los principales problemas de Rainbow es que no permite representar la funcionalidad del sistema.
Esto es debido a que Rainbow genera automáticamente el diagrama E-R partiendo de las interfaces diseñadas por los usuarios. Este tipo de diagrama solo representa la persistencia del sistema al tratarse de un tipo de diagrama no funcional.

OO-Method permite representar tanto la funcionalidad del sistema como la persistencia, pero una de las desventajas es que no resulta intuitivo para los usuarios. Los diagramas que se utilizan para el diseño y validación del modelo conceptual no son fácilmente entendibles por los usuarios.

En este trabajo proponemos aunar la potencia de ambos métodos, de forma que el usuario pueda participar fácilmente en la captura de requisitos mediante Rainbow, y después, obtener aplicaciones totalmente funcionales a partir de los formularios creados con OO-Method.

4.2. Representación de correspondencias

En la siguiente sección se van a presentar las diferentes correspondencias que hemos detectado entre *OO*-Method y Rainbow.

4.2.1. **Definición de reglas**

A continuación se definen las reglas que se utilizarán para realizar la transformación desde las interfaces diseñadas utilizando la metodología que propone Rainbow al modelo de objetos de OO-Method y al modelo de presentación.

4.2.1.1. Definición de reglas de transformación al modelo de objetos

- **Regla 1:** Cada elemento *FORM* da lugar a una clase.
- **Regla 2:** Los elementos *INPUT* dan lugar a los atributos de las clases.
- **Regla 3:** Los elementos *BUTTON* dan lugar a eventos de la clase.

- **Regla 4:** La inclusión de un elemento *FIELDSET* dentro de un *FORM* da lugar a una relación entre clases.

- **Regla 5:** Cada elemento *TABLE* da lugar a una clase.

- **Regla 6:** Cada columna de un elemento *TABLE* da lugar a un atributo de la clase.

- **Regla 7:** La cardinalidad de las relaciones viene determinada por el número de objetos de cada clase que se puede crear en el formulario.

- **Regla 8:** Una restricción de integridad estática viene determinada por los campos obligatorios (marcados con un asterisco).

4.2.1.2. Definición de reglas de transformación al modelo de presentación

- **Regla 9:** Un patrón de unidad de interacción de servicio viene dado por un elemento *FORM*.

- **Regla 10:** Un patrón de de entrada viene dado por un elemento *INPUT*.

- **Regla 11:** Un patrón de criterio de agrupación viene dado por un elemento *FIELDSET*.

- **Regla 12:** Un patrón de unidad de interacción de servicio viene dado por un elemento *TABLE* en caso de que permita la introducción de datos.

- **Regla 13:** Un patrón de conjunto de visualización viene dado por un elemento *TABLE* en caso de que permita la visualización de datos.

- **Regla 14:** Un elemento de selección definida viene dado por un elemento *RADIOBUTTON*, *CHECKBOX* o *LIST*.

4.2.2. Obtención del Modelo de Objetos de OO-Method

Para presentar las distintas correspondencias existentes, se va a utilizar el ejemplo de un formulario (figura 23) expresado en Rainbow y se mostrará cómo se representan los distintos elementos que lo componen en OO-Method.

El formulario 1 (figura 23) ha sido extraído de una aplicación para la gestión de una joyería.

Los usuarios de la aplicación son los dependientes de la joyería.
En esta joyería, habitualmente los dependientes suelen atender a los mismos clientes, por ese motivo, en los datos de los clientes se almacenan los datos de los dependientes que habitualmente suelen atenderles.

Primero se va a mostrar la representación del modelo de objetos en OO-Method.

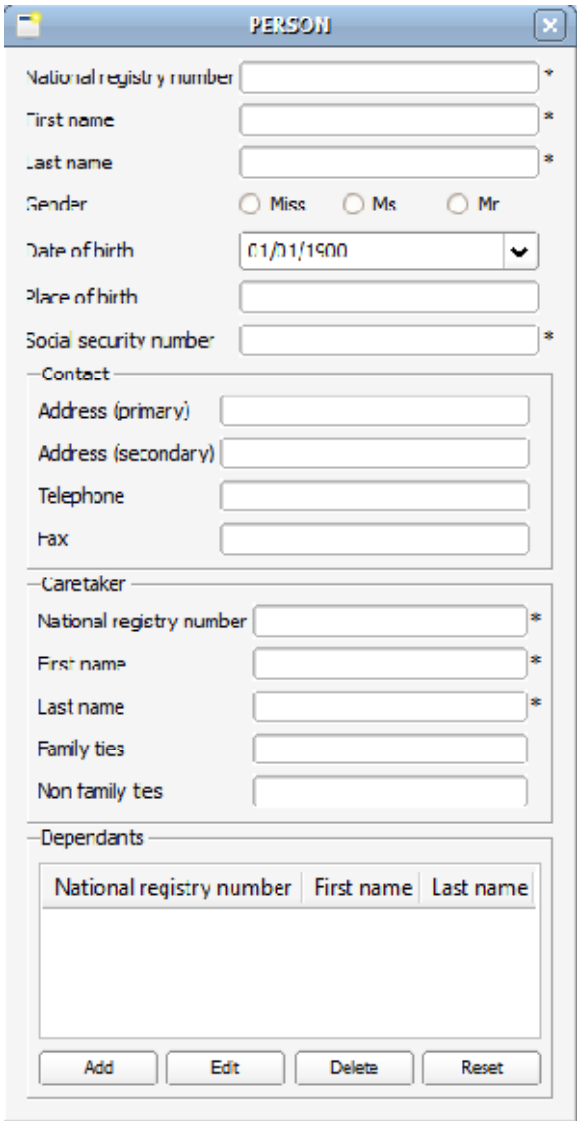

Figura 23. Formulario Persona

La representación del formulario sería la siguiente en OO-Method:

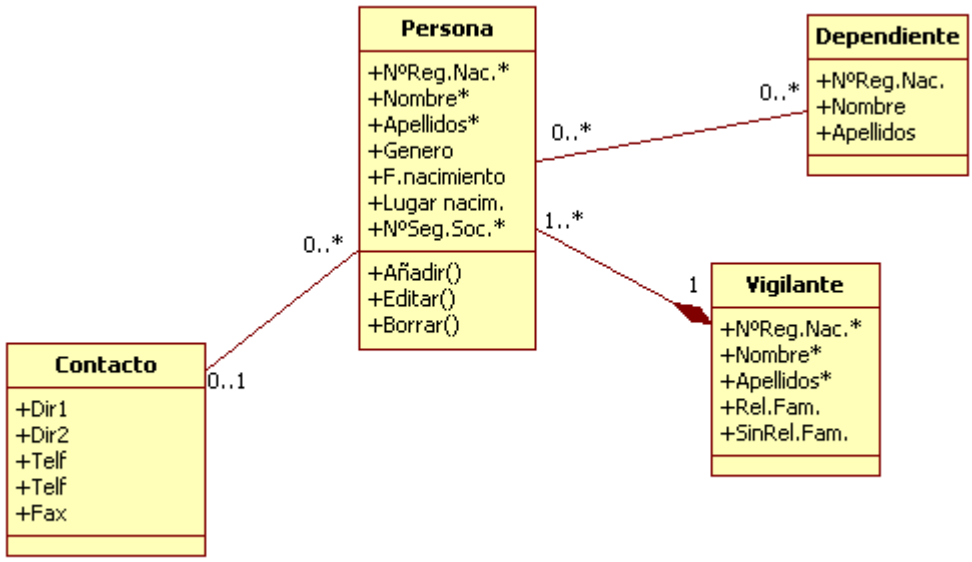

Figura 24. Representación gráfica del Modelo de Objetos en OO-Method

Para la obtención del modelo de objetos, se han aplicado las reglas definidas en la sección *4.1.1. Definición de reglas*.

El elemento *FORM* "Persona", tal y como indica la regla 1 ha dado lugar a la clase "Persona".

Tras la aplicación de la regla 2 los campos de entrada de los formularios dan lugar a los distintos atributos por los que están compuestas las clases.

Tal y como indica la regla 3, los botones de los formularios dan lugar a los eventos de la clase.

El elemento "*FIELDSET* Contacto" incluido en el formulario "Persona", tal y como indica la regla 4 a dado lugar a la relación entre ambas clases.

Como indica la regla 5, el elemento "*TABLE* Dependiente" ha dado lugar a la clase "Dependiente".

Para la obtención de los atributos de la clase "Dependiente", se aplica la regla 6, que indica que cada una de las columnas de la tabla es un atributo de la clase.

Tal como indica la regla 7, la cardinalidad entre las clases "Persona" y "Dependiente" es de 0 a * en ambos sentidos. Esto es debido a que el formulario dispone de un elemento tipo "TABLA"

para crear los objetos de la clase “Dependiente”, por lo tanto un objeto de la clase “Persona” puede estar relacionado con varios objetos de la clase “Dependiente” y viceversa.

La figura 23 contiene varias restricciones de integridad, tal y como indica la regla 8. Los campos obligatorios (figura 24) incluidos en los distintos subformularios obligan al usuario a que introduzca un valor para estos campos.

Figura 25. Elemento CAMPO OBLIGATORIO

4.2.3. Obtención del Modelo de Presentación de OO-Method

A continuación se presentan las correspondencias entre los elementos que componen la interfaz de Rainbow y los elementos del modelo de presentación de OO-Method.

El elemento FORM de Rainbow (figura 23), tal y como indica la regla 9, equivaldría en OO-Method a una unidad de interacción de servicio.
La correspondencia entre ambos elementos es debida a que ambos están pensados para permitir al usuario la introducción de datos.

A continuación vamos a detallar los distintos elementos que componen el elemento FORM de Rainbow.

La correspondencia del siguiente elemento de Rainbow de tipo FIELDSET (figura 26) del ejemplo que estamos utilizando con OO-Method, tal y como indica la regla 11, sería el de patrón de criterio de agrupación dado que presenta una estructura agrupando un conjunto de atributos.

Figura 26. Subformulario Contacto

En el ejemplo del formulario de Rainbow (figura 23) se presenta un elemento TABLE (Figura 27) cuya representación en OO-Method, tal y como indica la regla 13, sería la de un patrón de conjunto de visualización dado que está pensado para la visualización de datos.

Figura 27. Tabla DEPENDIENTE

La correspondencia del siguiente elemento de Rainbow de tipo INPUT (figura 28) del ejemplo que estamos utilizando (figura 23) con OO-Method es la de un dato de entrada, tal y como indica la regla 8.

Figura 28. Input LUGAR DE NACIMIENTO

En los siguientes campos (Figura 29) del formulario del ejemplo (Figura 23) aplica la regla 14. Los patrones de selección definida utilizan atributos de tipo enumerado. Los enumerados son atributos que sólo pueden adquirir un número cerrado de valores. Dependiendo del número de posibles valores que pueda obtener el atributo, se podría elegir "*Radiobuttons*", "*Checkboxes*" o "Listas de selección". Por ejemplo, hasta 5 valores sería recomendable usar *Radiobuttons o Checkboxes* pero para más posibles valores, la lista es la opción óptima.

Se observa como para proporcionar al usuario la opción del estado civil se utilizan los "*radiobuttons"* dado que solo se ofrecen 3 opciones de las cuales el usuario deberá seleccionar una, mientras que para la selección de la fecha de nacimiento se ha utilizado "lista de selección" dado que el número de opciones proporcionadas es superior.

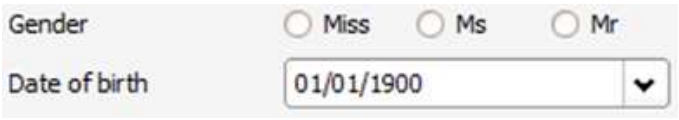

Figura 29. Radiobuttons "Estado Civil", Lista de selección "Fecha de Nacimiento"

4.3. Caso de estudio

En esta sección se presenta un caso de estudio para presentar las correspondencias, explicadas en la sección 4, entre las primitivas de OO-Method y Rainbow.

El caso de estudio presenta el modelo establecido para la gestión de una juguetería llamada "Green Kids".
El contexto de aplicación es una pequeña tienda de juguetes. La aplicación debe permitir realizar la gestión básica de la tienda como por ejemplo dar de alta clientes, dar de alta proveedores, realizar pedidos, emitir facturas, consultar el stock disponible de los productos…

Los usuarios que van a utilizar el sistema no son usuarios avanzados, por ese motivo se pretende que la interacción con el sistema sea lo más sencilla posible.

Durante el proceso de captura de requerimientos, los usuarios han participado activamente. Las interfaces para el sistema han sido creadas por ellos dado que son los que mejor conocen lo que realmente necesitan.
Siguiendo la propuesta de Rainbow se presentan las distintas interfaces basadas en formularios que componen el sistema:

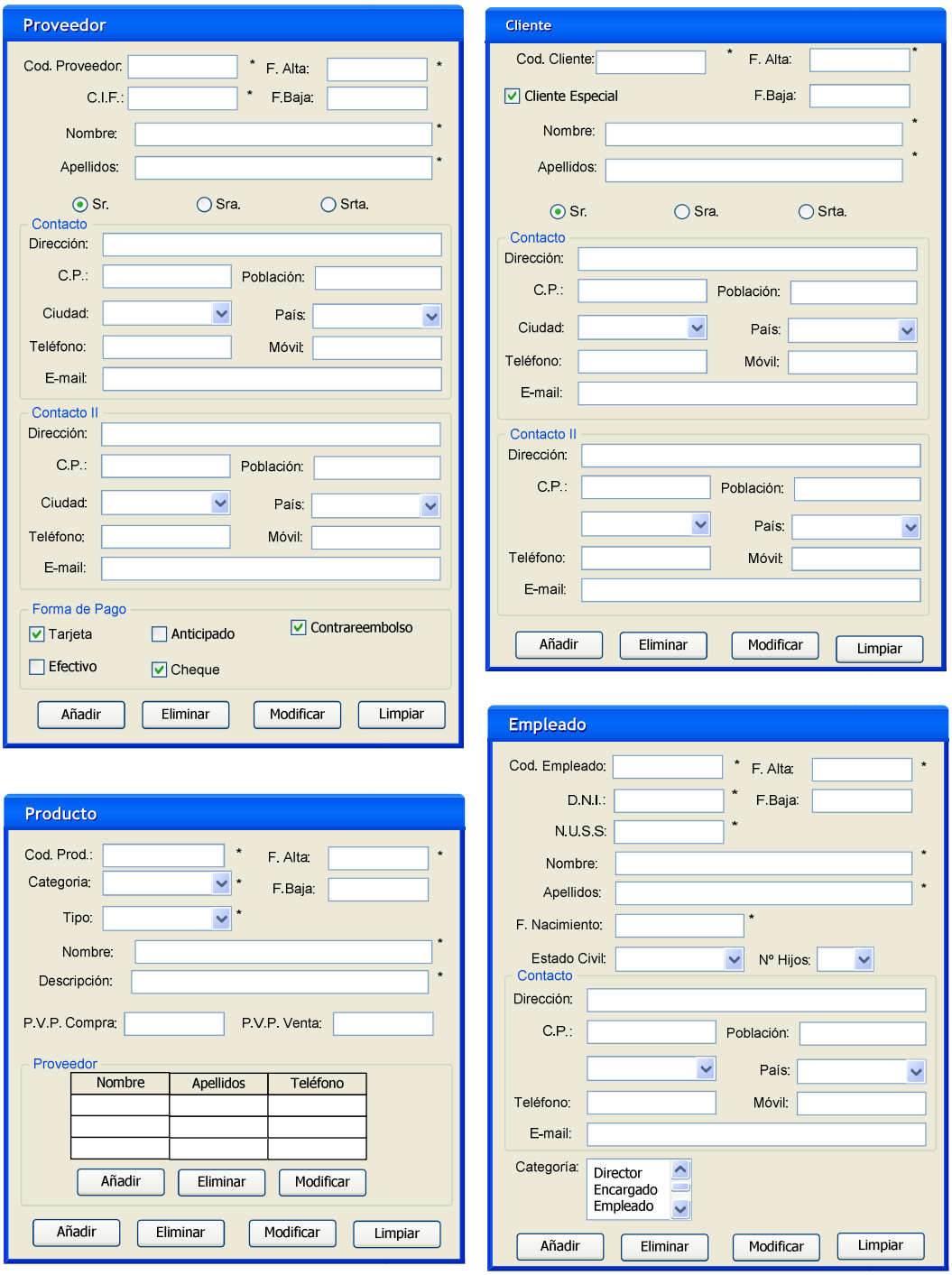

Figura 30. Interfaces de la aplicación para la gestión de la juguetería GreenKids

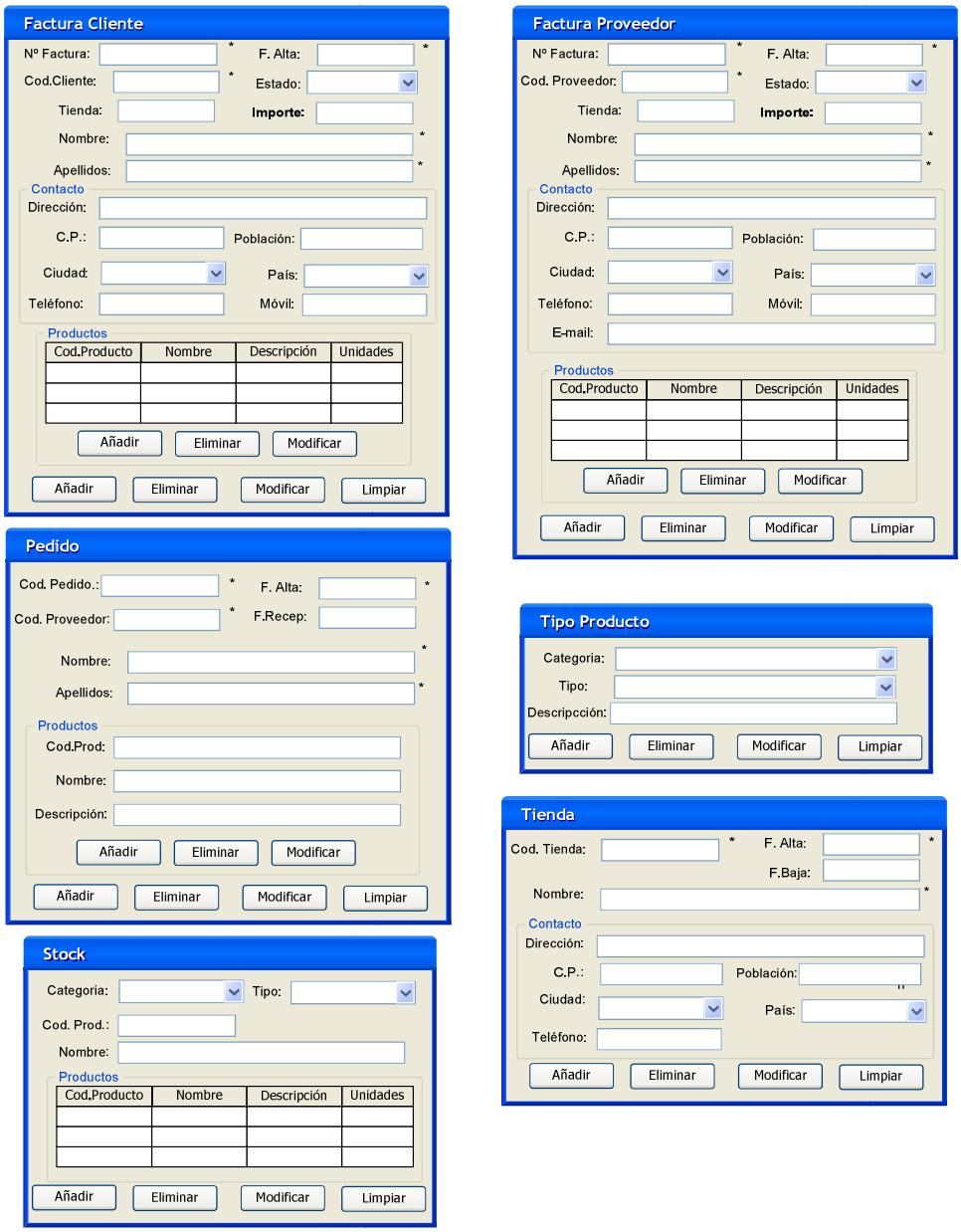

Figura 30b. Interfaces de la aplicación para la gestión de la juguetería GreenKids

A continuación se presenta cada una de las interfaces explicando brevemente su funcionalidad.

Formulario Proveedor

A través de este formulario se realiza el alta de proveedores en el sistema.
Para poder dar de alta un proveedor, es necesario cumplimentar los datos.
Algunos de los datos son campos obligatorios, como por ejemplo el dato “C.I.F.”, estos se diferencian del resto porque están marcados con un asterisco.
A través de este formulario también se puede realizar la consulta y modificación de proveedores.
En caso de dar de baja un proveedor en el sistema, no se realiza una baja física, sino que se realiza una baja lógica. Es decir, el proveedor no se elimina del sistema. Esto se consigue informando el dato “Fecha Baja”.

Formulario Cliente

A través de este formulario se realiza el alta de clientes en el sistema.
Para poder dar de alta un cliente, es necesario cumplimentar los datos.
Algunos de los datos son campos obligatorios, como por ejemplo el dato “Cod.Cliente”, estos se diferencian del resto porque están marcados con un asterisco.
A través de este formulario también se puede realizar la consulta y modificación de clientes.
En caso de dar de baja un cliente en el sistema, no se realiza una baja física, sino que se realiza una baja lógica. Es decir, el cliente no se elimina del sistema. Esto se consigue informando el dato “Fecha Baja”.

Formulario Empleado

A través de este formulario se realiza el alta de empleados en el sistema.
Para poder dar de alta un empleado, es necesario cumplimentar los datos.
Algunos de los datos son campos obligatorios, como por ejemplo el dato “D.N.I.”, estos se diferencian del resto porque están marcados con un asterisco.
A través de este formulario también se puede realizar la consulta y modificación de empleados.
En caso de dar de baja un empleado en el sistema, no se realiza una baja física, sino que se realiza una baja lógica. Es decir, el empleado no se elimina del sistema. Esto se consigue informando el dato “Fecha Baja”.

Formulario Producto

A través de este formulario se realiza el alta de productos en el sistema.
Para poder dar de alta un producto, es necesario cumplimentar los datos.

Algunos de los datos son campos obligatorios, como por ejemplo el dato “Categoría”, estos se diferencian del resto porque están marcados con un asterisco.
A través de este formulario también se puede realizar la consulta y modificación de productos.
En caso de dar de baja un producto en el sistema, no se realiza una baja física, sino que se realiza una baja lógica. Es decir, el producto no se elimina del sistema. Esto se consigue informando el dato “Fecha Baja”.

Formulario Factura Cliente

A través de este formulario se realiza el alta de facturas de clientes en el sistema.
Para poder dar de alta una factura, es necesario cumplimentar los datos.
Algunos de los datos son campos obligatorios, como por ejemplo el dato “Cod.Factura”, estos se diferencian del resto porque están marcados con un asterisco.
Los productos asociados a la factura se insertan a través de la tabla “Productos”.
A través de este formulario también se puede realizar la consulta y modificación de facturas asociadas a clientes.
Cuando una factura es cobrada, se debe modificar el estado actual de la factura a través del elemento “*List Box* Estado”.

Formulario Factura Proveedor

A través de este formulario se realiza el alta de facturas de proveedores en el sistema.
Para poder dar de alta una factura, es necesario cumplimentar los datos.
Algunos de los datos son campos obligatorios, como por ejemplo el dato “Cod.Factura”, estos se diferencian del resto porque están marcados con un asterisco.
Los productos asociados a la factura se insertan a través de la tabla “Productos”.
A través de este formulario también se puede realizar la consulta y modificación de facturas asociadas a proveedores.
Cuando una factura es pagada, se debe modificar el estado actual de la factura a través del elemento *“List Box* Estado”.

Formulario Pedido

A través de este formulario se realiza el alta de pedidos a proveedores en el sistema.
Para poder dar de alta un pedido, es necesario cumplimentar los datos.
Algunos de los datos son campos obligatorios, como por ejemplo el dato “Cod.Pedido”, estos se diferencian del resto porque están marcados con un asterisco.
Los productos asociados al pedido se insertan a través de la tabla “Productos”. Esta tabla permite la inserción de los productos de uno en uno.

Formulario Stock

A través de este formulario se realiza la consulta de existencias de productos en el sistema.
El formulario permite la consulta de dos formas distintas, por categoría - tipo de producto o por producto.
En caso de realizar la consulta por categoría – tipo de producto, es necesario cumplimentar los datos “Categoría” y “Tipo”. El resultado de la consulta será las existencias de todos los productos pertenecientes a esa categoría y tipo.
En caso de realizar la consulta por producto, será necesario cumplimentar los datos “Categoría”, “Tipo”, “Cod. Producto” y opcionalmente el nombre del producto.
Solo se permitirá la compra de tantas unidades de un determinado producto como unidades haya en stock.

Formulario Categoría Producto

A través de este formulario se realiza el alta de categorías de productos en el sistema.
Para poder dar de alta una categoría de producto, es necesario cumplimentar los datos.
A través de este formulario también se puede realizar la consulta, modificación y baja de categorías productos.

Formulario Tipo Producto

A través de este formulario se realiza el alta de tipos de productos en el sistema.
Para poder dar de alta un tipo de producto, es necesario cumplimentar los datos.
A través de este formulario también se puede realizar la consulta, modificación y baja de tipos productos.

Formulario Tienda

A través de este formulario se realiza el alta de los datos de la tienda en el sistema.
Para poder dar de alta una tienda, es necesario cumplimentar los datos.
Algunos de los datos son campos obligatorios, como por ejemplo el dato “Cod.Tienda”, estos se diferencian del resto porque están marcados con un asterisco.
A través de este formulario también se puede realizar la consulta y modificación de las distintas tiendas.
En caso de dar de baja una en el sistema, no se realiza una baja física, sino que se realiza una baja lógica. Es decir, la tienda no se elimina del sistema. Esto se consigue informando el dato “Fecha Baja”.

4.3.1. **Obtención del Modelo de Objetos**

El primer paso es la representación del modelo de objetos en OO-Method a partir de los formularios que componen el sistema.

La representación sería la siguiente en OO-Method:

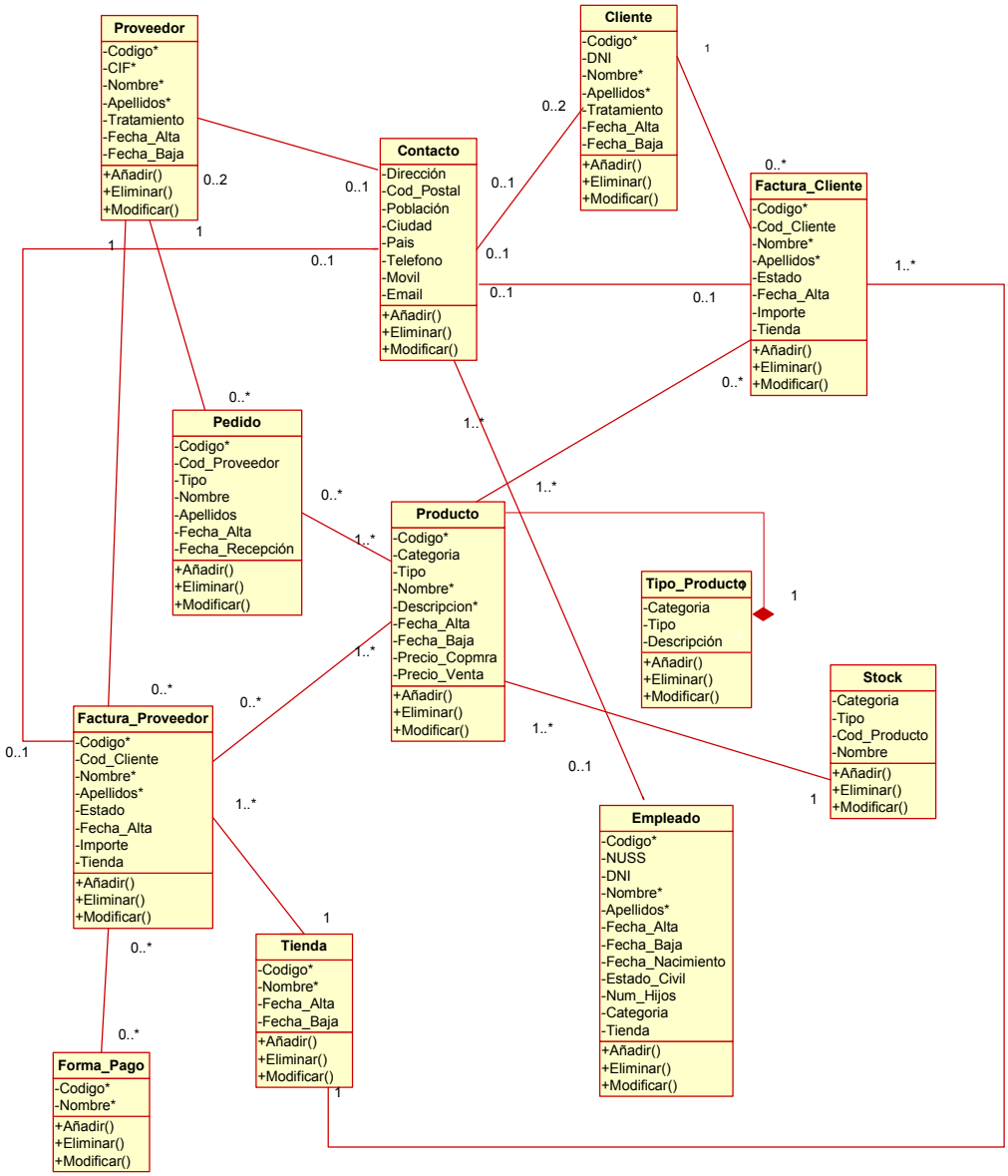

Figura 31. Representación gráfica del Modelo de Objetos en OO-Method

Para la obtención del modelo de objetos (figura 31) se han aplicado las reglas definidas en esta sección.

Los distintos formularios que componen la aplicación, tal y como indica la regla 1, dan lugar a las distintas clases que componen el sistema.
Por ejemplo, el formulario “Proveedor” ha dado lugar a la clase “Proveedor”.

Tras la aplicación de la regla 2 los campos de entrada de los formularios dan lugar a los distintos atributos por los que están compuestas las clases.
Por ejemplo, el formulario “Proveedor” está compuesto de distintos campos de entrada como por ejemplo, el campo de entrada “Nombre” o el campo de entrada “Apellidos”. Estos datos de entrada son algunos de los atributos que componen la clase “Proveedor”.

Existe un formulario “Producto” que tras aplicar la regla 1, ha dado lugar a la clase “Producto”. En caso de que este formulario no hubiera existido, la clase “Producto” se habría obtenido tras aplicar la regla 5 al elemento *TABLE* del formulario “Factura Cliente” y los atributos de la clase se habrían obtenido tras aplicar la regla 6 que indica que cada una de las columnas da lugar a los distintos atributos que componen la clase.

Tal y como indica la regla 3, los botones de los formularios dan lugar a los eventos de la clase. La clase “Cliente” dispone de tres eventos, el evento añadir, el evento modificar y el evento borrar.

El elemento “*FIELD SET* Dirección” de Rainbow aparece en varios formularios, en el formulario “Cliente”, en el formulario “Proveedor”, en el formulario “Empleado”, en el formulario “Tienda”, en el formulario “Factura de Cliente” y “Factura de Proveedor”. En OO-Method esto se representa con las relaciones entre clases, tal y como indica la regla 4.
La clase “Dirección” está relacionada con la clase “Cliente”, con la clase “Proveedor”, con la clase “Empleado”, con la clase “Factura de Cliente”, con la clase “Factura de Proveedor” y con la clase “Tienda”.
Lo mismo sucede con el subformulario “Producto”, este subformulario aparece en el formulario “Stock”, en el formulario “Pedido”, en el formulario “Factura de Cliente” y en el formulario “Factura de Proveedor”, por lo tanto, en la representación gráfica la clase “Producto” estará relacionada con la clase “Stock”, con la clase “Pedido”, con la clase “Factura de Cliente” y con la clase “Factura de Proveedor”.

Tal como indica la regla 5, la cardinalidad entre las clases “Producto” y “Factura de Compra” es de 1 a * en ambos sentidos. Esto es debido a que el formulario dispone de un elemento tipo “TABLE” para crear los objetos de la clase “Producto”, por lo tanto un objeto de la clase “Factura de Compra” puede estar relacionado con varios objetos de la clase “Producto” y viceversa.

Las figura 30 y 30b contienen varias restricciones de integridad, tal y como indica la Regla 6. Los campos obligatorios (marcados con un asterisco) incluidos en los distintos subformularios obligan al usuario a que introduzca un valor para estos campos.

4.3.2. **Obtención del Modelo de Presentación**

A continuación se detalla la transformación de cada una de las interfaces propuestas en Rainbow indicando su correspondencia con los distintos elementos del modelo de presentación de OO-Method.
Para la obtención del modelo de objetos se han aplicado las reglas definidas en el apartado *4.1.1. Definición de reglas*.

Formulario Proveedor

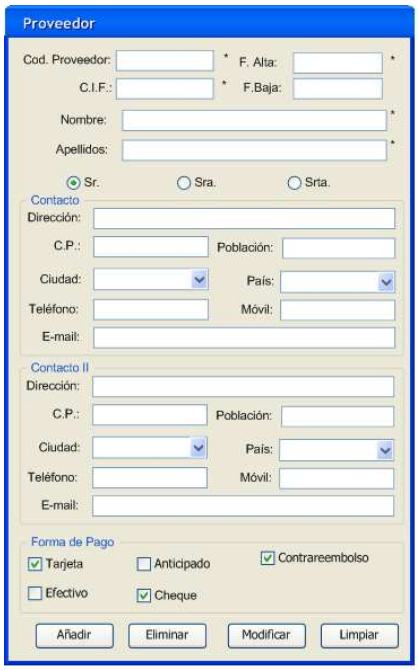

Figura 32. Formulario Proveedor

El elemento "*FORM*", tal y como indica la regla 9, da lugar a un patrón de interacción de servicio.

El elemento "*FORM*" está compuesto por varios elementos "*INPUT*" como por ejemplo el dato "Fecha de Baja". La correspondiente representación en OO-Method tal y como indica la regla 10 sería la de un patrón de entrada.

También hay varios elementos "*FIELDSET*". Para estos elementos se aplica la regla 9 que indica que estos elementos representan un patrón de criterio de agrupación.

Dentro del elemento *"FIELDSET* Forma de Pago" se observa un conjunto de "C*HECKBOXES*", tal como indica la regla 14, este tipo de elementos representa un patrón de selección definida en OO-Method.
En este caso dado que el proveedor puede tener diversas formas de pago, se ha seleccionado como forma de representación el elemento "*CHECKBOX*" para que pueda seleccionar varias opciones dado que la selección de una de ellas no es excluyente. Otro de los motivos por los que se ha seleccionado este elemento es que el número de opciones que se le proporciona al usuario es inferior a 5. En caso de que el número de opciones fuera superior a cinco sería recomendable utilizar otro elemento como por ejemplo una lista.

Esto sucede con el elemento *"LIST* Ciudad", al que también se le aplica la regla 14, representado dentro del elemento "*FIELD SET* Contacto". Al usuario se le ofrece un número elevado de opciones, por ese motivo se ha seleccionado esta forma de representación.

Para que el usuario seleccione la opción del "Tratamiento" para los proveedores, se ha optado por la opción de los elementos "*RADIO BUTTON*", al igual que a los otros dos elementos comentados anteriormente a este tipo de elementos también se les aplica la regla 14. Se ha optado por la representación con elementos "*RADIO BUTTON*" dado que únicamente se ofrecen 3 opciones y además el usuario solo debe de seleccionar una de las 3 opciones.

Formulario Cliente

El formulario "Cliente" es muy similar al formulario "Proveedor". Este formulario no presenta ningún tipo de elemento nuevo.

Formulario Producto

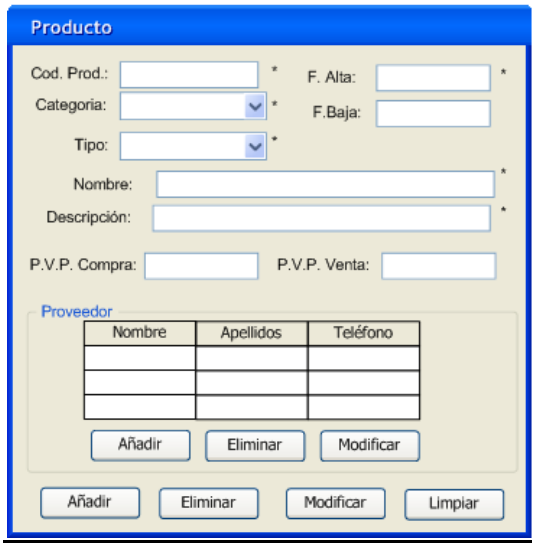

Figura 33. Formulario Producto

Muchos de los elementos que componen el formulario "Producto" son iguales que los explicados en el formulario "Proveedor".

Pero en este formulario aparece un tipo de elemento que no se ha visto en los anteriores formularios. Se trata del elemento *"TABLE"*. En este caso al elemento "*TABLE*" se utiliza para la visualización de datos. Al tratarse de un elemento de visualización se le aplica la regla 13, que indica que la correspondiente representación es la de un patrón de un conjunto de visualización.

Formulario Empleado

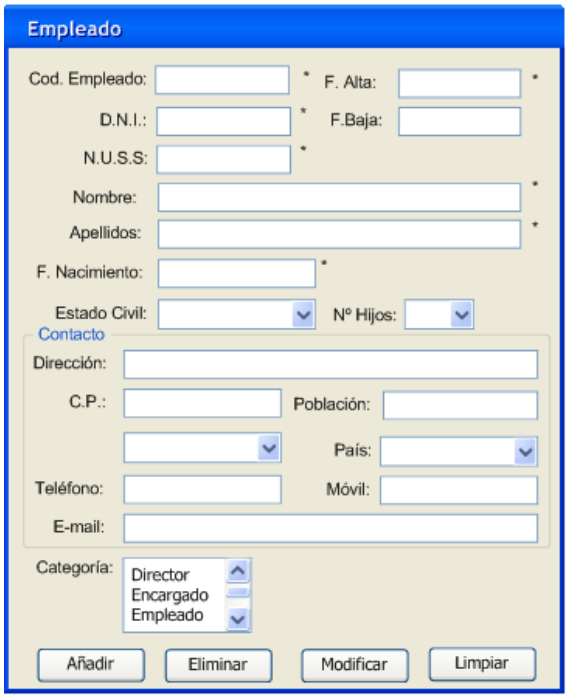

Figura 34. Formulario Empleado

Muchos de los elementos que componen el formulario "Empleado" son iguales que los explicados en los formularios anteriores.

Pero en este formulario aparece el elemento "*LIST* Categoría", tal como indica la regla 14, este tipo de elementos representa un patrón de selección definida en OO-Method.
Mediante este elemento se selecciona la categoría profesional del empleado.
Se ha seleccionado este elemento debido a que el número de opciones que se le proporciona al usuario es inferior a 5.

Formulario Factura Cliente

El formulario "Factura Cliente" es muy similar a los formularios presentados anteriormente. Este formulario no presenta ningún tipo de elemento nuevo.

Formulario Factura Proveedor

El formulario "Factura Proveedor" es muy similar a los formularios presentados anteriormente. Este formulario no presenta ningún tipo de elemento nuevo.

Formulario Stock

El formulario "Stock" es muy similar a los formularios presentados anteriormente. Este formulario no presenta ningún tipo de elemento nuevo.

Formulario Tipo de Producto

El formulario "Tipo de Producto" es muy similar a los formularios presentados anteriormente. Este formulario no presenta ningún tipo de elemento nuevo.

Formulario Tienda

El formulario "Tienda" es muy similar a los formularios presentados anteriormente. Este formulario no presenta ningún tipo de elemento nuevo.

Formulario Pedido

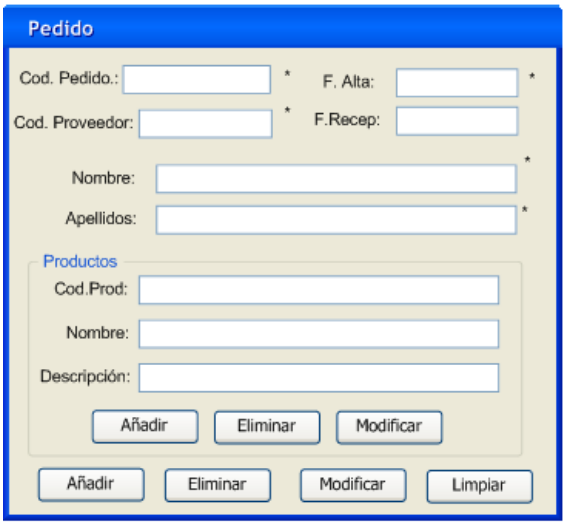

Figura 35. Formulario Proveedor

Muchos de los elementos que componen el formulario "Pedido" son iguales que los vistos en los formularios anteriores.

En el formulario "Producto", presentado anteriormente, se ha visto una representación del elemento "*TABLE*". En ese caso se utilizaba para la visualización de datos.
En este formulario aparece otra representación del elemento "*TABLE*" que no se ha visto en los anteriores formularios. En este caso al elemento "*TABLE* Productos" se utiliza para la introducción de datos. Se aplica la regla 12, que indica que se representa como un patrón de unidad de interacción de servicio.

5. CARENCIAS DE RAINBOW

En la sección 4 se ha realizado la correspondencia entre los elementos de Rainbow y los elementos OO-Method. Pero en OO-Method existen muchos elementos que no existen en Rainbow. A continuación se presentan aquellos elementos que existen de OO-Method pero que no son representables en Rainbow.

En la comparación realizada entre Rainbow y OO-Method aparecen diversas carencias por parte de Rainbow que se explicarán en esta sección.

OO-Method es una metodología que cubre todas las fases del proceso de desarrollo de software, desde la recogida de requerimientos, pasando por la definición del modelo conceptual hasta la generación del producto software final, a diferencia de Rainbow que se centra en la especificación y validación de requisitos.

En Rainbow no se puede definir la funcionalidad del sistema mientras que en OO-Method esto se consigue gracias al Modelo Dinámico y al Modelo Funcional. Rainbow únicamente "proporciona" el Modelo de Datos. Decimos "proporciona" dado que lo obtiene automáticamente aplicando técnicas de *Ingeniería Inversa* a las interfaces creadas por los usuarios.
Por lo tanto una de las principales carencias de Rainbow frente a OO-Method es que Rainbow no permite funcionalidad, solo persistencia.

5.1. Carencias

A continuación se presentan las carencias que presenta Rainbow para en un futuro, dada la utilidad que proporciona Rainbow, extender la metodología actual para que proporcione soporte a OO-Method.

Primero se presentarán aquellos elementos correspondientes al Modelo de Objetos que no son representables en RAINBOW.

Una de las carencias de Rainbow frente a OO-Method, es que en OO-Method se pueden representar "Transacciones".

Las "Transacciones" se tratan de unidades moleculares compuestas de servicios. Normalmente no ejecutan un único evento.

Tienen dos propiedades fundamentales:

- No se visualizan los estados intermedios.

- Ejecución “todo o nada”: cuando una transacción comienza a ejecutarse es necesario continuar hasta el final. En caso contrario se aborta la ejecución y el sistema se devuelve al estado inmediatamente anterior al inicio de la transacción).

Otra de las carencias de Rainbow son las “Operaciones”. El concepto de “Operación” en OO-Method es muy similar al concepto, explicado anteriormente, de “Transacción”. La diferencia es que las “Operaciones” no satisfacen las dos propiedades fundamentales de las “Transacciones”.

Rainbow tampoco dispone de “Precondiciones”. Las “Precondiciones” son condiciones que deben cumplir un evento, una transacción o una operación para ser ejecutada.
Una “Precondición” es una fórmula Booleana que puede ser definida usando constantes, funciones, atributos y argumentos de servicio
OO-Method permite representar un conjunto de vistas lo cual no es representable en Rainbow.

Tanto Rainbow como OO-Method permiten definir interfaces, y en ambas metodologías sirven para representar las relaciones entre clases pero en OO-Method además de definición de interfaces se pueden definir “Vistas”.

Para poder definir las vistas en OO-Method, previamente de definirán varios conceptos que son representables en OO-Method pero no en Rainbow y son necesarios para explicar el concepto de “Vistas”.

Rainbow al igual que OO-Method permite la representación de relaciones entre clases. En ambas metodologías se permite la representación de tipos de relación “Asociación”, Agregación”, “Composición” y “Especialización” pero en OO-Method además existe otro tipo diferente, se trata de los “Agentes de relación”. Los “Agentes de relación” representan que objetos están autorizados para activar los servicios de otras clases.

Una interfaz en OO-Method se compone de un “Agente de relación” dado entre dos clases (un parte actúa como agente; la otra parte actúa como servidor), y además define su propia visibilidad. De esta manera, cada clase agente tiene una interfaz para cada una de las clases a las que se puede acceder, ya sea a sus atributos o para ejecutar sus servicios.

Definidos los conceptos “Agente de relación” e “Interfaz” ya se pueden definir las “Vistas” en OO-Method.

Las “Vistas” en OO-Method son un conjunto de interfaces que se definen en dos pasos:

- Selección de los agentes de clase que participan en la vista.
- Selección de la interface para cada agente de clase.

La selección de agentes de clase determina que perfiles pueden interactuar con el sistema, mientras la selección de las interfaces de cada agente determina la visibilidad que cada perfil tendrá del sistema.

Si no se define una vista explícitamente, se asume la vista implícita, que incluye todos los agentes de clases del sistema y todas las interfaces.

A continuación se presentan los elementos correspondientes al Modelo de Presentación de OO-Method que no son representables en Rainbow.

En OO-Method se pueden definir “Argumentos de Agrupación”. Los “Argumentos de Agrupación” definen la manera en que para un servicio se presentan al usuario los argumentos de entrada.
Los argumentos de entrada del servicio son organizados en una lista ordenada que establece el orden y se muestra al usuario, de modo que puede informar los argumentos de entrada antes de la ejecución.

OO-Method también permite la definición de “Argumentos de Dependencia”. Los “Argumentos de Dependencia” permiten definir relaciones de dependencia entre los valores o el estado de un argumento de entrada de un servicio y los valores o el estado de otros argumentos de entrada del mismo servicio.
Usan las reglas tipo-ECA (evento, condición, acción) para cada uno de los argumentos de entrada del servicio. Cuando ocurre un evento interfaz para una acción específica, esta se ejecuta si la condición se cumple.
Este elemento permite especificar el comportamiento dinámico determinado por la interfaz de usuario. Siempre se aplican dentro de las “Unidades de Interacción de Servicio”.

Las “Acciones” en OO-Method son otra que se puede realizar sobre un objeto, además de la navegación, es ejecutar un servicio en él. El enfoque más sencillo sería permitir al usuario invocar, para un objeto particular, cualquier servicio definido por la clase del objeto (de acuerdo con la especificación en el modelo dinámico). Sin embargo, y con el fin de mejorar la interactividad y la interacción, es conveniente que el conjunto de servicios disponibles sea limitado y estén clasificados por la frecuencia de uso.
Una acción se define para una clase en particular y permite definir el conjunto de servicios que están disponibles para ser ejecutados en un objeto de esa clase. Para una misma clase se pueden definir múltiples acciones.

6. CONCLUSIONES

En este documento se ha presentado la integración de OO-Method y Rainbow.

Las ventajas que se obtienen de integrar Rainbow con OO-Method es que Rainbow aprovecha la experiencia de los usuarios para definir y validar el modelo de datos mediante Rainbow, y la potencia de OO-Method en cuanto a la generación de aplicaciones funcionales.

Rainbow utiliza las ventajas de las interfaces basadas en formularios para la captura de los requerimientos y aplica diversas técnicas de Ingeniería Inversa a estos formularios para generar un modelo E-R extendido.

Posteriormente se deriva al modelo de datos y de presentación de OO-Method. Para obtener ambos modelos, el analista completa los elementos que no son soportados por Rainbow.

Con la integración de ambas metodologías se pretende facilitar la actual captura de requerimientos de OO-Method.

Las lecciones aprendidas durante el desarrollo de este trabajo son las siguientes:

- La ambigüedad de las reglas puede ser un problema. Algunos elementos origen de Rainbow dan lugar a varios elementos destino en OO-Method.
  Esta ambigüedad puede conducir a transformaciones incorrectas entre las dos metodologías.

- Todas las reglas no se pueden aplicar automáticamente. Esto es debido a la ambigüedad existente entre algunas reglas.

- Las reglas definidas son específicas para OO-Method. La propuesta se puede aplicar a otros métodos MDD distintos a OO-Method, pero el esfuerzo requerido para adaptar las reglas aquí presentadas va a depender de la expresividad que tenga el método MDD para representar la interacción.
  OO-Method dispone del modelo de presentación, que facilita enormemente su integración con Rainbow. Otros métodos MDD sin modelo para representar la interacción con el usuario o con modelos de interacción poco expresivos serían más difíciles de integrar con Rainbow.

Uno de los trabajos futuros será la definición de las reglas presentadas en la sección *4.1.1. Definición de reglas*, en lenguaje ATL (*ATLAS Transformation Languaje*). Esto permitirá aplicar las reglas de transformación de forma semiautomática. El analista decidirá qué reglas aplicar y su aplicación será en base a transformaciones especificadas mediante metamodelos.

Tras realizar la definición de las reglas en lenguaje ATL, se pretende realizar la integración real entre OO-Method y Rainbow. La integración real supondrá superar todas las dificultades

encontradas. Para poder realizar está integración será necesario eliminar las carencias actuales de Rainbow, presentadas en la sección 5.

Una vez se haya realizado la integración de OO-Method y Rainbow se pretende mejorar la captura de requerimientos en OO-Method. Está mejora se consigue integrando en OO-Method las ventajas que proporciona Rainbow para la captura y validación de requerimientos.

7. REFERENCIAS

[1] Pastor, O., Molina, J.C. Model-Driven Architecture in Pratice: A Software Production Environment Based on Conceptual Modeling. Springer (2007)

[2] Ramdoyal, R., Reverse Engineering User-Drawn Form-Based Interfaces for Interactive Database Conceptual Analysis, Phd Thesis, Presses Universitaires de Namur, University of Namur, Namur, Belgium, (2010)

[3] The Rainbow Project: http://info.fundp.ac.be/~rra/rainbow/ Última visita: Sep-2012

[4] Pelechano V., Pastor O., Insfrán E. and Carsí J.A. *OO-Method: Una apuesta por la Integración de Técnicas Formales y Semi-formales en la Ingeniería de Requisito*s. in I Jornadas de Ingenieria del Software. Noviembre 1996. Sevilla, España

[5] Schaefer, R., Steffen, B., Wolfgang, M., Task Models and Diagrams for User Interface Design, Proceedings of 5th International Workshop, TAMODIA'2006 (Hasselt, Belgium, October 2006), Lecture Notes in Computer Science, Vol. 4385, Springer Verlag Berlin, 2006, pp. 39-53

[6] Kost, S. (2004), Dynamically generated multi-modal application interfaces. Ph.D. Thesis, Technical University of Dresden and Leipzig University of Applied Sciences, Germany

[7] Johnson, P. D. & Parekh, J. (2003), Multiple Device Markup Language a Rule Approach. SE MS Project & Thesis (SE690), DePaul University

[8] Luyten, K., Abrams, M., Vanderdonckt, J. & Limbourg, Q. (2004) Developing User Interfaces with XML: Advances on User Interface Description Languages, Sattelite workshop of Advanced Visual Interfaces 2004, Gallipoli, Italy

[9] Demler, G., Wasmund, M., Grassel, G., Spriestersbach, A. & Ziegert, T. (2003), Flexible pagination and layouting for device independent authoring, WWW2003 Emerging Applications for Wireless and Mobile access Workshop

[10] Picard, E., Fierstone, J., Pinna-Dery, A-M. & M. Riveill (2003). Atelier de composition d'ihm et évaluation du modèle de composants. Livrable I3, RNTL ASPECT, Laboratoire I3S, mai.

[11] Paternò, F. & Santoro, C. (2003), A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms, Interacting with Computers, Elsevier, 15, pp. 349-366.

[12] Vanderdonckt J., and Bodart F. (1993), Encapsulating knowledge for intelligent automatic interaction objects selection In Ashlund S., Mullet K., Henderson A., Hollnagel E., and White T. (Eds.), Proceedings of the ACM Conference on Human Factors in Computing Systems InterCHI'93 (Amsterdam, 24-29 April 1993), ACM Press pages, New York, 1993, pp. 424-429.

[13] Helms, J. Schaefer, R, Luyten, K., Vanderdonckt, J., Vermeulen, J. & Abrams, M. (2008), UIML Version 4.0: Committee Draft, Available at http://www.oasis-open.org/committees/download.php/28457/uiml-4.0-cd01.pdf

[14] IBM (2002), WSXL specification, April 2002, retrieved on Janury 2nd 2009

[15] Gomes de Sousa, L. & Leite, J. C. (2003), XICL: a language for the user's interfaces development and its components. Proceedings of the Latin American conference on Human-computer interaction (Rio de Janeiro, Brazil, August 17 - 20, 2003), ACM Press pages, New York, pp. 191-200

[16] Eisenstein J., Vanderdonckt J., Puerta A. (2001), Model-Based User-Interface Development Techniques for Mobile Computing, Proceedings of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001 (Santa Fe, 14-17 January 2001), Lester, J. (Ed.), ACM Press, New York, 2001, pp. 69-76

[17] Puerta A.R. (1996), The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development, Proceedings of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur, 5-7 June 1996), Presses Universitaires de Namur, 1996, pp. 19-35

[18] Pastor,O.,Hayes,F.,Bear,S*. OASIS: An Object Oriented Specification Language*, Springer Verlag

[19] OMG (2003) MDA Guide Version 1.0.1: http://www.omg.org/docs/omg/03-06-01.pdf

[20] Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide, Addison Wesley, 1999

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

[22] 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

ÍNDICE DE FIGURAS