Universidad Politécnica de Valencia

Departamento de Sistemas
Informáticos y Computación

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

Plantillas de Transformación
Añadiendo Flexibilidad a las Transformaciones de Modelos de Interfaces de Usuario

Nathalie Marylin Aquino Salvioni

Director
Óscar Pastor López

– Diciembre de 2008 –

Centro de Investigación
en Métodos de
Producción de Software

Correo Electrónico del autor: naquino@pros.upv.es

Dirección del autor:

Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera, s/n, Edificio 1F
46022 Valencia
España

Resumen

Un proceso de desarrollo de interfaces de usuario dirigido por modelos está basado en un conjunto de transformaciones de modelo a modelo y de modelo a código.

Típicamente, la lógica que guía estas transformaciones está implícita y fija en las herramientas que realizan las transformaciones, lo cual resulta en la generación de interfaces de usuario predeterminadas, siempre muy parecidas unas a otras, y sobre las que los diseñadores de interfaces de usuario deben aplicar modificaciones manuales a la hora de satisfacer requisitos de usuarios finales que escapan a la capacidad del proceso dirigido por modelos.

En otros casos, la lógica que guía las transformaciones está explícita, expresada, por ejemplo, con modelos de mapeos y de transformaciones o con lenguajes de transformaciones de modelos. En este caso, los diseñadores de interfaces de usuario deben añadir las nuevas opciones de transformación, o modificar las existentes, cuando necesitan dar soporte a un requisito de usuario final no cubierto con las opciones disponibles. Sin embargo, este es un proceso complejo, más orientado a especialistas en transformaciones de modelos que a diseñadores de interfaces de usuario.

En este trabajo se presenta una aproximación de Plantillas de Transformación que pretende, por una parte, externalizar y hacer adaptable y reutilizable la lógica de transformación de modelos de interfaces de usuario, y por otra parte, pretende facilitar a los diseñadores de interfaces de usuario en entornos de desarrollo dirigidos por modelos, la especificación de cómo deben ser las transformaciones.

Las Plantillas de Transformación están formadas por parámetros que parametrizan las transformaciones de modelos de interfaces de usuario. Además, pueden ser utilizadas con distintos métodos de desarrollo de interfaces de usuario dirigidos por modelos.

Las Plantillas de Transformación y sus conceptos asociados han sido claramente descritos y caracterizados con meta-meta modelos y meta modelos.

Se ha implementado un prototipo de editor de Plantillas de Transformación.

A modo de aplicación práctica de las Plantillas de Transformación, se ha definido un Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method, un método que permite la generación automática de software a partir de modelos conceptuales.

Abstract

A model-driven user interface development process is based on a set of model-to-model and model-to-code transformations.

The logic that guides these model transformations is usually implicit and fixed in tools that perform the transformations. This results in the generation of predetermined user interfaces, all of them looking alike. When user interface designers need to satisfy end-user requirements that go beyond the scope of the model-driven process, they perform manual tweaking.

In other cases, the logic that guides transformations is explicit and expressed, either with mapping models and transformation models or with model transformation languages.

When user interface designers need to give support to an end-user requirement which is not covered with the available transformation options they must enter new options, or edit existing ones.

However, this complex process is more oriented to model transformation specialists than to user interface designers.

This work presents a Transformation Templates approach which is intended to externalize the transformation logic of user interface models, and to make it editable, customizable and reusable. It is also intended to make it easier for user interface designers to specify transformations.

Transformation Templates are composed of parameters that parameterize user interface model transformations. Besides, they can be adopted by different model-driven user interface development methods.

Transformation Templates and their associated concepts have been clearly described with meta-meta models and meta models.

Moreover, a prototype of a Transformation Template editor has been implemented.

Finally, as a practical application of Transformation Templates, a catalog of parameter types has been defined for the OO-Method Presentation Model. OO-Method is a method that allows the automatic generation of software from conceptual models.

Índice general

Índice de figuras

1

Introducción

Los métodos tradicionales de desarrollo de interfaces de usuario son mayormente artesanales, y son típicamente llevados a cabo por los mismos programadores de la lógica y persistencia de los sistemas de información, por lo que, la calidad de las interfaces de usuario resultantes es altamente dependiente de la habilidad y experiencia de dichos programadores.

Por una parte, la calidad de las interfaces de usuario es un factor determinante para la implantación exitosa de un sistema de información, y para la aceptación y satisfacción del usuario final. Por otra parte, un porcentaje significativo de la actividad de desarrollo de software es invertido en el diseño e implementación de las interfaces de usuario. Por estos motivos, se puede afirmar que el desarrollo de las interfaces de usuario precisa de un proceso formal y bien definido que asegure la calidad de las interfaces de usuario resultantes y que sea automatizable y reaplicable.

Los métodos de desarrollo de software que se estructuran con una Arquitectura Dirigida por Modelos (MDA, por las siglas en inglés de *Model-Driven Architecture*) [Mukerji and Miller, 2003] sientan las bases para el proceso que se necesita.

Las tecnologías de desarrollo de interfaces de usuario dirigidas por modelos tienen el objetivo de proveer un ambiente donde los desarrolladores puedan diseñar e implementar interfaces de usuario de manera profesional y sistemática, y de esta manera son más fáciles de utilizar que las herramientas tradicionales de desarrollo de interfaces de usuario. A fin de conseguir el objetivo, las interfaces de usuario se describen utilizando modelos. En [da Silva, 2000] se distinguen tres ventajas principales derivadas del uso de modelos de interfaces de usuario:

- Pueden proveer una descripción más abstracta de una interfaz de usuario que las descripciones de interfaces de usuario provistas por las herramientas tradicionales de desarrollo de interfaces de usuario
- Facilitan la creación de métodos para diseñar e implementar una interfaz de usuario de manera sistemática ya que proveen capacidades para:
    - Modelar interfaces de usuario a distintos niveles de abstracción

  - Refinar los modelos de manera incremental
  - Reutilizar especificaciones de interfaces de usuario
- Proveen la infraestructura requerida para automatizar las tareas relacionadas al diseño de la interfaz de usuario y a los procesos de implementación.

Diversas aproximaciones se han propuesto para el desarrollo dirigido por modelos de las interfaces de usuario. Entre otras, se tienen las propuestas de [Abrams et al., 1999; Limbourg et al., 2004; Mori et al., 2004; Vanderdonckt, 2005].

A pesar de que no se cuenta con una aproximación de modelado ampliamente aceptada y difundida para el modelado de las interfaces de usuario (algo, por ejemplo, equivalente a lo que son los modelos Entidad-Relación para el modelado de bases de datos), el Marco de Referencia Cameleon (*Cameleon Reference Framework*) [Calvary et al., 2003] ha sentando las bases sobre las cuales se asientan muchos de los trabajos actuales sobre el tema.

El Marco de Referencia Cameleon descompone el contexto de uso de una interfaz de usuario en tres aspectos:

- Los usuarios finales
- La plataforma de computación hardware y software
- El ambiente físico

Además, estructura el ciclo de desarrollo de las interfaces de usuario en cuatro niveles de abstracción:

- Tareas y Conceptos (Dominio)
- Interfaz de Usuario Abstracta
- Interfaz de Usuario Concreta
- Interfaz de Usuario Final

Estos niveles se estructuran con relaciones de reificación que van de los niveles más abstractos a los niveles más concretos, y relaciones de abstracción que van de los niveles más concretos a los más abstractos.

Un método de desarrollo de interfaces de usuario dirigido por modelos conforme al Marco de Referencia Cameleon, sigue las directrices de MDA, pues empieza definiendo un modelo de interfaces de usuario independiente de computación, a nivel del dominio y de las tareas que se deben llevar a cabo en el dominio. A partir de ese modelo es posible derivar otros modelos de interfaces de

usuario más especializados (bajando el nivel de abstracción), para modalidades de interacción, y para plataformas de computación, a través de transformaciones de modelo a modelo. Estas transformaciones se realizan sucesivamente hasta llegar a la obtención del código fuente de la interfaz de usuario (transformación de modelo a código). Las transformaciones se realizan, idealmente, de forma automática.

Por otra parte, en la literatura existen también modelos de interfaces de usuario que están basados en patrones de interfaces de usuario. Estos modelos basados en patrones también pueden ser utilizados en procesos de desarrollo de interfaces de usuario dirigidos por modelos. En particular, el modelo de interfaces de usuario basado en patrones definido en [Molina et al., 2002], constituye el Modelo de Presentación de OO-Method [Pastor and Molina, 2007], un método de desarrollo de software dirigido por modelos.

La generación de las interfaces de usuario con un método MDA ayuda a disminuir la variabilidad que introduce la poca o mucha habilidad y experiencia de un programador que desarrolla interfaces de usuario, así como también introduce un proceso formal y automatizable en el cual es posible introducir buenas prácticas de generación de interfaces de usuario, las que se aplicarán automáticamente al utilizar el método, y redundarán en beneficio de la calidad de las interfaces de usuario generadas.

1.1. Planteamiento del Problema

La Figura 1.1, presentada en [Szekely, 1996; da Silva, 2000], constituye un diagrama tradicional para describir el proceso de desarrollo de interfaces de usuario en ambientes de desarrollo dirigidos por modelos. Ilustra un proceso de diseño de interfaces de usuario totalmente automatizado. En la misma, se ve que una herramienta de diseño abstracto puede generar el modelo de interfaz de usuario abstracto a partir de los modelos de tareas y dominio, utilizando una base de datos de conocimientos de diseño que provee la información requerida durante el proceso de diseño de la interfaz de usuario. Además, una herramienta de diseño concreto puede generar el modelo de interfaz de usuario concreto a partir del abstracto utilizando una base de datos de guías de presentación. Los conocimientos de diseño y las guías de presentación no forman parte de los modelos de interfaces de usuario, son externos.

Ya en el trabajo de [da Silva, 2000] se ha reconocido que a la vez que el diseño automatizado descrito facilita el trabajo de los desarrolladores de interfaces de usuario, también crea un nuevo problema: ¿cómo podrán, los desarrolladores de interfaces de usuario, interferir en el proceso automatizado de diseño de interfaces de usuario cuando tengan que diseñar interfaces de usuario con características diferentes a las proveídas por los conocimientos de diseño y guías de presentación de las bases de datos?

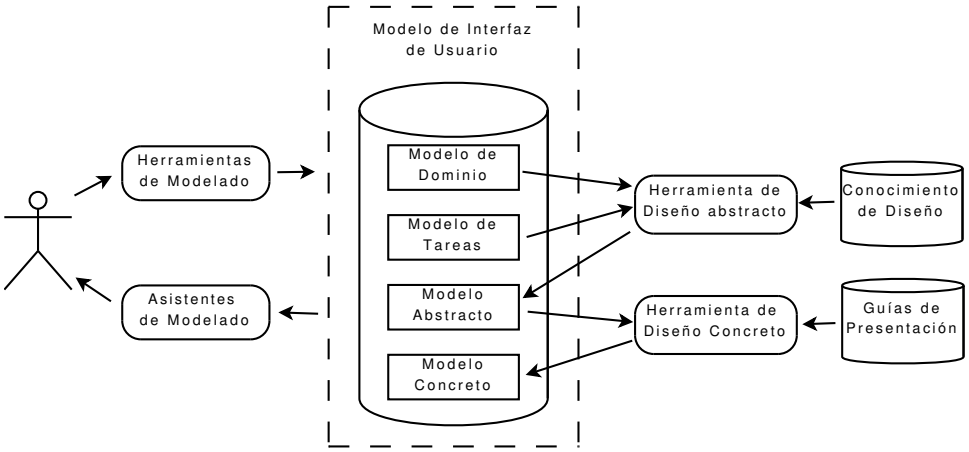


Figura 1.1: Diseño de interfaces de usuario en ambientes de desarrollo de interfaces de usuario dirigidos por modelos

Este problema se agudiza al considerar que muchas aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos tienen la lógica de transformaciones de modelos implícitas en las herramientas que dan soporte a dichas transformaciones ([Griffiths et al., 2001; Mori et al., 2004], entre otras). Es decir, los conocimientos de diseño y guías de presentación que se utilizan en las transformaciones no son externas.

Existen también aproximaciones que tienen la lógica de las transformaciones de modelos explícitas, por ejemplo en modelos de mapeos y de transformaciones. En estos casos, los diseñadores deben definir nuevas opciones de transformación cuando las necesitan, y ese es un proceso complejo, más orientado a especialistas en transformaciones de modelo que a diseñadores de interfaces de usuario. Por lo tanto, en general, se necesitan mecanismos para especificar la lógica de las transformaciones de modelos que sean más flexibles y abordables por los diseñadores. De hecho, recientemente, en [Vanderdonckt, 2008], se identifican las promesas, sucesos, fallos y desafíos de la ingeniería dirigida por modelos de interfaces de usuario, y uno de los desafíos identificados tiene que ver con la necesidad de disponer de mecanismos de transformación más flexibles. Esto se debe a que, mientras que una persona puede quedar satisfecha con la interfaz de usuario generada con una herramienta de desarrollo de interfaces de usuario dirigido por modelos, siempre habrá otra que quiera cambiar la interfaz resultante. Varias razones pueden explicar esta necesidad, como por ejemplo, la necesidad de mantenerse conforme a guías corporativas de estilo, o la necesidad de satisfacer preferencias de los usuarios que no fueron consideradas en el desarrollo automatizado de la interfaz de usuario.

Es por esto que, siempre se tiene la necesidad de modificar, por aquí o por allí,

las interfaces de usuario resultantes de un proceso de generación automatizado. A este proceso se le conoce como *embellecimiento* (*beautification*) [Pederiva et al., 2007]. El *embellecimiento* se refiere a cualquier modificación que se hace a la interfaz de usuario a fin de adecuarla a los requisitos del usuario final, y no se limita a modificaciones de estilo (como colores, tipos de letras, etc.), puede implicar modificaciones más serias.

El trabajo recopilatorio y comparativo que se hace en [Schaefer, 2007] de herramientas de transformación de modelos de interfaces de usuario, muestra claramente, que la mayoría de estas herramientas de transformación dan poco o nada de soporte al *embellecimiento* [Vanderdonckt, 2008].

El *embellecimiento*, típicamente, se aborda con modificaciones manuales sobre el código generado. Estas modificaciones manuales pueden llevar a inconvenientes tales como problemas para entender y modificar el código generado, inconsistencias entre la interfaz de usuario y su modelo, y la puesta en peligro de características de calidad garantizadas por una construcción que sigue un esquema dirigido por modelos, como la ausencia de errores en el código. Además, si se tiene una interfaz de usuario que ha sido generada automáticamente a partir de un modelo y luego se modifica manualmente, y más adelante se cambia su modelo, la interfaz de usuario regenerada cambiará de acuerdo al nuevo modelo, pero se perderán los cambios manuales hechos previamente.

Se puede decir entonces, que el soporte completo a los requisitos de interfaces de usuario es un problema que no está del todo resuelto en el ámbito del desarrollo de interfaces de usuario dirigido por modelos.

Esto plantea un interesante trabajo en el que los intereses de investigación de la comunidad de la Ingeniería del Software (SE - *Software Engineering*), y la de la Interacción Persona-Ordenador (HCI - *Human-Computer Interaction*) convergen.

1.2. Propuesta

A fin de abordar el problema descrito, este trabajo propone una aproximación de Plantillas de Transformación.

Una Plantilla de Transformación está formada por Parámetros que especifican detalles de estructura, disposición y estilo de las interfaces de usuario.

Las Plantillas de Transformación son una entrada a las herramientas de transformaciones de modelos de interfaces de usuario, y parametrizan el proceso de generación de interfaces de usuario en un proceso de desarrollo dirigido por modelos. Con esto, se logra añadir flexibilidad a las transformaciones de modelos de interfaces de usuario más abstractos a modelos de interfaces de usuario más concretos, o a código.

Además, las Plantillas de Transformación externalizan la lógica de las transformaciones, haciendo que los diseñadores las puedan adaptar, de una

manera sencilla, a las características del proyecto en el que están trabajando, y las puedan reutilizar en otros proyectos con características similares.

Los requisitos relacionados a las interfaces de usuario que no están soportados en un proceso de desarrollo dirigido por modelos y que se repiten frecuentemente pueden ser representados con un tipo de Parámetro, que al ser implementado, dará soporte automatizado el requisito de usuario en cuestión y disminuirá la necesidad de realizar modificaciones manuales en el código de las interfaces de usuario generadas con un proceso de desarrollo dirigido por modelos. Los tipos de Parámetros representan también a conocimientos de diseño y guías de presentación (ver Figura 1.1). En la aproximación de Plantillas de Transformación, los tipos de Parámetros que pueden ser utilizados, no son fijos, se pueden definir los que se necesiten, ampliando el conjunto de acuerdo a los requisitos de los usuarios.

Las Plantillas de Transformación proveen mecanismos flexibles para que los diseñadores especifiquen las características de las interfaces de usuario a ser generadas, y además, dan soporte a las transformaciones que involucran a modelos de interfaces de usuario basados en patrones de interfaces de usuario.

Las Plantillas de Transformación han sido claramente caracterizadas a nivel de meta-meta modelo y de meta modelo. Además, se las ha relacionado a un meta-meta modelo y a un meta modelo simplificados de interfaces de usuario, a fin de definir como utilizar la aproximación de Plantillas de Transformación con diversas aproximaciones de modelado de interfaces de usuario.

También se ha implementado un prototipo de editor de plantillas de Transformación.

Como aplicación práctica de la aproximación de Plantillas de Transformación, se ha definido un Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method.

1.3. Contexto de Investigación

Este trabajo ha sido desarrollado en el contexto del grupo de investigación OO-Method, perteneciente al Centro de Investigación en Métodos de Producción de Software (ProS) de la Universidad Politécnica de Valencia. Las principales áreas de investigación del Centro ProS son Ingeniería de Requisitos, Ingeniería Web, Interacción Persona-Ordenador, Calidad de Software, y Desarrollo de Sistemas Ubicuos, todas ellas abordadas desde la perspectiva del modelado conceptual.

Además, se ha contado con la valiosa colaboración del Profesor Jean Vanderdonckt de la Université catholique de Louvain, Bélgica, quien tiene una vasta experiencia en el área del desarrollo de interfaces de usuario.

El trabajo realizado ha sido posible gracias a convenios de investigación que tiene el centro ProS con la empresa CARE Technologies, y gracias a proyectos

financiados por el Centro de Investigación Científica y Tecnológica de España (CICYT) y por el Ministerio de Industria, Turismo y Comercio de España (MITyC). A continuación se listan los convenios que han soportado este trabajo de investigación:

- PISA: Producción Industrial de Software en Ambientes MDA. PROFIT de MITyC referenciado como FIT-340000-2007-110 desde 2007 a 2008.
- SESAMO: Construcción de Servicios Software a partir de Modelos. Proyecto CICYT referenciado como TIN2007-62894 desde 2008 a 2010.

Este trabajo también tiene una perspectiva industrial, ya que forma parte de convenios de investigación que tiene el Centro ProS con la empresa CARE Technologies, la que está interesada en propuestas relacionadas a mejorar los procesos de desarrollo de interfaces de usuario, y de software en general, dirigidos por modelos.

1.4. Estructura de la Tesis

Esta tesis está estructurada en 7 capítulos, los que se describen brevemente a continuación.

- Capítulo 1. Introducción

  El presente capítulo introduce el contexto tecnológico en el que se desarrolla este trabajo, y expone las motivaciones y el problema que dan origen a este trabajo. Luego, se describe brevemente la propuesta para abordar el problema. Además, se describe el contexto de investigación en el que se ha desarrollado este trabajo. Finalmente, se describe la estructura de este trabajo.

- Capítulo 2. Estado del Arte

  En este capítulo se describen las herramientas que utilizan una aproximación basada en transformaciones de modelos para generar interfaces de usuario. En primer lugar, se hace una clasificación de los trabajos en base a si las reglas de transformación están implícitas o explicitas en las herramientas. También se describen los principales lenguajes de transformación de modelos (que no son específicos de las interfaces de usuarios), pero que han sido utilizados con éxito en el contexto de interfaces de usuarios. También una sección está dedicada a las aproximaciones basdas en plantillas. Finalmente, se resumen las comparaciones de nuestra propuestas con respectos a estos trabajos relaciones.

- Capítulo 3. Fundamentos

  En este capítulo se presentan los trabajos en los cuales se ha basado el desarrollo del presente trabajo. Primero se describe brevemente a MDA (*Model-Driven Architecture*). Luego, se presentan los conceptos fundamentales del Marco de Referencia Cameleon, en especial los de contexto de uso y niveles de abstracción de modelos de interfaces de usuario, ya que esos contextos de uso son utilizados en la aproximación de Plantillas de Transformación, y los niveles de abstracción de los modelos describen los tipos de modelos de interfaces de usuario que son considerados por la aproximación de Plantillas de Transformación. A continuación se introduce UsiXML, un lenguaje para la descripción de interfaces de usuario a varios niveles de abstracción. Y finalmente se introducen OO-Method, un método para el desarrollo de software dirigido por modelos, y su Modelo de Presentación. Los modelos de UsiXML y el Modelo de Presentación de OO-Method son ejemplos de los modelos de interfaces de usuario que la aproximación de Plantillas de Transformación pretende abarcar. Además, el modelo de contexto de UsiXML es adoptado como modelo de contexto en este trabajo.

- Capítulo 4. Plantillas de Transformación

  Este capítulo presenta a las Plantillas de Transformación y a todos sus conceptos relacionados. Las Plantillas de Transformación son preparadas para un contexto de uso y se pueden utilizar con diversas aproximaciones de desarrollo de interfaces de usuario dirigidos por modelos. Por lo tanto, además de caracterizar a las Plantillas de Transformación a nivel de meta-meta modelo y de meta modelo, este capítulo también presenta el meta modelo de contexto considerado en este trabajo, y el meta-meta modelo y meta modelo de interfaces de usuario (simplificados) que representan a las aproximaciones de desarrollo de interfaces de usuario dirigidos por modelos con los que se pueden usar las Plantillas de Transformación.

- Capítulo 5. Automatización de Plantillas de Transformación

  En este capítulo se esquematizan las herramientas necesarias para implementar la aproximación de Plantillas de Transformación. Además se presenta un prototipo de editor de Plantillas de Transformación.

- Capítulo 6. Plantillas de Transformación para OO-Method

  En este capítulo se propone una aplicación práctica de la aproximación de Plantillas de Transformación al Modelo de Presentación de OO-Method. Como parte de esta puesta en práctica se ha elaborado un Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method, y en este capítulo se presentan, a modo de ejemplo, algunos de ellos.

- Capítulo 7. Conclusiones

  En este capítulo se presentan las contribuciones de este trabajo junto con una discusión sobre las mismas, los trabajos futuros identificados y las publicaciones realizadas.

- Anexo A. Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method

  En este anexo se presenta el Catálogo de Tipos de Parámetros que ha sido elaborado para el Modelo de Presentación de OO-Method.

2

Estado del Arte

Este trabajo presenta una aproximación de Plantillas de Transformación que añade flexibilidad a las transformaciones de modelos de interfaces de usuario.

La flexibilidad se logra al externalizar las reglas de transformación de las herramientas que las procesan para que puedan ser adaptadas según las características del proyecto en el que se está trabajando. Como adicional, esto permite que las reglas se puedan reutilizar en otros proyectos similares.

Existen diversos enfoques en las transformaciones de modelos de interfaces de usuarios, y para cada uno de ellos una comunidad de investigadores.

Por una parte, existen herramientas software que utilizan una aproximación basada en transformaciones de modelos para generar una interfaz de usuario. En este sentido clasificamos los trabajos en base a si las reglas de transformación están implícitas o explícitas con respecto a las herramientas que realizan las transformaciones. A su vez, también se presentan algunos de los lenguajes de transformación de modelos (que no son específicos de las interfaces de usuarios), pero que han sido utilizados con éxito en el contexto de interfaces de usuarios.

Otro grupo de trabajos corresponden con las aproximaciones basadas en plantillas.

Por otra parte, existen herramientas software que dan soporte a una aproximación basada en plantillas.

A continuación, se describe un trabajo que define opciones de diseño aplicados en el contexto de aplicaciones Web multimodales.

Finalmente, se resumen las comparaciones de nuestra propuesta con respecto a los trabajos relacionados aquí descritos.

2.1. Aproximaciones basadas en transformaciones de modelos

Entre las herramientas software que utilizan una aproximación basada en transformaciones de modelos para generar una interfaz de usuario, se distinguen dos categorías: las que tienen las reglas de transformación implícitas en las

herramientas que realizan las transformaciones, y las que hacen a las reglas explícitas.

2.1.1. Aproximaciones con reglas de transformación implícitas

Las herramientas que utilizan una aproximación basada en transformaciones de modelos para generar una interfaz de usuario que tienen las reglas de transformación implícitas en el código de los compiladores de modelo a modelo o de modelo a código, no permiten que las reglas de transformación sean adaptadas a las características de un proyecto en concreto y, por lo tanto, generan la interfaz de usuario final siempre de la misma manera. Esto podría llegar a ser un inconveniente para un usuario final en el momento de adaptar la interfaz a sus necesidades. En este tipo de herramientas tenemos: Teallach, TERESA, MOBI-D y OLIVANOVA. A continuación describimos cada una de ellas.

Teallach

El proyecto Teallach [Griffiths et al., 2001; Barclay et al., 2003], adapta técnicas de desarrollo de interfaces de usuarios basadas en modelos para la creación sistemática de interfaces de usuarios de aplicaciones de base de datos orientadas a objetos. En Teallach se construyen tres tipos de modelos: el modelo de dominio, el modelo de tareas y el modelo de presentación. Este último a su vez se divide en concreto y abstracto.

El modelo de dominio corresponde con el esquema específico de la aplicación. Se definen las relaciones entre los objetos del dominio. La construcción de este modelo es automática utilizando una API específica de la base de datos.

El modelo de tareas describe el flujo de la aplicación, y asocia una tarea a las operaciones de los elementos del modelo de dominio.

Un modelo de presentación describe la apariencia y la navegación de la interfaz de usuario. Tiene dos niveles, un nivel abstracto y uno concreto. El nivel concreto son las librerías propias del lenguaje (Java Swing widget). En el nivel abstracto, define categorías abstractas que luego son asociadas a los widget del nivel concreto.

La interacción entre modelos está fija en el código, y su forma de trabajo consiste en asociar elemento de un modelo con elemento de otro. De esta manera, se genera el código de la interfaz de usuario. La interfaces desarrolladas están escritas en Java utilizando la librería de Java Swing. La herramienta es flexible para el diseñador, permite la incorporación de nuevos elementos gráficos, y es posible variar entre los distintos elementos antes de generar el código final.

TERESA

TERESA [Mori et al., 2004; Berti et al., 2004a; Berti et al., 2004b] es una herramienta basada en transformaciones que soporta el diseño de una aplicación interactiva a diferente niveles de abstracción y genera la interfaz de usuario concreta para varios tipos de plataforma. El objetivo que los autores se plantean es dar soporte al desarrollo a través del uso de abstracciones y de esta manera evitar tratar con detalles de bajo nivel.

Las transformaciones son del estilo *top down*, en donde los diseñadores comienzan con una descripción lógica y luego se mueven a representaciones más concretas, así hasta llegar al nivel del sistema.

Para el uso de la herramienta se debe seguir un método compuesto de cuatro pasos, en donde en cada paso se obtiene un modelo que sirve como entrada del siguiente paso. Los pasos son:

- Modelar las tareas de alto nivel de una aplicación multiplataforma. En este paso se obtiene el modelo de tareas del sistema. El objeto es direccionar los posibles contextos de uso de las plataformas involucradas.
- Refinar el modelo para las plataformas específicas consideradas. De esta manera se obtiene una interfaz de usuario abstracta compuesta de un conjunto de componentes abstractos.
- Transformación. El objeto de este paso es pasar del modelo abstracto al modelo concreto de interfaz. Esta fase es dependiente de la plataforma y se consideran las propiedades específicas de cada una.
- La generación de código. Este paso es completamente automático y depende de la plataforma seleccionada. Para la incorporación de nuevas plataformas sólo es necesario implementar la transformación de este paso.

Los modelos son representados usando un lenguaje propio basado en XML llamado *Teresa XML*, y las transformaciones están implícitas en la herramienta. Para mas información visitar http://giove.isti.cnr.it/teresa.html.

MOBI-D

MOBI-D (Model-Based Interface Designer) [Puerta and Eisenstein, 1999a], es un ambiente que soporta el diseño centrado en el usuario a través del desarrollo de interfaces basadas en modelos. En MOBI-D, una serie de modelos declarativos tales como tareas de usuario, diálogos y presentación, están relacionados a una representación formal de un diseño de interfaz.

El modelo de interfaz es una colección de los elementos relevantes de una interfaz de usuario. Cada elemento es representado con un modelo que describe

sus características, estos modelos son: modelo de tareas de usuario, modelo de dominio, modelo de usuario, modelo de presentación y modelo de diálogo. De la representación de estos modelos, se define una función de mapeo que permite pasar de cada uno de los modelo a una representación concreta del mismo.

El ciclo de desarrollo para obtener la interfaz de usuario es interactivo con el usuario. El ciclo se divide en cuatro partes: (1) definir las tareas del usuario, (2) modelar las tareas y definir el modelo de dominio, (3) integrar las tareas y el dominio, y (4) diseñar la presentación.

La transformación de modelos de MOBI-D está integrada en el código de la herramienta, y la generación de código es directa a partir de los modelos.

OLIVANOVA

OLIVANOVA (http://www.care-t.com) es una herramienta comercial que da soporte al método de desarrollo de software OO-Method. OO-Method y OLIVANOVA son presentados con más detalle en la Sección 3.4.

OO-Method tiene un Modelo de Presentación que permite modelar una interfaz de usuario de manera abstracta utilizando los patrones de interfaces de usuario definidos en [Molina et al., 2002].

A partir de la especificación abstracta y basada en patrones del Modelo de Presentación, OLIVANOVA genera el código de la interfaz de usuario. Las reglas de transformación están implícitas en el compilador.

2.1.2. Aproximaciones con reglas de transformación explícitas

TransformiXML

TransformiXML [Limbourg et al., 2004], es una aplicación Java que permite definir, almacenar, manipular y ejecutar producciones contenidas en gramáticas de grafos a fin de dar soporte a transformaciones de grafos (transformaciones de modelo a modelo). TransformiXML es la herramienta utilizada por UsiXML para la transformación de sus modelos.

Los modelos de UsiXML están basados en grafos y de esta manera los mapeos de modelos están especificados con transformaciones de grafos, los cuales consisten de un conjunto de reglas de transformación. Para externalizan los mapeos entre los elementos de los diversos modelos de UsiXML y las transformaciones correspondientes, se utiliza un *mappingModel* y un *transformationModel*.

- *mappingModel*: es un modelo que contiene una serie de mapeos entre modelos o elementos de modelos. Un modelo de mapeos recoge un conjunto de relaciones entre modelos que están conectados de manera semántica.

Expresa reificaciones, abstracciones y traslaciones. Adicionalmente, se pueden definir otros mapeos [Limbourg, 2004].

- *transformationModel*: es un modelo que contiene descripciones de transformaciones de modelos tal como se definen en el CRF (reificaciones, abstracciones y traslaciones). Para formalizar las transformaciones de modelo a modelo se utilizan técnicas de transformaciones de grafos.

2.1.3. Lenguajes de transformación de modelos, aplicados en modelos de interfaces de usuario

Diferentes lenguajes de transformación de modelos se han utilizados con éxito en transformación de modelos de interfaces de usuarios. Una recopilación y comparación de ellos se encuentra en [Schaefer, 2007]. En nuestro trabajo se describen los lenguajes más relevantes, estos son: ATL [Jouault and Kurtev, 2005], UIML Peers [Abrams et al., 1999], RDL/TT [Schaefer et al., 2002a] y XSLT [Clark, 1999].

ATL

ATL (ATLAS Transformation Language) es un lenguaje usado para la transformación de modelos [Jouault and Kurtev, 2005]. ATL no es una herramienta para el desarrollo de interfaces de usuario, pero ha sido utilizada con éxito en ingeniería dirigida por modelos de interfaces de usuarios plásticas [Sottet et al., 2005].

ATL sigue una aproximación híbrida en el sentido de que un usuario puede seleccionar si usar ATL puramente declarativo o emplear características imperativas. El aspecto declarativo está soportado por una aproximación de reglas de mapeo compuestas de una parte izquierda o patrón fuente (*source pattern*) y una parte derecha o patrón objetivo (*target pattern*). Un patrón fuente es descrito a través de un conjunto de tipos fuente y una guarda con una expresión OCL. El patrón objetivo es construido de la misma manera especificando un conjunto de tipos desde el meta-modelo y un conjunto de enlaces (*bindings*) que son usados para iniciar las características de los tipos.

El aspecto declarativo es más directo de especificar pero es más duro en reglas complejas, por ello, ATL permite adicionar a las reglas bloques con constructores imperativos o llamadas a procedimientos externos.

ATL cuenta con un motor de transformación ATL, un IDE (*integrated development environment*) para Eclipse y un depurador ATL.

UIML Peers

El UIML (User Interface Markup Language) [Abrams et al., 1999], es un meta-lenguaje que permite a los diseñadores describir la interfaz de usuario en términos genéricos y usar una descripción de estilo para mapear la interfaz de usuario a varios sistemas operativos, lenguajes o dispositivos.

Un documento UIML contiene tres partes diferentes: (1) una descripción de la interfaz de usuario, (2) una sección que describe el mapeo de las interfaces de usuario a entidades externas, y (3) una plantilla que permite el uso de elementos escritos previamente. En UIML una interfaz de usuario es descrita como un conjunto de elementos de la interfaz con los cuales el usuario final interactúa. Por cada parte, un estilo de presentación es dado (posiciones, tipos de letra, colores), junto con su contenido (texto, imágenes, etc.), posibles eventos de entrada del usuario y las acciones resultantes.

La descripción de la interface es *renderizada* acorde a las especificaciones de los componentes de presentación, y se comunica con la lógica de la aplicación por medio de definiciones lógicas. De esta manera, el nuevo documento puede ser visualizado en una aplicación cliente (por ejemplo un navegador) o en otro lenguaje (como WML o HMTL).

RDL/TT

RDL/TT (Rule Description Language for Tree Transformation) [Schaefer et al., 2002a] ha evolucionado desde un lenguaje de dominio específico para adaptar contenido Web a diferentes contenidos, a un lenguaje de transformación para múltiples propósitos, incluyendo transformaciones dependientes de contexto de descripciones de interfaces de usuario basadas en XML. RDL/TT emplea una sintaxis orientada a Java para definir las reglas de transformación, las cuales operan sobre árboles DOM de documentos XML. La idea es definir simple patrones de búsqueda basados en los nombres de los etiquetas (*tag*) o colecciones de tags con reglas de reestructuración complejas sobre las asociaciones encontradas.

Un propiedad de RDL/TT es el uso de variables, las cuales pueden transmitir información de contexto que permita diferentes flujos de operaciones de transformación para diferentes preferencias, plataformas y contextos de uso. La reglas de transformación son especificadas de una manera imperativa y proveen distintas estructuras de control (ramificación, ciclos y llamadas a funciones de transformación específicas). Este conjunto de funciones de transformación pueden ser extendidas.

En la práctica RDL/TT a sido usado en adaptaciones basadas en contexto de aplicaciones Web [Schaefer et al., 2002b] y en formatos de interfaces de usuario genéricas [Plomp et al., 2002].

XSLT

XSLT (Extensible Stylesheet Language Transformations) [Clark, 1999] o Transformaciones XSL es un estándar de la organización W3C que presenta una forma de transformar documentos XML a otros, e incluso a formatos que no son XML. Las hojas de estilo XSLT realizan la transformación del documento utilizando una o varias reglas de una plantilla. Estas reglas de la plantilla unidas al documento fuente a transformar alimentan un procesador de XSLT, el que realiza las transformaciones deseadas poniendo el resultado en un archivo de salida, o, como en el caso de una página web, las hace directamente en un dispositivo de presentación tal como el navegador.

Una definición XSLT define un conjunto de reglas (plantillas) que asocian patrones con plantillas. Cada regla consiste de un patrón de asociación (*matching pattern*) una plantilla. La expresiones de patrón de asociación son definidas por un subconjunto del lenguaje XPath [DeRose and Clark, 1999] y son evaluadas con respecto al nodo corriente o el nodo raíz.

El proceso de asociación considera el nombre de los nodos, atributos y posición, y el resultado es un conjunto de nodos que pude ser usado como parámetro para las plantillas o como base para futuras asociaciones. Durante la ejecución, XSLT provee variables y parámetros que pueden ser pasados entre las reglas.

2.2. Aproximaciones basadas en plantillas

Existen herramientas software que dan soporte a una aproximación basada en plantillas para la especificación de interfaces de usuario. Típicamente están restringidas solo a modificar los valores de algunas propiedades de *widgets*.

Genova

Genova [Arisholm et al., 1998], es una herramienta CASE (*Computer Aided Software Engineering*) para la generación rápidas de prototipos de interfaces de usuario basada en modelos de objetos usando UML. El objetivo de Genova es obtener un prototipo de la interfaz de usuario en etapas tempranas del desarrollo de un producto y, de esta manera, mejorar la interacción con el usuario y la comprensión del sistema. En Genova se define una plantilla con valores predefinidos para propiedades de interfaces de usuario, como color y tipos de letras. La plantilla se aplica luego a una interfaz de usuario.

La transformación desde el modelo de objetos al prototipo de interfaz de usuario se realiza en tres pasos:

- Primer paso: Un conjunto de objetos asociados es seleccionado desde el modelo de objeto usando lo que llaman un selector de objetos.

- Segundo paso: La generación del modelo de diálogo requiere un objeto seleccionado y una guía de estilo de diálogo como entrada. La guía de estilo de diálogo define las reglas para la creación del modelo de diálogo desde el objeto seleccionado.

- Tercer paso: La transformación automática desde el modelo de diálogo a un prototipo de interfaz de usuario requiere la especificación de una plantilla de diálogo predefinida. Esta plantilla define el estilo y el diseño para cada componente de la interfaz. En el diseño se determina como los grupos de componentes son posicionados uno con respecto del otro, mientras que en el estilo se define el tipo de letra y el color para cada uno de los componentes.

Navegadores con soporte a CSS

Hojas de Estilo en Cascada (*Cascading Style Sheets*) [Bos et al., 2007], es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre el estilo y formato de sus documentos.

CSS se utiliza para dar estilo a documentos en la Web, separando el contenido de la presentación. Los Estilos definen la forma de mostrar los elementos.

CSS funciona a base de reglas, es decir, declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas reglas aplicadas a un documento HTML o XML. La regla tiene dos partes: un selector y la declaración. A su vez la declaración está compuesta por una propiedad y el valor que se le asigne. El selector funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. La declaración es la parte de la regla que establece cuál será el efecto, allí se definen los valores de las propiedades del elemento.

Los tipos de selectores posibles son: universal, de tipo, descendente, hijo, adyacente, atributo, id, seudo-elemento y seudo-clase.

Valores de las propiedades pueden ser: enteros, reales, medidas, porcentaje, url, contador, color y cadena.

Subconjuntos de estos tipos de selectores y de valores posibles de las propiedades han sido adoptados en este trabajo.

2.3. Opciones de diseño de aplicaciones multimodales

En [Stanciulescu and Vanderdonckt, 2008], se presentan opciones de diseño para interfaces de usuario gráficas que están descritas acorde a cinco parámetros específicos: presentación de la sub-tarea, navegación de la sub-tarea, concretización de la navegación y el control, concretización de la navegación y concretización del control.

De esta manera se define un espacio de diseño para aplicaciones Web multimodales basadas en opciones de diseño teniendo múltiples valores de diseño, los cuales pueden ser reutilizados.

Independientemente de esto, estas opciones de diseño tienen que ser traducidas en parámetros que dirijan las reglas de diseño codificadas como gramáticas de grafos. Cada gramática de grafo está compuesta por reglas de transformación de grafos, de esta manera cada regla se ejecuta sobre el grafo correspondiente de la especificación UsiXML de la interfaz de usuario bajo estudio. Un motor de transformación podría procesar estas reglas.

2.4. Comparaciones con respecto a nuestra propuesta

En esta Sección se presentan algunas comparaciones de la propuesta de Plantillas de Transformación con respecto a las aproximaciones que se han considerado en este Capítulo.

Con respecto a las aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos que tienen la lógica de las transformaciones implícitas en las herramientas de transformación, esta propuesta tiene la ventaja de que externaliza esa lógica, haciéndola adaptable según las características del proyecto en el que se está trabajando. A la vez, permite que la lógica apropiada sea reutilizada en otros proyectos de características similares.

Con respecto a las aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos que tienen la lógica de las transformaciones explícitas en modelos de mapeos y de transformaciones, y que utilizan lenguajes de transformaciones de modelos, se puede decir que en el caso de que el diseñador de la interfaz de usuario tenga que manipular estos modelos y lenguajes de transformaciones, esta propuesta tiene la ventaja de que la especificación de los detalles de estructura, disposición y estilo de las interfaces de usuario se realiza de una manera que es bastante sencilla al compararla con el complejo proceso de definir transformaciones de modelos. Con nuestra propuesta, aunque de fondo el compilador de modelos utilice modelos de mapeos y de transformaciones y

lenguajes de transformaciones de modelos, esto queda oculto para el diseñador de las interfaces de usuario, que solo debe manipular Plantillas de Transformación.

Con respecto a otras aproximaciones que también utilizan plantillas para definir detalles de las interfaces de usuario, esta aproximación tiene la ventaja de que no está sujeta a un método de desarrollo de interfaces de usuario en particular, sino que puede ser utilizada en diversas aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos, pudiendo, también guiar las transformaciones hacia diversos tipos de plataformas de cómputo, y para diversos tipos de usuarios. Además, los parámetros de una Plantilla de Transformación no solo pueden modificar propiedades de *widgets*, sino que también afectar a la estructura y disposición de los elementos de la interfaz de usuario. Además, en nuestra aproximación, los parámetros no están restringidos a lenguajes de programación o de marcado, y nuevos parámetros pueden ser definidos cuando se necesitan.

Por otra parte, esta aproximación también es aplicable a las transformaciones de modelos en las que interviene un modelo de interfaces de usuario basado en patrones de interfaces de usuario.

3

Fundamentos

En este capítulo se presentan los trabajos en los cuales se ha basado el desarrollo del presente trabajo.

La aproximación de Plantillas de Transformación pretende añadir flexibilidad en las transformaciones de modelos de interfaces de usuario que son conformes a una arquitectura MDA, la cual es brevemente descrita en la Sección 3.1

El Marco de Referencia Cameleon constituye una referencia para la especificación y el entendimiento unificado de las diversas aproximaciones que utilizan un enfoque dirigido por modelos para el desarrollo de interfaces de usuario que dan soporte a múltiples contextos de uso. Las definiciones que el Marco de Referencia Cameleon hace del contexto de uso de las aplicaciones, de los niveles de abstracción utilizados para modelar interfaces de usuario, y de los tipos de transformaciones posibles, son utilizados en este trabajo, y se presentan en la Sección 3.2.

UsiXML es un lenguaje de descripción de interfaces de usuario conforme al Marco de Referencia Cameleon. Una vista del modelo de contexto de UsiXML es adoptado como modelo de contexto de las Plantillas de Transformación. Los modelos de interfaces de usuario abstractos, concretos, entre otros, de UsiXML constituyen ejemplos de los modelos con los que la aproximación de Plantillas de Transformación pretende trabajar. UsiXML se presenta en la Sección 3.3.

Finalmente, la Sección 3.4 presenta un método de desarrollo de sistemas llamado OO-Method. OO-Method tiene un Modelo de Presentación basado en patrones de interfaces de usuario con el que las Plantillas de Transformación también pretenden funcionar. Una aplicación práctica de la aproximación de Plantillas de Transformación se ha iniciado utilizando este Modelo. Específicamente, se ha elaborado un Catálogo de Tipos de Parámetros para ser utilizado con el Modelo de Presentación de OO-Method. Este Catálogo se presenta en el Capítulo 6 y el Apéndice A. Además, un prototipo de editor de Plantillas de Transformación ha sido implementado utilizando OO-Method (ver Sección 5.1.1).

3.1. Arquitectura Dirigida por Modelos

En 1997, el *Object Management Group* (OMG - www.omg.org) ha lanzado una iniciativa llamada Arquitectura Dirigida por Modelos (MDA - *Model Driven Architecture*) [Mukerji and Miller, 2003] a fin de dar soporte al desarrollo de sistemas software interactivos de gran escala y complejos. MDA propone una arquitectura estandarizada con la cual:

- Los sistemas pueden evolucionar fácilmente a fin de dar soporte a los constantemente variables requisitos de usuario.
- Las tecnologías antiguas, actuales y nuevas se pueden armonizar.
- La lógica de negocios se puede mantener constante o puede evolucionar independientemente de los cambios tecnológicos.
- Los sistemas legados se pueden integrar y unificar con los nuevos sistemas.

En la propuesta MDA, se utilizan modelos en todas las etapas del desarrollo hasta llegar a una plataforma destino para la cual se proveen el código fuente, los archivos de configuración, etc.

MDA ha sido aplicada a muchos dominios y ha sido integrada con un amplio abanico de tecnologías de computación, incluyendo el campo de las interfaces de usuario.

MDA recomienda un método sistemático para guiar el ciclo de vida del desarrollo a fin de garantizar cierto grado de calidad del sistema software resultante. Cuatro principios subyacen la visión de OMG acerca de MDA [Mellor et al., 2004]:

1. Los modelos se expresan con una notación bien formada y unificada y conforman la piedra angular para entender los sistemas software a nivel de escala empresarial. La semántica de los modelos se basa en meta modelos.
2. La construcción de sistemas software se organiza con un conjunto de modelos a los que se aplica una serie de transformaciones entre modelos. Las transformaciones se organizan en un marco arquitectónico de capas de transformación: las transformaciones de modelo a modelo dan soporte a cualquier cambio entre modelos mientras que las transformaciones de modelo a código están asociadas a la producción de código, automatizada o no.
3. La descripción de modelos con una estructura formal, como son los meta modelos, facilita la integración significativa y las transformaciones entre modelos, y es la base para la automatización a través de software.

4. La aceptación y la adopción de esta aproximación guiada por modelos requiere estándares industriales para proveer franqueza a los consumidores y fomentar la competición entre vendedores.

3.2. El Marco de Referencia Cameleon

El Marco de Referencia Cameleon (CRF - *Cameleon Reference Framework*) [Calvary et al., 2003] es una referencia para la clasificación de interfaces de usuario que dan soporte a múltiples contextos de uso en el área de la computación sensible al contexto.

En este marco, un contexto de uso se descompone en las siguientes tres facetas:

- Los usuarios finales del sistema interactivo
- La plataforma hardware y software sobre la que se ejecuta el sistema, incluyendo a los dispositivos que se utilizan para la interacción con el sistema
- El ambiente físico en el que la interacción con el sistema toma lugar

Una interfaz de usuario sensible al contexto tiene algunas capacidades de sensibilidad al contexto y de reacción ante los cambios del mismo.

Por otro lado, el marco estructura el ciclo de vida del desarrollo de las interfaces de usuario en cuatro niveles de abstracción que se corresponden con pasos en el proceso de desarrollo:

- Tareas y Conceptos
- Interfaz de Usuario Abstracta (AUI - *Abstract User Interface*)
- Interfaz de Usuario Concreta (CUI - *Concrete User Interface*)
- Interfaz de Usuario Final (FUI - *Final User Interface*)

Entre estos niveles se pueden dar tres tipos de transformaciones:

- Abstracción: proceso con el que se obtienen artefactos que son más abstractos que los artefactos que sirven de entrada al proceso.
- Reificación: proceso con el que se obtienen artefactos que son más concretos que los artefactos que sirven de entrada al proceso. La abstracción y la reificación son opuestos.
- Traslación: proceso con el que se obtienen artefactos para un contexto de uso en particular a partir de artefactos de una fase de desarrollo igual pero de un contexto de uso distinto.

La Figura 3.1 ilustra una versión simplificada del CRF en la que el proceso de desarrollo de interfaces de usuario se estructura para dos contextos de uso (A y B) y con los cuatro pasos o niveles correspondientes (Tareas y Conceptos, AUI, CUI y FUI). Las flechas indican las abstracciones, reificaciones y traslaciones posibles.

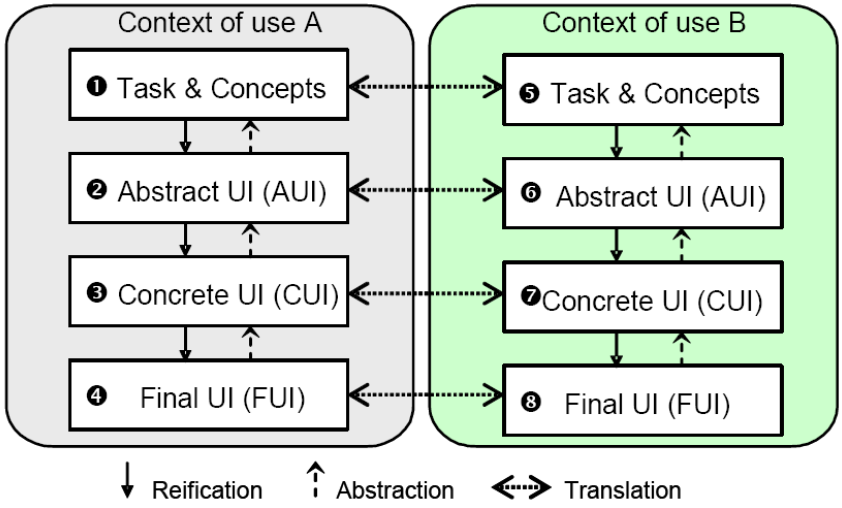


Figura 3.1: Versión simplificada del Marco de Referencia Cameleon

Considerando los niveles de abstracción y los tipos de transformaciones que el CRF contempla, se puede notar que no es obligatorio iniciar el proceso de desarrollo de una interfaz de usuario en el nivel de Tareas y Conceptos. De hecho, el proceso podría iniciarse a partir de cualquiera de los niveles de abstracción. Por ejemplo, una opción válida sería iniciar el proceso con la FUI 4 del contexto de uso A (ver Figura 3.1). Utilizando ingeniería reversa la FUI 4 se puede abstraer a la CUI 3, la que a su vez se puede abstraer a la AUI 2. Utilizando traslación, la AUI 2 se puede transformar a la AUI 6 teniendo en cuenta las restricciones y oportunidades del contexto de uso B. A partir de allí, la AUI 6 se puede reificar a la CUI 7, la que a su vez se puede reificar a la FIU 8.

Por otro lado, más que prescribir maneras o métodos para abordar las distintas fases del desarrollo de las interfaces de usuario sensibles al contexto, el CRF pretende proveer una referencia para el entendimiento unificado de los mismos. De hecho, en [Calvary et al., 2003], varios modelos, métodos y herramientas relacionados al desarrollo de interfaces de usuario sensibles al contexto [Bastide et al., 2003; Penner and Steinmetz, 2003; Luyten et al., 2003; Paternò and Santoro, 2003] se expresan en términos del marco, lo que resulta en un entendimiento claro, sobre todo, al hacer comparaciones entre unos y otros.

Por último, el marco introduce, define y ejemplifica la noción de interfaces de usuario plásticas. Estas interfaces dan soporte a algunas adaptaciones ante

cambios del contexto de uso pero se cuidan de preservar un conjunto predefinido de propiedades de usabilidad.

En el contexto de este trabajo, son especialmente relevantes las definiciones de contexto y de los niveles de abstracción que se corresponden con los pasos del proceso de desarrollo de las interfaces de usuario. Por ese motivo, estos conceptos son expuestos con más detalle a continuación.

3.2.1. Contextos de uso de un sistema interactivo

Como se ha mencionado previamente, en el CRF un contexto de uso incluye a los usuarios finales del sistema interactivo, a la plataforma hardware y software sobre la que se ejecuta el sistema, junto con los dispositivos que se utilizan para la interacción, y al ambiente físico en el que la interacción toma lugar.

A continuación, las tres facetas del contexto se describen con más detalle, tal como están descritas en el CRF.

- El *usuario* representa a un usuario estereotipo del sistema, el que puede ser descrito con un conjunto de valores que caracterizan sus capacidades de percepción, cognoscitivas y de acción [Card et al., 1983]. En particular, se pueden expresar las discapacidades de percepción, cognoscitivas y de acción del usuario a fin de seleccionar las mejores opciones para la representación y manipulación del sistema interactivo.

- La *plataforma* se modela en términos de recursos, los que a su vez, determinan la manera en la que la información se computa, transmite, representa y manipula por los usuarios. Algunos ejemplos de recursos son el tamaño de la memoria, el ancho de banda de la red y los dispositivos de interacción de entrada y de salida. Los recursos determinan la elección de un conjunto de modalidades de entrada y de salida, y para cada modalidad, la cantidad de información disponible. Típicamente, el tamaño de la pantalla es un factor determinante para el diseño de páginas web. En el caso de DynaWall [Streitz et al., 1999] la plataforma incluye tres pantallas táctiles de igual tamaño montadas una al lado de la otra (ver Figura 3.2). Con las superficies aumentadas de Rekimoto [Rekimoto and Saitoh, 1999] los usuarios pueden usar proyecciones sobre mesas y paredes como extensiones de espacio continuas de sus ordenadores portátiles (ver Figura 3.3). Las superficies aumentadas se construyen a partir de un conjunto heterogéneo de pantallas cuya topología puede variar. En Pebbles, los PDAs se pueden usar como dispositivos de entrada para controlar la información que se despliega en un tablero electrónico montado en la pared [Myers et al., 1998]. En Kidpad [Benford et al., 2000], los objetos gráficos, que se despliegan en una única pantalla, se pueden manipular con un número variable de ratones.

Estos ejemplos muestran que la plataforma no siempre se limita a un único ordenador personal, sino que abarca todos los recursos de cómputo y de interacción que están disponibles en un momento determinado para realizar un conjunto de tareas relacionadas.

Figura 3.2: Muro electrónico interactivo DynaWall

- El *ambiente* se refiere a "el conjunto de objetos, personas y eventos que son secundarios para la actividad actual pero que podrían tener un impacto sobre el sistema y/o el comportamiento de los usuarios, ya sea ahora o en el futuro" [Coutaz and Rey, 2002]. De acuerdo a esta definición, el ambiente puede abarcar el mundo entero. En la práctica, los límites son fijados por los analistas del dominio quienes deben decidir cuáles serán las entidades relevantes para el caso actual. Para ello se deberán observar las prácticas de los usuarios [Beyer and Holtzblatt, 1999; Cockton et al., 1996; Dey et al., 2001; Johnson et al., 1993; Lim and Long, 1997], así como también se deberán considerar restricciones técnicas. Por ejemplo, la condición de iluminación es un aspecto a considerar cuando puede influenciar en la robustez de un sistema de seguimiento basado en la visión [Crowley et al., 2000].

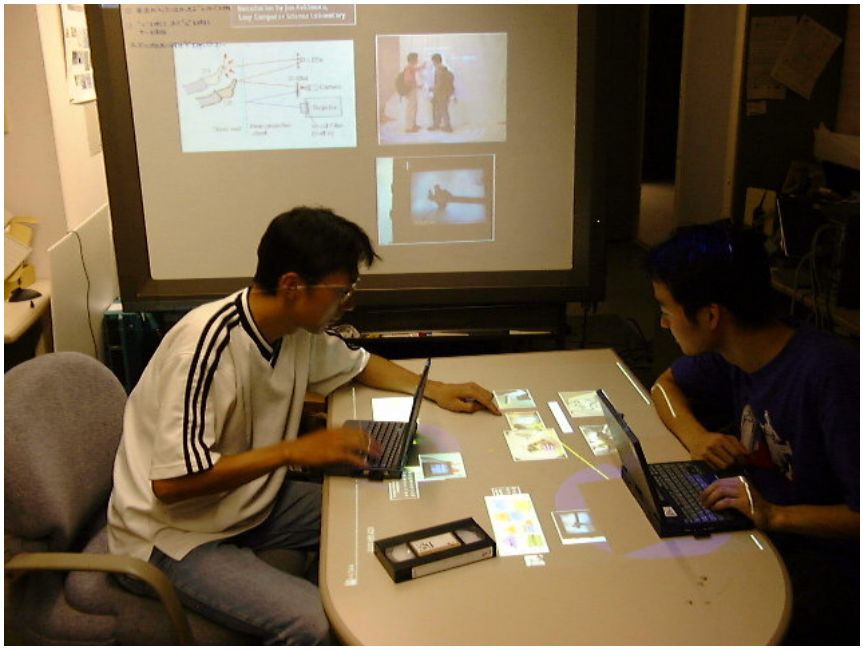


Figura 3.3: Superficies aumentadas de Rekimoto

3.2.2. Niveles de abstracción o pasos del proceso de desarrollo de interfaces de usuario

El CRF define cuatro pasos o fases en el desarrollo de las interfaces de usuario. Cada uno de estos pasos corresponde a un nivel de abstracción distinto. Y en cada uno de los niveles, la interfaz de usuario se representa con un modelo que corresponde al nivel de abstracción.

A continuación se describe cada fase o nivel.

- Interfaz de Usuario Final (FUI - *Final User Interface*): es la interfaz de usuario operativa, es decir una interfaz de usuario ejecutándose en una plataforma de cómputo particular, ya sea utilizando interpretación, como por ejemplo en un navegador web, o bien luego de un proceso de compilación.

- Interfaz de Usuario Concreta (CUI - *Concrete User Interface*): concretiza una AUI en Objetos de Interacción Concretos (CIOs - *Concrete Interaction Objects*) [Vanderdonckt and Bodart, 1993] para un contexto de uso dado a fin de definir la disposición de los *widgets* y la navegación entre las interfaces. Abstrae una FUI a una definición de interfaz de usuario que es independiente de las plataformas de cómputo. Aunque una CUI hace explícito el *look & feel* final de una FUI, sigue siendo solo un *mock-up*

que se ejecuta solo en un ambiente en particular. Una CUI también se puede considerar como una reificación de una AUI del siguiente nivel de abstracción y como una abstracción de la FUI con respecto a la plataforma.

- Interfaz de Usuario Abstracta (AUI - *Abstract User Interface*): define contenedores y componentes individuales abstractos [Limbourg et al., 2004; Limbourg, 2004], dos tipos de Objetos de Interacción Abstractos (AIOs - *Abstract Interaction Objects*) [Vanderdonckt and Bodart, 1993], agrupando sub-tareas de acuerdo a varios criterios (por ejemplo, patrones estructurales del modelo de tareas, identificación de relaciones semánticas, análisis de carga cognoscitiva). También define un esquema de navegación entre contenedores, y selecciona componentes individuales abstractos para cada concepto de manera que sean independientes de cualquier modalidad. Una AUI abstrae una CUI a una definición de interfaz de usuario que es independiente de cualquier modalidad de interacción (por ejemplo: interacción gráfica; interacción vocal; síntesis y reconocimiento de discurso; interacción basada en vídeo; realidad virtual, aumentada o mixta). Una AUI también puede ser considerada como una expresión canónica de la representación de los conceptos del dominio y las tareas en una manera que es independiente de cualquier modalidad de interacción. En este nivel, la interfaz de usuario básicamente consiste de definiciones de entradas y salidas, junto con acciones que deben ser realizadas sobre la información.

- Tareas y Conceptos: describe las varias tareas que los usuarios deben realizar y los conceptos del dominio involucrados en la realización de las tareas.

3.3. UsiXML

Un Lenguaje de Descripción de Interfaces de Usuario (UIDL - *User Interface Description Language*) especifica los distintos aspectos de una interfaz de usuario y de conceptos relacionados, y permite que los diseñadores y desarrolladores definan, comuniquen, intercambien, y compartan fragmentos de especificaciones. Además, los UIDLs hacen posible que las especificaciones de interfaces de usuario sean manipuladas por herramientas.

UsiXML (*USer Interface eXtensible Markup Language* - http://www.usixml.org) [Limbourg et al., 2004; Vanderdonckt, 2005] es un UIDL. Más específicamente, UsiXML es un lenguaje de marcas conforme a XML que describe una interfaz de usuario para múltiples contextos de uso tales como Interfaces de Usuario basadas en Caracteres, Interfaces de Usuario Gráficas (GUIs - *Graphical User Interfaces*), Interfaces de Usuario Auditivas y Vocales, Realidad Virtual, e Interfaces de Usuario Multimodales.

En este trabajo se ha utilizado la versión 1.8.0 de UsiXML, lanzada en febrero del 2007, y a la fecha de hoy (diciembre del 2008), la última disponible.

UsiXML es un lenguaje conforme al CRF y tiene las siguientes características:

- UsiXML está pensado para ser utilizado por personas que no son desarrolladores de interfaces de usuario, tales como analistas, diseñadores, expertos en factores humanos, líderes de proyectos y programadores novatos.
- UsiXML también puede ser utilizado por diseñadores y desarrolladores experimentados.
- Gracias a UsiXML, personas que no son desarrolladores pueden dar forma o definir la interfaz de usuario de cualquier aplicación interactiva nueva, especificándola y describiéndola con el UIDL, sin necesitar la experiencia en programación que típicamente se necesita para trabajar con otros lenguajes de marcas o de programación.
- UsiXML es un UIDL declarativo que captura la esencia de lo que una interfaz de usuario debe ser independientemente de las características físicas.
- UsiXML describe en un nivel de abstracción elevado, los elementos que constituyen la interfaz de usuario de una aplicación: *widgets*, controles, contenedores, modalidades, y técnicas de interacción.
- La interfaz de usuario de cualquier aplicación conforme a UsiXML se puede ejecutar en todos los *toolkits* que la implementen: compiladores e intérpretes.
- UsiXML da soporte a la independencia de dispositivos: una interfaz de usuario puede ser descrita de manera autónoma respecto a los dispositivos que se utilizan para las interacciones (ratón, pantalla, teclado, sistema de reconocimiento de vos, etc.). En caso de necesidad, una referencia a un dispositivo en particular puede ser incorporada.
- UsiXML da soporte a la independencia de plataforma: una interfaz de usuario puede ser descrita de manera autónoma respecto a las varias plataformas de cómputo existentes (teléfonos móviles, Pocket PC, Tablet PC, ordenadores portátiles, ordenadores de escritorio, etc.). En caso de necesidad, una referencia a una plataforma de cómputo en particular puede ser incorporada.
- UsiXML da soporte a la independencia de modalidad: una interfaz de usuario puede ser descrita de manera autónoma respecto a la modalidad de interacción (interacción gráfica, interacción vocal, interacción 3D,

interacción con realidad virtual, etc.). En caso de necesidad, una referencia a una modalidad en particular puede ser incorporada.

- UsiXML permite reutilizar elementos previamente descritos en interfaces de usuario anteriores a fin de componer una interfaz de usuario en aplicaciones nuevas.

Por otra parte, UsiXML no cubre todas las características de todos los tipos de interfaces de usuario:

- UsiXML no pretende introducir otro lenguaje más de implementación de interfaces de usuario. En lugar de eso, propone la integración de algunos de estos formatos: cHTML, WML, HTML, XHTML, VoiceXML, VRML, Java, C++, ... El soporte a la transformación de UsiXML a tales formatos depende de la implementación subyacente.

- UsiXML no describe los detalles de bajo nivel de los elementos involucrados en las distintas modalidades, tales como atributos o eventos de los sistemas operativos.

- UsiXML no puede ser directamente ejecutado: depende de una implementación de motor de terceros.

- UsiXML no pretende dar soporte a todos los atributos, eventos y primitivas de todos los *widgets* que existen en los *toolkits*. En lugar de eso, pretende dar soporte a un sub-conjunto común, representativo y significativo.

3.3.1. Modelos

UsiXML permite definir los siguientes modelos para describir una interfaz de usuario (ver Figura 3.4) [Limbourg et al., 2004; Limbourg, 2004; Vanderdonckt, 2005]:

- taskModel: es un modelo que describe las tareas interactivas tal como son vistas por el usuario final que interactúa con el sistema. Un modelo de tareas representa una descomposición de tareas en sub-tareas conectadas con relaciones. La relación de descomposición es la relación utilizada para expresar la jerarquía de tareas, mientras que las relaciones temporales expresan las restricciones de tiempo entre las sub-tareas de una misma tarea padre.

- domainModel: es una descripción de las clases o tipos de objetos que son manipulados por el usuario cuando interactúa con un sistema.

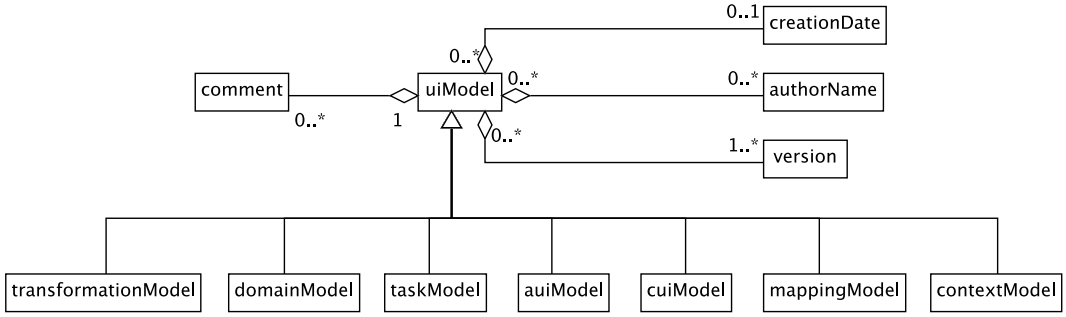


Figura 3.4: Colección de modelos para especificar una interfaz de usuario con UsiXML

- mappingModel: es un modelo que contiene una serie de mapeos entre modelos o elementos de modelos. Un modelo de mapeos recoge un conjunto de relaciones entre modelos que están conectados de manera semántica. Expresa reificaciones, abstracciones y traslaciones. Adicionalmente, se definen y se pueden definir otros mapeos [Limbourg, 2004].

- contextModel: es un modelo que describe los tres aspectos de un contexto de uso en el que un usuario final está llevando a cabo una tarea interactiva con una plataforma de cómputo específica en un ambiente dado. En consecuencia, un modelo de contexto consiste de un modelo de usuarios, un modelo de plataformas y un modelo de ambiente.

- auiModel: es el modelo que describe a la interfaz de usuario a nivel abstracto.

- cuiModel: es el modelo que describe a la interfaz de usuario a nivel concreto.

- transformationModel: es un modelo que contiene descripciones de transformaciones de modelos tal como se definen en el CRF (reificaciones, abstracciones y traslaciones). Para formalizar las transformaciones de modelo a modelo se utilizan técnicas de transformaciones de grafos.

- uiModel: es la superclase superior que contiene las características comunes compartidas por todos los modelos que componen una interfaz de usuario. Un uiModel puede estar constituido por un conjunto con cualquier número de modelos componentes, y en cualquier orden. Por ejemplo, un uiModel puede estar constituido por un modelo de tareas, un modelo de dominio, un modelo de interfaz de usuario abstracto, un modelo de interfaz de usuario concreto, un modelo de mapeos y un modelo de contexto. Un uiModel no necesita incluir un modelo de cada tipo de modelo, y además, podría incluir más de un modelo de un determinado tipo.

La semántica de UsiXML está definida de acuerdo a los modelos previamente explicados y sus relaciones en términos de un diagrama de clases UML, el cual conforma el meta modelo del UIDL. Todos los diagramas de clases se mantienen en Rational Rose. Además, a partir de los diagramas de clases, utilizando una serie de transformaciones sistemáticas, se han definido los XML Schemas correspondientes. Los diagramas de clases y XML Schemas están disponibles en http://www.usixml.org/index.php?mod=pages&id=5.

3.3.2. Método

La metodología de ingeniería de interfaces de usuario de UsiXML está acorde con los cuatro principios de MDA (presentados en la Sección 3.1) de la siguiente manera:

1. Todos los modelos involucrados se expresan en UsiXML, un UIDL bien formado basado en XML Schema. La semántica de los modelos UsiXML está basada en meta modelos expresados en términos de diagramas de clases UML, a partir de los cuales, se derivan las definiciones de XML Schemas.

2. Todas las transformaciones de modelo a modelo se especifican en UsiXML, y esto permite utilizar solo un UIDL a través de todo el ciclo de vida del desarrollo. Las transformaciones de modelo a código están soportadas por herramientas apropiadas (ver Sección 3.3.3) que producen código para la plataforma o contexto de uso destino. Para el caso de la ingeniería reversa, las transformaciones de código a modelo se realizan fundamentalmente con reglas de derivación basadas en un mapeo entre el meta modelo del lenguaje origen (por ejemplo, HTML, WML, VoiceXML, etc.) y el meta modelo del lenguaje destino (en este caso UsiXML).

3. Todas las transformaciones se definen explícitamente basadas en una serie de relaciones semánticas predefinidas y un conjunto de tres primitivas (abstracción, reificación y traslación). El modelo de transformación puede, a su vez, contener reglas de transformación.

4. El último principio, el proceso de estandarización, está iniciado, pero aún no finalizado.

Además de la adherencia a los principios básicos de MDA, la metodología de ingeniería de interfaces de usuario de UsiXML clasifica a los modelos involucrados de manera similar. La Figura 3.5 representa gráficamente la distribución de modelos de acuerdo al paradigma MDA y su contraparte UsiXML.

Los modelos de tareas y de dominio se consideran como el modelo independiente de computación (CIM) ya que la definición de los mismos es independiente de una implementación de sistema interactivo.

El modelo independiente de la plataforma (PIM) se interpreta como el modelo de interfaz de usuario abstracto (AUI) de UsiXML en el sentido de que este modelo es independiente de cualquier modalidad de interacción. A este nivel, aún no se determina si la interfaz de usuario será gráfica, vocal, virtual o multimodal.

El modelo específico de plataforma (PSM) se interpreta como el modelo de interfaz de usuario concreto (CUI) de UsiXML en el sentido de que este modelo es independiente de cualquier vocabulario de lenguaje de marcas o de programación. A este nivel, ya se ha determinado el tipo de modalidad de interacción que será utilizado, pero aún no se determina la plataforma de computación física sobre la que se ejecutará la interfaz de usuario. Por este motivo se puede considerar que el CUI no es, en realidad, específico de plataforma. Sólo algunos aspectos de la plataforma destino se seleccionan, y la plataforma misma se modela con el modelo de plataformas. En contraste con otras arquitecturas compatibles con MDA, UsiXML puede, o desplegar la interfaz de usuario directamente (por interpretación), o generar código de manera automática (por generación). Este código generado deberá luego ser compilado, enlazado con el resto de la aplicación y ejecutado. Por lo tanto, no hay una transformación de modelo a código *per se*. En lugar de ello, distintas herramientas producen diferentes resultados a partir de especificaciones en UsiXML. Para el caso de las transformaciones de modelo a modelo, técnicas de transformaciones de grafos son utilizadas a lo largo del ciclo de vida del desarrollo para mantener consistencia.

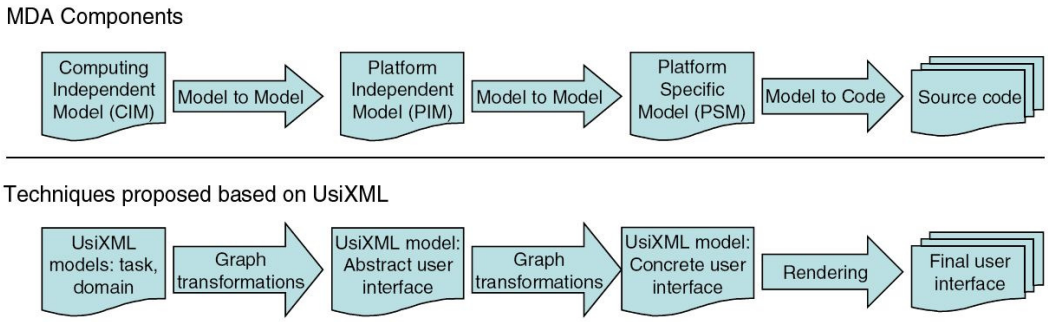


Figura 3.5: Distribución de los modelos de UsiXML de acuerdo a la clasificación MDA

3.3.3. Herramientas

La Figura 3.5 representa uno de los caminos posibles para hacer ingeniería de interfaces de usuario, iniciando el trabajo con los modelos de tareas y dominio y continuando hacia adelante hasta obtener la interfaz de usuario final (*fordward*

engineering). Sin embargo, la metodología de ingeniería de interfaces de usuario de UsiXML no se restringe a este único camino de desarrollo. Otros caminos podrían ser utilizados. La Figura 3.6 generaliza el ciclo de vida del desarrollo para un contexto de uso a la vez.

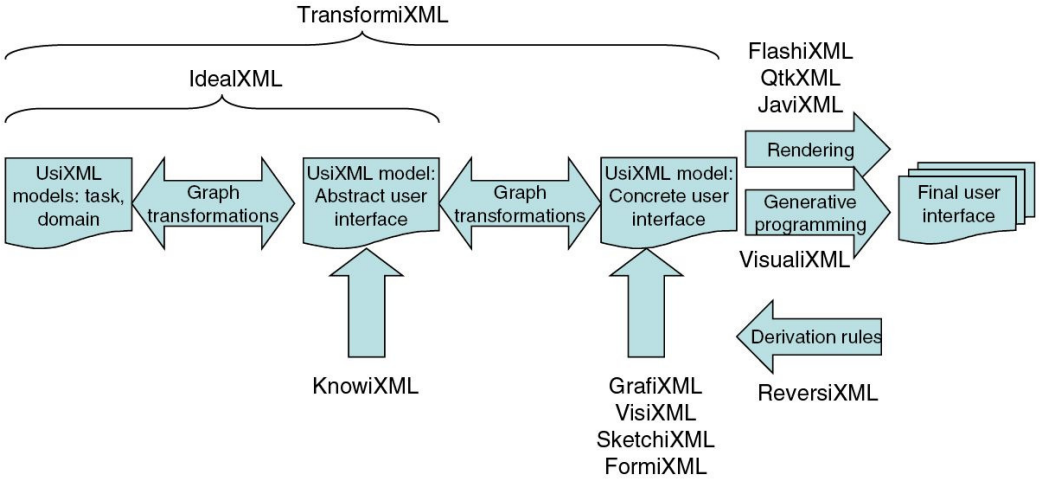


Figura 3.6: Conjunto de herramientas de UsiXML estructuradas de acuerdo a la clasificación MDA

A fin de dar soporte a la aplicación de un camino de desarrollo en particular, un conjunto de herramientas ha sido desarrollado y sigue expandiéndose (ver http://www.usixml.org/ para más información). Las herramientas más significativas de este conjunto son las siguientes (ver Figura 3.6):

- TransformiXML es una aplicación Java que permite definir, almacenar, manipular y ejecutar producciones contenidas en gramáticas de grafos a fin de dar soporte a transformaciones de grafos (transformaciones de modelo a modelo).
- IdealXML [Montero and López-Jaquero, 2007] es una aplicación Java que cuenta con un editor gráfico para los modelos de tareas, de dominio y abstracto. Además permite establecer mapeos entre estos modelos, ya sea manualmente (por manipulación directa), o semiautomáticamente (utilizando TransformiXML).
- KnowiXML [Furtado et al., 2004] es un sistema experto basado en Protégé que, a partir de un modelo de tareas y de dominio, genera automáticamente varias AUIs para varios contextos.

- GrafiXML [Limbourg et al., 2004] es el editor de alta fidelidad más elaborado de UsiXML para la edición de CUI, modelo de contexto y relaciones entre ellos. Es capaz de generar automáticamente código de interfaz de usuario en HTML, XHTML, XUL y Java utilizando una serie de *plug-ins*.
- VisiXML es un *plug-in* para Microsoft Visio que permite dibujar, con fidelidad media, interfaces de usuario gráficas. Además, permite exportar la definición de la interfaz de usuario en UsiXML a nivel de CUI, a fin de que sea editada con GrafiXML u otro editor.
- SketchiXML [Coyette and Vanderdonckt, 2005] es una herramienta Java de baja fidelidad que sirve para esquematizar una interfaz de usuario para múltiples usuarios, plataformas y contextos de uso.
- FormiXML es un editor Java dedicado a formularios interactivos con un sistema de técnicas de copiado/pegado que dan soporte a la reutilización de componentes. Genera automáticamente el código completo de interfaces de usuario para Java/Swing.
- FlashiXML abre un archivo de una especificación CUI de UsiXML y lo representa en Flash, QtkXML en el ambiente Tcl/Tk, y JaviXML para Java.
- VisualiXML [Schlee and Vanderdonckt, 2004] personaliza una interfaz de usuario y la produce para Visual C++ V6.0 utilizando técnicas de programación generativa.
- ReversiXML [Bouillon et al., 2004] abre un archivo HTML y aplicando ingeniería reversa lo convierte a UsiXML en los siguientes niveles: AUI y CUI.

3.4. El Método de Desarrollo de Software OO-Method

OO-Method es un método orientado a objetos que permite la generación automática de software a partir de modelos conceptuales [Pastor and Molina, 2007].

Este método pone en práctica la tecnología MDA, separando claramente la lógica de las aplicaciones de la plataforma tecnológica en que serán desarrolladas esas aplicaciones. Para realizar esto, OO-Method provee un conjunto de modelos conceptuales que permiten abstraer la lógica de las aplicaciones de la plataforma de implementación, centrando el proceso productivo en el espacio del problema,

que define qué es lo que hace el sistema, en lugar del espacio de la solución, que define cómo se implementa la solución.

El proceso de producción de software utilizando OO-Method comprende cuatro grandes fases, que en términos de MDA, respectivamente, corresponden al desarrollo de modelos independientes de computación (CIM - *Computing Independent Model*), modelos independientes de plataforma (PIM - *Platform Independent Model*), modelos específicos de plataforma (PSM - *Platform Specific Model*) y modelos de implementación (IM - *Implementation Model*).

En la fase de desarrollo de modelos independientes de computación se encuentra el Modelo de Requisitos [Insfrán et al., 2002; Díaz et al., 2005], el cual es realizado por el analista del sistema con la ayuda de la herramienta RETO (http://reto.dsic.upv.es/reto/).

En la fase de desarrollo de los modelos independientes de plataforma se encuentra el Modelo Conceptual, el cual es desarrollado por el analista del sistema con la ayuda de herramientas CASE que facilitan el modelado, el intercambio de modelos, la validación de los modelos y la posterior generación de la documentación asociada al sistema que se está modelando. Estas herramientas han sido desarrolladas por la empresa CARE Technologies y en conjunto son denominadas OLIVANOVA Suite (http://www.care-t.com/).

En la fase de desarrollo de modelos específicos de plataforma se encuentra el Modelo de Ejecución, que se realiza de manera completamente automática a partir del Modelo Conceptual y que está soportado por un compilador de modelos desarrollado por la empresa CARE Technologies denominado *The Programming Machine*.

La fase de desarrollo del modelo de implementación también se realiza de manera completamente automática por el compilador de modelos *The Programming Machine*.

La Figura 3.7 (extraída de [Marín, 2008]) esquematiza el proceso de producción de software de OO-Method, mostrando claramente los diferentes modelos que lo componen y las transformaciones entre modelos. Esta figura también refleja las correspondencias entre los modelos OO-Method y los modelos de la arquitectura MDA.

Como se puede observar en la Figura 3.7, el modelo de requisitos se compone de técnicas como la definición de la Misión del Sistema, el Árbol de Refinamiento de Funciones, los Diagramas de Casos de Uso y los Diagramas de Secuencia. Con la utilización de estas técnicas es posible definir los requisitos funcionales del sistema, que mediante transformaciones de modelos permiten conseguir las bases de los diagramas que componen el modelo conceptual.

El modelo conceptual (ver Figura 3.7) captura las propiedades estáticas y dinámicas del sistema utilizando un Modelo de Objetos, un Modelo Dinámico y un Modelo Funcional. Además, el modelo conceptual permite la especificación de las interfaces de usuario de manera abstracta a través del Modelo de Presentación.

Una vez que el analista ha especificado estos cuatro modelos, el modelo conceptual tiene todos los detalles necesarios para la generación automática de la aplicación de software. La definición completa de los elementos que componen el modelo conceptual de OO-Method está descrita en [Pastor and Molina, 2007].

El modelo de ejecución permite especificar la forma en que será utilizado el sistema de manera abstracta, es decir, permite especificar cómo los usuarios accederán al sistema, qué acciones tendrán disponibles cuando accedan al sistema, y qué secuencia de acciones debe realizar para interactuar con el sistema. Para esto el modelo de ejecución utiliza un lenguaje formal denominado OASIS [Pastor et al., 1992], que permite la especificación exacta de las características de implementación de los objetos del sistema de acuerdo a la forma en que serán utilizados estos objetos.

A partir del modelo de ejecución se obtiene automáticamente el modelo de implementación, que permite la transformación desde el espacio del problema (representado por el modelo conceptual) hacia el espacio de la solución (el producto de software correspondiente). El modelo de implementación realiza correspondencias entre las primitivas conceptuales y sus representaciones de software para un entorno de desarrollo específico, por ejemplo Java o C♯. Cabe destacar que el compilador de modelos OO-Method genera aplicaciones en arquitecturas de tres capas: una capa para el componente cliente, que contiene la interfaz gráfica; una capa para el componente servidor, que contiene las reglas de negocio y las conexiones a la base de datos; y una capa para el componente base de datos, que contiene los aspectos de persistencia de las aplicaciones.

La siguiente Sección presenta más detalles del Modelo de Presentación de OO-Method.

3.4.1. El Modelo de Presentación de OO-Method

El Modelo de Presentación de OO-Method permite especificar de manera abstracta la interfaz gráfica de usuario que tendrá la aplicación modelada. Para realizar esto, el modelo de presentación cuenta con un conjunto de patrones de presentación que están organizados jerárquicamente en tres niveles: estructura de acceso, unidades de interacción y elementos básicos. Estos patrones de presentación están definidos en [Molina et al., 2002]. A continuación se los describe brevemente.

Los patrones del primer nivel permiten especificar la estructura de acceso al sistema. En este nivel se especifica el menú que tendrá cada agente (rol) del sistema mediante un árbol jerárquico de acciones (*Hierarchy Action Tree* - HAT).

El segundo nivel permite especificar las unidades de interacción del sistema. Las unidades de interacción pueden ser:

- Unidades de Interacción de Servicio (UIS): Representan la interacción entre

el usuario de la aplicación y la ejecución de un servicio del sistema.

- Unidades de Interacción de Instancia (UIS): Representa la interacción entre el usuario de la aplicación y la información de un objeto específico.
- Unidades de Interacción de Población (UIP): Representa la interacción entre el usuario de la aplicación y la información de varios objetos de una clase.
- Unidades de Interacción de Maestro Detalle (UIMD): Representa la interacción entre el usuario de la aplicación y un grupo de unidades de interacción del sistema, siguiendo una estructura de maestro-detalle.

El tercer nivel permite especificar los elementos que pueden componer las unidades de interacción del sistema. Estos elementos pueden ser:

- Entrada: Este elemento permite capturar los datos que deben ser ingresados por el usuario del sistema.
- Selección definida: Este elemento permite al usuario seleccionar un valor desde una colección de datos existente.
- Agrupador de argumentos: Este elemento permite definir la forma en que los argumentos de los servicios serán presentados al usuario.
- Dependencia de argumentos: Permite definir relaciones de dependencias entre los valores o estado de un argumento de entrada de un servicio y los valores o estado de otros argumentos de entrada del mismo servicio. La dependencia de argumentos utiliza reglas de tipo ECA (Evento - Condición - Acción) para cada argumento de entrada de un servicio, que tienen la siguiente semántica: cuando un evento de interfaz ocurre en un argumento de entrada (por ejemplo cuando el usuario ingresa un valor) se realiza una acción si se cumple una determinada condición.
- Precarga de argumentos: Este elemento permite definir un conjunto de objetos que pueden ser seleccionados como argumentos de entrada de un servicio.
- Filtro: Este elemento permite definir una condición de selección sobre una población de objetos de una clase. El filtro puede tener variables dato-valuadas y variables objeto-valuadas que pueden tener definido un valor por defecto, una población asociada para seleccionar los valores de las variables objeto-valuadas y precarga de los valores de las variables objeto-valuadas.
- Criterio de ordenación: Este elemento permite definir el orden de los objetos presentados en una población, considerando el orden ascendente/descendente sobre los valores de los atributos de los objetos presentes en la población.

- Conjunto de visualización: Permite especificar las propiedades de una clase que serán visibles al usuario, y el orden en que se mostrarán dichas propiedades.
- Navegación: Permite la visualización de información de los objetos de las clases relacionadas a un objeto de una clase.
- Acción: Permite la ejecución de los servicios definidos en las clases sobre un objeto de la clase.
- Filtrado Navegacional: Permite acotar la navegación hacia los objetos de las clases relacionadas a un objeto según una condición de filtro.
- Inicialización de Argumentos: Permite asignar un valor inicial a los argumentos de entrada de un servicio que se accede directamente desde otro servicio.
- Navegación Condicional: Permite navegar a las unidades de interacción luego de la ejecución de un servicio que haya terminado de manera exitosa o fallida. Para seleccionar la unidad de interacción a la que se navegará, es necesario especificar una condición que debe cumplirse cuando se haya ejecutado el servicio.

Los elementos del tercer nivel se pueden agrupar en elementos componentes individuales (que no se pueden descomponer en patrones de presentación) y elementos contenedores intermedios (que se pueden descomponer en patrones de presentación). Los elementos componentes individuales son: entrada, selección definida, agrupador de elementos, dependencia de argumentos, criterio de ordenación, conjunto de visualización, filtrado navegacional, e inicialización de argumentos. Los elementos contenedores intermedios son: filtro, navegación, acción, precarga de argumentos, y navegación condicional. Los elementos del segundo nivel también son elementos contenedores.

La Figura 3.8 (extraída de [Marín, 2008]) esquematiza los tres niveles jerárquicos del modelo de presentación y los patrones que se pueden utilizar en cada nivel, representando en color celeste los elementos que pueden contener a otros y en color amarillo los elementos componentes individuales.

Como se puede ver en la Figura 3.8, una unidad de interacción de servicio puede tener patrones de entrada, selección definida, agrupador de argumentos, dependencia de argumentos, precarga de argumentos y navegación condicional. A su vez, la precarga de argumentos puede tener asociada una unidad de interacción de población y un criterio de ordenación. Asimismo, la navegación condicional puede tener asociadas unidades de interacción de servicio, de instancia, de población y de maestro detalle, y también inicialización de argumentos y filtrado navegacional.

Las unidades de interacción de población pueden tener patrones de filtro, criterio de ordenación, conjunto de visualización, navegación y acción (ver Figura 3.8). Los filtros pueden tener precarga en las variables de filtro, las acciones pueden tener asociadas unidades de interacción de instancia y filtrado navegacional, y por último, las navegaciones pueden tener asociadas unidades de interacción de servicio, de instancia, de población y de maestro detalle, y también filtrado navegacional.

Las unidades de interacción de instancia pueden tener patrones de conjunto de visualización, navegación y acción (ver Figura 3.8). Al igual que en las unidades de interacción de población, los patrones de navegación y acción están compuestos por otros patrones de presentación.

En último lugar, la Figura 3.8 muestra que las unidades de interacción de maestros detalle pueden tener patrones de unidad de interacción de instancia y de población en su parte maestra; y unidades de interacción de instancia, de población, o de maestro detalle en su parte de detalle.

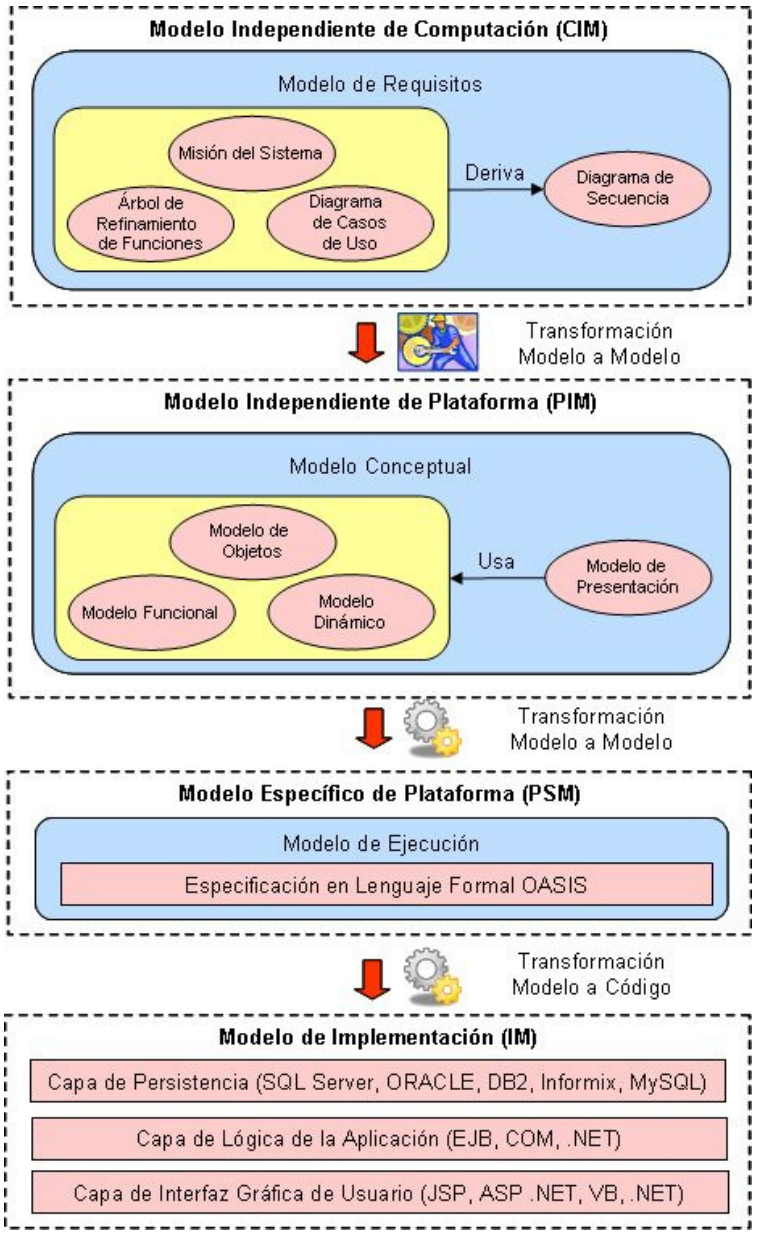


Figura 3.7: Proceso de Desarrollo de Software utilizando OO-Method

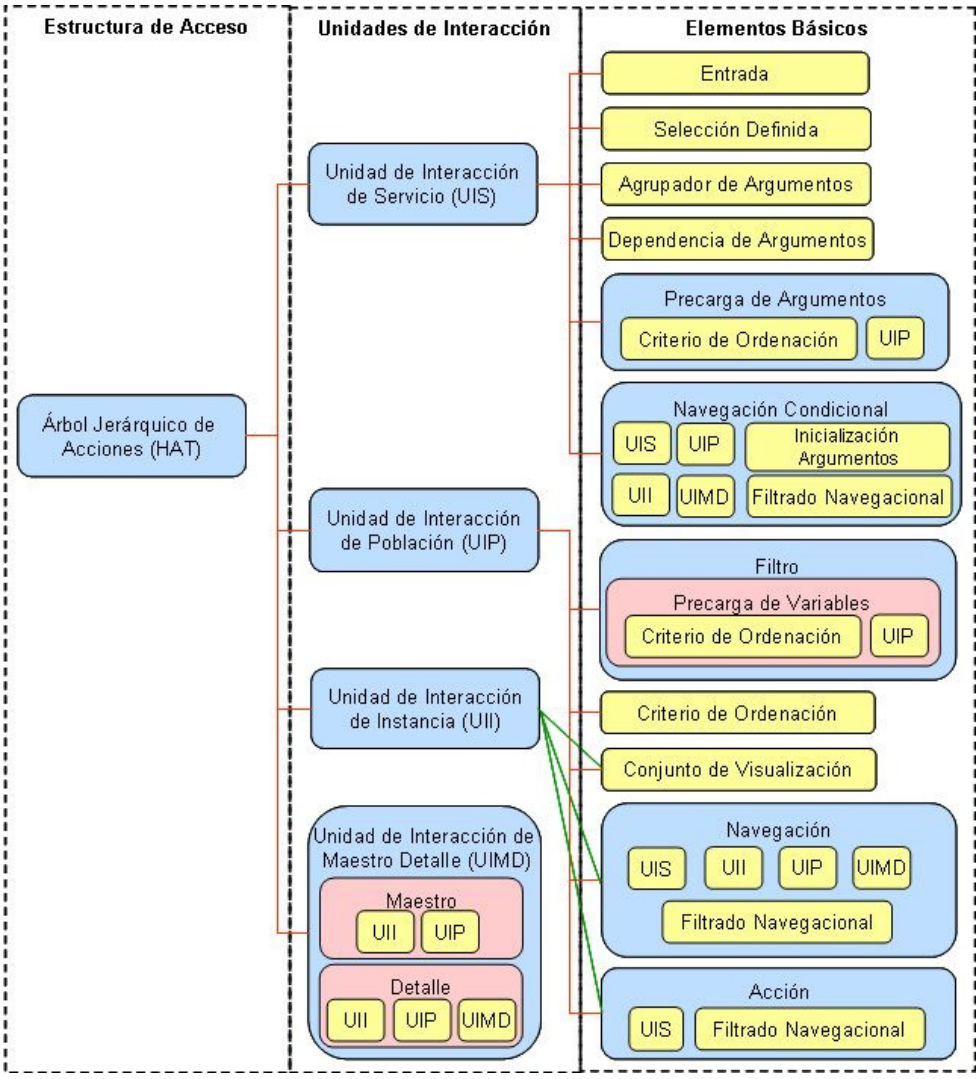


Figura 3.8: Niveles y elementos del Modelo de Presentación de OO-Method

4

Plantillas de Transformación

Como se ha visto previamente, en la Sección 3.2, el CRF (*Cameleon Reference Framework*) contempla cuatro niveles de abstracción distintos para estructurar el ciclo de vida del desarrollo de las interfaces de usuario junto con los modelos que las representan: Tareas y Conceptos, AUI (*Interfaz de Usuario Abstracta*), CUI (*Interfaz de Usuario Concreta*) y FUI (*Interfaz de Usuario Final*).

Además, el CRF permite iniciar el proceso de desarrollo de una interfaz de usuario desde cualquiera de los niveles de abstracción citados, pues provee de relaciones de reificación para ser utilizadas en un proceso de ingeniería directa de interfaces de usuario (*forward engineering*), y relaciones de abstracción para ser utilizadas en un proceso de ingeniería reversa de interfaces de usuario (*backward engineering*).

En este trabajo se considera el caso de la ingeniería directa de interfaces de usuario. Este tipo de ingeniería da lugar a tres tipos de transformaciones en un proceso DE de desarrollo de interfaces de usuario:

- Transformación del modelo de Tareas y Conceptos al modelo AUI
- Transformación del modelo AUI al modelo CUI
- Transformación del modelo CUI al código de la FUI

Este trabajo se enfoca en estos tres tipos de transformaciones. Además también considera las transformaciones a partir de un modelo de interfaces de usuario basado en patrones, como son los casos del Modelo de Presentación de OO-Method [Molina et al., 2002], y de las siguientes propuestas para el modelado de la interacción a nivel abstracto [Valverde et al., 2007; Valverde et al., 2009].

Por una parte, existen aproximaciones en las que estas transformaciones están implícitas en las herramientas que realizan las transformaciones lo cual resulta en una falta de flexibilidad para adaptar las transformaciones según el tipo de proyecto en el que se esté trabajando [Puerta and Eisenstein, 1999b; Limbourg and Vanderdonckt, 2004]. Por otra parte, típicamente existen similitudes y diferencias entre distintos proyectos de desarrollo de interfaces de usuario. Por lo tanto, no resulta muy razonable definir las reglas de transformación para las

transformaciones de modelos de interfaces de usuario cada vez para cada proyecto, como tampoco resulta muy razonable usar siempre el mismo conjunto de reglas de transformación para todos los proyectos de desarrollo de interfaces de usuario.

El objetivo principal de este trabajo es el de otorgar flexibilidad y reusabilidad a las transformaciones entre modelos de interfaces de usuario consideradas (Tareas y Conceptos a AUI, AUI a CUI y CUI a FUI, incluyendo modelos basados en patrones). La flexibilidad se logra externalizando las reglas de transformación, ya que cuando estas son externas a las herramientas que realizan las transformaciones, pueden ser editadas y adaptadas a las características del proyecto de desarrollo de interfaces de usuario que se está llevando a cabo. Y luego, las reglas adaptadas pueden ser reutilizadas en otros proyectos de desarrollo de interfaces de usuario que tengan características similares.

A fin de lograr el objetivo se propone la utilización de *Plantillas de Transformación* [Aquino et al., 2008].

En términos generales, una Plantilla de Transformación tiene el objetivo de especificar la estructura y el estilo de una interfaz de usuario de acuerdo a las preferencias y los requisitos de los usuarios finales, y a las características de las distintas plataformas de cómputo y ambientes en los que se podrá usar la interfaz de usuario.

Una Plantilla de Transformación está compuesta por *Parámetros* con *Valores* asociados que parametrizan las transformaciones entre modelos de interfaces de usuario.

Los parámetros pueden ser de bajo nivel o de alto nivel. Los parámetros de bajo nivel se asocian, típicamente, a características de estilo, como colores o tipos de letras. Por su parte, los parámetros de alto nivel pueden determinar la estructura de la interfaz de usuario y la disposición de sus componentes.

Entonces, las Plantillas de Transformación con sus Parámetros externalizan las reglas de transformación entre modelos de interfaces de usuario y las hacen modificables. La asignación de distintos valores a los parámetros permite adaptar las reglas de transformación de acuerdo a las características del proyecto que se está desarrollando. Y, por supuesto, las Plantillas de Transformación generadas para un proyecto, pueden ser almacenadas y reutilizadas posteriormente en proyectos con características similares.

La Figura 4.1 ilustra el uso de una Plantilla de Transformación en el caso de una transformación de un modelo AUI a un modelo CUI. Una herramienta de transformación toma como entradas al modelo AUI y a la Plantilla de Transformación. La Plantilla de Transformación provee especificaciones que determinan cómo transformar el modelo AUI en un modelo CUI. Siguiendo esas especificaciones la herramienta de transformación genera el modelo CUI.

Los conceptos que caracterizan a las Plantillas de Transformación se estructuran en los siguientes tres grupos:

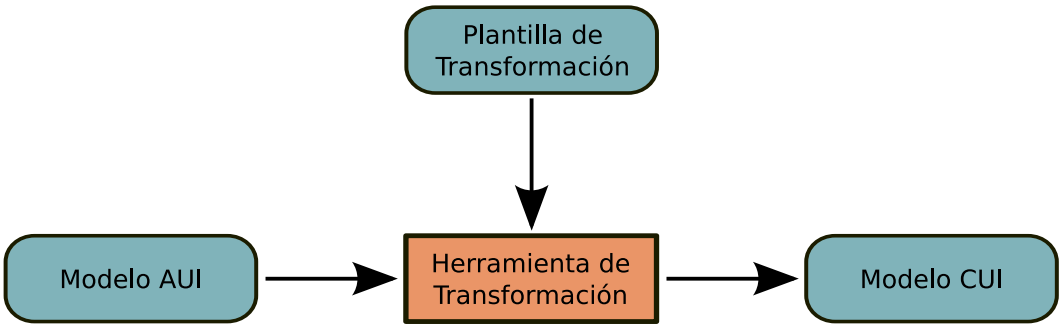


Figura 4.1: Transformación de un Modelo AUI a un Modelo CUI utilizando una Plantilla de Transformación

- *Contextos* de uso de interfaces de usuario
- Modelos de *interfaces de usuario*
- *Plantillas de Transformación*

Los grupos de contextos de uso de interfaces de usuario y de modelos de interfaces de usuario proveen conceptos de soporte a los conceptos específicos de Plantillas de Transformación, y permiten especificar clara y precisamente la semántica de las Plantillas de Transformación.

A fin de describir y caracterizar de manera precisa a todos los conceptos relacionados a las Plantillas de Transformación se han elaborado los meta modelos de los tres grupos de conceptos. Para los grupos de modelos de interfaces de usuario y Plantillas de Transformación, se han elaborado también los meta-meta modelos correspondientes.

Un meta modelo es una definición estructural de los elementos que pueden estar contenidos en un modelo, con sus propiedades y las relaciones entre ellos [Selic, 2007]. A su vez, un meta-meta modelo es una abstracción que considera a un meta modelo como modelo, y por lo tanto, define la estructura de los elementos que pueden estar contenidos en un meta modelo, con sus propiedades y las relaciones entre ellos.

A continuación, se presentan cada uno de los tres grupos de conceptos relacionados a las Plantillas de Transformación junto con los meta-meta modelos y meta modelos que los describen. A medida que se van presentado los elementos de los meta-meta modelos y meta modelos, se va dando una descripción detallada de los mismos.

4.1. Contexto

En este trabajo el concepto de *Contexto* se refiere al contexto de uso de una aplicación o sistema interactivo y se define, tal como en el CRF [Calvary et al., 2003], por tres tipos de entidades:

- Los usuarios finales del sistema
- La plataforma hardware y software sobre la que se ejecuta el sistema, incluyendo a los dispositivos que se utilizan para la interacción con el sistema
- El ambiente físico en el que la interacción con el sistema toma lugar

En la Sección 3.2.1 los tres tipos de entidades mencionadas se describen con más detalle.

Se ha adoptado la definición de contexto del CRF ya que la misma descompone el concepto de contexto en tres ejes (usuario, plataforma y ambiente) significativos y convenientes para contextualizar la interacción de una persona con un sistema. Además, la definición de contexto del CRF tiene una amplia aceptación en la comunidad de HCI.

Con respecto a la relación entre el concepto de contexto y las Plantillas de Transformación, el objetivo es que sea posible definir diferentes Plantillas de Transformación para distintos contextos de uso.

A continuación se describe el meta modelo de contexto utilizado en el marco de este trabajo.

4.1.1. Meta modelo de Contexto

En este trabajo se ha adoptado una vista del meta modelo de Contexto de UsiXML a fin de caracterizar los contextos de uso de una aplicación interactiva.

La Sección 3.3 presenta más información sobre UsiXML.

Desde http://www.usixml.org/index.php?mod=pages&id=5 se puede acceder al meta modelo de Contexto de UsiXML completo.

La vista considerada del meta modelo de contexto de UsiXML da soporte al concepto de contexto tal como está definido en el CRF.

El meta modelo de contexto de UsiXML también da soporte a los dominios de plasticidad [Calvary et al., 2001]. El dominio de las interfaces de usuario se refiere al abanico de contextos de uso en los que la interfaz de usuario se puede desplegar. El sub-conjunto de este dominio para el cual la usabilidad se preserva se denomina dominio de plasticidad.

Por ahora, este trabajo de tesis no incorpora los conceptos de plasticidad. Los mismos han sido excluidos de la vista considerada del meta modelo de contexto

de UsiXML. Estos aspectos de plasticidad podrían ser abordados en futuras extensiones de este trabajo.

La Figura 4.2 ilustra los elementos y relaciones que componen la vista del meta modelo de contexto considerada. A continuación se describen brevemente los elementos y relaciones del meta modelo, así como también las propiedades de los elementos. Para obtener una especificación más detallada del meta modelo, representado como un Diagrama de Clases de UML en Rational Rose, se puede acceder a: http://www.usixml.org/index.php?mod=pages&id=5.

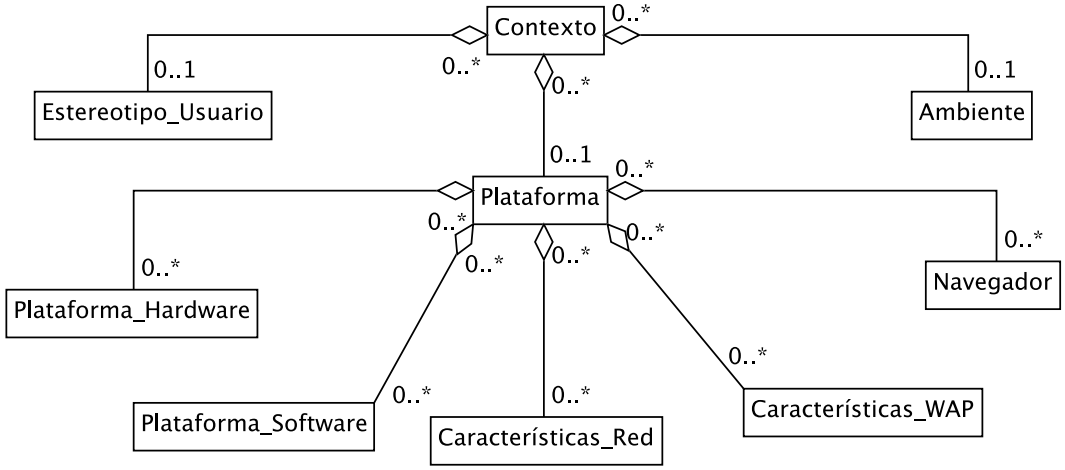


Figura 4.2: Elementos y relaciones de la vista del meta modelo de Contexto de UsiXML considerada en este trabajo

Como se puede apreciar en la Figura 4.2, un *Contexto* de uso está conformado por un *Estereotipo de Usuario* que realiza una tarea interactiva con una *Plataforma* de cómputo específica en un *Ambiente* dado.

Además, un contexto de uso tiene las siguientes propiedades:

- Identificador: identificador único del contexto de uso.
- Nombre: nombre del contexto de uso.

Un *Estereotipo de Usuario* representa a un conjunto de usuarios que comparten los mismos valores en unos parámetros de descripción o propiedades.

Las propiedades de un estereotipo de usuario son las siguientes:

- Identificador: identificador único de los estereotipos de usuario.
- Lenguaje: lenguaje del estereotipo de usuarios.
- Nombre: nombre del estereotipo de usuarios que comparten la misma descripción.

- Experiencia en la tarea: describe el nivel de experiencia que tienen los usuarios del estereotipo para realizar su tarea de manera independiente de cualquier sistema de soporte. Puede asumir un valor del rango de 1 (bajo) a 5 (alto).

- Experiencia con el sistema: describe el nivel de experiencia que tienen los usuarios del estereotipo para manipular el sistema interactivo. Puede asumir un valor de 1 (bajo) a 5 (alto). Este atributo es diferente al de la experiencia en la tarea en el sentido de que este atributo se refiere a la habilidad que el usuario puede tener con el sistema independientemente de una tarea particular. Podría ser con otra tarea.

- Experiencia con dispositivos: describe el nivel de experiencia que tienen los usuarios del estereotipo para manipular el sistema con dispositivos de interacción avanzados. Los dispositivos de interacción avanzados incluyen, por ejemplo, a las interacciones basadas en lápiz, interacciones basadas en gestos, etc; mientras que los dispositivos tradicionales incluyen el monitor, teclado y ratón. Puede asumir un valor de 1 (bajo) a 5 (alto).

- Motivación para la tarea: especifica el nivel de motivación suscitado por la tarea. Puede asumir un valor de 1 (bajo) a 5 (alto). Por ejemplo, se puede considerar que una tarea repetitiva y tediosa suscitará una motivación baja, mientras que un juego podría resultar altamente motivante.

El *Ambiente* describe cualquier propiedad de interés del ambiente físico en el que el usuario está utilizando el sistema. Las propiedades pueden ser físicas (como las condiciones de iluminación), psicológicas (como el nivel de estrés) y organizativas (como la definición del rol y de la posición en un organigrama de una organización).

El ambiente se describe con las siguientes propiedades:

- Tipo: es el tipo de ambiente clasificado de acuerdo a un catálogo de ambientes posibles. Este catálogo puede ser expandido cuando se requiera.

- Identificador: es el identificador del ambiente físico.

- Nombre: nombre del ambiente físico.

- Es ruidoso: indica si el ambiente tiene un nivel significativo de ruido que podría afectar la realización de la tarea.

- Nivel de iluminación: expresa el nivel de iluminación del ambiente.

- Es estresante: indica si en el ambiente se tiene un nivel de estrés que podría influenciar significativamente a la realización de las tareas.

La *Plataforma* captura las propiedades relevantes de cada par de plataformas hardware y software junto con los dispositivos que puedan influir significativamente en el contexto de uso en el que el usuario está utilizando el sistema.

Una plataforma tiene las siguientes propiedades:

- Identificador: es el identificador de la plataforma.
- Nombre: es el nombre de la plataforma.

Una plataforma puede estar formada por una serie de dispositivos hardware físicos (*Plataforma Hardware*), una serie de componentes software (*Plataforma Software*), las características de la *Red* a la que la plataforma está conectada, la capacidad de soporte a conectividad inalámbrica *(WAP)*, y la capacidad de navegación en páginas web (*Navegador*).

La *Plataforma Hardware* se describe con las siguientes propiedades:

- Modelo del hardware.
- Fabricante del hardware.
- Tipo de CPU.
- Tipo de teclado.
- Tipo de dispositivo apuntador.
- Capacidad de color.
- Capacidad de almacenamiento.
- Despliega imágenes: indica si el hardware es capaz de desplegar imágenes.
- Emite sonido: indica si el hardware es capaz de emitir sonidos.
- Recibe texto como entrada: indica si el hardware es capaz de recibir texto como entrada.
- Recibe sonido como entrada: indica si el hardware es capaz de recibir sonido como entrada.
- Tiene pantalla táctil: indica si el hardware tiene pantalla táctil.
- Ancho de pantalla.
- Largo de pantalla.
- Conjunto de caracteres de entrada.

- Conjunto de caracteres de salida.

La *Plataforma Software* se caracteriza con las siguientes propiedades:

- Nombre del sistema operativo.
- Fabricante del sistema operativo.
- Versión del sistema operativo.
- Versión de JVM: es la versión de la máquina virtual Java (JVM - *Java Virtual Machine*).
- Codificador de entrada de audio.
- Codificador de entrada de vídeo.

Las características de la *Red* se pueden describir con las siguientes propiedades:

- Capacidad: es la cantidad máxima de datos que se puede enviar por segundo a través de un canal de comunicación dado (kb/s).
- Costo por volumen: expresa el costo por unidad de datos transferidos. Sus valores posibles son: "Alto", "Medio", "Bajo".
- Costo por tiempo: se refiere al costo por tiempo de utilización del canal de comunicación. Sus valores posibles son: "Alto", "Medio", "Bajo".

En caso de utilizar el protocolo *WAP*, las siguientes propiedades lo describen:

- Conjunto de pictogramas soportado: se refiere al conjunto de las clases de pictogramas que son soportados por el dispositivo, según como se define en la especificación de pictogramas WAP (*WAP Pictogram specification*). Un pictograma es una imagen similar a un icono que se despliega entre el texto.
- Clase de dispositivo WAP: es la clase de dispositivo, según sus capacidades, tal como se identifica en la especificación WAP 1.1.
- Versión de WML: es la lista de versiones del lenguaje WML que son soportadas por el dispositivo.
- Tamaño de documento WML: especifica el tamaño máximo de un documento WML que se puede descargar al dispositivo.
- Versión de WMLScript: es la lista de versiones de WMLScript que son soportadas por el dispositivo.

- Librerías WMLScript: es la lista de librerías requeridas y opcionales soportadas en el WMLScript VM del dispositivo.

Las siguientes son propiedades relacionadas al *Navegador* utilizado en una plataforma:

- Nombre: es el nombre del navegador.
- Versión: indica la versión del navegador.
- Versión de HTML: es la versión de HTML (*HyperText Markup Language*) soportada por el navegador.
- Despliega marcos (*frames*): indica si el navegador es capaz de desplegar marcos de HTML.
- Despliega tablas: indica si el navegador es capaz de desplegar tablas de HTML.
- Despliega Java Applets: indica si el navegador da soporte a los Applets de Java.
- Maneja JavaScript: indica si el navegador da soporte a JavaScript.
- Versión de JavaScript: es la versión del lenguaje JavaScript soportada por el navegador.
- Versión de XHTML: es la versión de XHTML soportada por el navegador.
- Módulos XHTML: consiste en la lista de módulos XHTML soportados por el navegador.

4.2. Interfaces de Usuario

Como se ha visto en el Capítulo 2, existen diversas aproximaciones para modelar interfaces de usuario. Estas aproximaciones ofrecen meta modelos de interfaces de usuario, a partir de los cuales se pueden crear modelos de interfaces de usuario.

Este trabajo presenta una aproximación de Plantillas de Transformación que se utilizan para parametrizar transformaciones entre modelos de interfaces de usuario. Por lo tanto, se tiene la intención de que las Plantillas de Transformación se puedan utilizar con las distintas aproximaciones de modelado de interfaces de usuario. Para que esto sea posible, se debe especificar cómo relacionar las Plantillas de Transformación con estas aproximaciones.

Las siguientes dos Secciones presentan un meta-meta modelo y un meta modelo de interfaces de usuario que, de manera simplificada, representan algunos de los conceptos que caracterizan a diversos meta modelos y modelos de interfaces de usuario, respectivamente, tanto a nivel abstracto como a nivel concreto.

Esta versión simplificada de los meta-meta modelos y meta modelos de interfaces de usuario provee solamente los conceptos que son necesarios a fin de poder explicar cómo se definen y cómo se utilizan las Plantillas de Transformación. Por lo tanto, define también los conceptos o requisitos que debe incorporar o cumplir una aproximación de modelado de interfaces de usuario para poder utilizar Plantillas de Transformación.

4.2.1. Meta-meta modelo de Interfaces de Usuario

La aproximación de Plantillas de Transformación asume que a nivel de meta-meta modelo de interfaces de usuario existen *Meta Modelos de Interfaces de Usuario* que se componen de *Meta Elementos* y de *Meta Relaciones* entre los Meta Elementos (ver Figura 4.3).

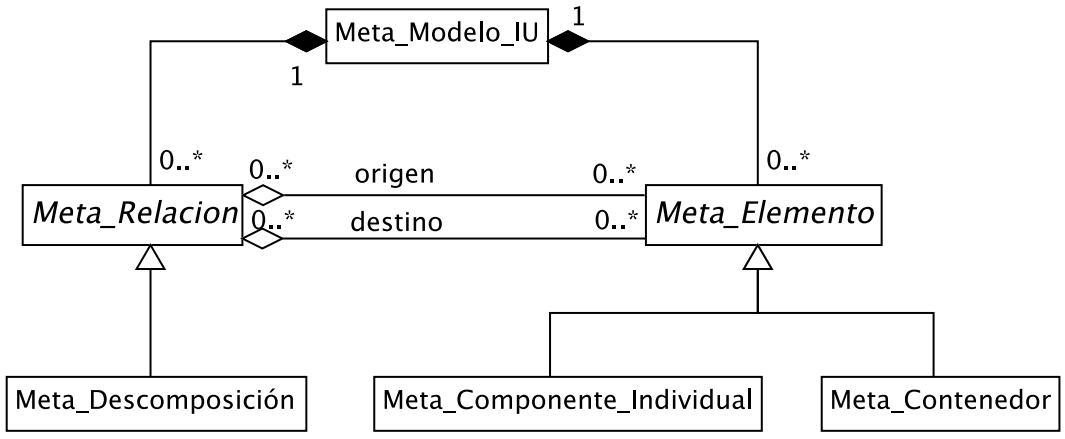


Figura 4.3: Elementos y relaciones del meta-meta modelo de Interfaces de Usuario considerado en este trabajo

En este contexto, el Meta Modelo de Interfaz de Usuario representa, de manera general y simplificada, a diversos meta modelos de interfaces de usuario abstractos y concretos existentes, como por ejemplo el meta modelo del Modelo de Presentación de OO-Method (ver Sección 3.4.1) y los meta modelos de los auiModel y cuiModel de UsiXML (ver Sección 3.3.1).

Un Meta Elemento representa a un elemento de un Meta Modelo de Interfaces de Usuario. Para los fines de este trabajo, un Meta Elemento es un elemento

abstracto, y sus especializaciones son el *Meta Componente Individual* y el *Meta Contenedor*.

Tanto a nivel abstracto, como a nivel concreto y final, una interfaz de usuario puede ser caracterizada por componentes individuales y contenedores [Vanderdonckt, 2005]. Los componentes individuales son atómicos y no pueden contener o estar formados por otros componentes. Por su parte, los contenedores sí pueden contener a componentes individuales y a otros contenedores. Por lo tanto, un Meta Componente Individual representa a un componente individual, y un Meta Contenedor a un contenedor.

Otras especializaciones de Meta Elemento son posibles en un meta-meta modelo de interfaces de usuario, pero a efectos de este trabajo las de Meta Componente Individual y Meta Contenedor son suficientes.

Por su parte, una Meta Relación tiene un conjunto de Meta Elementos que son el origen de la relación, y un conjunto de Meta Elementos que son el destino de la relación.

Las Meta Relaciones deben poder especificar las cardinalidades de todos los Meta Elementos que intervienen, pero a fin de simplificar el meta-meta modelo de interfaces de usuario, se ha asumido que todos los Meta Elementos orígenes y destinos de una Meta Relación tienen cardinalidades de 0 (cero) o muchos.

En el contexto de este trabajo, una Meta Relación es un elemento abstracto y su especialización es la *Meta Descomposición*.

Una Meta Descomposición representa la estructuración lógica de los componentes de una interfaz de usuario. En una Meta Descomposición la relación con Meta Elementos orígenes de la Meta Relación, se especializa a una relación con Meta Elementos contenedores de la Meta Descomposición. Debido a esto, se añade la restricción de que todos los Meta Elementos contenedores de la Meta Descomposición deben ser Meta Contenedores. De manera análoga, la relación con Meta Elementos destino de la Meta Relación, se especializa a una relación con Meta Elementos contenidos en la Meta Descomposición. Los Meta Elementos contenidos en una Meta Descomposición pueden ser Meta Contenedores y Meta Componentes Individuales.

También en este caso, otras especializaciones de Meta Relación son posibles en un meta-meta modelo de interfaces de usuario, pero a efectos de este trabajo la de Meta Descomposición es suficiente.

Los elementos de un meta-meta modelo o meta modelo se pueden describir con el formato siguiente, el cual ha sido esquematizado a partir de una versión simplificada de la descripción de un modelo de objetos que aparece en [Pastor and Molina, 2007]:

- Nombre del elemento: nombra a un elemento de un meta-meta modelo o meta modelo. Este nombre está orientado a ser utilizado, sobre todo, por herramientas software. En este trabajo, en las Figuras de meta-meta

modelos y meta modelos se utiliza el nombre del elemento para identificar a cada elemento.

- Alias: es una alternativa al nombre de un elemento de un meta-meta modelo o meta modelo. Este alias está orientado a ser utilizado por personas.

- Descripción: describe un elemento de un meta-meta modelo o meta modelo.

- Propiedades: se listan las propiedades o atributos de los elementos de un meta-meta modelo o meta modelo. Cada propiedad se caracteriza con los siguientes datos:

  - Alias: alias de la propiedad, orientado a personas.
  - Descripción: describe la propiedad.
  - Tipo de propiedad: Constante, Variable o Derivada. El valor de una propiedad Constante permanece inalterado luego de que la instancia del elemento es creada. El valor de una propiedad Variable puede cambiar en el tiempo a consecuencia de la ocurrencia de servicios que alteran este valor. El valor de una propiedad Derivada se calcula a partir del valor de otras propiedades del elemento. Cuando la propiedad es Derivada se debe explicar también la lógica de la derivación.
  - Tipo de dato: define el tipo de dato de la propiedad.
  - Es identificador: indica si la propiedad forma parte del identificador único de un elemento.
  - Acepta nulo: indica si la propiedad acepta el valor nulo.
  - Valor por defecto: define un valor por defecto para el elemento.

- Relaciones: se listan las relaciones entre los elementos de un meta-meta modelo o meta modelo. Para cada relación se especifican las cardinalidades correspondientes, y se explican las restricciones relacionadas. En general, y para evitar muchas repeticiones, si en la descripción de un elemento $E1$ ya se ha descripto su relación R con otro elemento $E2$, al describir el elemento $E2$ ya no se describe la relación R con $E1$.

- Servicios: se listan los servicios o métodos que ofrece un elemento de un meta-meta modelo o meta modelo. Los servicios son descriptos con su nombre y lista de argumentos de entrada. También se presenta una descripción textual del propósito del servicio, y de las condiciones que se deben cumplir para poder ejecutar un servicio, y las que se deben cumplir por los argumentos de entrada del servicio.

- Observaciones: se mencionan aspectos del meta-meta modelo o meta modelo que se deben tener en cuenta para fines de implementación, o se comentan las asunciones realizadas, etc.

A continuación se describe cada elemento del meta-meta modelo de interfaces de usuario considerado en este trabajo, junto con sus propiedades, relaciones, servicios y restricciones, siguiendo el formato presentado. Este mismo formato será utilizado más adelante para describir los elementos de los demás meta-meta modelos y meta modelos que se presentan en este trabajo.

Elemento: Meta Modelo de Interfaz de Usuario

- Nombre del elemento: Meta_Modelo_IU
- Alias: Meta Modelo de Interfaz de Usuario
- Descripción: representa, de manera general y simplificada, a diversos meta modelos de interfaces de usuario abstractos y concretos existentes.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único de un meta modelo de interfaz de usuario
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Nombre
    - Descripción: Nombre del meta modelo de interfaz de usuario
    - Tipo de propiedad: Variable
    - Tipo de dato: String
    - Es identificador: no
    - Acepta nulo: no
- Servicios:
  - Nuevo(String pIdentificador, String pNombre): crea un Meta Modelo de Interfaz de Usuario asignando a las propiedades Identificador y Nombre el valor de los argumentos pIdentificador y pNombre, respectivamente.

- Editar(Meta_Modelo_IU THIS, String pNombre): asigna a la propiedad Nombre del Meta Modelo de Interfaz de Usuario actual, referenciado por THIS, el valor del argumento pNombre.
- Eliminar(Meta_Modelo_IU THIS): elimina el Meta Modelo de Interfaz de Usuario actual, referenciado por THIS.

Elemento: Meta Elemento

- Nombre del elemento: Meta_Elemento
- Alias: Meta Elemento
- Descripción: representa a un elemento de un meta modelo de interfaz de usuario.

  Es un elemento abstracto. Sus especializaciones son: Meta Componente Individual y Meta Relación.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único de un meta elemento
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Nombre
    - Descripción: Nombre del meta elemento
    - Tipo de propiedad: Variable
    - Tipo de dato: String
    - Es identificador: no
    - Acepta nulo: no
- Relaciones:
  - Un Meta Elemento corresponde (se asocia) a un Meta Modelo de Interfaz de Usuario.

    Un Meta Modelo de Interfaz de Usuario se compone de 0 o muchos Meta Elementos.

- Servicios:

  - Nuevo(String pIdentificador, String pNombre, Meta_Modelo_IU pMetaModeloIU): crea un Meta Elemento asignando a las propiedades Identificador y Nombre del nuevo Meta Elemento, el valor de los argumentos pIdentificador y pNombre, respectivamente. Además, asocia el Meta Elemento creado al Meta Modelo de Interfaz de Usuario especificado por el argumento pMetaModeloIU.
  - Editar(Meta_Elemento THIS, String pNombre): asigna a la propiedad Nombre del Meta Elemento actual, referenciado por THIS, el valor del argumento pNombre.
  - Eliminar(Meta_Elemento THIS): elimina el Meta Elemento actual, referenciado por THIS.

Elemento: Meta Componente Individual

- Nombre del elemento: Meta_Componente_Individual
- Alias: Meta Componente Individual
- Descripción: representa a un componente individual de un meta modelo de interfaz de usuario. La expresión componente individual se refiere a que el componente es atómico, no está formado por otros componentes.
- Propiedades: las que hereda de Meta Elemento: Identificador y Nombre.
- Relaciones:

  - Meta Componente Individual especializa a Meta Elemento. Por lo tanto hereda sus relaciones.

- Servicios: los que hereda de Meta Elemento: Editar y Eliminar.

  Además, define el siguiente servicio:

  - Nuevo(String pIdentificador, String pNombre, Meta_Modelo_IU pMetaModeloIU): primero crea un Meta Elemento invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador, pNombre y pMetaModeloIU, en ese orden. Luego crea el Meta Componente Individual correspondiente.

Elemento: Meta Contenedor

- Nombre del elemento: Meta_Contenedor
- Alias: Meta Contenedor
- Descripción: representa a un contenedor de un meta modelo de interfaz de usuario. Un contenedor puede contener otros contenedores o componentes individuales.
- Propiedades: las que hereda de Meta Elemento: Identificador y Nombre.
- Relaciones:
  - Meta Contenedor especializa a Meta Elemento. Por lo tanto hereda sus relaciones.
- Servicios: los que hereda de Meta Elemento: Editar y Eliminar.

  Además, define el siguiente servicio:
  - Nuevo(String pIdentificador, String pNombre, Meta_Modelo_IU pMeta-ModeloIU): primero crea un Meta Elemento invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador, pNombre y pMetaModeloIU, en ese orden. Luego crea el Meta Contenedor correspondiente.

Elemento: Meta Relación

- Nombre del elemento: Meta_Relacion
- Alias: Meta Relación
- Descripción: representa a una relación entre elementos de un meta modelo de interfaz de usuario.

  Es un elemento abstracto. Meta Descomposición especializa a Meta Relación.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único de una meta relación
    - Tipo de propiedad: Constante
    - Tipo de dato: String

    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Nombre
    - Descripción: Nombre de la meta relación
    - Tipo de propiedad: Variable
    - Tipo de dato: String
    - Es identificador: no
    - Acepta nulo: no

- Relaciones:
  - Una Meta Relación corresponde (se asocia) a un Meta Modelo de Interfaz de Usuario.

    Un Meta Modelo de Interfaz de Usuario se compone de 0 o muchas Meta Relaciones.
  - Una Meta Relación se compone de 0 o muchos Meta Elementos que juegan el papel de origen de la Meta Relación.

    Un Meta Elemento puede ser origen en (se asocia a) 0 o muchas Meta Relaciones.

    En cada Meta Relación, todos los Meta Elementos origen deben ser distintos unos de otros.

    Cada Meta Elemento origen de una Meta Relación debe estar asociado al mismo Meta Modelo de Interfaz de Usuario al cual está asociada la Meta Relación.
  - Una Meta Relación se compone de 0 o muchos Meta Elementos que juegan el papel de destino de la Meta Relación.

    Un Meta Elemento puede ser destino en (se asocia a) 0 o muchas Meta Relaciones.

    En cada Meta Relación, todos los Meta Elementos destino deben ser distintos unos de otros.

    Cada Meta Elemento destino de una Meta Relación debe estar asociado al mismo Meta Modelo de Interfaz de Usuario al cual está asociada la Meta Relación.

- Servicios:
  - Nuevo(String pIdentificador, String pNombre, Meta_Modelo_IU pMeta-ModeloIU): crea una Meta Relación asignando a las propiedades Identificador y Nombre de la nueva Meta Relación, el valor de los argumentos pIdentificador y pNombre, respectivamente. Además, asocia la

Meta Relación creada al Meta Modelo de Interfaz de Usuario especificado por el argumento pMetaModeloIU.

- Editar(Meta_Relacion THIS, String pNombre): asigna a la propiedad Nombre de la Meta Relación actual, referenciada por THIS, el valor del argumento pNombre.
- Eliminar(Meta_Relacion THIS): elimina la Meta Relación actual, referenciada por THIS.
- Agregar_Origen(Meta_Relacion THIS, Meta_Elemento pMetaElemento): asocia, con el papel de origen, el Meta Elemento especificado por el argumento pMetaElemento a la Meta Relación actual, referenciada por THIS.

  pMetaElemento debe ser un Meta Elemento que aún no es origen en la Meta Relación actual.

  pMetaElemento debe estar asociado al mismo Meta Modelo de Interfaz de Usuario al cual está asociada la Meta Relación actual.
- Quitar_Origen(Meta_Relacion THIS, Meta_Elemento pMetaElementoOrigen): quita, en su papel de origen, el Meta Elemento especificado por el argumento pMetaElementoOrigen de la Meta Relación actual, referenciada por THIS.

  pMetaElementoOrigen debe ser un Meta Elemento origen en la Meta Relación actual.
- Agregar_Destino(Meta_Relacion THIS, Meta_Elemento pMetaElemento): asocia, con el papel de destino, el Meta Elemento especificado por el argumento pMetaElemento a la Meta Relación actual, referenciada por THIS.

  pMetaElemento debe ser un Meta Elemento que aún no es destino en la Meta Relación actual.

  pMetaElemento debe estar asociado al mismo Meta Modelo de Interfaz de Usuario al cual está asociada la Meta Relación actual.
- Quitar_Destino(Meta_Relacion THIS, Meta_Elemento pMetaElementoDestino): quita, en su papel de destino, el Meta Elemento especificado por el argumento pMetaElementoDestino de la Meta Relación actual, referenciada por THIS.

  pMetaElementoDestino debe ser un Meta Elemento destino en la Meta Relación actual.

- Observaciones: se asume que todos los Meta Elementos orígenes y destinos de una Meta Relación tienen cardinalidades de 0 (cero) o muchos.

Elemento: Meta Descomposición

- Nombre del elemento: Meta_Descomposicion
- Alias: Meta Descomposición
- Descripción: representa a una relación de descomposición lógica (no física) entre elementos de un meta modelo de interfaz de usuario.
- Propiedades: las que hereda de Meta Relación: Identificador y Nombre.
- Relaciones:
  - Meta Descomposición especializa a Meta Relación. Por lo tanto hereda sus relaciones, y agrega la siguiente restricción:
    - En una Meta Descomposición todos los Meta Elementos origen deben ser Meta Contenedores.
- Servicios: los que hereda de Meta Relación: Editar, Eliminar, Agregar_Origen, Quitar_Origen, Agregar_Destino y Quitar_Destino.

  En los servicios de Agregar_Origen y Quitar_Origen se agrega la restricción de que los Meta Elementos origen deben ser Meta Contenedores.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, String pNombre, Meta_Modelo_IU pMetaModeloIU): primero crea una Meta Relación invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador, pNombre y pMetaModeloIU, en ese orden. Luego crea la Meta Descomposición correspondiente.

4.2.2. Meta modelo de Interfaces de Usuario

A nivel de meta modelo de interfaces de usuario, la aproximación de Plantillas de Transformación asume que existen los elementos que se corresponden a los del nivel de meta-meta modelo de interfaces de usuario, y que son necesarios y suficientes para describir la definición y comportamiento de las Plantillas de Transformación.

La Figura 4.4 muestra que se tiene un *Modelo de Interfaz de Usuario* que se compone de *Elementos* y de *Descomposiciones*. Además, se ilustran las relaciones de estos elementos con sus elementos correspondientes en el nivel de meta-meta modelado.

Un Modelo de Interfaz de Usuario representa, de manera general y simplificada, a un modelo de interfaz de usuario abstracto o concreto, como por

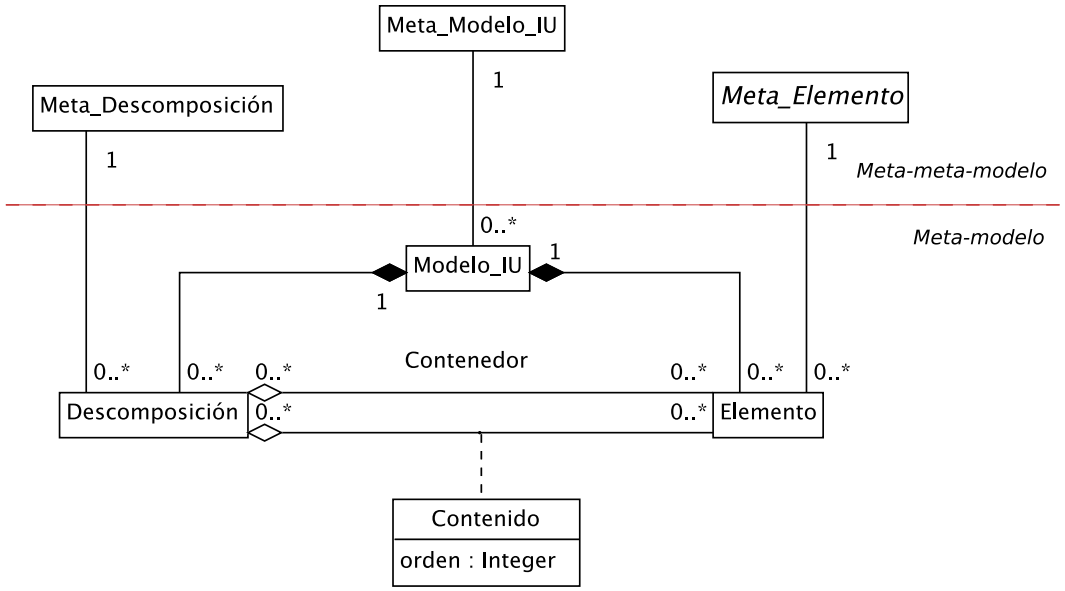


Figura 4.4: Elementos y relaciones del meta modelo de Interfaces de Usuario considerado en este trabajo

ejemplo, un Modelo de Presentación de OO-Method (ver Sección 3.4.1) o un auiModel o cuiModel de UsiXML (ver Sección 3.3.1).

Un Modelo de Interfaz de Usuario se asocia a su Meta Modelo de Interfaz de Usuario correspondiente en el nivel de meta-meta modelo de interfaces de usuario.

Un Elemento representa a un elemento de un Modelo de Interfaz de Usuario.

Un Elemento está asociado a su Meta Elemento correspondiente del nivel de meta-meta modelo de interfaces de usuario. El Meta Elemento asociado es, o un Meta Componente Individual, o un Meta Contenedor.

Una Descomposición tiene un conjunto de Elementos que son contenedores en la Descomposición. Una Descomposición tiene, también, un conjunto de Elementos que son contenido de la Descomposición. Los Elementos contenidos en una Descomposición están ordenados.

Una Descomposición también está asociada a su Meta Descomposición correspondiente del nivel de meta-meta modelo de interfaces de usuario. Esta asociación determina los Elementos que pueden ser contenedores y contenidos en la Descomposición.

A continuación se describe cada uno de los elementos del meta modelo de interfaces de usuario considerado en este trabajo, junto con sus propiedades, relaciones, servicios y restricciones.

Elemento: Modelo de Interfaz de Usuario

- Nombre del elemento: Modelo_IU
- Alias: Modelo de Interfaz de Usuario
- Descripción: representa un modelo de interfaz de usuario.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único del modelo de interfaz de usuario
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Fecha de creación
    - Descripción: Fecha de creación del modelo de interfaz de usuario
    - Tipo de propiedad: Constante
    - Tipo de dato: DateTime
    - Es identificador: no
    - Acepta nulo: no
    - Valor por defecto: fecha y hora del sistema
- Relaciones:
  - Un Modelo de Interfaz de Usuario corresponde (se asocia) a un Meta Modelo de Interfaz de Usuario.

    Un Meta Modelo de Interfaz de Usuario puede estar asociado a 0 o muchos Modelos de Interfaz de Usuario.
- Servicios:
  - Nuevo(String pIdentificador, DateTime pFechaCreacion, Meta_Modelo_IU pMetaModeloIU): crea un Modelo de Interfaz de Usuario asignando a las propiedades Identificador y Fecha de creación el valor de los argumentos pIdentificador y pFechaCreacion, respectivamente. Además, asocia el Modelo de Interfaz de Usuario creado al Meta Modelo de Interfaz de Usuario especificado por el argumento pMetaModeloIU.

- Eliminar(Modelo_IU THIS): elimina el Modelo de Interfaz de Usuario actual, referenciado por THIS.

Elemento: Elemento

- Nombre del elemento: Elemento
- Alias: Elemento
- Descripción: representa a un elemento de un modelo de interfaz de usuario.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único del elemento
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Fecha de creación
    - Descripción: Fecha de creación del elemento
    - Tipo de propiedad: Constante
    - Tipo de dato: DateTime
    - Es identificador: no
    - Acepta nulo: no
    - Valor por defecto: fecha y hora del sistema
- Relaciones:
  - Un Elemento corresponde (se asocia) a un Modelo de Interfaz de Usuario.

    Un Modelo de Interfaz de Usuario se compone de 0 o muchos Elementos.
  - Un Elemento corresponde (se asocia) a un Meta Elemento.

    Un Meta Elemento se asocia a 0 o muchos Elementos.

    El Meta Elemento que se asocia a un Elemento debe estar asociado al mismo Meta Modelo de Interfaz de Usuario que está asociado al Modelo de Interfaz de Usuario del cual forma parte el Elemento.

- Servicios:
  - Nuevo(String pIdentificador, DateTime pFechaCreacion, Modelo_IU pModeloIU, Meta_Elemento pMetaElemento): crea un Elemento asignando a las propiedades Identificador y Fecha de creación el valor de los argumentos pIdentificador y pFechaCreacion, respectivamente. Además, asocia el Elemento creado al Modelo de Interfaz de Usuario especificado por el argumento pModeloIU, y al Meta Elemento especificado por el argumento pMetaElemento.

    El Meta Elemento referenciado por pMetaElemento debe estar asociado al mismo Meta Modelo de Interfaz de Usuario que está asociado al Modelo de Interfaz de Usuario referenciado por pModeloIU.
  - Eliminar(Elemento THIS): elimina el Elemento actual, referenciado por THIS.

Elemento: Descomposición

- Nombre del elemento: Descomposicion
- Alias: Descomposición
- Descripción: representa a una relación de descomposición lógica (no física) entre elementos de un modelo de interfaz de usuario.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único de una descomposición
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Fecha de creación
    - Descripción: Fecha de creación de la descomposición
    - Tipo de propiedad: Constante
    - Tipo de dato: DateTime
    - Es identificador: no
    - Acepta nulo: no

- Valor por defecto: fecha y hora del sistema

- Relaciones:

  - Una Descomposición corresponde (se asocia) a un Modelo de Interfaz de Usuario.

    Un Modelo de Interfaz de Usuario se compone de 0 o muchas Descomposiciones.

  - Una Descomposición corresponde (se asocia) a una Meta Descomposición.

    Una Meta Descomposición se asocia a 0 o muchas Descomposiciones.

    La Meta Descomposición que se asocia a una Descomposición debe estar asociada al mismo Meta Modelo de Interfaz de Usuario que está asociado al Modelo de Interfaz de Usuario del cual forma parte la Descomposición.

  - Una Descomposición se compone de 0 o muchos Elementos que juegan el papel de contenedores de la Descomposición.

    Un Elemento puede ser contenedor en (se asocia a) 0 o muchas Descomposiciones.

    En cada Descomposición, todos los Elementos contenedores deben ser distintos unos de otros.

    Cada Elemento contenedor de una Descomposición debe estar asociado al mismo Modelo de Interfaz de Usuario al cual está asociada la Descomposición.

    Cada Elemento contenedor de una Descomposición debe estar asociado a un Meta Elemento que es origen en la Meta Descomposición que está asociada a la Descomposición. A consecuencia de esto y del hecho de que en una Meta Descomposición todos los Meta Elementos origen son Meta Contenedores, en una Descomposición todos los Elementos contenedores están asociados a Meta Elementos que son Meta Contenedores.

  - Una Descomposición se compone de 0 o muchos Elementos que juegan el papel de contenido de la Descomposición.

    Un Elemento puede estar contenido en (se asocia a) 0 o muchas Descomposiciones.

    En cada Descomposición, todos los Elementos contenidos deben ser distintos unos de otros.

    Cada Elemento contenido en una Descomposición debe estar asociado al mismo Modelo de Interfaz de Usuario al cual está asociada la Descomposición.

Cada Elemento contenido en una Descomposición debe estar asociado a un Meta Elemento que es destino de la Meta Descomposición que está asociada a la Descomposición.

Esta relación tiene, además, la siguiente propiedad:

- Alias: Orden
- Descripción: Orden o posición del elemento contenido dentro de la descomposición
- Tipo de propiedad: Variable
- Tipo de dato: Natural
- Es identificador: no
- Acepta nulo: no

- Servicios:

  - Nuevo(String pIdentificador, DateTime pFechaCreacion, Modelo_IU pModeloIU, Meta_Descomposicion pMetaDescomposicion): crea una Descomposición asignando a las propiedades Identificador y Fecha de creación el valor de los argumentos pIdentificador y pFechaCreacion, respectivamente. Además, asocia la Descomposición creada al Modelo de Interfaz de Usuario especificado por el argumento pModeloIU, y a la Meta Descomposición especificada por el argumento pMetaDescomposicion.

    La Meta Descomposición referenciada por pMetaDescomposicion debe estar asociada al mismo Meta Modelo de Interfaz de Usuario que está asociado al Modelo de Interfaz de Usuario referenciado por pModeloIU.

  - Eliminar(Descomposicion THIS): elimina la Descomposición actual, referenciada por THIS.

  - Agregar_Contenedor(Descomposicion THIS, Elemento pElemento): asocia, con el papel de contenedor, el Elemento especificado por el argumento pElemento a la Descomposición actual, referenciada por THIS.

    pElemento debe ser un Elemento que aún no es contenedor en la Descomposición actual.

    pElemento debe estar asociado al mismo Modelo de Interfaz de Usuario al cual está asociada la Descomposición actual.

    pElemento debe estar asociado a un Meta Elemento que es origen en la Meta Descomposición que está asociada a la Descomposición actual. Al satisfacerse esta precondición, y debido al hecho de que en una Meta Descomposición todos los Meta Elementos origen son Meta

Contenedores, también se cumple que pElemento está asociado a un Meta Elemento que es un Meta Contenedor.

- Quitar_Contenedor(Descomposicion THIS, Elemento pElementoContenedor): quita, en su papel de contenedor, el Elemento especificado por el argumento pElementoContenedor de la Descomposición actual, referenciada por THIS.

  pElementoContenedor debe ser un Elemento contenedor en la Descomposición actual.

- Agregar_Contenido(Descomposicion THIS, Elemento pElemento, Natural pOrden): asocia, con el papel de contenido, el Elemento especificado por el argumento pElemento a la Descomposición actual, referenciada por THIS. Además, asigna al Elemento referenciado por pElemento la posición u orden especificado por el argumento pOrden entre los contenidos de la Descomposición actual.

  pElemento debe ser un Elemento que aún no está contenido en la Descomposición actual.

  pElemento debe estar asociado al mismo Modelo de Interfaz de Usuario al cual está asociada la Descomposición actual.

  pElemento debe estar asociado a un Meta Elemento que es destino en la Meta Descomposición que está asociada a la Descomposición actual.

- Quitar_Contenido(Descomposicion THIS, Elemento pElementoContenido): quita, en su papel de contenido, el Elemento especificado por el argumento pElementoContenido de la Descomposición actual, referenciada por THIS.

  pElementoContenido debe ser un Elemento contenido en la Descomposición actual.

- Cambiar_Orden_Contenido(Descomposicion THIS, Elemento pElementoContenido, Natural pOrden): asigna a la propiedad Orden de la relación entre la Descomposición actual, referenciada por THIS, y el Elemento referenciado por pElementoContenido, el valor del argumento pOrden.

  pElementoContenido debe ser un Elemento contenido en la Descomposición actual.

4.3. Plantillas de Transformación

Una Plantilla de Transformación [Aquino et al., 2008] tiene el objetivo de especificar la estructura y el estilo de una interfaz de usuario de acuerdo a las preferencias y los requisitos de los usuarios finales, y a las características de las

distintas plataformas de cómputo y ambientes en los que se podrá usar la interfaz de usuario, es decir, el contexto de uso de la interfaz de usuario.

Una Plantilla de Transformación está compuesta por Parámetros con Valores asociados que parametrizan las transformaciones entre modelos de interfaces de usuario. Esto externaliza la lógica que utilizan las herramientas de transformación de modelos y hace que esta lógica pueda ser adaptada según las características del proyecto que se está llevando a cabo.

Una Plantilla de Transformación que se define en el proceso de desarrollo de una interfaz de usuario, puede luego ser reutilizada en el proceso de desarrollo de otras interfaces de usuario con características similares, es decir, un contexto de uso similar.

Además, la aproximación de Plantillas de Transformación provee mecanismos que evitan sobrecargar, en demasía, a los analistas con la especificación de los aspectos concretos de las interfaces de usuario. Es decir, el analista no tiene que especificar la estructura y el estilo de una interfaz de usuario, elemento por elemento, sino que se proveen mecanismos de selección que le permiten realizar la especificación tanto para elementos individuales como para grupos de elementos.

Tal como aparece definida en la ISO 9126-1 [ISO, 2001], la usabilidad es una de las características que definen la calidad de un sistema software. La aproximación de Plantillas de Transformación permite establecer guías de uso a fin de guiar a un analista cuando está definiendo parámetros para poder elegir los valores más adecuados para un contexto dado, a fin de obtener interfaces de usuario con buena usabilidad. En una implementación de Plantillas de Transformación, las guías de uso asociadas a un parámetro podrían desplegarse visualmente al analista para que las considere al hacer su elección de valor para el parámetro, o bien, todas o algunas de ellas, podrían ser automatizadas, a fin de que la misma herramienta de edición de Plantillas de Transformación permita o no, asignar ciertos valores a ciertos parámetros en determinados contextos.

Así como la caracterización de modelos de interfaces de usuario, las Plantillas de Transformación han sido modeladas a nivel de meta-meta modelo y de meta modelo.

Las dos Secciones siguientes presentan el meta-meta modelo y el meta modelo de las Plantillas de Transformación, respectivamente.

4.3.1. Meta-meta modelo de Plantillas de Transformación

Como ya se ha mencionado previamente, una Plantilla de Transformación está compuesta por Parámetros que tienen un Valor.

Estos parámetros y valores son conceptos que se sitúan en el nivel de meta modelado de Plantillas de Transformación. En el nivel superior de meta-meta modelado de Plantillas de Transformación, estos conceptos tienen su correspondencia en los Tipos de Parámetro y Tipos de Valor.

La Figura 4.5 ilustra los elementos que representan estos conceptos y otros más que componen el meta-meta modelo de las Plantillas de Transformación. Se ven también las relaciones entre estos elementos, e incluso las relaciones de estos elementos con elementos del meta-meta modelo de interfaces de usuario, del meta modelo de contexto, y del meta modelo de Plantillas de Transformación.

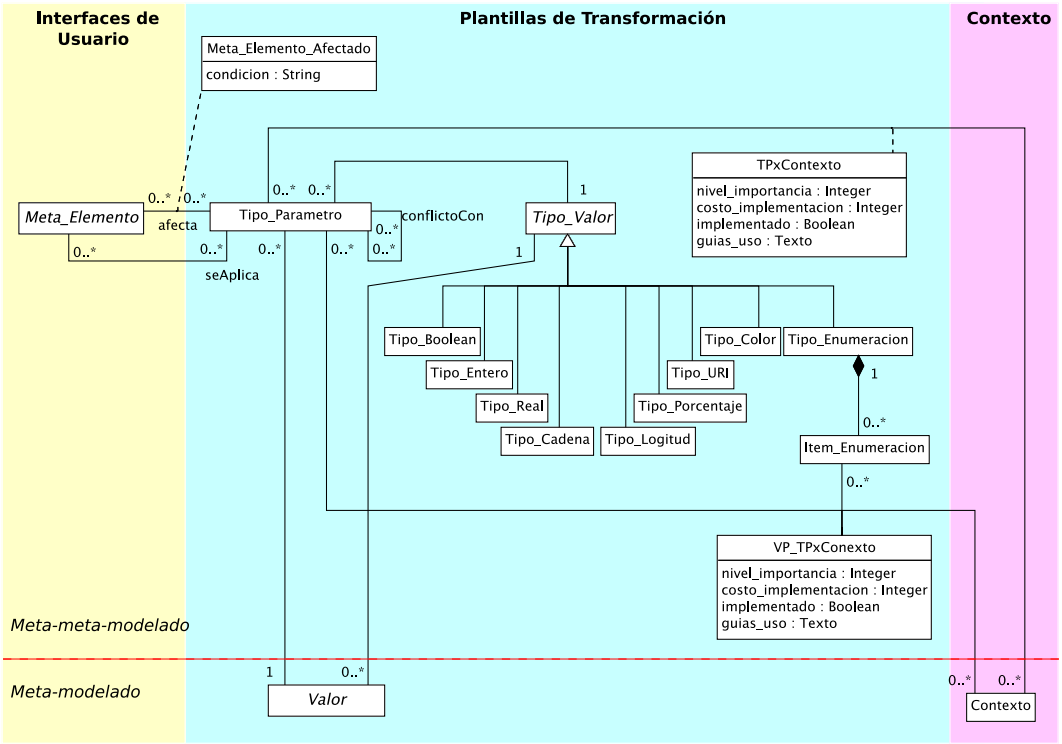


Figura 4.5: Elementos y relaciones del meta-meta modelo de Plantillas de Transformación

Un *Tipo de Parámetro* especifica las características compartidas por sus Parámetros correspondientes.

Con la definición de un Tipo de Parámetro se definen cuáles son los Meta Elementos de los Meta Modelos de Interfaces de Usuario sobre los que el Tipo de Parámetro causa un efecto. Esta información es descriptiva, y puede ser útil para las personas que se encargan de incorporar la aproximación de Plantillas de Transformación a sus compiladores de modelo a modelo o de modelo a código.

Al decir que un Tipo de Parámetro afecta a un conjunto de Meta Elementos, se quiere decir que sus Parámetros correspondientes tendrán un efecto (por ejemplo, un efecto visual en las interfaces de usuario gráficas) sobre los Elementos asociados a los Meta Elementos en cuestión.

Se puede también definir una condición para que el Tipo de Parámetro afecte a

los Elementos correspondientes. Esta condición deberá ser una expresión booleana que puede involucrar a Meta Elementos y a Tipos de Parámetros.

Si se ha especificado que un Tipo de Parámetro TP afecta a un conjunto de Meta Elementos $ME = \{ME_1, ME_2, ..., ME_z\}$ con una condición C entonces un Parámetro P de tipo TP podrá afectar al conjunto de Elementos $E = \{A_1, A_2, ..., A_n, B_1, B_2, ..., B_m, ..., P_1, P_2, ..., P_r\}$ donde A_i está relacionado a ME_1 y cumple C $\forall i$ tal que $1 \leq i \leq n$, B_j está relacionado a ME_2 y cumple C $\forall j$ tal que $1 \leq j \leq m$ y P_K está relacionado a ME_z y cumple C $\forall k$ tal que $1 \leq k \leq r$.

Por otra parte, la definición de un Tipo de Parámetro también especifica los Meta Elementos de los Meta Modelos de Interfaces de Usuario sobre los que el Tipo de Parámetro se puede aplicar.

Un Tipo de Parámetro TP podría afectar a un Meta Elemento ME_1, y ser aplicable a ese mismo Meta Elemento ME_1 y a otros Meta Elementos que pueden contener a ME_1, a cualquier nivel de contención, por ejemplo ME_2. De esta manera, si consideramos que se tiene un Parámetro P de tipo TP, que ME_2 puede contener a ME_1, que E_1 está asociado a ME_1, que E_2 está asociado a ME_2 y que E_2 contiene a E_1, si P se aplica a E_1 su efecto se reflejará directamente en E_1, y si P se aplica a E_2 su efecto se reflejará, igualmente, en E_1.

Si se ha especificado que un Tipo de Parámetro TP se puede aplicar a un conjunto de Meta Elementos $ME = \{ME_1, ME_2, ..., ME_z\}$ entonces un Parámetro P de tipo TP se podrá aplicar al conjunto de Elementos $E = \{A_1, A_2, ..., A_n, B_1, B_2, ..., B_m, ..., P_1, P_2, ..., P_r\}$ donde A_i está relacionado a ME_1 $\forall i$ tal que $1 \leq i \leq n$, B_j está relacionado a ME_2 $\forall j$ tal que $1 \leq j \leq m$ y P_K está relacionado a ME_z $\forall k$ tal que $1 \leq k \leq r$.

Un Tipo de Parámetro puede entrar en conflicto con otros Tipos de Parámetros. Esto puede suceder cuando ambos afectan a un mismo Meta Elemento y le ocasionan el mismo efecto. Por ejemplo, un Tipo de Parámetro TP_1 que afecta a un Meta Elemento ME y determina el tipo de *widget* que será utilizado para representar gráficamente a sus Elementos correspondientes, no entra en conflicto con otro Tipo de Parámetro TP_2 que afecta al mismo Meta Elemento ME pero que determina el tipo de letra que será utilizado en la representación gráfica de sus Elementos correspondientes, mientras que sí entra en conflicto con otro Tipo de Parámetro TP_3 que afecta al mismo Meta Elemento ME y determina, también, el tipo de *widget* que será utilizado para representar gráficamente a sus Elementos correspondientes.

Un Tipo de Parámetro tiene una *Prioridad*. En el caso en el que un Elemento E es afectado por dos Parámetros, P_1 que corresponde a un Tipo de Parámetro TP_1 con prioridad n, y P_2 que corresponde a un Tipo de Parámetro TP_2 que entra en conflicto con TP_1 y tiene prioridad m, entonces si $n > m$, P_1 será el que afecte a E. Si $n < m$ P_2 afectará a E. Y si $n = m$ se puede definir aleatoriamente, o según un criterio decidido por quien implemente la aproximación de Plantillas de Transformación en un compilador de modelo a modelo o de modelo a código.

Por otra parte, un Tipo de Parámetro tiene asociado un *Tipo de Valor* que define el tipo de Valor que los Parámetros del Tipo de Parámetro en cuestión pueden asumir. La noción de Tipo de Valor es semejante a la de tipo de dato, o tipo de objeto (clase).

Un Tipo de Valor es un elemento abstracto. Actualmente, el meta-meta modelo de Plantillas de Transformación considera las siguientes especializaciones de Tipo de Valor: *Tipo Booleano*, *Tipo Entero*, *Tipo Real*, *Tipo Cadena de Caracteres*, *Tipo Longitud*, *Tipo Porcentaje*, *Tipo URI*, *Tipo Color* y *Tipo Enumeración*. Este conjunto abarca a tipos comúnmente utilizados en especificaciones relacionadas a interfaces de usuario [Bos et al., 2007], y a futuro podría ser ampliado.

El Tipo Enumeración tiene asociado un conjunto de *Ítems de Enumeración*.

Un Tipo de Parámetro debe tener un *Valor por defecto*. Este Valor por defecto debe estar asociado al Tipo de Valor del Tipo de Parámetro.

Cuando un Tipo de Parámetro tiene un Tipo de Valor que es un Tipo Enumeración, los Ítems de Enumeración del Tipo Enumeración constituyen el conjunto de valores posibles del Tipo de Parámetro en cuestión.

Cada uno de estos valores posibles es implementable o no en distintos Contextos. Es decir, se podría dar el caso en el que algunos de los valores posibles de un Tipo de Parámetro sean implementables en unos Contextos y no implementables en otros Contextos, por ejemplo, debido a restricciones tecnológicas.

Luego, a partir del conjunto de valores posibles que son implementables en un Contexto dado, se debe decidir cuáles serán, en efecto, implementados. Esta decisión es muy importante debido a que el costo de implementación de cada valor posible de un Tipo de Parámetro puede llegar a ser alto, debido a que implica modificaciones en las herramientas de transformación de modelo a modelo o de modelo a código.

Por lo tanto, es necesario un mecanismo que ayude a decidir si un valor posible de un Tipo de Parámetro será implementado o no para un Contexto, y el orden de la implementación, ya que en una primera etapa se podría implementar un conjunto de valores posibles y los demás en etapas posteriores.

Este trabajo propone un mecanismo sencillo para ayudar a tomar esta decisión. A cada valor posible de un Tipo de Parámetro que es implementable en un Contexto se le asigna una estimación de *Nivel de Importancia* y de *Costo de Implementación*. La estimación de nivel de importancia se debe basar en lo significativo que es el valor posible para el Contexto en cuestión.

En un proceso de desarrollo de interfaces de usuario dirigido por modelos que ya tiene implementados sus compiladores de modelos se pueden identificar nuevos Tipos de Parámetros con sus valores posibles cuando los usuarios finales de las interfaces de usuario generadas piden que las mismas sean modificadas.

En principio, estos cambios se pueden realizar manualmente modificando el

código final de la interfaz de usuario que ha resultado del proceso de desarrollo dirigido por modelos. Sin embargo, esta opción tiene asociados los inconvenientes que han sido discutidos en la Sección 1.1.

Cuando el mismo cambio es solicitado repetidas veces, se puede definir un Tipo de Parámetro con los valores posibles adecuados e implementarlos en los compiladores correspondientes a fin de evitar las modificaciones manuales.

La frecuencia con la que los usuarios piden determinados cambios también debe influir en la estimación del nivel de importancia de cada valor posible del Tipo de Parámetro que se introduce para hacer frente al cambio en cuestión.

Por su parte, el costo de implementación del valor posible del Tipo de Parámetro en un Contexto determinado debe ser estimado por un programador que tenga experiencia programando para el Contexto en cuestión.

Las estimaciones de nivel de importancia y de costo de implementación pueden utilizar una escala de 1 a 5, siendo 1 un valor bajo, y 5 un valor alto.

Como la Figura 4.6 sugiere, los valores posibles con un alto nivel de importancia y un bajo costo de implementación pueden implementarse en una primera etapa, seguidos de los valores posibles que tienen alta importancia y costo de implementación, en la segunda etapa. En la tercera etapa se pueden implementar los valores posibles con baja importancia y costo de implementación, y en la cuarta y última etapa, los valores posibles con baja importancia y alto costo de implementación. Los valores posibles de la cuarta etapa, podrían, incluso, no ser implementados. Así como también se podría decidir implementar solo los valores posibles de la primera etapa, o los de la primera y segunda, etc.

Por otra parte, a un valor posible de un Tipo de Parámetro en un Contexto dado, se le puede asignar una *Guía de Uso*, que sirva al diseñador que está creando una Plantilla de Transformación para el Contexto en cuestión, para elegir el valor a asignar a un Parámetro del Tipo de Parámetro en cuestión. Estas guías deben orientar la selección de los valores a fin de obtener interfaces de usuario finales con buena usabilidad.

Ya se ha visto como se relacionan los valores posibles de un Tipo de Parámetro con el Contexto. Se verá ahora como un Tipo de Parámetro se relaciona de manera similar con un Contexto.

Un Tipo de Parámetro es implementable o no en distintos Contextos. Un Tipo de Parámetro podría no ser implementable en un Contexto debido, por ejemplo, a restricciones tecnológicas.

A partir del conjunto de Tipos de Parámetros implementables en un Contexto dado, se debe decidir cuáles serán, en efecto, implementados.

Para el caso de los Tipos de Parámetros que tienen un Tipo de Valor que no es un Tipo Enumeración, el proceso que ayuda a decidir los Tipos de Parámetros a ser implementados en un Contexto y el orden de la implementación, es exactamente el mismo que el utilizado para los valores posibles de los Tipos de Parámetros que sí tienen un Tipo de Valor que es un Tipo Enumeración. Es decir,

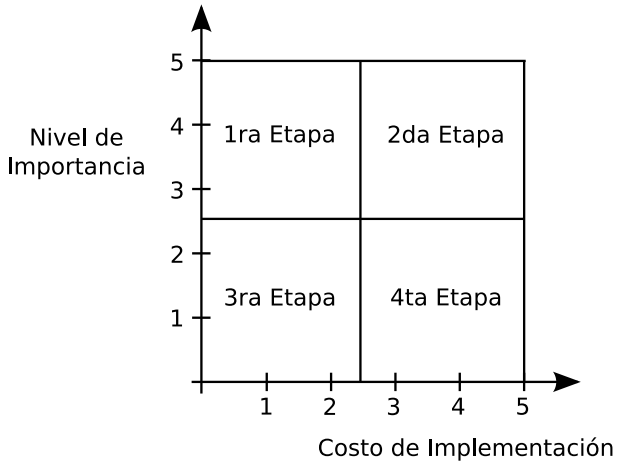


Figura 4.6: Etapas propuestas para la implementación de valores posibles de Tipos de Parámetros y de Tipos de Parámetros en un Contexto, basadas en estimaciones de Nivel de Importancia y de Costo de Implementación

a cada Tipo de Parámetro que es implementable en un Contexto se le asignan las estimaciones de nivel de importancia y de costo de implementación siguiendo los mismos criterios que en el caso de estas estimaciones para los valores posibles de un Tipo de Parámetro en un Contexto. Y de manera análoga a como en ese caso, los Tipos de Parámetros con un alto nivel de importancia y un bajo costo de implementación pueden implementarse en una primera etapa, seguidos de los Tipos de Parámetro que tienen alta importancia y costo de implementación, en la segunda etapa. En la tercera etapa se pueden implementar los Tipos de Parámetro con baja importancia y costo de implementación, y en la cuarta y última etapa, los Tipos de Parámetro con baja importancia y alto costo de implementación (ver Figura 4.6).

En el caso de los Tipos de Parámetros que sí tienen un Tipo de Valor que es un Tipo Enumeración, los datos de su relación con un Contexto se derivan de los datos de sus valores posibles en ese Contexto. Es decir, un Tipo de Parámetro que tiene un Tipo de Valor que es un Tipo Enumeración es implementable en un Contexto si alguno de sus valores posibles es implementable en ese Contexto. Y sus estimaciones de nivel de importancia y de costo de implementación deben ser una agregación de los niveles de importancia y costos de implementación de sus valores posibles, respectivamente. Asimismo, un Tipo de Parámetro que tiene un Tipo de Valor que es un Tipo Enumeración se considera implementado en un Contexto si alguno de sus valores posibles está implementado para ese Contexto.

Finalmente, un Tipo de Parámetro puede tener asociadas unas *Guías de Uso*

para un Contexto dado. Estas guías pueden aconsejar qué valores asociar a los Parámetros del Tipo de Parámetro en cuestión a fin de obtener interfaces de usuario con una usabilidad adecuada en el Contexto en cuestión.

A continuación se describe cada uno de los elementos del meta-meta modelo de Plantillas de Transformación, junto con sus propiedades, relaciones, servicios y restricciones.

Elemento: Tipo de Valor

- Nombre del elemento: Tipo_Valor
- Alias: Tipo de Valor
- Descripción: representa un tipo de valor. Este concepto es similar al de tipo de datos.

  El elemento Tipo de Valor es abstracto. Más adelante se presentarán los elementos Tipo Booleano, Tipo Entero, Tipo Real, Tipo Cadena de Caracteres, Tipo Longitud, Tipo Porcentaje, Tipo URI, Tipo Color y Tipo Enumeración, todos ellos, especializaciones de Tipo de Valor.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único de un tipo de valor
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
- Servicios:
  - Nuevo(String pIdentificador): Crea un Tipo de Valor asignando a la propiedad Identificador del nuevo Tipo de Valor, el valor del argumento pIdentificador.
  - Eliminar(Tipo_Valor THIS): elimina el Tipo de Valor actual, referenciado por THIS.

Elemento: Tipo Booleano

- Nombre del elemento: Tipo_Booleano
- Alias: Tipo Booleano

- Descripción: representa el tipo de valor booleano.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo Booleano especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Booleano correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo Booleano.

Elemento: Tipo Entero

- Nombre del elemento: Tipo_Entero
- Alias: Tipo Entero
- Descripción: representa el tipo de valor entero.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo Entero especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Entero correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo Entero.

Elemento: Tipo Real

- Nombre del elemento: Tipo_Real
- Alias: Tipo Real
- Descripción: representa el tipo de valor real.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo Real especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Real correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo Real.

Elemento: Tipo Cadena de Caracteres

- Nombre del elemento: Tipo_Cadena
- Alias: Tipo Cadena de Caracteres
- Descripción: representa el tipo de valor de cadena de caracteres.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo Cadena de Caracteres especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Cadena de Caracteres correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo Cadena de Caracteres.

Elemento: Tipo Longitud

- Nombre del elemento: Tipo_Longitud
- Alias: Tipo Longitud
- Descripción: representa el tipo de valor de longitud.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo Longitud especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Longitud correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo Longitud.

Elemento: Tipo Porcentaje

- Nombre del elemento: Tipo_Porcentaje
- Alias: Tipo Porcentaje
- Descripción: representa el tipo de valor de porcentaje.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo Porcentaje especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Porcentaje correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo Porcentaje.

Elemento: Tipo URI

- Nombre del elemento: Tipo_URI
- Alias: Tipo URI
- Descripción: representa el tipo de valor de URI.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo URI especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo URI correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo URI.

Elemento: Tipo Color

- Nombre del elemento: Tipo_Color
- Alias: Tipo Color
- Descripción: representa el tipo de valor de Color.
- Propiedades: la que hereda de Tipo de Valor: Identificador.
- Relaciones:
  - Tipo Color especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Color correspondiente.
- Observaciones: se debe poder crear solo una instancia del elemento Tipo Color.

Elemento: Tipo Enumeración

- Nombre del elemento: Tipo_Enumeracion
- Alias: Tipo Enumeración
- Descripción: representa el tipo de valor Enumeración.
- Propiedades: la que hereda de Tipo de Valor: Identificador.

  Además agrega la siguiente propiedad:

  - Alias: Nombre
  - Descripción: Nombre de la enumeración
  - Tipo de propiedad: Variable
  - Tipo de dato: String
  - Es identificador: no
  - Acepta nulo: no
- Relaciones:

  - Tipo Enumeración especializa a Tipo de Valor.
- Servicios: el que hereda de Tipo de Valor: Eliminar.

  Además, se definen los siguientes servicios:

  - Nuevo(String pIdentificador, String pNombre): primero crea un Tipo de Valor invocando su servicio de creación Nuevo y pasándole el argumento pIdentificador. Luego crea el Tipo Enumeración correspondiente, y asigna a la propiedad Nombre del nuevo Tipo Enumeración, el valor del argumento pNombre.
  - Editar(Tipo_Enumeracion THIS, String pNombre): asigna a la propiedad Nombre del Tipo Enumeración actual, referenciado por THIS, el valor del argumento pNombre.

Elemento: Ítem de Enumeración

- Nombre del elemento: Item_Enumeracion
- Alias: Ítem de Enumeración
- Descripción: constituye uno de los valores o ítems de un tipo enumeración.
- Propiedades:

- Propiedad 1
  - Alias: Identificador
  - Descripción: Identificador único de un ítem de un tipo enumeración
  - Tipo de propiedad: Constante
  - Tipo de dato: String
  - Es identificador: si
  - Acepta nulo: no
- Propiedad 2
  - Alias: Nombre
  - Descripción: Nombre del ítem del tipo enumeración
  - Tipo de propiedad: Variable
  - Tipo de dato: String
  - Es identificador: no
  - Acepta nulo: no

- Relaciones:
  - Un Ítem de Enumeración corresponde (se asocia a) un Tipo Enumeración.

    Un Tipo Enumeración se compone de 0 o muchos Ítems de Enumeración.

- Servicios:
  - Nuevo(String pIdentificador, String pNombre, Tipo_Enumeracion pTipoEnumeracion): crea un Ítem de Enumeración asignando a las propiedades Identificador y Nombre el valor de los argumentos pIdentificador y pNombre, respectivamente. Además, asocia el Ítem de Enumeración creado al Tipo Enumeración especificado por el argumento pTipoEnumeracion.
  - Editar(Item_Enumeracion THIS, String pNombre): asigna a la propiedad Nombre del Ítem de Enumeración actual, referenciado por THIS, el valor del argumento pNombre.
  - Eliminar(Item_Enumeracion THIS): elimina el Ítem de Enumeración actual, referenciado por THIS.

Elemento: Tipo de Parámetro

- Nombre del elemento: Tipo_Parametro
- Alias: Tipo de Parámetro
- Descripción: caracteriza a los Parámetros que le corresponden. Define cuáles son los Meta Elementos sobre los que el Tipo de Parámetro tiene un efecto, y sobre cuáles se puede aplicar. Define los Tipos de Parámetros con los que se pueden tener conflictos. Define también el Tipo de Valor y Valor por defecto para el Tipo de Parámetro. Además, permite definir si un valor posible de un Tipo de Parámetro, y si el Tipo de Parámetro mismo son implementables o no en un Contexto dado. Para los valores posibles de un Tipo de Parámetro y los Tipos de Parámetro que son implementables en un Contexto, permite estimar sus niveles de importancia y costo de implementación con respecto al Contexto en cuestión, e indicar si están, efectivamente, implementados o no en el Contexto.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: identificador único de un tipo de parámetro.
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Nombre
    - Descripción: nombra al tipo de parámetro. Es un nombre orientado a ser utilizado por herramientas software.
    - Tipo de propiedad: Variable
    - Tipo de dato: String
    - Es identificador: no
    - Acepta nulo: no
  - Propiedad 3
    - Alias: Alias
    - Descripción: constituye una alternativa al nombre, y está orientado a ser utilizado por personas.
    - Tipo de propiedad: Variable

        - Tipo de dato: String
        - Es identificador: no
        - Acepta nulo: si
    - Propiedad 4
        - Alias: Descripción
        - Descripción: describe el tipo de parámetro.
        - Tipo de propiedad: Variable
        - Tipo de dato: Texto
        - Es identificador: no
        - Acepta nulo: si
    - Propiedad 5
        - Alias: Comentarios
        - Descripción: añade comentarios relacionados al tipo de parámetro, por ejemplo, notas relacionadas a su implementación.
        - Tipo de propiedad: Variable
        - Tipo de dato: Texto
        - Es identificador: no
        - Acepta nulo: si
    - Propiedad 6
        - Alias: Prioridad
        - Descripción: establece una prioridad para el Tipo de Parámetro.
        - Tipo de propiedad: Variable
        - Tipo de dato: Natural, restringido a las siguientes opciones: 1 (muy baja), 2 (baja), 3 (media), 4 (alta) y 5 (muy alta).
        - Es identificador: no
        - Acepta nulo: no
        - Valor por defecto: 1 (muy baja)

- Relaciones:

    - Un Tipo de Parámetro puede afectar (asociación) a 0 o muchos Meta Elementos de Meta Modelos de Interfaces de Usuario.

      Un Meta Elemento puede ser afectado (asociación) por 0 o muchos Tipos de Parámetros.

      Todos los Meta Elementos afectados por un Tipo de Parámetro deben ser distintos unos de otros.

      Esta relación tiene, además, la siguiente propiedad:

  - Alias: Condición
  - Descripción: Condición que se debe cumplir para que el Tipo de Parámetro afecte a los Elementos correspondientes al Meta Elemento de la relación. Esta condición deberá conformar una expresión booleana que puede involucrar a Meta Elementos y Tipos de Parámetros.
  - Tipo de propiedad: Variable
  - Tipo de dato: String
  - Es identificador: no
  - Acepta nulo: si

- Un Tipo de Parámetro se puede aplicar (asociación) a 0 o muchos Meta Elementos de Meta Modelos de Interfaces de Usuario.

  Un Meta Elemento puede recibir la aplicación (asociación) de 0 o muchos Tipos de Parámetros.

  Todos los Meta Elementos que pueden recibir la aplicación de un Tipo de Parámetro deben ser distintos unos de otros.

- Un Tipo de Parámetro puede tener conflicto (asociación) con 0 o muchos Tipos de Parámetros.

  Un Tipo de Parámetro puede estar en conflicto (asociación) con 0 o muchos Tipos de Parámetros.

  Todos los Tipos de Parámetros que pueden estar en conflicto deben ser distintos unos de otros.

  Una condición necesaria, pero no suficiente, para que un Tipo de Parámetro esté en conflicto con otro Tipo de Parámetro, es que ambos afecten a un mismo Meta Elemento.

- Un Tipo de Parámetro tiene (asociación) un Tipo de Valor.

  Un Tipo de Valor puede estar asociado a 0 o muchos Tipos de Parámetros.

- Un Tipo de Parámetro tiene (asociación) un Valor por defecto.

  Un Valor puede ser el Valor por defecto (asociación) de 0 o muchos Tipos de Parámetros.

  El Valor por defecto de un Tipo de Parámetro debe estar asociado a su Tipo de Valor.

- Un Tipo de Parámetro puede estar asociado a 0 o muchos Ítems de Enumeración.

  Un Ítem de Enumeración puede estar asociado a 0 o muchos Tipos de Parámetros.

Los Ítems de Enumeración que se asocian a un Tipo de Parámetro deben ser distintos unos de otros.

Los Ítems de Enumeración que se pueden asociar a un Tipo de Parámetro son aquellos asociados al Tipo Enumeración que constituye el Tipo de Valor del Tipo de Parámetro. Por lo tanto, los Ítems de Enumeración que se asocian a un Tipo de Parámetro son sus valores posibles.

Cada Ítem de Enumeración asociado a un Tipo de Parámetro se puede implementar (asociación) en 0 o muchos Contextos.

En un Contexto pueden ser implementados 0 o muchos Ítems de Enumeración asociados a un Tipo de Parámetro.

Los Contextos en los que puede ser implementado un Ítem de Enumeración asociado a un Tipo de Parámetro deben ser distintos unos de otros.

Esta relación ternaria tiene, además, las siguientes propiedades:

- Propiedad 1
  - Alias: Nivel de Importancia
  - Descripción: Estimación del nivel de importancia del valor posible (Ítem de Enumeración) del Tipo de Parámetro con respecto al Contexto.
  - Tipo de propiedad: Variable
  - Tipo de dato: Natural, restringido a las siguientes opciones: 1 (muy bajo), 2 (bajo), 3 (medio), 4 (alto) y 5 (muy alto).
  - Es identificador: no
  - Acepta nulo: no
  - Valor por defecto: 1 (muy bajo)
- Propiedad 2
  - Alias: Costo de Implementación
  - Descripción: Estimación del costo de implementación del valor posible (Ítem de Enumeración) del Tipo de Parámetro con respecto al Contexto.
  - Tipo de propiedad: Variable
  - Tipo de dato: Natural, restringido a las siguientes opciones: 1 (muy bajo), 2 (bajo), 3 (medio), 4 (alto) y 5 (muy alto).
  - Es identificador: no
  - Acepta nulo: no
  - Valor por defecto: 1 (muy bajo)
- Propiedad 3

- ◇ Alias: Implementado
- ◇ Descripción: Indica si el valor posible (Ítem de Enumeración) del Tipo de Parámetro está implementado o no para el Contexto.
- ◇ Tipo de propiedad: Variable
- ◇ Tipo de dato: Booleano
- ◇ Es identificador: no
- ◇ Acepta nulo: no
- ◇ Valor por defecto: FALSO

- ○ Propiedad 4
  - ◇ Alias: Guías de Uso
  - ◇ Descripción: Proporciona guías para el uso adecuado del valor posible (Ítem de Enumeración) del Tipo de Parámetro en el Contexto, por ejemplo, recomienda usar o no el valor posible del Tipo de Parámetro en el Contexto, a fin de lograr una usabilidad adecuada para el Contexto.
  - ◇ Tipo de propiedad: Variable
  - ◇ Tipo de dato: Texto
  - ◇ Es identificador: no
  - ◇ Acepta nulo: si

- Un Tipo de Parámetro se puede implementar (asociación) en 0 o muchos Contextos.

  En un Contexto pueden estar implementados (asociación) 0 o muchos Tipos de Parámetros.

  Todos los Contextos en los que se puede implementar un Tipo de Parámetro deben ser distintos unos de otros.

  Cuando el Tipo de Parámetro tiene un Tipo de Valor que es un Tipo Enumeración, el conjunto de Contextos en los que el Tipo de Parámetro puede ser implementado deberá coincidir con el conjunto de Contextos en los que los valores posibles (Ítems de Enumeración) del Tipo de Parámetro pueden ser implementados.

  Esta relación tiene, además, las siguientes propiedades:

  - ○ Propiedad 1
    - ◇ Alias: Nivel de Importancia
    - ◇ Descripción: Estimación del nivel de importancia del Tipo de Parámetro con respecto al Contexto. Cuando el Tipo de Parámetro tiene un Tipo de Valor que es un Tipo Enumeración, su estimación de nivel de importancia con

respecto a un Contexto, será una función de las estimaciones del nivel de importancia de cada uno de sus valores posibles (Ítems de Enumeración) con respecto al Contexto en cuestión.
    - ◇ Tipo de propiedad: Variable
    - ◇ Tipo de dato: Natural, restringido a las siguientes opciones: 1 (muy bajo), 2 (bajo), 3 (medio), 4 (alto) y 5 (muy alto).
    - ◇ Es identificador: no
    - ◇ Acepta nulo: no
    - ◇ Valor por defecto: 1 (muy bajo)
  - ○ Propiedad 2
    - ◇ Alias: Costo de Implementación
    - ◇ Descripción: Estimación del costo de implementación del Tipo de Parámetro con respecto al Contexto. Cuando el Tipo de Parámetro tiene un Tipo de Valor que es un Tipo Enumeración, su estimación de costo de implementación con respecto a un Contexto, será una función de las estimaciones del costo de implementación de cada uno de sus valores posibles (Ítems de Enumeración) con respecto al Contexto en cuestión.
    - ◇ Tipo de propiedad: Variable
    - ◇ Tipo de dato: Natural, restringido a las siguientes opciones: 1 (muy bajo), 2 (bajo), 3 (medio), 4 (alto) y 5 (muy alto).
    - ◇ Es identificador: no
    - ◇ Acepta nulo: no
    - ◇ Valor por defecto: 1 (muy bajo)
  - ○ Propiedad 3
    - ◇ Alias: Implementado
    - ◇ Descripción: Indica si el Tipo de Parámetro está o no implementado para el Contexto. Cuando el Tipo de Parámetro tiene un Tipo de Valor que es un Tipo Enumeración, el Tipo de Parámetro estará implementado para el Contexto si alguno de sus valores posibles (Ítems de Enumeración) está implementado para el Contexto en cuestión.
    - ◇ Tipo de propiedad: Variable
    - ◇ Tipo de dato: Booleano
    - ◇ Es identificador: no
    - ◇ Acepta nulo: no
    - ◇ Valor por defecto: FALSO

- Propiedad 4
  - Alias: Guías de Uso
  - Descripción: Proporciona guías para el uso adecuado del Tipo de Parámetro en el Contexto, por ejemplo, aconseja qué valores asociar a un Parámetro del Tipo de Parámetro en cuestión a fin de lograr una usabilidad adecuada para el Contexto.
  - Tipo de propiedad: Variable
  - Tipo de dato: Texto
  - Es identificador: no
  - Acepta nulo: si

- Servicios:

  - Nuevo(String pIdentificador, String pNombre, String pAlias, Texto pDescripcion, Texto pComentarios, Natural pPrioridad, Tipo_Valor pTipoValor, Valor pValorDefecto): crea un Tipo de Parámetro asignando a las propiedades Identificador, Nombre, Alias, Descripción, Comentarios y Prioridad del nuevo Tipo de Parámetro, el valor de los argumentos pIdentificador, pNombre, pAlias, pDescripcion, pComentarios y pPrioridad, respectivamente. Además, asocia el Tipo de Valor especificado por el argumento pTipoValor, y el Valor por defecto especificado por el argumento pValorDefecto, al Tipo de Parámetro creado.

    pValorDefecto debe ser un Valor asociado al Tipo de Valor referenciado por el argumento pTipoValor.

  - Editar(Tipo_Parametro THIS, String pNombre, String pAlias, Texto pDescripcion, Texto pComentarios, Natural pPrioridad): asigna a las propiedades Nombre, Alias, Descripción, Comentarios y Prioridad del Tipo de Parámetro actual, referenciado por THIS, el valor de los argumentos pNombre, pAlias, pDescripcion, pComentarios y pPrioridad, respectivamente.

  - Eliminar(Tipo_Parametro THIS): elimina el Tipo de Parámetro actual, referenciado por THIS.

  - Agregar_Meta_Elemento_Afectado(Tipo_Parametro THIS, Meta_Elemento pMetaElemento, String pCondicion): asocia, con el papel de afectado, el Meta Elemento especificado por el argumento pMetaElemento al Tipo de Parámetro actual, referenciado por THIS. Además, establece la condición (especificada por el argumento pCondicion y asignada a la propiedad Condición de la relación) que se debe

cumplir para que el Tipo de Parámetro actual afecte, efectivamente, al Meta Elemento referenciado por pMetaElemento.

pMetaElemento debe ser un Meta Elemento no afectado aún por el Tipo de Parámetro actual.

- Quitar_Meta_Elemento_Afectado(Tipo_Parametro THIS, Meta_Elemento pMetaElementoAfectado): desasocia, en su papel de afectado, el Meta Elemento especificado por el argumento pMetaElementoAfectado del Tipo de Parámetro actual, referenciado por THIS.

  pMetaElementoAfectado debe ser un Meta Elemento afectado por el Tipo de Parámetro actual.

- Cambiar_Condicion_Meta_Elemento_Afectado(Tipo_Parametro THIS, Meta_Elemento pMetaElementoAfectado, String pCondicion): asigna a la propiedad Condición de la relación entre el Tipo de Parámetro actual, referenciado por THIS, y el Meta Elemento referenciado por pMetaElementoAfectado, el valor del argumento pCondicion.

  pMetaElementoAfectado debe ser un Meta Elemento afectado por el Tipo de Parámetro actual.

- Agregar_Meta_Elemento_Aplicable(Tipo_Parametro THIS, Meta_Elemento pMetaElemento): asocia, con el papel de aplicable, el Meta Elemento especificado por el argumento pMetaElemento al Tipo de Parámetro actual, referenciado por THIS.

  pMetaElemento debe ser un Meta Elemento al cual el Tipo de Parámetro actual no es aplicable aún.

- Quitar_Meta_Elemento_Aplicable(Tipo_Parametro THIS, Meta_Elemento pMetaElementoAplicable): desasocia, en su papel de aplicable, el Meta Elemento especificado por el argumento pMetaElementoAplicable del Tipo de Parámetro actual, referenciado por THIS.

  pMetaElementoAplicable debe ser un Meta Elemento al cual el Tipo de Parámetro actual es aplicable.

- Agregar_Tipo_Parametro_Conflicto(Tipo_Parametro THIS, Tipo_Parametro pTipoParametro): asocia, con el papel de conflictivo, el Tipo de Parámetro especificado por el argumento pTipoParametro al Tipo de Parámetro actual, referenciado por THIS.

  pTipoParametro debe ser un Tipo de Parámetro con el cual el Tipo de Parámetro actual no tiene conflicto aún.

  pTipoParametro debe afectar a un Meta Elemento afectado también por el Tipo de Parámetro actual.

- Quitar_Tipo_Parametro_Conflicto(Tipo_Parametro THIS, Tipo_Parametro pTipoParametroConflicto): desasocia, en su papel

de conflictivo, el Tipo de Parámetro especificado por el argumento pTipoParametroConflicto del Tipo de Parámetro actual, referenciado por THIS.

pTipoParametroConflicto debe ser un Tipo de Parámetro con el cual el Tipo de Parámetro puede tener un conflicto.

- Agregar_Contexto_Implementacion_Valor_Posible(Tipo_Parametro THIS, Item_Enumeracion pValorPosible, Contexto pContexto, Natural pNivelImportancia, Natural pCostoImplementacion, Booleano pImplementado, Texto pGuiasUso): asocia, con el papel de implementable, el Contexto especificado por el argumento pContexto al Ítem de Enumeración especificado por el argumento pValorPosible, el cual es un valor posible del Tipo de Parámetro actual, referenciado por THIS. Además, especifica el nivel de importancia, costo de implementación, indicador de implementación y guías de uso (especificados por los argumentos pNivelImportancia, pCostoImplementacion, pImplementado y pGuiasUso, y asignados a las propiedades Nivel de Importancia, Costo de Implementación, Implementado y Guías de Uso, respectivamente) del valor posible (Ítem de Enumeración) del Tipo de Parámetro actual con respecto al Contexto referenciado por pContexto.

  pValorPosible debe ser un Ítem de Enumeración asociado a un Tipo Enumeración que es el Tipo de Valor del Tipo de Parámetro actual.

  La dupla pValorPosible y pContexto no debe estar asociada aún al Tipo de Parámetro actual.

- Quitar_Contexto_Implementacion_Valor_Posible(Tipo_Parametro THIS, Item_Enumeracion pValorPosible, Contexto pContextoImplementacion): desasocia, en su papel de implementable, el Contexto especificado por el argumento pContextoImplementacion del valor posible (Ítem de Enumeración referenciado por pValorPosible) del Tipo de Parámetro actual, referenciado por THIS.

  La dupla pValorPosible y pContextoImplementacion debe estar asociada al Tipo de Parámetro actual.

- Cambiar_Informacion_Contexto_Implementacion_Valor_Posible(Tipo_Parametro THIS, Item_Enumeracion pValorPosible, Contexto pContextoImplementacion, Natural pNivelImportancia, Natural pCostoImplementacion, Booleano pImplementado, Texto pGuiasUso): asigna a las propiedades Nivel de Importancia, Costo de Implementación, Implementado y Guías de Uso de la relación entre el valor posible (Ítem de Enumeración referenciado por pValorPosible) del Tipo de Parámetro actual, referenciado por THIS, y el Contexto referenciado por pContextoImplementacion, el valor de los argumentos

pNivelImportancia, pCostoImplementacion, pImplementado y pGuiasUso, respectivamente.

La dupla pValorPosible y pContextoImplementacion debe estar asociada al Tipo de Parámetro actual.

- Agregar_Contexto_Implementacion(Tipo_Parametro THIS, Contexto pContexto, Natural pNivelImportancia, Natural pCostoImplementacion, Booleano pImplementado, Texto pGuiasUso): asocia, con el papel de implementable, el Contexto especificado por el argumento pContexto al Tipo de Parámetro actual, referenciado por THIS. Además, especifica el nivel de importancia, costo de implementación, indicador de implementación y guías de uso (especificados por los argumentos pNivelImportancia, pCostoImplementacion, pImplementado y pGuiasUso, y asignados a las propiedades Nivel de Importancia, Costo de Implementación, Implementado y Guías de Uso, respectivamente) del Tipo de Parámetro actual con respecto al Contexto referenciado por pContexto.

  pContexto debe ser un Contexto en el cual el Tipo de Parámetro actual no es implementable aún.

  Si el Tipo de Parámetro actual tiene un Tipo de Valor que es un Tipo Enumeración, pContexto deberá ser uno de los Contextos en los que pueden ser implementados los valores posibles (Ítems de Enumeración) del Tipo de Parámetro actual. Además, las estimaciones de nivel de importancia y de costo de implementación del Tipo de Parámetro actual con respecto al Contexto pContexto deberán corresponder a una función de las estimaciones del nivel de importancia y de costo de implementación de cada uno de los valores posibles del Tipo de Parámetro actual con respecto al Contexto pContexto. Por su parte, el indicador de implementación del Tipo de Parámetro actual con respecto al Contexto pContexto será VERDADERO si alguno de los valores posibles del Tipo de Parámetro actual está implementado para el Contexto pContexto.

- Quitar_Contexto_Implementacion(Tipo_Parametro THIS, Contexto pContextoImplementacion): desasocia, en su papel de implementable, el Contexto especificado por el argumento pContextoImplementacion del Tipo de Parámetro actual, referenciado por THIS.

  pContextoImplementacion debe ser un Contexto en el cual el Tipo de Parámetro actual es implementable.

- Cambiar_Informacion_Contexto_Implementacion(Tipo_Parametro THIS, Contexto pContextoImplementacion, Natural pNivelImportancia, Natural pCostoImplementacion, Booleano pImplementado, Texto

pGuiasUso): asigna a las propiedades Nivel de Importancia, Costo de Implementación, Implementado y Guías de Uso de la relación entre el Tipo de Parámetro actual, referenciado por THIS, y el Contexto referenciado por pContextoImplementacion, el valor de los argumentos pNivelImportancia, pCostoImplementacion, pImplementado y pGuiasUso, respectivamente.

pContextoImplementacion debe ser un Contexto en el cual el Tipo de Parámetro actual es implementable.

Si el Tipo de Parámetro actual tiene un Tipo de Valor que es un Tipo Enumeración, pContextoImplementacion deberá ser uno de los Contextos en los que pueden ser implementados los valores posibles (Ítems de Enumeración) del Tipo de Parámetro actual. Además, las estimaciones de nivel de importancia y de costo de implementación del Tipo de Parámetro actual con respecto al Contexto pContextoImplementacion deberán corresponder a una función de las estimaciones del nivel de importancia y de costo de implementación de cada uno de los valores posibles del Tipo de Parámetro actual con respecto al Contexto pContextoImplementacion. Por su parte, el indicador de implementación del Tipo de Parámetro actual con respecto al Contexto pContextoImplementacion será VERDADERO si alguno de los valores posibles del Tipo de Parámetro actual está implementado para el Contexto pContextoImplementacion.

4.3.2. Meta modelo de Plantillas de Transformación

En esta Sección se presenta el meta modelo de las Plantillas de Transformación.

La Figura 4.7 ilustra los elementos y las relaciones entre los elementos del meta modelo de Plantillas de Transformación. Ilustra, también, las relaciones de los elementos de este meta modelo con los elementos de los meta-meta modelos de Plantillas de Transformación y de interfaces de usuario, y con los meta modelos de interfaces de usuario y de contexto.

Una *Plantilla de Transformación* contiene un conjunto de *Parámetros* que definen aspectos de estructura y de estilo de interfaces de usuario. Una Plantilla de Transformación es utilizada por un compilador que transforma un modelo de interfaz de usuario más abstracto a uno más concreto o a código. El compilador toma como entradas al modelo más abstracto y a la Plantilla de Transformación, y los Parámetros de la Plantilla de Transformación proveen al compilador las directivas para realizar el proceso de transformación.

Una Plantilla de Transformación se prepara para un Contexto de uso y se puede asociar, o no, a un Modelo de Interfaz de Usuario.

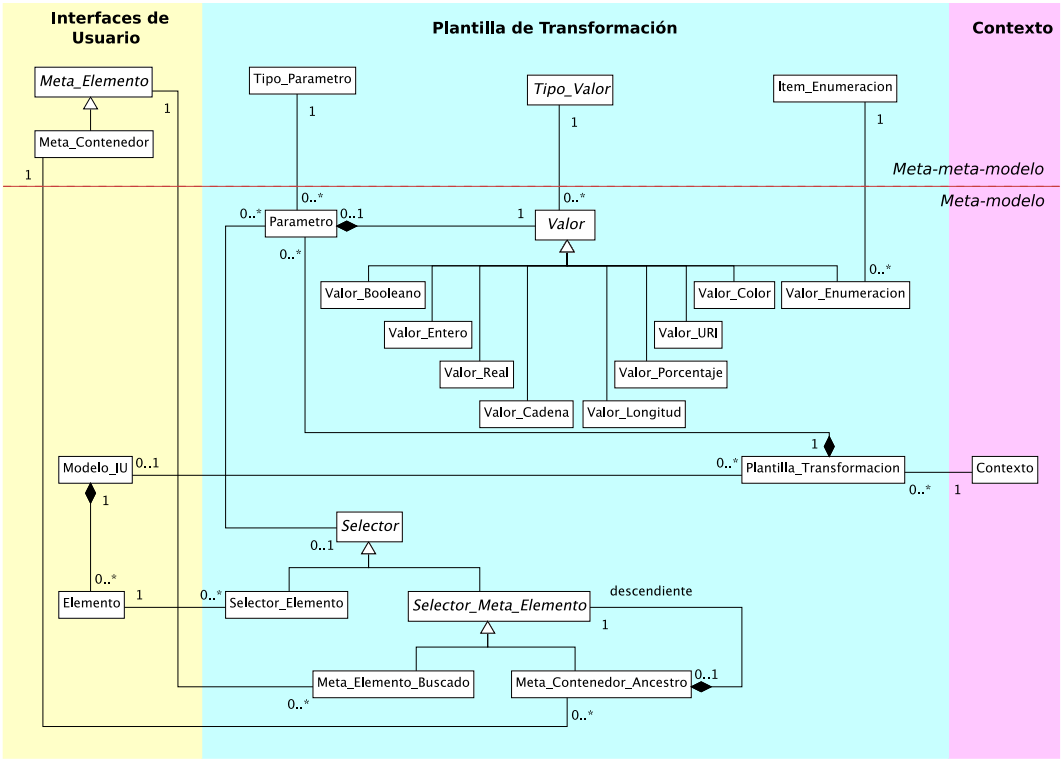


Figura 4.7: Elementos y relaciones del meta modelo de Plantillas de Transformación

Una Plantilla de Transformación está compuesta por Parámetros. Los Parámetros corresponden a un Tipo de Parámetro, tienen un Valor y pueden tener un Selector.

A continuación se explican los conceptos de Valor y de Selector.

El *Valor* que se podrá asociar a un Parámetro de una Plantilla de Transformación corresponde a un Tipo de Valor del nivel de meta-meta modelo de Plantillas de Transformación. Las nociones de Tipo de Valor y Valor son semejantes a las de tipo de dato y dato, o tipo de objeto (clase) y objeto.

Un Valor es un elemento abstracto. Actualmente, el meta modelo de Plantillas de Transformación considera las siguientes especializaciones de Valor: *Valor Booleano*, *Valor Entero*, *Valor Real*, *Valor Cadena de Caracteres*, *Valor Longitud*, *Valor Porcentaje*, *Valor URI*, *Valor Color* y *Valor Enumeración*.

Cada uno de estos valores se corresponden con un Tipo de Valor que se ha definido en el nivel de meta-meta modelo de Plantillas de Transformación.

El Valor Booleano, Valor Entero, Valor Real y Valor Cadena de Caracteres, permiten especificar un valor booleano, entero, real y de cadena de caracteres, respectivamente.

El Valor Longitud permite especificar un número real que representa la longitud, y una unidad de medida: “em”, “ex”, “px”, “in”, “cm”, “mm”, “pt” o “pc”.

El Valor Porcentaje permite especificar un número real que representa un porcentaje.

El Valor URI especifica una cadena de caracteres que es un URI.

El Valor Color permite especificar una cadena de caracteres que es un color en notación hexadecimal RGB.

El Valor Enumeración se asocia a un Ítem de Enumeración, que constituye su valor. Este Ítem de Enumeración debe estar relacionado al Tipo Enumeración del Valor Enumeración.

El Valor que se asigna a un Parámetro debe estar relacionado al Tipo de Valor del Tipo de Parámetro que corresponde al Parámetro en cuestión.

Cuando se asigna a un Parámetro un Valor que es un Valor Enumeración, el Ítem de Enumeración asociado debe estar implementado para el Contexto de la Plantilla de Transformación de la cual forma parte el Parámetro. Si el Valor no es un Valor Enumeración, basta con que el Tipo de Parámetro correspondiente al Parámetro esté implementado para el Contexto de la Plantilla de Transformación.

Un *Selector* tiene la intención de acotar el conjunto de Elementos de un Modelo de Interfaz de Usuario sobre los que se aplica un Parámetro de una Plantilla de Transformación. Esto, a su vez, acota el conjunto de Elementos que son afectados por el Valor del Parámetro.

Un Parámetro corresponde a un Tipo de Parámetro. Este Tipo de Parámetro, como se ha explicado en la Sección 4.5, define un conjunto de Meta Elementos cuyos Elementos pueden ser afectados por los Parámetros del Tipo de Parámetro en cuestión. Define también, un conjunto de Meta Elementos sobre cuyos Elementos se pueden aplicar los Parámetros del Tipo de Parámetro en cuestión.

Sin embargo, no siempre se querrá aplicar un Parámetro con un Valor determinado a todo el conjunto de Elementos de un Modelo de Interfaz de Usuario sobre los cuales es factible aplicar el Parámetro. En muchas oportunidades será necesario seleccionar un subconjunto de estos Elementos sobre los cuales aplicar un Parámetro con un cierto valor. Los Selectores se utilizan para definir estos subconjuntos.

Si un Parámetro tiene asociado un Selector, el Parámetro será aplicado a los Elementos seleccionados por el Selector. Si un Parámetro no tiene asociado un Selector, el Parámetro será aplicado a todos los Elementos de los Meta Elementos sobre los que el Tipo de Parámetro correspondiente al Parámetro es aplicable.

Existen dos tipos de Selectores: Selector de Elemento y Selector de Meta Elemento.

Un Selector de Elemento permite seleccionar un Elemento específico de un Modelo de Interfaz de Usuario. El Parámetro asociado a este Selector se aplicará sobre el Elemento seleccionado. Para ello, es necesario que el Elemento

seleccionado corresponda a un Meta Elemento sobre el cual el Tipo de Parámetro del Parámetro es aplicable.

Como se ha mencionado previamente, cuando se está creando una Plantilla de Transformación, es opcional asociarla a un Modelo de Interfaz de Usuario. Si en esa Plantilla van a ser incluidos Selectores de Elemento, la misma debería ser asociada al Modelo de Interfaz de Usuario cuyos Elementos van a ser seleccionados por los Selectores de Elemento. Si la Plantilla de Transformación se asocia a un Modelo de Interfaz de Usuario, entonces todos los Selectores de Elementos que se usen en esa Plantilla de Transformación, deben hacer referencia solo a Elementos de dicho Modelo de Interfaz de Usuario.

Un Selector de Meta Elemento permite especificar un Meta Elemento y un Número de Elemento Buscado. Básicamente el Selector de Meta Elemento permite seleccionar los Elementos del Meta Elemento especificado, y dentro de ellos, en el caso de que el Meta Elemento especificado sea un Meta Contenedor, al Elemento contenido que está en la posición especificada por el Número de Elemento Buscado. Estos Elementos contenidos son los que el Selector de Meta Elemento selecciona.

El Número de Elemento Buscado puede tener los siguientes valores:

- n (n-ésimo Elemento): selecciona a los n-ésimos Elementos contenidos en los Elementos que corresponden al Meta Elemento especificado por el Selector de Meta Elemento.

- 0 (todos los Elementos): en este caso, no se buscan Elementos contenidos en los Elementos que corresponden al Meta Elemento especificado por el Selector de Meta Elemento, sino que se selecciona a todos los Elementos que corresponden al Meta Elemento especificado por el Selector de Meta Elemento.

- -1 (todos menos el primer Elemento): selecciona a todos los Elementos menos los primeros, contenidos en los Elementos que corresponden al Meta Elemento especificado por el Selector de Meta Elemento.

- -2 (último Elemento): selecciona a los últimos Elementos contenidos en los Elementos que corresponden al Meta Elemento especificado por el Selector de Meta Elemento.

- -3 (todos menos el último Elemento): selecciona a todos los Elementos menos los últimos, contenidos en los Elementos que corresponden al Meta Elemento especificado por el Selector de Meta Elemento.

Cuando el Meta Elemento asociado al Selector de Meta Elemento es un Meta Componente Individual, el valor del Número de Elemento Buscado no tiene efecto,

y el Selector de Meta Elemento selecciona, simplemente, a todos los Elementos que corresponden al Meta Componente Individual.

Un Parámetro al que se le asocia a un Selector de Meta Elemento se aplicará sobre todos los Elementos seleccionados. Para ello, es necesario que los Meta Elementos asociados a los Elementos seleccionados sean Meta Elementos sobre los cuales el Tipo de Parámetro del Parámetro es aplicable.

Además, un Selector de Meta Elemento puede tener asociado un Meta Contenedor Ancestro con un Nivel de Contención. En este caso para que los Elementos queden seleccionados por el Selector de Meta Elemento, además de cumplir las condiciones previas, deben estar contenidos, al nivel de contención especificado por Nivel de Contención, en el Meta Contenedor Ancestro especificado.

A su vez, el Meta Contenedor Ancestro, puede tener otro Meta Contenedor Ancestro con Nivel de Contención, y así sucesivamente hasta el nivel que se precise.

Un Parámetro determina los Elementos sobre los que será aplicado gracias a su Selector, sea este de Elemento o de Meta Elemento. El Parámetro afectará a los Elementos sobre los que se aplica y/o a sus Elementos contenidos, según la definición del Tipo de Parámetro del Parámetro en cuestión, y su relación "afecta" con Meta Elemento y según que se cumpla la condición para el efecto (ver Sección 4.5).

A fin de ejemplificar el uso de los Selectores considérese la Figura 4.8 que representa un modelo genérico de interfaces de usuario, con la típica estructura de elementos contenedores y contenidos.

Considérese también que un Tipo de Parámetro TP afecta al Meta Elemento N y puede ser aplicado a N y a los Meta Elementos A, B, C y D. Por lo tanto un Parámetro P de tipo TP puede afectar a los Elementos con Meta Elemento N. Y se puede aplicar a los Elementos con Meta Elemento N, A, B, C, o D.

N es un Meta Componente Individual y A, B, C y D son Meta Contenedores. Sea E también un Meta Contenedor y F un Meta Componente Individual.

A puede contener a B y C. B puede contener a D. C puede contener a D y N. D puede contener a D, E y N. E puede contener a F.

$A1$ es un Elemento con Meta Elemento A. $B1$ es un Elemento con Meta Elemento B. $C1$ es un Elemento con Meta Elemento C. $D1$, $D2$, $D3$ y $D4$ son Elementos con Meta Elemento D. $E1$ es un Elemento con Meta Elemento E. $F1$ es un Elemento con Meta Elemento F. $N1$, $N2$, $N3$, $N4$, $N5$, $N6$ y $N7$ son Elementos con Meta Elemento N (ver Figura 4.8).

A continuación se listan los Elementos sobre los que se aplica y que son afectados por P al asociarle un Selector con las siguientes características:

- Si se asocia a P un Selector de Elemento, y el Selector de Elemento se asocia al Elemento:
  - $A1$: P se aplica a $A1$ y afecta a Ni $\forall 1 \leq i \leq 7$.

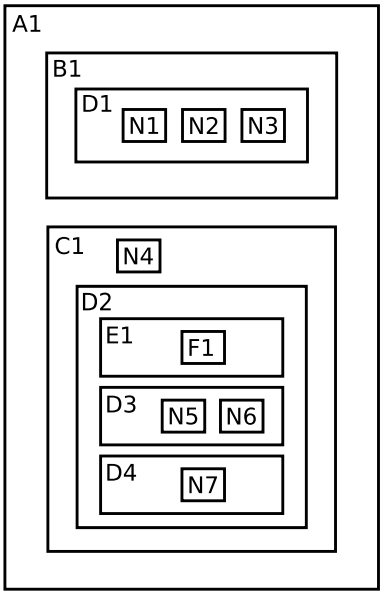


Figura 4.8: Representación de un modelo genérico de interfaces de usuario con elementos contenedores y contenidos

- $B1$: P se aplica a $B1$ y afecta a $N1$, $N2$, $N3$.
- $C1$: P se aplica a $C1$ y afecta a $N4$, $N5$, $N6$, $N7$.
- $D1$: P se aplica a $D1$ y afecta a $N1$, $N2$, $N3$.
- $D2$: P se aplica a $D2$ y afecta a $N5$, $N6$, $N7$.
- $D3$: P se aplica a $D3$ y afecta a $N5$, $N6$.
- $D4$: P se aplica a $D4$ y afecta a $N7$.
- $E1$: P se aplica a nada, ya que TP no es aplicable a E.
- $F1$: P se aplica a nada, ya que TP no es aplicable a F.
- Ni: P se aplica a Ni y afecta a Ni $\forall 1 \leq i \leq 7$

- Si se asocia a P un Selector de Meta Elemento con Número de Elemento Buscado 0 (opción de todos los Elementos), y el Selector de Meta Elemento se asocia al Meta Elemento:

  - A: P se aplica a $A1$ y afecta a Ni $\forall 1 \leq i \leq 7$.
  - B: P se aplica a $B1$ y afecta a $N1$, $N2$, $N3$.
  - C: P se aplica a $C1$ y afecta a $N4$, $N5$, $N6$, $N7$.
  - D: P se aplica a $D1$, $D2$, $D3$, $D4$ y afecta a $N1$, $N2$, $N3$, $N5$, $N6$, $N7$.

- E: P se aplica a nada, ya que TP no es aplicable a E.
- F: P se aplica a nada, ya que TP no es aplicable a F (cuando el Meta Elemento asociado a Selector de Meta Elemento es un Meta Componente Individual, como es el caso de N, el Número de Elemento Buscado no tiene efecto).
- N: P se aplica a Ni y afecta a Ni $\forall 1 \leq i \leq 7$ (cuando el Meta Elemento asociado a Selector de Meta Elemento es un Meta Componente Individual, como es el caso de N, el Número de Elemento Buscado no tiene efecto).

- Si se asocia a P un Selector de Meta Elemento con Número de Elemento Buscado 1 (opción de primer Elemento), y el Selector de Meta Elemento se asocia al Meta Elemento:
  - A: P se aplica a $B1$ y afecta a $N1$, $N2$, $N3$.
  - B: P se aplica a $D1$ y afecta a $N1$, $N2$, $N3$.
  - C: P se aplica a $N4$ y afecta a $N4$.
  - D: P se aplica a $N1$, $N5$, $N7$ y afecta a $N1$, $N5$, $N7$.
  - E: P se aplica a nada ya que TP no es aplicable a F (en este caso $F1$ es el primer elemento).
  - F: P se aplica a nada ya que TP no es aplicable a F (cuando el Meta Elemento asociado a Selector de Meta Elemento es un Meta Componente Individual, como es el caso de F, el Número de Elemento Buscado no tiene efecto).
  - N: P se aplica a Ni y afecta a Ni $\forall 1 \leq i \leq 7$ (cuando el Meta Elemento asociado a Selector de Meta Elemento es un Meta Componente Individual, como es el caso de N, el Número de Elemento Buscado no tiene efecto).

- Si se asocia a P un Selector de Meta Elemento con Número de Elemento Buscado -3 (opción de todos menos el último Elemento), y el Selector de Meta Elemento se asocia al Meta Elemento:
  - A: se excluye a $C1$, entonces P se aplica a $B1$ y afecta a $N1$, $N2$, $N3$.
  - B: se excluye a $D1$, entonces P se aplica a nada.
  - C: se excluye a $D2$, entonces P se aplica a $N4$ y afecta a $N4$.
  - D: se excluye a $N3$, $D4$, $N6$ entonces P se aplica a $N1$, $N2$, $D3$, $N5$ y afecta a $N1$, $N2$, $N5$.
  - E: se excluye a $F1$, entonces P se aplica a nada.

  - F: P se aplica a nada ya que TP no es aplicable a F (cuando el Meta Elemento asociado a Selector de Meta Elemento es un Meta Componente Individual, como es el caso de F, el Número de Elemento Buscado no tiene efecto).
  - N: P se aplica a Ni y afecta a Ni $\forall 1 \leq i \leq 7$ (cuando el Meta Elemento asociado a Selector de Meta Elemento es un Meta Componente Individual, como es el caso de N, el Número de Elemento Buscado no tiene efecto).

- Si se asocia a P un Selector de Meta Elemento asociado a:
  - el Meta Elemento N con Meta Contenedor Ancestro C con Nivel de Descendencia 1: P se aplica a $N4$ y afecta a $N4$.
  - el Meta Elemento D con Meta Contenedor Ancestro A con Nivel de Descendencia 1: P se aplica a nada.
  - el Meta Elemento D con Meta Contenedor Ancestro C con Nivel de Descendencia 2: P se aplica a $D3$ y $D4$ y afecta a $N5$, $N6$ y $N7$.
  - el Meta Elemento D con Número de Elemento Buscado -1 (opción todos menos el primero) con Meta Contenedor Ancestro C con Nivel de Descendencia 2: P se aplica a $N6$ y afecta a $N6$.

Cabe notar que los Parámetros de una Plantilla de Transformación que han sido asociados a un Selector de Elemento no son reutilizables en distintos Modelos de Interfaces de Usuario, pues los Selectores de Elemento hacen referencia a un Elemento específico de un Modelo de Interfaz de Usuario. Por su parte, los Parámetros que se asocian a un Selector de Meta Elemento sí son reutilizables en distintos Modelos de Interfaces de Usuario.

A continuación se describe cada uno de los elementos del meta modelo de Plantillas de Transformación, junto con sus propiedades, relaciones, servicios y restricciones.

Elemento: Plantilla de Transformación

- Nombre del elemento: Plantilla_Transformacion
- Alias: Plantilla de Transformación
- Descripción: define un conjunto de Parámetros con Valores asociados y Selectores que serán utilizados para guiar una transformación de un modelo de interfaz de usuario más abstracto a uno más concreto o a código, especificando aspectos de estructura y de estilo de las interfaces de usuario.
- Propiedades:

- Propiedad 1
  - Alias: Identificador
  - Descripción: identificador único de la plantilla de transformación.
  - Tipo de propiedad: Constante
  - Tipo de dato: String
  - Es identificador: si
  - Acepta nulo: no
- Propiedad 2
  - Alias: Nombre
  - Descripción: nombre descriptivo de la plantilla de transformación.
  - Tipo de propiedad: Variable
  - Tipo de dato: String
  - Es identificador: no
  - Acepta nulo: no
- Propiedad 3
  - Alias: Fecha de creación
  - Descripción: Fecha de creación de la plantilla de transformación.
  - Tipo de propiedad: Constante
  - Tipo de dato: DateTime
  - Es identificador: no
  - Acepta nulo: no
  - Valor por defecto: fecha y hora del sistema
- Propiedad 4
  - Alias: Descripción
  - Descripción: describe la plantilla de transformación.
  - Tipo de propiedad: Variable
  - Tipo de dato: Texto
  - Es identificador: no
  - Acepta nulo: si

- Relaciones:
  - Una Plantilla de Transformación se prepara (asocia) para un Contexto.

    Un Contexto puede estar asociado a 0 o muchas Plantillas de Transformación.

- Una Plantilla de Transformación se puede asociar o no a un Modelo de Interfaz de Usuario.

  Un Modelo de Interfaz de Usuario puede estar asociado a 0 cero o muchas Plantillas de Transformación.

- Servicios:

  - Nuevo(String pIdentificador, String pNombre, DateTime pFechaCreacion, Texto pDescripcion, Contexto pContexto, Modelo_IU pModeloIU): crea una Plantilla de Transformación asignando a las propiedades Identificador, Nombre, Fecha de Creación y Descripción, el valor de los argumentos pIdentificador, pNombre, pFechaCreacion y pDescripcion, respectivamente. Además, asocia el Contexto especificado por el argumento pContexto, y el Modelo de Interfaz de Usuario especificado por el argumento pModeloIU, a la Plantilla de Transformación creada.

    pModeloIU puede ser nulo.

  - Editar(Plantilla_Transformacion THIS, String pNombre, Texto pDescripcion): asigna a las propiedades Nombre y Descripción de la Plantilla de Transformación actual, referenciada por THIS, el valor de los argumentos pNombre y pDescripcion, respectivamente.
  - Eliminar(Plantilla_Transformacion THIS): elimina la Plantilla de Transformación actual, referenciada por THIS.

Elemento: Valor

- Nombre del elemento: Valor
- Alias: Valor
- Descripción: representa un valor que se puede asignar como valor por defecto a un tipo de parámetro, o como valor a un parámetro.

  El elemento Valor es abstracto. Más adelante se presentarán los elementos Valor Booleano, Valor Entero, Valor Real, Valor Cadena de Caracteres, Valor Longitud, Valor Porcentaje, Valor URI, Valor Color y Valor Enumeración, todos ellos, especializaciones de Valor.

- Propiedades:

  - Propiedad 1
    - Alias: Identificador
    - Descripción: Identificador único de un valor

    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no

- Relaciones:

  - Un Valor corresponde (se asocia) a un Tipo de Valor.
    Un Tipo de Valor puede tener 0 o muchos Valores asociados.

- Servicios:

  - Nuevo(String pIdentificador, Tipo_Valor pTipoValor): Crea un Valor asignando a la propiedad Identificador del nuevo Valor, el valor del argumento pIdentificador. Además, asocia el Tipo de Valor especificado por el argumento pTipoValor al Valor creado.
  - Eliminar(Valor THIS): elimina el Valor actual, referenciado por THIS.

Elemento: Valor Booleano

- Nombre del elemento: Valor_Booleano
- Alias: Valor Booleano
- Descripción: representa un valor booleano.
- Propiedades: la que hereda de Valor: Identificador.
  Además, define las siguientes propiedades:

  - Propiedad 1
    - Alias: Valor
    - Descripción: el valor booleano
    - Tipo de propiedad: Constante
    - Tipo de dato: Booleano
    - Es identificador: no
    - Acepta nulo: no

- Relaciones:

  - Valor Booleano especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Booleano.

- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, Tipo_Booleano pTipoBooleano, Booleano pValor): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoBooleano. Luego crea el Valor Booleano correspondiente y asigna a la propiedad Valor el valor del argumento pValor.

Elemento: Valor Entero

- Nombre del elemento: Valor_Entero
- Alias: Valor Entero
- Descripción: representa un valor entero.
- Propiedades: la que hereda de Valor: Identificador.

  Además, define las siguientes propiedades:

  - Propiedad 1
    - Alias: Valor
    - Descripción: el valor entero
    - Tipo de propiedad: Constante
    - Tipo de dato: Entero
    - Es identificador: no
    - Acepta nulo: no
- Relaciones:

  - Valor Entero especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Entero.
- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, Tipo_Entero pTipoEntero, Entero pValor): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoEntero. Luego crea el Valor Entero correspondiente y asigna a la propiedad Valor el valor del argumento pValor.

Elemento: Valor Real

- Nombre del elemento: Valor_Real
- Alias: Valor Real
- Descripción: representa un valor real.
- Propiedades: la que hereda de Valor: Identificador.

  Además, define las siguientes propiedades:

  - Propiedad 1
    - Alias: Valor
    - Descripción: el valor real
    - Tipo de propiedad: Constante
    - Tipo de dato: Real
    - Es identificador: no
    - Acepta nulo: no
- Relaciones:

  - Valor Real especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Real.
- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, Tipo_Real pTipoReal, Real pValor): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoReal. Luego crea el Valor Real correspondiente y asigna a la propiedad Valor el valor del argumento pValor.

Elemento: Valor Cadena de Caracteres

- Nombre del elemento: Valor_Cadena
- Alias: Valor Cadena de Caracteres
- Descripción: representa un valor de cadena de caracteres.
- Propiedades: la que hereda de Valor: Identificador.

  Además, define las siguientes propiedades:

- Propiedad 1
  - Alias: Valor
  - Descripción: el valor de cadena de caracteres
  - Tipo de propiedad: Constante
  - Tipo de dato: String
  - Es identificador: no
  - Acepta nulo: no

- Relaciones:
  - Valor Cadena de Caracteres especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Cadena de Caracteres.

- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador, Tipo_Cadena pTipoCadena, String pValor): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoCadena. Luego crea el Valor Cadena de Caracteres correspondiente y asigna a la propiedad Valor el valor del argumento pValor.

Elemento: Valor Longitud

- Nombre del elemento: Valor_Longitud
- Alias: Valor Longitud
- Descripción: representa un valor de longitud.
- Propiedades: la que hereda de Valor: Identificador.

  Además, define las siguientes propiedades:
  - Propiedad 1
    - Alias: Valor
    - Descripción: el valor de longitud
    - Tipo de propiedad: Constante
    - Tipo de dato: Real
    - Es identificador: no
    - Acepta nulo: no

- Propiedad 2
  - Alias: Unidad de medida
  - Descripción: la unidad de medida de la longitud
  - Tipo de propiedad: Constante
  - Tipo de dato: String, restringido a: “em”, “ex”, “px”, “in”, “cm”, “mm”, “pt”, “pc”
  - Es identificador: no
  - Acepta nulo: no

- Relaciones:
  - Valor Longitud especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Longitud.

- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador, Tipo_Longitud pTipoLongitud, Real pValor, String pUnidadMedida): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoLongitud. Luego crea el Valor Longitud correspondiente y asigna a la propiedad Valor el valor del argumento pValor, y a la propiedad Unidad de medida el valor del argumento pUnidadMedida.

Elemento: Valor Porcentaje

- Nombre del elemento: Valor_Porcentaje
- Alias: Valor Porcentaje
- Descripción: representa un valor de porcentaje.
- Propiedades: la que hereda de Valor: Identificador.

  Además, define las siguientes propiedades:
  - Propiedad 1
    - Alias: Valor
    - Descripción: el valor de porcentaje
    - Tipo de propiedad: Constante
    - Tipo de dato: Real
    - Es identificador: no

        - Acepta nulo: no

- Relaciones:

    - Valor Porcentaje especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Porcentaje.

- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:

    - Nuevo(String pIdentificador, Tipo_Porcentaje pTipoPorcentaje, Real pValor): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoPorcentaje. Luego crea el Valor Porcentaje correspondiente y asigna a la propiedad Valor el valor del argumento pValor.

Elemento: Valor URI

- Nombre del elemento: Valor_URI
- Alias: Valor URI
- Descripción: representa un valor de URI.
- Propiedades: la que hereda de Valor: Identificador.

  Además, define las siguientes propiedades:

    - Propiedad 1
        - Alias: Valor
        - Descripción: el valor de URI
        - Tipo de propiedad: Constante
        - Tipo de dato: String
        - Es identificador: no
        - Acepta nulo: no

- Relaciones:

    - Valor URI especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo URI.

- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:

- Nuevo(String pIdentificador, Tipo_URI pTipoURI, String pValor): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoURI. Luego crea el Valor URI correspondiente y asigna a la propiedad Valor el valor del argumento pValor.

Elemento: Valor Color

- Nombre del elemento: Valor_Color
- Alias: Valor Color
- Descripción: representa un valor de Color.
- Propiedades: la que hereda de Valor: Identificador.

  Además, define las siguientes propiedades:

  - Propiedad 1
    - Alias: Valor
    - Descripción: el valor de Color
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: no
    - Acepta nulo: no
- Relaciones:

  - Valor Color especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Color.
- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, Tipo_Color pTipoColor, String pValor): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoColor. Luego crea el Valor Color correspondiente y asigna a la propiedad Valor el valor del argumento pValor.

Elemento: Valor Enumeración

- Nombre del elemento: Valor_Enumeracion
- Alias: Valor Enumeración
- Descripción: representa un valor de Enumeración.
- Propiedades: la que hereda de Valor: Identificador.
- Relaciones:
  - Valor Enumeración especializa a Valor. Por lo tanto hereda su relación con Tipo de Valor y añade la restricción de que el Tipo de Valor debe ser un Tipo Enumeración.
  - Un Valor Enumeración se corresponde (asocia) a un Ítem de Enumeración.
    Un Ítem de Enumeración puede estar asociado a 0 o muchos Valores de Enumeración.
    El Ítem de Enumeración que se asocia a un Valor de Enumeración, debe estar relacionado al Tipo Enumeración del Valor de Enumeración.
- Servicios: el que hereda de Valor: Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador, Tipo_Enumeracion pTipoEnumeracion, Item_Enumeracion pItemEnumeracion): primero crea un Valor invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador y pTipoEnumeracion. Luego crea el Valor Enumeración correspondiente y lo asocia al Ítem de Enumeración especificado por el argumento pItemEnumeracion.
    pItemEnumeracion debe ser un Ítem de Enumeración que está asociado al Tipo Enumeración referenciado por el argumento pTipoEnumeracion.

Elemento: Selector

- Nombre del elemento: Selector
- Alias: Selector
- Descripción: acota el conjunto de elementos de un modelo de interfaz de usuario sobre los que se aplica un parámetro.

  Selector es un elemento abstracto. Sus especializaciones son: Selector de Elemento y Selector de Meta Elemento.

- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: identificador único del selector
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Nombre
    - Descripción: nombre descriptivo del selector
    - Tipo de propiedad: Variable
    - Tipo de dato: String
    - Es identificador: no
    - Acepta nulo: no
  - Propiedad 3
    - Alias: Fecha de creación
    - Descripción: Fecha de creación del selector
    - Tipo de propiedad: Constante
    - Tipo de dato: DateTime
    - Es identificador: no
    - Acepta nulo: no
    - Valor por defecto: fecha y hora del sistema
- Servicios:
  - Nuevo(String pIdentificador, String pNombre, DateTime pFechaCreacion): crea un Selector asignando a las propiedades Identificador, Nombre y Fecha de Creación el valor de los argumentos pIdentificador, pNombre y pFechaCreacion respectivamente.
  - Editar(Selector THIS, String pNombre): asigna a la propiedad Nombre del Selector actual, referenciado por THIS, el valor del argumento pNombre.
  - Eliminar(Selector THIS): elimina el Selector actual, referenciado por THIS.

Elemento: Selector de Elemento

- Nombre del elemento: Selector_Elemento
- Alias: Selector de Elemento
- Descripción: selecciona un elemento específico de un modelo de interfaz de usuario.
- Propiedades: las que hereda de Selector: Identificador, Nombre y Fecha de Creación.
- Relaciones:
  - Selector de Elemento especializa a Selector.
  - Un Selector de Elemento selecciona (se asocia) a un Elemento.

    Un Elemento puede estar asociado a 0 o muchos Selectores de Elemento.
- Servicios: los que hereda de Selector: Editar y Eliminar.

  Además, se define el siguiente servicio:
  - Nuevo(String pIdentificador, String pNombre, DateTime pFechaCreacion, Elemento pElemento): primero crea un Selector invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador, pNombre y pFechaCreacion. Luego crea el Selector de Elemento correspondiente y lo asocia al Elemento especificado por el argumento pElemento.

Elemento: Selector de Meta Elemento

- Nombre del elemento: Selector_Meta_Elemento
- Alias: Selector de Meta Elemento
- Descripción: permite seleccionar elementos de un modelo de interfaz de usuario en base a sus meta elementos.

  Selector de Meta Elemento es un elemento abstracto. Sus especializaciones son: Meta Elemento Buscado y Meta Contenedor Ancestro.
- Propiedades: las que hereda de Selector: Identificador, Nombre y Fecha de Creación.
- Relaciones:

- Selector de Elemento especializa a Selector.

- Servicios: los que hereda de Selector: Editar y Eliminar.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, String pNombre, DateTime pFechaCreacion): primero crea un Selector invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador, pNombre y pFechaCreacion. Luego crea el Selector de Meta Elemento correspondiente.

Elemento: Meta Elemento Buscado

- Nombre del elemento: Meta_Elemento_Buscado

- Alias: Meta Elemento Buscado

- Descripción: permite seleccionar, en un modelo de interfaz de usuario, los elementos que corresponden a un meta elemento especificado, y dentro de ellos, en el caso de que el meta elemento especificado sea un meta contenedor, al elemento contenido que está en la posición especificada por un número de elemento buscado. Si el meta elemento especificado es un meta componente individual, el número de elemento buscado no tiene efecto y, simplemente, se seleccionan todos los elementos que corresponden al meta componente individual.

- Propiedades: las que hereda de Selector de Meta Elemento: Identificador, Nombre y Fecha de Creación.

  Además, define las siguientes propiedades:

  - Propiedad 1
    - Alias: Número Elemento Buscado
    - Descripción: posición del elemento buscado dentro de otro elemento que corresponde a un meta elemento especificado
    - Tipo de propiedad: Constante
    - Tipo de dato: Entero
      - Sea n el número entero positivo que se asigna a esta propiedad. Entonces el Elemento Buscado es el n-ésimo contenido en un Elemento que está asociado al Meta Elemento asociado al Meta Elemento Buscado.
      - Si a esta propiedad se le asigna el valor 0, entonces no se busca a un Elemento que esté contenido en otro Elemento que esté asociado al Meta Elemento asociado al Meta Elemento

Buscado. Sino que se busca a todos los Elementos que están asociados al Meta Elemento asociado al Meta Elemento Buscado.

- ◇ Si a esta propiedad se le asigna el valor -1 (todos menos el primero), entonces los Elementos Buscados son todos menos los primeros contenidos en los Elementos que están asociados al Meta Elemento asociado al Meta Elemento Buscado.
- ◇ Si a esta propiedad se le asigna el valor -2 (el último), entonces los Elementos Buscados son los últimos contenidos en los Elementos que están asociados al Meta Elemento asociado al Meta Elemento Buscado.
- ◇ Si a esta propiedad se le asigna el valor -3 (todos menos el último), entonces los Elementos Buscados son todos menos los últimos contenidos en los Elementos que están asociados al Meta Elemento asociado al Meta Elemento Buscado.
- ◇ Los demás enteros negativos no se permiten como valor de esta propiedad.

- Es identificador: no
- Acepta nulo: no
- Valor por defecto: 0

- Relaciones:

  - Meta Elemento Buscado especializa a Selector de Meta Elemento.
  - Un Meta Elemento Buscado se asocia a un Meta Elemento.

    Un Meta Elemento puede estar asociado a 0 o muchos Meta Elementos Buscados.

- Servicios: los que hereda de Selector de Meta Elemento: Editar y Eliminar.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, String pNombre, DateTime pFechaCreacion, Meta_Elemento pMetaElemento, Entero pNumeroElementoBuscado): primero crea un Selector de Meta Elemento invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador, pNombre y pFechaCreacion. Luego crea el Meta Elemento Buscado correspondiente, le asocia al Meta Elemento especificado por el argumento pMetaElemento, y asigna a la propiedad de Número Elemento Buscado el valor del argumento pNumeroElementoBuscado.

Elemento: Meta Contenedor Ancestro

- Nombre del elemento: Meta_Contenedor_Ancestro
- Alias: Meta Contenedor Ancestro
- Descripción: un Selector de Meta Elemento puede tener asociado un Meta Contenedor Ancestro con un Nivel de Contención. En este caso para que los Elementos queden seleccionados por el Selector de Meta Elemento, deben estar contenidos, al nivel de contención especificado por Nivel de Contención, en el Meta Contenedor Ancestro especificado.

  A su vez, el Meta Contenedor Ancestro, puede tener otro Meta Contenedor Ancestro con Nivel de Contención, y así sucesivamente hasta el nivel que se precise.

- Propiedades: las que hereda de Selector de Meta Elemento: Identificador, Nombre y Fecha de Creación.

  Además, define las siguientes propiedades:

  - Propiedad 1
    - Alias: Nivel Descendencia
    - Descripción: es el nivel de contención al que debe encontrarse el descendiente de este ancestro
    - Tipo de propiedad: Constante
    - Tipo de dato: Natural
      - ◇ Si el valor de esta propiedad es $N <> 0$, entonces se está especificando un nivel de contención N.
      - ◇ Si el valor es 0, entonces se está especificando cualquier nivel de contención.
    - Es identificador: no
    - Acepta nulo: no
    - Valor por defecto: 1

- Relaciones:

  - Meta Contenedor Ancestro especializa a Selector de Meta Elemento.
  - Un Meta Contenedor Ancestro se asocia a un Meta Contenedor.

    Un Meta Contenedor puede estar asociado a 0 o muchos Meta Contenedores Ancestros.

- Un Meta Contenedor Ancestro tiene como descendiente (se asocia) a un Selector de Meta Elemento.

  Un Selector de Meta Elemento es descendiente de un Meta Contenedor Abstracto o de ninguno.

  El Selector de Meta Elemento descendiente de un Meta Contenedor Ancestro podrá ser un Meta Elemento Buscado u otro Meta Contenedor Ancestro (diferente del original). Si es un Meta Elemento Buscado, el mismo deberá estar asociado a un Meta Elemento asociado al mismo Meta Modelo de Interfaz de Usuario al cual está asociado el Meta Contenedor asociado al Meta Contenedor Ancestro original. Si es un Meta Contenedor Ancestro, el mismo deberá estar asociado a un Meta Contenedor asociado al mismo Meta Modelo de Interfaz de Usuario al cual está asociado el Meta Contenedor asociado al Meta Contenedor Ancestro original.

- Servicios: los que hereda de Selector de Meta Elemento: Editar y Eliminar.

  Además, se define el siguiente servicio:

  - Nuevo(String pIdentificador, String pNombre, DateTime pFechaCreacion, Meta_Contenedor pMetaContenedor, Natural pNivelDescendencia, Selector_Meta_Elemento pSelectorMetaElemento): primero crea un Selector de Meta Elemento invocando su servicio de creación Nuevo y pasándole los argumentos pIdentificador, pNombre y pFechaCreacion. Luego crea el Meta Contenedor Ancestro correspondiente y le asocia el Meta Contenedor especificado por el argumento pMetaContenedor. Además, asigna a la propiedad Nivel Descendencia el valor del argumento pNivelDescendencia. Y finalmente, asocia al Meta Contenedor Ancestro el Selector de Meta Elemento especificado por el argumento pSelectorMetaElemento.

    pSelectorMetaElemento debe ser un Meta Elemento Buscado que está asociado a un Meta Elemento asociado a un Meta Modelo de Interfaz de Usuario que debe ser el mismo que está asociado al Meta Contenedor especificado por el argumento pMetaContenedor. O bien, pSelectorMetaElemento debe ser un Meta Contenedor Ancestro que está asociado a un Meta Contenedor asociado a un Meta Modelo de Interfaz de Usuario que debe ser el mismo que está asociado al Meta Contenedor especificado por el argumento pMetaContenedor.

Elemento: Parámetro

- Nombre del elemento: Parametro

- Alias: Parámetro
- Descripción: define un aspecto de estructura o estilo de una interfaz de usuario. Está en una Plantilla de Transformación. Corresponde a un Tipo de Parámetro. Y tiene un Valor y un Selector.
- Propiedades:
  - Propiedad 1
    - Alias: Identificador
    - Descripción: identificador único del parámetro
    - Tipo de propiedad: Constante
    - Tipo de dato: String
    - Es identificador: si
    - Acepta nulo: no
  - Propiedad 2
    - Alias: Fecha de creación
    - Descripción: Fecha de creación del parámetro
    - Tipo de propiedad: Constante
    - Tipo de dato: DateTime
    - Es identificador: no
    - Acepta nulo: no
    - Valor por defecto: fecha y hora del sistema
- Relaciones:
  - Un Parámetro forma parte de una Plantilla de Transformación.

    Una Plantilla de Transformación está compuesta por 0 o muchos Parámetros.
  - Un Parámetro corresponde (se asocia) a un Tipo de Parámetro.

    Un Tipo de Parámetro puede estar asociado a 0 o muchos Parámetros.

    En una Plantilla de Transformación solo se pueden poner Parámetros cuyos Tipos de Parámetros están implementados en el Contexto asociado a la Plantilla de Transformación.
  - Un Parámetro tiene un Valor.

    Un Valor está asociado a un Parámetro o a ninguno.

    El Valor de un Parámetro debe estar asociado al Tipo de Valor del Tipo de Parámetro del Parámetro.

Si el Valor de un Parámetro es un Valor Enumeración, el mismo debe estar asociado a un Ítem de Enumeración que está implementado para el Tipo de Parámetro del Parámetro en el Contexto asociado a la Plantilla de Transformación de la que el Parámetro forma parte.

- Un Parámetro tiene (está asociado a) un Selector o ninguno.

  Un Selector se asocia a 0 o muchos Parámetros.

  Si el Selector de un Parámetro es un Selector de Elemento su Elemento asociado debe corresponder al mismo Modelo de Interfaz de Usuario que corresponde a la Plantilla de Transformación de la que el Parámetro forma parte.

  Si el Selector de un Parámetro es un Selector de Elemento su Elemento asociado debe corresponder a un Meta Elemento que es uno de los Meta Elementos sobre los que el Tipo de Parámetro del Parámetro puede ser aplicado.

  Si el Selector de un Parámetro es un Selector de Meta Elemento su Meta Elemento Buscado debe corresponder a un Meta Elemento que es uno de los Meta Elementos sobre los que el Tipo de Parámetro del Parámetro puede ser aplicado.

- Servicios:

  - Nuevo(String pIdentificador, DateTime pFechaCreacion, Tipo_Parametro pTipoParametro, Plantilla_Transformacion pPlantillaTransformacion, Valor pValor, Selector pSelector): crea un Parámetro asignando a las propiedades Identificador y Fecha de Creación del nuevo Parámetro, el valor de los argumentos pIdentificador y pFechaCreacion, respectivamente. Además, asocia al Parámetro creado el Tipo de Parámetro especificado por el argumento pTipoParametro, la Plantilla de Transformación especificada por el argumento pPlantillaTransformacion, el Valor especificado por el argumento pValor y el Selector especificado por el argumento pSelector.

    pTipoParametro debe estar implementado para el Contexto asociado a la Plantilla de Transformación especificada por pPlantillaTransformacion.

    Si pValor es un Valor Enumeración, el Ítem de Enumeración asociado debe estar implementado para el Tipo de Parámetro especificado por pTipoParametro en el Contexto relacionado a la Plantilla de Transformación especificada por pPlantillaTransformacion.

    pValor debe estar relacionado al Tipo de Valor asociado al Tipo de Parámetro especificado por pTipoParametro.

    pSelector puede ser nulo.

Si pSelector es un Selector de Elemento su Elemento asociado debe corresponder al mismo Modelo de Interfaz de Usuario que corresponde a la Plantilla de Transformación especificada por el argumento pPlantillaTransformacion.

Si pSelector es un Selector de Elemento su Elemento asociado debe corresponder a un Meta Elemento que es uno de los Meta Elementos sobre los que el Tipo de Parámetro especificado por pTipoParametro puede ser aplicado.

Si pSelector es un Selector de Meta Elemento su Meta Elemento Buscado debe corresponder a un Meta Elemento que es uno de los Meta Elementos sobre los que el Tipo de Parámetro especificado por pTipoParametro puede ser aplicado.

- Eliminar(Parametro THIS): elimina el Parámetro actual, referenciado por THIS.

5

Automatización de Plantillas de Transformación

La automatización de la aproximación de Plantillas de Transformación implica dos tipos de herramientas: Editores de Plantillas de Transformación y Compiladores de modelo a modelo, o de modelo a código de interfaces de usuario que utilicen una Plantilla de Transformación en su proceso de compilación.

A continuación se describe como deberían funcionar estas herramientas. También se presenta un prototipo de un editor de Plantillas de Transformación.

5.1. Editores de Plantillas de Transformación

Un Editor de Plantillas de Transformación permite crear una Plantilla de Transformación.

El proceso de creación de una Plantilla de Transformación implica los siguientes pasos:

1. Definir el Modelo de Contexto para el cual se pretende crear una Plantilla de Transformación.

   Si las características del contexto de uso para el cual se pretende crear la Plantilla de Transformación no han sido modeladas previamente, este Modelo de Contexto debe ser creado. De lo contrario, se debe reutilizar el Modelo de Contexto correspondiente creado previamente.

2. Definir el (o los) Meta Modelo(s) de Interfaces de Usuario para el (los) cual(es) se pretende crear una Plantilla de Transformación.

   Como los Tipos de Parámetros de los Parámetros de una Plantilla de Transformación se pueden aplicar y pueden afectar a Meta Elementos de distintos Meta Modelos de Interfaces de Usuario, una misma Plantilla de Transformación se puede crear para varios Meta Modelos de Interfaces de Usuario.

Si estos Meta Modelos no ha sido modelados previamente, se deben crear los Meta Modelos. De lo contrario, se deben utilizar los que ya fueron previamente creados.

Por otra parte, se asume que esos Meta Modelos de Interfaces de Usuario tienen implementadas herramientas asociadas de compilación de modelo a modelo o de modelo a código.

3. Definir el Modelo de Interfaz de Usuario para el cual se pretende crear una Plantilla de Transformación.

   Este paso es opcional, ya que una Plantilla de Transformación puede ser creada de manera específica para un Modelo de Interfaz de Usuario en particular, haciendo referencia a sus Elementos, o bien, puede ser creada de manera general para Meta Modelos de Interfaces de Usuario, sin hacer mención específica a los elementos de un Modelo de Interfaz de Usuario en particular.

   Si la Plantilla va ser creada para un Modelo de Interfaz de Usuario en particular, y este Modelo no ha sido utilizado previamente, este Modelo debe ser importado. De lo contrario, se debe utilizar el Modelo previamente importado.

   Se asume que este Modelo ha sido creado con otra herramienta de definición de Modelos de Interfaces de Usuario, por lo en este paso se realiza un proceso de importación o de reutilización de un Modelo previamente importado.

4. Definir el Meta Modelo de las Plantillas de Transformación.

   Básicamente, esto implica definir los Tipos de Parámetros con sus Tipos de Valores y especificar los Meta Elementos de Meta Modelos de Interfaces de Usuario que serán afectados por el Tipo de Parámetro, y los Meta Elementos sobre los que el Tipo de Parámetro se podrá aplicar.

   Además, implica asociar guías de uso a los Tipos de Parámetro en base a diversos Contextos de Uso.

   En base a estimaciones del nivel de importancia y del costo de implementación de un Tipo de Parámetro en un Contexto dado, se debe decidir si el Tipo de Parámetro en cuestión será implementado en el compilador de modelo a modelo o de modelo a código asociado al Meta Modelo de Interfaz de Usuario y al Contexto considerados para la Plantilla de Transformación.

   Si se ha decidido que un Tipo de Parámetro será implementado en un compilador para un Contexto de uso en particular, se debe proceder a las modificaciones necesarias en el compilador a fin de dar soporte al Tipo de Parámetro. Este es un proceso que escapa del alcance del Editor de Plantillas de Transformación. Obviamente, se realiza de manera

independiente del proceso de creación de una Plantilla de Transformación, utilizando las herramientas de desarrollo apropiadas.

Sin embargo, una vez terminada la implementación de un Tipo de Parámetro para un contexto dado en un compilador, es necesario volver al Editor de Plantillas de Transformación e indicar que dicho Tipo de Parámetro ya ha sido implementado, y que queda, por lo tanto, disponible para ser utilizado cuando se crean Plantillas de Transformación para ese contexto en particular.

Todos los Tipos de Parámetros que se van definiendo, podrán luego utilizarse en la creación de Plantillas de Transformación.

5. Crear la Plantilla de Transformación.

   Una Plantilla de Transformación se crea para un Contexto de uso en particular.

   La creación de una Plantilla de Transformación implica crear un conjunto de Parámetros y asignarles un Valor y un Selector.

   Los Parámetros que pueden ser utilizados en una Plantilla de Transformación son aquellos que han sido implementados para el Contexto de la Plantilla de Transformación.

Evidentemente, los pasos previamente expuestos no siempre serán seguidos exactamente en ese orden, pues cuando muchos Contextos de Uso y Meta Modelos de Interfaces de Usuario hayan sido modelados, muchos Modelos de Interfaces de Usuario hayan sido importados, y el Meta Modelo de Plantillas de Transformación esté formado por un conjunto relevante de Tipos de Parámetros con sus datos asociados, se podrá realizar solo el paso de la creación de una Plantilla de Transformación nueva.

De los pasos involucrados en la creación de una Plantilla de Transformación, se deduce que un Editor de Plantillas de Transformación debe permitir la edición de:

- Modelos de Contexto
- Meta Modelos de Interfaces de Usuario
- Modelos de Interfaces de Usuario (importación, en lugar de edición)
- Meta Modelo de Plantillas de Transformación: esto abarca a los Tipos de Parámetros y todos sus datos asociados (Tipos de Valor, Meta Elementos sobre los que se aplica, Meta Elementos que afecta, guías de uso e información relacionada a su implementación en diversos contextos)
- Plantillas de Transformación: esto abarca a Plantillas de Transformación, Parámetros, Valores y Selectores.

5.1.1. Prototipo de Editor de Plantillas de Transformación

En el contexto de este trabajo se ha implementado un Prototipo de un Editor de Plantillas de Transformación.

El prototipo ha sido desarrollado siguiendo el método de desarrollo de software OO-Method y la tecnología OLIVANOVA (ver Sección 3.4).

Los meta-meta modelos y meta modelos de Contexto, de Interfaces de Usuario y de Plantillas de Transformación, presentados en el Capítulo 4, han sido modelados utilizando el OLIVANOVA *Modeler* y se han definido las cuatro vistas que componen el esquema conceptual: modelo de objetos, modelo dinámico, modelo funcional y modelo de presentación.

A partir del esquema conceptual, utilizando la tecnología de compilación de modelos de OLIVANOVA, se ha generado automáticamente el Prototipo de Edición de las Plantillas de Transformación.

Tal como se ha definido en la Sección 5.1, el Prototipo permite la edición de:

- Modelos de Contexto: Plataformas, Estereotipos de Usuarios, Ambientes y Contextos propiamente dichos.

  Las Figuras 5.1 y 5.2 muestran las pantallas que listan y permiten crear, editar y eliminar Plataformas y Contextos, respectivamente.

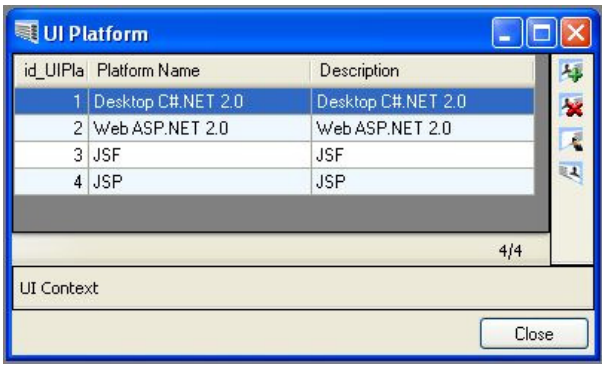


Figura 5.1: Pantalla de Plataformas del Prototipo de Editor de Plantillas de Transformación

- Meta Modelos de Interfaces de Usuario con sus Meta Elementos que pueden ser Meta Componentes Individuales o Meta Contenedores, y con sus Meta Relaciones que son Meta Descomposiciones.

  Las Figuras 5.3 y 5.4 muestran las pantallas que listan y permiten crear, editar y eliminar Meta Modelos de Interfaces de Usuario y Meta Elementos, respectivamente.

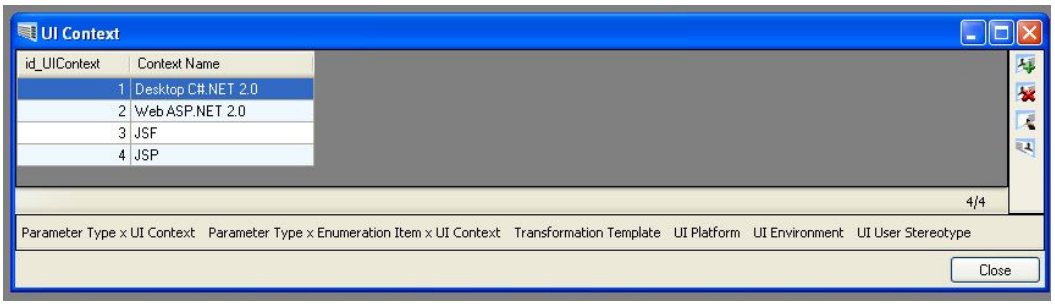


Figura 5.2: Pantalla de Contextos del Prototipo de Editor de Plantillas de Transformación

- Modelos de Interfaces de Usuario: actualmente el Prototipo de Editor no implementa una importación automática de Modelos de Interfaces de Usuario. La importación debe realizarse de manera manual.

  El Prototipo permite definir Modelos de Interfaz de Usuario, Elementos y Descomposiciones.

  Las Figuras 5.5 y 5.6 muestran la pantalla que lista y permite crear, editar y eliminar Modelos de Interfaces de Usuario, y la pantalla para crear nuevos Modelos de Interfaces de Usuario, respectivamente.

- Meta Modelo de Plantillas de Transformación: Tipo de Parámetro, Tipo de Valor (Tipo Booleano, Tipo Entero, Tipo Real, Tipo Cadena, Tipo Longitud, Tipo Porcentaje, Tipo URI, Tipo Color, Tipo Enumeración) e Ítem de Enumeración.

  Las Figuras 5.7 y 5.8 muestran la pantalla que permite crear nuevos Tipos de Parámetros, y la pantalla que despliega la información de estimación de nivel de importancia, costo de implementación e indicador de implementación de un Tipo de Parámetro en un Contexto, respectivamente.

- Plantillas de Transformación, junto con Valor (Valor Booleano, Valor Entero, Valor Real, Valor Cadena, Valor Longitud, Valor Porcentaje, Valor URI, Valor Color, Valor Enumeración), Selector, Selector de Elemento, Selector de Meta Elemento, Meta Elemento Buscado, Meta Contenedor Ancestro y Parámetros.

  Las Figuras 5.9 y 5.10 muestran las pantallas que permiten crear nuevas Plantillas de Transformación y nuevos Parámetros, respectivamente.

Finalmente, cabe mencionar que al Prototipo de Editor de Plantillas de Transformación se le deberían incorporar las facilidades de importación de Modelos de Interfaces de Usuario.

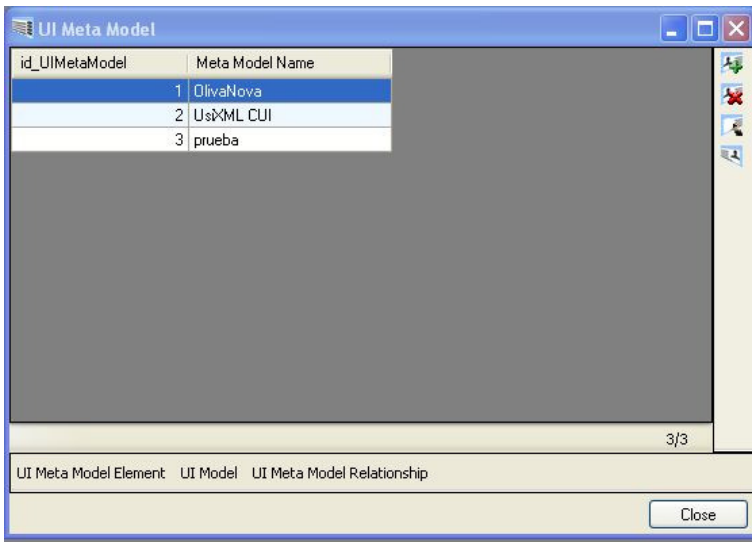


Figura 5.3: Pantalla de Meta Modelos de Interfaces de Usuario del Prototipo de Editor de Plantillas de Transformación

También sería importante y muy útil incorporar facilidades de previsualización, que den al analista que está creando una Plantilla de Transformación una vista previa de cómo quedarían las interfaces de usuario generadas utilizando la Plantilla.

5.2. Compiladores que utilizan Plantillas de Transformación

Además de los Editores, la aproximación de Plantillas de Transformación precisa que los compiladores de modelo a modelo o de modelo a código que corresponden a los Meta Modelos de Interfaces de Usuario con los cuales se trabajan, implementen la posibilidad de trabajar con los Parámetros de una Plantilla de Transformación.

Estos compiladores, deberían tomar como entrada un modelo más abstracto de interfaz de usuario y una Plantilla de Transformación, y generar como salida un modelo más concreto o código de interfaz de usuario. La Figura 4.1 del Capítulo 4 ilustra el uso de una Plantilla de Transformación en el caso de una transformación de un modelo AUI a un modelo CUI.

Las Plantillas de Transformación, con sus Parámetros, Valores y Selectores, proveen indicaciones que determinan cómo transformar el modelo más abstracto al más concreto o código. Siguiendo esas indicaciones la herramienta de

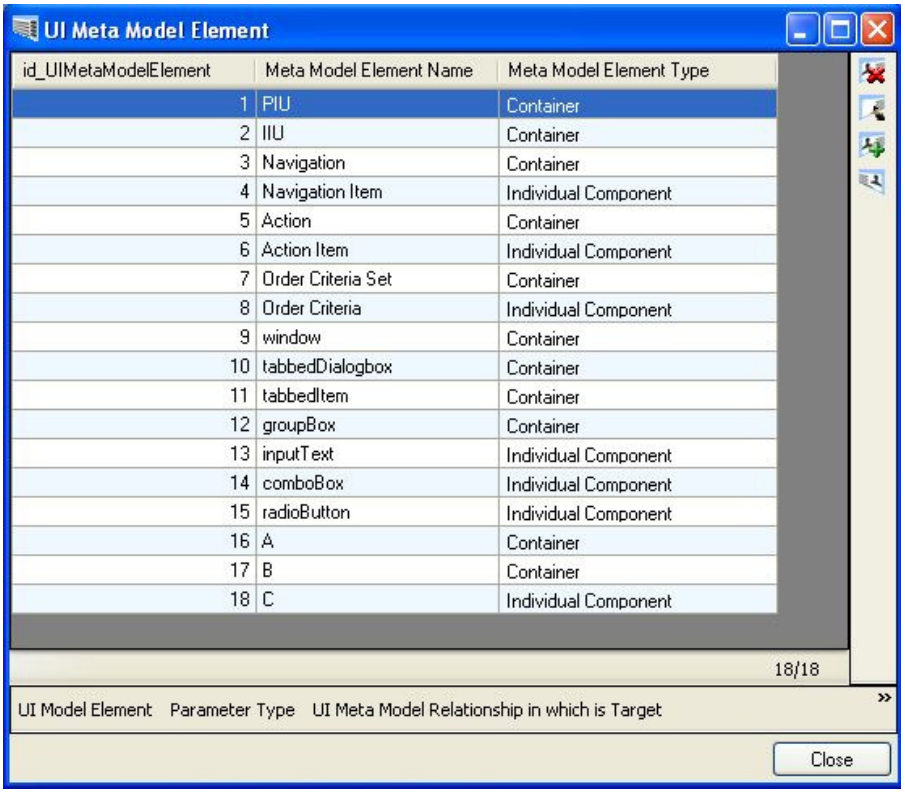


Figura 5.4: Pantalla de Meta Elementos del Prototipo de Editor de Plantillas de Transformación

transformación genera el modelo más concreto o código.

La implementación en un compilador de todos los Tipos de Parámetros para todos los Contextos, puede llegar a tener un costo económico, de tiempo, y de esfuerzo, considerable.

Es por ello que se ha propuesto el mecanismo de clasificar, según estimaciones de su nivel de importancia y costo de implementación en un contexto dado, a los Tipos de Parámetros en cuatro categorías, que se corresponden con etapas sugeridas de implementación (ver Sección 4.3.1). Los Tipos de Parámetro con más importancia y menos costo de implementación son los que se deben implementar en una primera etapa, seguidos de los de gran importancia y alto costo. Luego se podrían implementar los de baja importancia y bajo costo, y finalmente los de baja importancia y alto costo. Evidentemente, las implementaciones pueden parar antes de llegar a la última etapa.

La aproximación de Plantillas de Transformación otorga, entonces, flexibilidad a los responsables de la implementación de los Tipos de Parámetros en los compiladores, y permite que estos compiladores vayan incorporando más Tipos

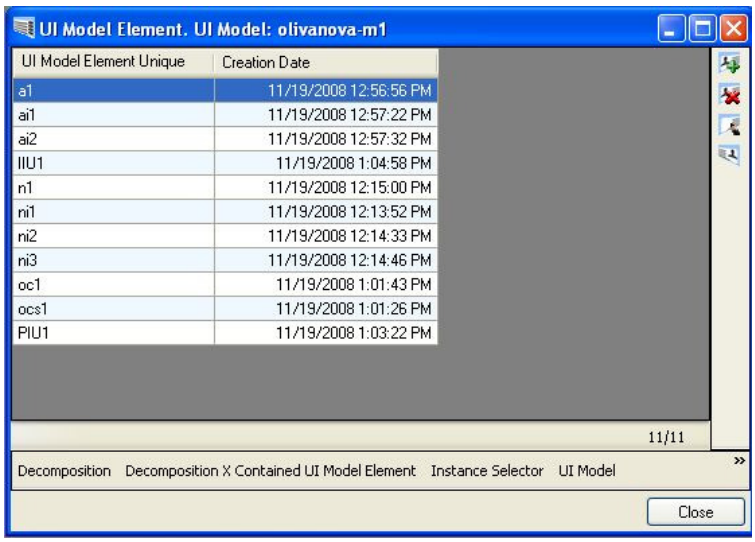


Figura 5.5: Pantalla de Modelos de Interfaces de Usuario del Prototipo de Editor de Plantillas de Transformación

de Parámetros de manera gradual y según las necesidades.

Actualmente, todavía no se han llegado a implementar compiladores de modelo a modelo o de modelo a código de interfaces de usuario que implementen esta aproximación de Plantillas de Transformación.

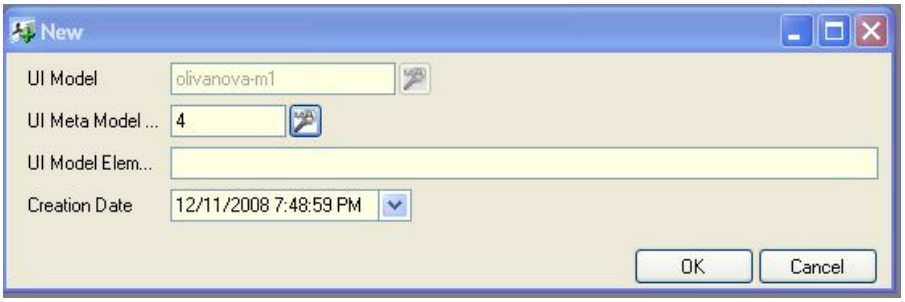


Figura 5.6: Pantalla de creación de un Modelo de Interfaz de Usuario del Prototipo de Editor de Plantillas de Transformación

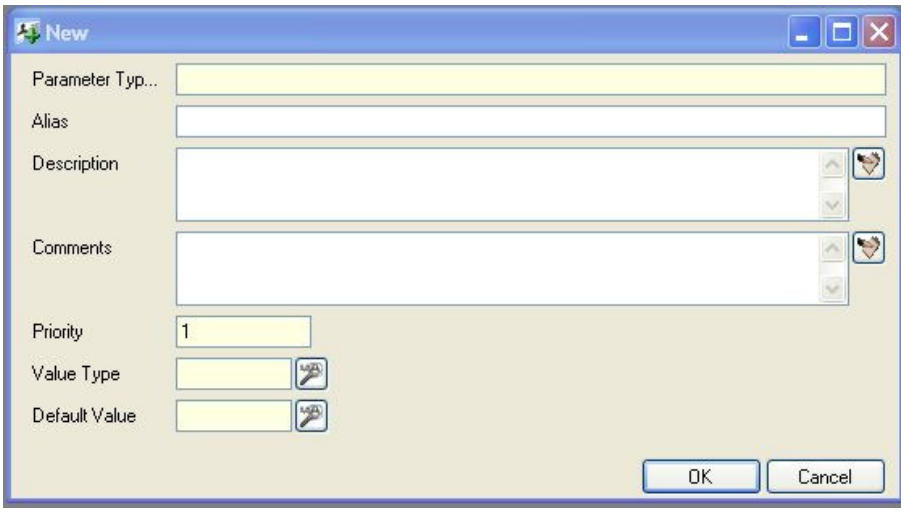


Figura 5.7: Pantalla de creación de un Tipo de Parámetro del Prototipo de Editor de Plantillas de Transformación

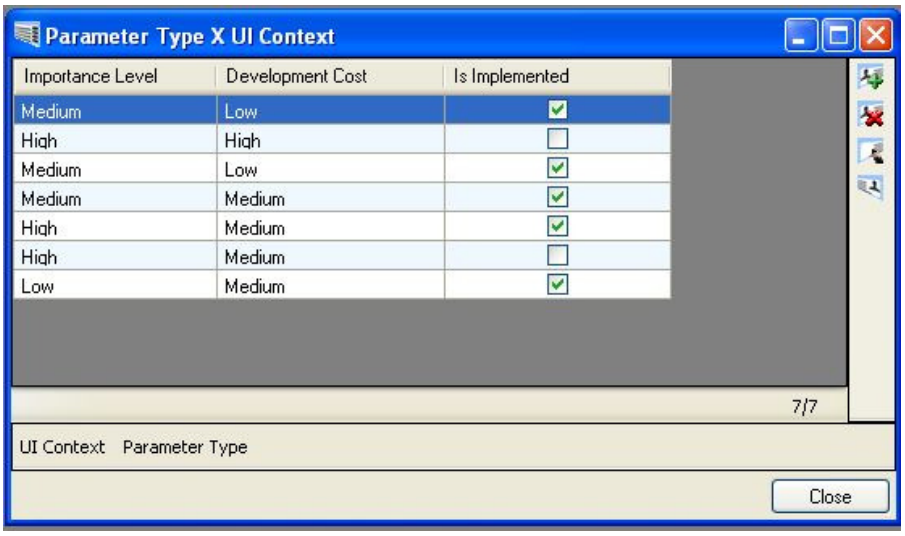


Figura 5.8: Pantalla de información de un Tipo de Parámetro con respecto a un Contexto

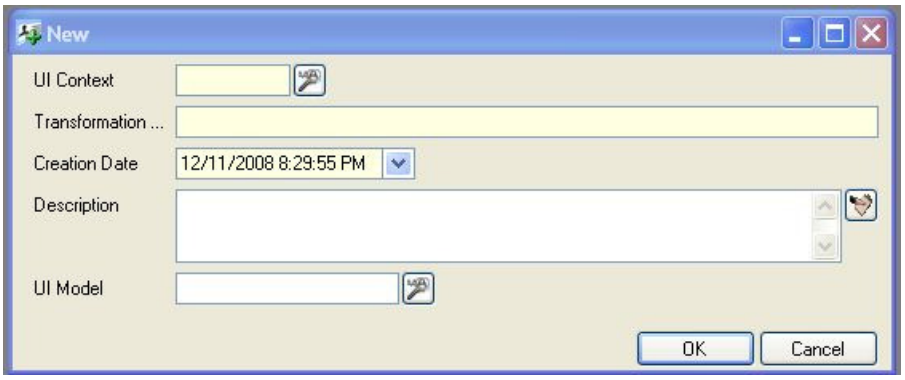


Figura 5.9: Pantalla de creación de una Plantilla de Transformación del Prototipo de Editor de Plantillas de Transformación

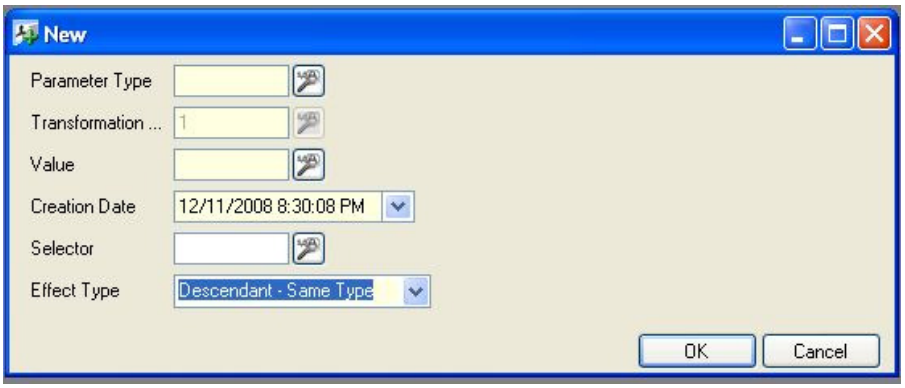


Figura 5.10: Pantalla de creación de un Parámetro del Prototipo de Editor de Plantillas de Transformación

6

Plantillas de Transformación para OO-Method

En este Capítulo se propone una aplicación práctica de la aproximación de Plantillas de Transformación al Modelo de Presentación de OO-Method presentado en la Sección 3.4.1.

La aplicación de la aproximación de Plantillas de Transformación a OO-Method se justifica ya que su Modelo de Presentación constituye una representación abstracta de la interfaz de usuario, basada en patrones de interfaces de usuario, sin mayores detalles sobre el aspecto visual de las mismas, y hoy en día, a partir de este modelo, el compilador de modelos OLIVANOVA *The Programming Machine* genera el código final de las interfaces de usuario gráficas utilizando reglas de transformación que están implícitas en el código mismo del compilador. Como consecuencia, las interfaces de usuario generadas son bastante parecidas unas a otras, y a la hora de satisfacer requisitos de los usuarios finales relacionados con cambios en las interfaces de usuario generadas, es necesario recurrir a modificaciones manuales sobre el código.

Se puede decir entonces, que el soporte completo a los requisitos de las interfaces de usuario es un problema que no está del todo resuelto en ese contexto. Las modificaciones manuales realizadas para cubrir los requisitos no satisfechos en el proceso MDA pueden llevar a los inconvenientes que han sido comentados en la Sección 1.1.

De todo esto se desprende que la aproximación de Plantillas de Transformación resultará útil, por una parte, para externalizar del compilador de modelos (o parametrizar) las decisiones de cómo estructurar y dar estilo a las interfaces de usuario, y por otra parte, para que estas especificaciones de estructura y estilo se puedan manejar y adaptar en la etapa de modelado.

La aplicación de las Plantillas de Transformación a una aproximación de desarrollo MDA de interfaces de usuario, en este caso, al Modelo de Presentación de OO-Method, implica definir los Tipos de Parámetros a ser utilizados en las Plantillas de Transformación. En este caso, este conjunto de Tipos de Parámetros deberá dar soporte a los requisitos de interfaces de usuario que hoy en día no

son soportados y que son frecuentes y relevantes, para evitar, lo más que se pueda, las modificaciones manuales del código de las interfaces de usuario. Al definir los Tipos de Parámetros para el Modelo de Presentación de OO-Method, se deberá especificar la siguiente información (ver Sección 4.3.1) para cada uno de ellos:

- Nombre
- Alias
- Descripción
- Comentarios
- Prioridad
- Conjunto de Meta Elementos del Modelo de Presentación de OO-Method que son afectados por el Tipo de Parámetro y condición para el efecto en el caso de ser necesaria
- Conjunto de Meta Elementos del Modelo de Presentación de OO-Method sobre los que el Tipo de Parámetro puede ser aplicado
- Conjunto de Tipos de Parámetros con los que se puede tener conflicto
- Tipo de valor
- Valor por defecto
- Información relacionada a un Contexto:
  - Estimación del nivel de importancia
  - Estimación de costo de implementación
  - Guías de uso
  - Indicador de implementación

Actualmente, OLIVANOVA puede generar interfaces de usuario gráficas para los siguientes lenguajes y plataformas:

- Web JSF
- Web ASP .Net 2.0
- Desktop C♯ .Net 2.0
- .NET User Interface Thick Forms Architecture

- User Interface Thin Web Architecture ASP.NET Platform
- User Interface Thin Web Architecture JSP platform
- User Interface Thick Forms Architecture Windows platform

Estos lenguajes y plataformas constituyen los contextos ha tener en cuenta al definir los Tipos de Parámetros.

A partir del conjunto de Tipos de Parámetros propuestos se deberá decidir cuáles serán implementados y en cuáles lenguajes y plataformas. Luego de tomar esta decisión, el compilador de modelos de la Suite OLIVANOVA deberá ser modificado para dar soporte a los Tipos de Parámetros que se hayan decidido implementar, en los contextos en los que se haya decidido implementarlos.

Llegando a este punto, se podrán crear Plantillas de Transformación que utilizan Parámetros correspondientes a los Tipos de Parámetros implementados, y estas podrán ser utilizas en el proceso de generación de las interfaces de usuario de las aplicaciones generadas con OLIVANOVA.

Finalmente, para verificar si la aplicación de la aproximación de Plantillas de Transformación al Modelo de Presentación de OO-Method resulta positiva, se deberán realizar evaluaciones de las interfaces de usuario generadas utilizando Plantillas de Transformación.

Como parte del proceso de aplicación de la aproximación de Plantillas de Transformación a OO-Method, se ha definido un Catálogo de Tipos de Parámetros para el Modelo de Presentación.

Este Catálogo de Tipos de Parámetros no incorpora aún las informaciones de conflicto y prioridades, ni las estimaciones de nivel de importancia, costo de implementación y guías de uso.

Los patrones (Meta Elementos) para los que se han definido Tipos de Parámetros son los siguientes:

- Unidad de Interacción de Servicio
- Unidad de Interacción de Instancia
- Unidad de Interacción de Población
- Entrada
- Selección definida
- Agrupador de argumentos
- Filtro
- Criterio de ordenación

- Conjunto de visualización
- Navegación
- Acción

Estos patrones han sido presentados en la Sección 3.4.1.

En la siguiente Sección se presentan, a manera de ejemplo, dos de los Tipos de Parámetros del Catálogo. En estos ejemplos sí se han incorporado las estimaciones de nivel de importancia, costo de implementación, guías de uso y prioridad.

En el Apéndice A se presenta el Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method, completo.

6.1. Tipos de Parámetros para el Modelo de Presentación de OO-Method

En esta Sección se presentan, a manera de ejemplos, dos de los Tipos de Parámetros del Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method.

La Figura 6.1 presenta el Tipo de Parámetro *Selección* que tiene la intención de seleccionar el *widget* que será utilizado para elegir un valor de un conjunto definido de valores posibles.

En el contexto del Modelo de Presentación de OO-Method, este parámetro afecta al patrón de *Selección definida*, y puede ser aplicado sobre patrones de *Selección definida* y de *Unidad de Interacción de Servicio*.

El Tipo de Parámetro de *Selección* tiene como tipo de valor a una enumeración que define cuatro valores posibles:

- *Botones de opción*
- *Lista desplegable* (es el valor por defecto)
- *Cuadro de lista*
- *Árbol selector de meses*

El *Árbol selector de meses* [1] es un *widget* que puede ser utilizado para la selección de un mes del año. Es un *widget* de selección con una semántica asociada. Otros valores posibles con semántica asociada podrían añadirse al Tipo de Parámetro *Selección*, por ejemplos selectores de un día de la semana o de sexo.

[1] El *widget Árbol selector de meses* ha sido desarrollado en el *Belgian Laboratory of Computer-Human Interaction* - http://www.isys.ucl.ac.be/bchi

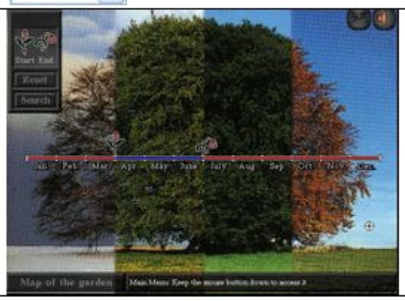
Tipo de Parámetro			
Nombre	Selección		
Descripción	Los parámetros de este tipo permitirán seleccionar el *widget* que será utilizado para elegir un valor de un conjunto definido de valores posibles		
Prioridad	Media		
Afecta a	Selección definida		
Se aplica a	Selección definida, Unidad de Interacción de Servicio		
Estimación de nivel de importancia (NI)	4		
Estimación de costo de implementación (CI)	2		
Enumeración de Valores Posibles			
Valor	*Descripción gráfica*	*NI*	*CI*
Botones de opción	Option 1 Option 2 Option 3	5	1
Lista desplegable (valor por defecto)	Option 1 Option 2 Option 3	5	1
Cuadro de lista	Option 1 Option 2 Option 3	4	1
Árbol selector de meses		2	5

Figura 6.1: Tipo de Parámetro Selección

La estimación del nivel de importancia del Tipo de Parámetro *Selección* es alta mientras que la del costo de implementación es baja, por lo tanto, este Tipo de Parámetro es un buen candidato ha ser implementado en una primera etapa de implementación (ver Sección 4.3.1).

Considerando las estimaciones del nivel de importancia y de costo de implementación de los valores posibles del Tipo de Parámetro *Selección*, las opciones *Botones de opción*, *Lista desplegable* y *Cuadro de lista* podrían ser implementadas en la primera etapa de implementación, y la opción de *Árbol selector de meses* podría ser implementada más tarde, en la cuarta etapa.

Las estimaciones de nivel de importancia y de costo de implementación se hicieron, de manera general, para todos los lenguajes y plataformas para los que el compilador OLIVANOVA *The Programming Machine* puede generar interfaces de usuario.

La Figura 6.2 lista las guías de uso asociadas a cada valor posible del Tipo de Parámetro *Selección*. Las guías de uso tienen la intención de especificar las

condiciones en las cuales sería conveniente usar un valor determinado para un parámetro. La mayoría de las guías de uso fueron extraídas de [Galitz, 2002].

Tipo de Parámetro	
Nombre	Selección
Enumeración de Valores Posibles	
Valor	*Guías de Uso (usar el valor cuando ...)*
Botones de opción	• El número de opciones no es muy grande (2 a 8) • Se necesita acceder fácilmente a las opciones disponibles • Se necesita comparar fácilmente las opciones disponibles • Se dispone del espacio adecuado en pantalla
Lista desplegable	• El número de opciones es grande • Un acceso fácil y rápido a las opciones no es imprescindible • Una comparación fácil y rápida de las opciones no es imprescindible • El espacio disponible en pantalla es limitado • Las opciones no serán seleccionadas con mucha frecuencia
Cuadro de lista	• El número de opciones es grande • Un acceso fácil y rápido a las opciones no es imprescindible • Una comparación fácil y rápida de las opciones no es imprescindible • Se dispone del espacio adecuado en pantalla • Las opciones no serán seleccionadas con mucha frecuencia
Árbol selector de meses	• Una apariencia visual llamativa es importante • Se dispone de espacio adecuado en pantalla

Figura 6.2: Guías de uso para los valores posibles del Tipo de Parámetro Selección

La Figura 6.3 presenta el Tipo de Parámetro *Disposición de grupos* que tiene la intención de seleccionar el tipo de contenedor que será utilizado para agrupar los argumentos de entrada de un servicio.

En el contexto del Modelo de Presentación de OO-Method, este parámetro afecta al patrón de *Agrupación de argumentos* y puede ser aplicado al patrón de *Unidad de Interacción de Servicio*.

El Tipo de Parámetro de *Disposición de grupos* tiene como tipo de valor a una enumeración que define cuatro valores posibles:

- *Cuadros de grupo* (es el valor por defecto)
- *Pestañas*
- *Asistente*
- *Acordeón*

La estimación del nivel de importancia del Tipo de Parámetro *Disposición de grupos* es alta y la del costo de implementación es también un poco alta, por lo tanto, este Tipo de Parámetro podría ser implementado en la segunda etapa de implementación (ver Sección 4.3.1).

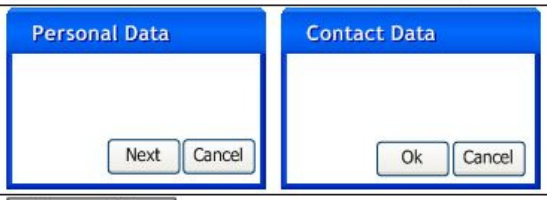
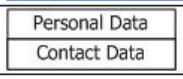
Tipo de Parámetro			
Nombre	Disposición de grupos		
Descripción	Los parámetros de este tipo permiten elegir el tipo de contenedor que será utilizado para agrupar los argumentos de entrada de un servicio		
Prioridad	Baja		
Afecta a	Agrupación de argumentos		
Se aplica a	Unidad de Interacción de Servicio		
Estimación de nivel de importancia (NI)	4		
Estimación de costo de implementación (CI)	3		
Enumeración de Valores Posibles			
Valor	*Descripción gráfica*	*NI*	*CI*
Cuadro de grupo (valor por defecto)		5	2
Pestañas		5	2
Asistente		3	3
Acordeón		2	3

Figura 6.3: Tipo de Parámetro Disposición de grupos

Considerando las estimaciones del nivel de importancia y de costo de implementación de los valores posibles del Tipo de Parámetro *Disposición de grupos*, las opciones *Cuadros de grupo* y *Pestañas* podrían ser implementadas en la primera etapa de implementación, seguidas por *Asistente* en la segunda etapa, y finalmente *Acordeón* en la cuarta etapa.

La Figura 6.4 lista las guías de uso asociadas a cada valor posible del Tipo de Parámetro *Disposición de grupo*. Estas guías de uso fueron extraídas de [Galitz, 2002] y de la Librería de Patrones para Diseño de Interacción (*Pattern Library for Interaction Design*) de Martijn van Welie (http://www.welie.com).

Para ilustrar los Tipos de Parámetros presentados en el contexto del Modelo de Presentación de OO-Method, considérese una *Unidad de Interacción de Servicio* para el registro de fotógrafos en un sistema administrativo de una agencia fotográfica. En el proceso de registro los fotógrafos deben proveer información personal (nombre, D.N.I., edad y sexo), y datos de contacto (teléfono y dirección de correo). La *Unidad de Interacción de Servicio* se estructura con

Tipo de Parámetro	
Nombre	Disposición de grupos
Enumeración de Valores Posibles	
Valor	*Guías de Uso (usar el valor cuando ...)*
Cuadro de grupo	• La distinción visual de grupos es importante • El número total de grupos será pequeño
Pestañas	• La distinción visual de grupos es importante • El número total de grupos no será mayor a 10
Asistente	• El número total de grupos estará entre 3 y 10 • La complejidad de la tarea es significativa • Las tareas implican varias decisiones importantes • El costo de errores es alto • La tarea se realizará de manera poco frecuente • El conocimiento o experiencia para realizar la tarea es limitado
Acordeón	• La distinción visual de grupos es importante • El número total de grupos no será mayor a 10

Figura 6.4: Guías de uso para los valores posibles del Tipo de Parámetro Disposición de grupos

dos *Agrupaciones de argumentos* (Datos Personales y Datos de Contacto) que contienen a los argumentos de entrada del servicio (nombre, teléfono, etc.). Un patrón de *Selección definida* se utiliza para especificar el conjunto de valores posibles para sexo: hombre y mujer.

La Figura 6.5 reproduce la interfaz de usuario generada por OLIVANOVA, para la plataforma Desktop C♯ .Net 2.0, para la *Unidad de Interacción de Servicio* descrita.

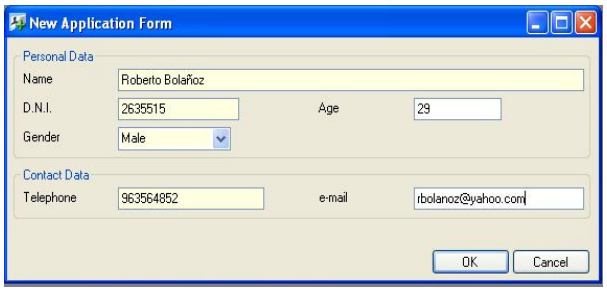


Figura 6.5: Interfaz de usuario generada por OLIVANOVA para el ejemplo de registro de fotógrafo

La Figura 6.6 ilustra la interfaz de usuario que podría ser generada para la *Unidad de Interacción de Servicio* descrita, si un parámetro de *Selección* con

valor *Botones de opción* fuera aplicado a la *Selección definida*, y un parámetro de *Disposición de grupos* con valor *Pestañas* fuera aplicado a la *Unidad de Interacción de Servicio*.

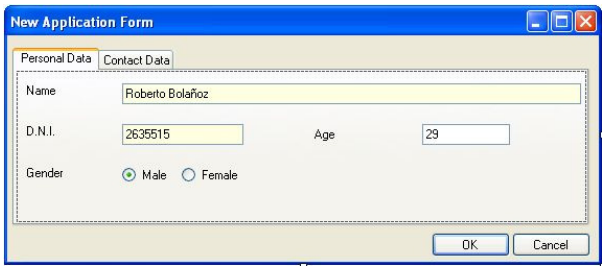


Figura 6.6: Interfaz de usuario esperada para el ejemplo de registro de fotógrafo aplicando parámetros de Selección y de Disposición de grupos

7

Conclusiones

En este Capítulo se presentan las contribuciones del trabajo, junto con una discusión acerca de las mismas. También se presentan los trabajos futuros, y finalmente, las publicaciones realizadas.

7.1. Contribuciones

Las contribuciones de este trabajo son las siguientes:

- Se ha definido una aproximación de Plantillas de Transformación para añadir flexibilidad en las transformaciones de modelos de interfaces de usario.

  Estas Plantillas permiten especificar detalles de la estructura, disposición y estilo de las interfaces de usuario, y se deben utilizar como entrada para un compilador de modelo a modelo de interfaz de usuario, o de modelo a código, de forma tal a parametrizar la lógica que sigue el compilador para hacer la transformación.

  Con respecto a las aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos que tienen la lógica de las transformaciones implícitas en las herramientas de transformación, esta propuesta tiene la ventaja de que externaliza esa lógica, haciéndola adaptable según las características del proyecto en el que se está trabajando. A la vez, permite que la lógica apropiada sea reutilizada en otros proyectos de características similares.

  Con respecto a las aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos que tienen la lógica de las transformaciones explícitas en modelos de mapeos y de transformaciones, esta propuesta tiene la ventaja de que la especificación de los detalles de estructura, disposición y estilo de las interfaces de usuario se realiza de una manera que es bastante sencilla al compararla con el complejo proceso de definir transformaciones de modelos.

  Y con respecto a otras aproximaciones que también utilizan plantillas para definir detalles de las interfaces de usuario, esta aproximación

tiene la ventaja de que no está sujeta a un método de desarrollo de interfaces de usuario en particular, sino que puede ser utilizada en diversas aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos, pudiendo, también guiar las transformaciones hacia diversos tipos de plataformas de cómputo, y para diversos tipos de usuarios.

Por otra parte, esta aproximación también es aplicable a las transformaciones de modelos en las que interviene un modelo de interfaces de usuario basado en patrones de interfaces de usuario.

Los tipos de parámetros que se pueden utilizar en una Plantilla de Transformación no son fijos. Esto permite que en un proceso de desarrollo automatizado de interfaces de usuario, se identifiquen los cambios más frecuentes y relevantes que se hacen sobre el código de las interfaces generadas, y que estos cambios puedan ser abordados como tipos de parámetros nuevos, que una vez implementados, automatizarán el soporte al cambio. Los conocimientos de diseño, y las guías de presentación también pueden ser implementados como tipos de parámetros. Se espera que cuanto más parametrizado esté el proceso de desarrollo de las interfaces de usuario, menos cambios manuales deberán hacerse sobre el código de las interfaces generadas.

Las Plantillas de Transformación proveen mecanismos flexibles para que los diseñadores especifiquen las características de las interfaces de usuario a ser generadas, cuidando que el proceso de especificación de los detalles de la interfaz no sea tedioso. Para esto se proveen valores por defecto para los tipos de parámetros, y varios mecanismos de selección de los elementos de la interfaz que serán afectados por un parámetro.

Además, la aproximación propuesta también contempla la posibilidad de asociar guías de uso a los tipos de parámetros, a fin de guiar al diseñador hacia una Plantilla de Transformación con la cual puedan generarse interfaces de usuario con buena usabilidad.

Las Plantillas de Transformación han sido claramente caracterizadas a nivel de meta-meta modelo y de meta modelo. Además, se las ha relacionado a un meta-meta modelo y a un meta modelo simplificados de interfaces de usuario, a fin de definir como utilizar la aproximación de Plantillas de Transformación con diversas aproximaciones de modelado de interfaces de usuario.

Como la implementación de cada tipo de parámetro que se pretende utilizar en una Plantilla de Transformación implica una modificación en las herramientas de compilación de modelos, se puede identificar al costo de implementación, como el principal problema que plantea esta aproximación. Sin embargo, estas modificaciones pueden realizarse de manera tal a

aumentar las capacidades del compilador de manera incremental. Además, a pesar de que el costo pueda resultar relativamente alto, esta aproximación permite a los diseñadores aplicar una Plantilla de Transformación para adaptar el proceso de generación de las interfaces de usuario de acuerdo a las necesidades de los usuarios finales. A los usuarios finales les gusta especificar sus necesidades y apreciarán verlas incorporadas en los procesos de desarrollo de interfaces de usuario dirigidos por modelos, a diferencia de los procesos tradicionales de desarrollo de interfaces de usuario dirigidos por modelos donde todas las transformaciones están predefinidas y llevan a la generación de interfaces de usuario predeterminadas.

- Se ha implementado un prototipo de editor de Plantillas de Transformación.
- Como aplicación práctica de la aproximación de Plantillas de Transformación, se ha definido un Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method.

7.2. Trabajos Futuros

Este trabajo deja abiertas varias líneas de trabajos futuros. Las mismas se describen a continuación.

- Se puede ampliar la jerarquía de tipos de valores que pueden ser utilizados en las Plantillas de Transformación, a fin de otorgar más poder a la propuesta.
- Se puede ampliar la información que se recopila sobre un tipo de parámetro para decidir si el mismo será implementado o no en un compilador para un contexto determinado. Los costos conativos, cognitivos y computacionales asociados al tipo de parámetro son candidatos para esto.
- Se deben mejorar las capacidades del prototipo de editor de Plantillas de Transformación, incorporando facilidades para importar modelos de interfaces de usuario, y facilidades de previsualización, que den al diseñador que está creando una Plantilla de Transformación una vista previa de cómo quedarían las interfaces de usuario generadas utilizando la Plantilla.
- Se deben analizar las modificaciones necesarias a OLIVANOVA *The Programming Machine*, a fin de incorporar los Tipos de Parámetros del Catálogo presentado en el Anexo A, y proceder a las modificaciones necesarias.
- Se deben realizar evaluaciones relacionadas a la aproximación de Plantillas de Transformación, en términos de su aporte al proceso de desarrollo de

interfaces de usuario dirigido por modelos, de satisfacción del usuario final, y de usabilidad de las interfaces de usuario generadas.

7.3. Trabajos desarrollados en el marco de esta tesis

El trabajo más relacionado con esta tesis es el siguiente:

- Aquino, N.; Vanderdonckt, J.; Valverde, F. y Pastor, Ó.
  Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces.
  Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces CADUI 2008 (Albacete, 11-13 June 2008)
  Springer, 2008 (to appear).

Asimismo, en el periodo en el que se estuvo realizando este trabajo de tesis se realizaron también otros trabajos relacionados al desarrollo de interfaces de usuario dirigo por modelos, como trabajos de definición de modelos de interacción abstractos y de evaluación temprana de usabilidad.

La aproximación de Plantillas de Transformación puede aplicarse en las transformaciones que involucren a los modelos de interacción abstractos definidos, y se podrá analizar la aplicabilidad de los trabajos de evaluación temprana de usabilidad cuando se realicen las evaluaciones relacionadas a las Plantillas de Transformación mencionadas entre los trabajos futuros.

- Valverde, F.; Panach, J. I.; Aquino, N. y Pastor, Ó.
  Hacia un Modelo de Interacción Abstracto para la Definición de Interfaces Multiplataforma
  Proc. of VIII Congreso Internacional de Interacción Persona-Ordenador Interacción'07 (Zaragoza, September 11-14, 2007)
  International Thomson Editores Spain, 2007, 251-260

- Panach, J. I.; Condori-Fernández, N.; Valverde, F.; Aquino, N. y Pastor, O.
  Towards an Early Usability Evaluation for Web Applications
  IWSM/Mensura 2007
  Springer, 2007, LNCS 4895, 32-45

- Panach, J. I.; Condori-Fernández, N.; Valverde, F.; Aquino, N. y Pastor, O.
  Understandability measurement in an early usability evaluation for model-driven development: an empirical study
  ESEM 2008
  ACM, 2008, 354-356

- Pastor, O.; España, S.; Panach, J. I. y Aquino, N.
  Model-Driven Development: piecing together the MDA jigsaw
  Informatik Spektrum, 2008, Vol 31, 394-407

- Iñesta, L.; Sánchez, J. y Aquino, N.
  Descripción de una Herramienta de Autor para una Extensión del Lenguaje UIML
  Actas del IX Congreso Internacional de Interacción Persona-Ordenador INTERACCIÓN 2008 (Albacete, 9-11 Junio 2008)
  2008, 443-452

- Valverde, F.; Panach, I.; Aquino, N. y Pastor, O.
  Dealing with Abstract Interaction Modelling in an MDE Development Process: a Pattern-based Approach
  New Trends on Human-Computer Interaction. Research, Development, New Tools and Methods
  Springer, 2008 (to appear)

Bibliografía

[Abrams et al., 1999] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E.: 1999, Uiml: An appliance-independent xml user interface language, *Computer Networks* **31(11-16)**, 1695–1708

[Aquino et al., 2008] Aquino, N., Vanderdonckt, J., Valverde, F., and Óscar Pastor: 2008, Using profiles to support model transformations in the model-driven development of user interfaces, in V. López, J. P. Molina, F. Montero, and J. Vanderdonckt (eds.), *Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces CADUI 2008 (Albacete, 11-13 June 2008)*, p. (to appear), Springer, ISBN: 978-1-84882-205-4

[Arisholm et al., 1998] Arisholm, E., Benestad, H. C., Skandsen, J., and Fredhall, H.: 1998, Incorporating rapid user interface prototyping in object-oriented analysis and design with genova, in *In Proceedings of NWPER'98 Nordic Workshop on Programming Environment Research (Eds, Mughal*, pp 155–161

[Ashlund et al., 1993] Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E., and White, T. N. (eds.): 1993, *Human-Computer Interaction, INTERACT '93, IFIP TC13 International Conference on Human-Computer Interaction, 24-29 April 1993, Amsterdam, The Netherlands, jointly organised with ACM Conference on Human Aspects in Computing Systems CHI'93*, ACM

[Barclay et al., 2003] Barclay, P. J., Griffiths, T., McKirdy, J., Kennedy, J. B., Cooper, R., Paton, N. W., and Gray, P. D.: 2003, Teallach - a flexible user-interface development environment for object database applications, *J. Vis. Lang. Comput.* **14(1)**, 47–77

[Bastide et al., 2003] Bastide, R., Navarre, D., and Palanque, P. A.: 2003, A tool-supported design framework for safety critical interactive systems, *Interacting with Computers* **15(3)**, 309–328

[Benford et al., 2000] Benford, S., Bederson, B. B., Akesson, K.-P., Bayon, V., Druin, A., Hansson, P., Hourcade, J. P., Ingram, R., Neale, H., O'Malley, C., Simsarian, K. T., Stanton, D., Sundblad, Y., and Taxén, G.: 2000, Designing storytelling technologies to encouraging collaboration between young children, in *CHI*, pp 556–563

[Berti et al., 2004a] Berti, S., Correani, F., Mori, G., Paternò, F., and Santoro, C.: 2004a, Teresa: a transformation-based environment for designing and

developing multi-device interfaces, in E. Dykstra-Erickson and M. Tscheligi (eds.), *CHI Extended Abstracts*, pp 793–794, ACM

[Berti et al., 2004b] Berti, S., Mori, G., Paternò, F., and Santoro, C.: 2004b, A transformation-based environment for designing multi-device interactive applications, in [Vanderdonckt et al., 2004], pp 352–353

[Beyer and Holtzblatt, 1999] Beyer, H. and Holtzblatt, K.: 1999, Contextual design, *interactions* **6(1)**, 32–42

[Bos et al., 2007] Bos, B., Çelik, T., Lie, H. W., and Hickson, I.: 2007, *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*, Technical report, World Wide Web Consortium (W3C)

[Bouillon et al., 2004] Bouillon, L., Vanderdonckt, J., and Chow, K. C.: 2004, Flexible re-engineering of web sites, in [Vanderdonckt et al., 2004], pp 132–139

[Calvary et al., 2001] Calvary, G., Coutaz, J., and Thevenin, D.: 2001, A unifying reference framework for the development of plastic user interfaces, in M. R. Little and L. Nigay (eds.), *EHCI*, Vol. 2254 of *Lecture Notes in Computer Science*, pp 173–192, Springer

[Calvary et al., 2003] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J.: 2003, A unifying reference framework for multi-target user interfaces, *Interacting with Computers* **15(3)**, 289–308

[Card et al., 1983] Card, S. K., Newell, A., and Moran, T. P.: 1983, *The Psychology of Human-Computer Interaction*, L. Erlbaum Associates Inc., Hillsdale, NJ, USA

[Clark, 1999] Clark, J.: 1999, *XSL Transformations (XSLT) Version 1.0*, W3C recommendation, W3C, http://www.w3.org/TR/1999/REC-xslt-19991116

[Cockton et al., 1996] Cockton, G., Clarke, S., Gray., P., and Johnson, C.: 1996, *Critical Issues in User Interface Systems Engineering*, Chapt. Literate Development: Weaving Human Context into Design Specifications, Springer

[Coutaz and Rey, 2002] Coutaz, J. and Rey, G.: 2002, Foundations for a theory of contextors, in [Kolski and Vanderdonckt, 2002], pp 13–34

[Coyette and Vanderdonckt, 2005] Coyette, A. and Vanderdonckt, J.: 2005, A sketching tool for designing anyuser, anyplatform, anywhere user interfaces, in M. F. Costabile and F. Paternò (eds.), *INTERACT*, Vol. 3585 of *Lecture Notes in Computer Science*, pp 550–564, Springer

[Crowley et al., 2000] Crowley, J. L., Coutaz, J., and Bérard, F.: 2000, Things that see, *Commun. ACM* **43(3)**, 54–64

[da Silva, 2000] da Silva, P. P.: 2000, User Interface Declarative Models and Development Environments: A Survey, in *DSV-IS*, pp 207–226

[DeRose and Clark, 1999] DeRose, S. and Clark, J.: 1999, *XML Path Language (XPath) Version 1.0*, W3C recommendation, W3C, http://www.w3.org/TR/1999/REC-xpath-19991116

[Dey et al., 2001] Dey, A. K., Abowd, G. D., and Salber, D.: 2001, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human-Computer Interaction* **16(2)**, 97–166

[Díaz et al., 2005] Díaz, I., Sánchez, J., and Pastor, O.: 2005, Metamorfosis: Un marco para el análisis de requisitos funcionales, in J. Araújo, A. D. Toro, and J. F. e Cunha (eds.), *WER*, pp 233–244

[Furtado et al., 2004] Furtado, E., Furtado, V., Sousa, K. S., Vanderdonckt, J., and Limbourg, Q.: 2004, Knowixml: a knowledge-based system generating multiple abstract user interfaces in usixml, in [Slavík and Palanque, 2004], pp 121–128

[Galitz, 2002] Galitz, W. O.: 2002, *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*, John Wiley & Sons, Inc., New York, NY, USA

[Griffiths et al., 2001] Griffiths, T., Barclay, P. J., Paton, N. W., McKirdy, J., Kennedy, J. B., Gray, P. D., Cooper, R., Goble, C. A., and da Silva, P. P.: 2001, Teallach: a model-based user interface development environment for object databases, *Interacting with Computers* **14(1)**, 31–68

[Insfrán et al., 2002] Insfrán, E., Pastor, O., and Wieringa, R.: 2002, Requirements engineering-based conceptual modelling, *Requir. Eng.* **7(2)**, 61–72

[ISO, 2001] ISO: 2001, *ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model*

[Johnson et al., 1993] Johnson, P., Wilson, S., Markopoulos, P., and Pycock, J.: 1993, Adept: Advanced design environment for prototyping with task models, in [Ashlund et al., 1993], p. 56

[Jouault and Kurtev, 2005] Jouault, F. and Kurtev, I.: 2005, Transforming models with atl, in J.-M. Bruel (ed.), *MoDELS Satellite Events*, Vol. 3844 of *Lecture Notes in Computer Science*, pp 128–138, Springer

[Kolski and Vanderdonckt, 2002] Kolski, C. and Vanderdonckt, J. (eds.): 2002, *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France*, Kluwer

[Lim and Long, 1997] Lim, K. Y. and Long, J. B.: 1997, The muse method for usability engineering, in S. Howard, J. Hammond, and G. Lindgaard (eds.), *INTERACT*, Vol. 96 of *IFIP Conference Proceedings*, pp 676–677, Chapman & Hall

[Limbourg, 2004] Limbourg, Q.: 2004, *Multi-path Development of User Interfaces*, *Ph.D. thesis*, Université catholique de Louvain, IAG-School of Management

[Limbourg and Vanderdonckt, 2004] Limbourg, Q. and Vanderdonckt, J.: 2004, Addressing the mapping problem in user interface design with usixml, in [Slavík and Palanque, 2004], pp 155–163

[Limbourg et al., 2004] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and López-Jaquero, V.: 2004, Usixml: A language supporting multi-path development of user interfaces, in R. Bastide, P. A. Palanque, and J. Roth (eds.), *EHCI/DS-VIS*, Vol. 3425 of *Lecture Notes in Computer Science*, pp 200–220, Springer

[Luyten et al., 2003] Luyten, K., Laerhoven, T. V., Coninx, K., and Reeth, F. V.: 2003, Runtime transformations for modal independent user interface migration, *Interacting with Computers* **15(3)**, 329–347

[Marín, 2008] Marín, B.: 2008, Un Procedimiento de Medición de Tamaño Funcional para Modelos Conceptuales en entornos MDA, *Master's thesis*, Universidad Politécnica de Valencia

[Mellor et al., 2004] Mellor, S. J., Kendall, S., Uhl, A., and Weise, D.: 2004, *MDA Distilled: Principles of Model-Driven Architecture*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA

[Molina et al., 2002] Molina, P. J., Meliá, S., and Pastor, O.: 2002, Just-ui : A user interface specification model, in [Kolski and Vanderdonckt, 2002], pp 63–74

[Montero and López-Jaquero, 2007] Montero, F. and López-Jaquero, V.: 2007, *IdealXml: An Interaction Design Tool - A Task-based Approach to User Interface Design*, Chapt. 20, pp 245–252, ISBN: 978-1-4020-5819-6 (Print) 978-1-4020-5820-2 (Online)

[Mori et al., 2004] Mori, G., Paterno, F., and Santoro, C.: 2004, Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions, *IEEE Trans. Software Eng.* **30(8)**, 507–520

[Mukerji and Miller, 2003] Mukerji, J. and Miller, J.: 2003, *MDA Guide V1.0.1*

[Myers et al., 1998] Myers, B. A., Stiel, H., and Gargiulo, R.: 1998, Collaboration using multiple pdas connected to a pc, in *CSCW*, pp 285–294

[Pastor et al., 1992] Pastor, O., Hayes, F., and Bear, S.: 1992, Oasis: An object-oriented specification language, in *CAiSE*, pp 348–363

[Pastor and Molina, 2007] Pastor, O. and Molina, J. C.: 2007, *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*, Springer-Verlag New York, Inc., Secaucus, NJ, USA

[Paternò and Santoro, 2003] Paternò, F. and Santoro, C.: 2003, A unified method for designing interactive systems adaptable to mobile and stationary platforms, *Interacting with Computers* **15(3)**, 349–366

[Pederiva et al., 2007] Pederiva, I., Vanderdonckt, J., España, S., Panach, J. I., and Pastor, O.: 2007, The beautification process in model-driven engineering of user interfaces, in M. C. C. Baranauskas, P. A. Palanque, J. Abascal, and S. D. J. Barbosa (eds.), *INTERACT (1)*, Vol. 4662 of *Lecture Notes in Computer Science*, pp 411–425, Springer

[Penner and Steinmetz, 2003] Penner, R. R. and Steinmetz, E. S.: 2003, Implementation of automated interaction design with collaborative models, *Interacting with Computers* **15(3)**, 367–385

[Plomp et al., 2002] Plomp, J., Schaefer, R., and Mueller, W.: 2002, Comparing Transcoding Tools for Use with a Generic User Interface Format, in *Extreme Markup Languages*

[Puerta and Eisenstein, 1999a] Puerta, A. R. and Eisenstein, J.: 1999a, Towards a general computational framework for model-based interface development systems, in *Intelligent User Interfaces*, pp 171–178

[Puerta and Eisenstein, 1999b] Puerta, A. R. and Eisenstein, J.: 1999b, Towards a general computational framework for model-based interface development systems, *Knowl.-Based Syst.* **12(8)**, 433–442

[Rekimoto and Saitoh, 1999] Rekimoto, J. and Saitoh, M.: 1999, Augmented surfaces: A spatially continuous work space for hybrid computing environments, in *CHI*, pp 378–385

[Schaefer, 2007] Schaefer, R.: 2007, A survey on transformation tools for model based user interface development, in J. A. Jacko (ed.), *HCI (1)*, Vol. 4550 of *Lecture Notes in Computer Science*, pp 1178–1187, Springer

[Schaefer et al., 2002a] Schaefer, R., Dangberg, A., and Mueller, W.: 2002a, Dangberg A.: RDL/TT - A Description Language for the Profile-Dependent Transcoding of XML Documents, in *In International ITEA Workshop on Virtual Home Environments*

[Schaefer et al., 2002b] Schaefer, R., Dangberg, A., and Mueller, W.: 2002b, Fuzzy rules for html transcoding, in *In Proceedings of the Thirty-Fifth Annual Hawaii International Conference on System Sciences*

[Schlee and Vanderdonckt, 2004] Schlee, M. and Vanderdonckt, J.: 2004, Generative programming of graphical user interfaces, in M. F. Costabile (ed.), *AVI*, pp 403–406, ACM Press

[Selic, 2007] Selic, B.: 2007, A systematic approach to domain-specific language design using uml, in *ISORC*, pp 2–9, IEEE Computer Society

[Slavík and Palanque, 2004] Slavík, P. and Palanque, P. A. (eds.): 2004, *Task Models and Diagrams for User Interface Design: Proceedings of the Third International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2004, November 15 - 16, 2004, Prague, Czech Republic*, ACM

[Sottet et al., 2005] Sottet, J.-S., Calvary, G., and Favre, J.-M.: 2005, Towards model-driven engineering of plastic user interfaces, in A. Pleuß, J. V. den Bergh, H. Hußmann, and S. Sauer (eds.), *MDDAUI*, Vol. 159 of *CEUR Workshop Proceedings*, CEUR-WS.org

[Stanciulescu and Vanderdonckt, 2008] Stanciulescu, A. and Vanderdonckt, J.: 2008, *Design Options For Multimodal Web Applications*, Chapt. Computer-Aided Design Of User Interfaces V, pp 41–56, Springer Netherlands

[Streitz et al., 1999] Streitz, N. A., Geißler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., and Steinmetz, R.: 1999, i-land: An interactive landscape for creativity and innovation, in *CHI*, pp 120–127

[Szekely, 1996] Szekely, P. A.: 1996, Retrospective and challenges for model-based interface development, in F. Bodart and J. Vanderdonckt (eds.), *DSV-IS*, pp 1–27, Springer

[Valverde et al., 2009] Valverde, F., Panach, I., Aquino, N., and Pastor, O.: 2009, *New Trends on Human-Computer Interaction. Research, Development, New Tools and Methods*, Chapt. Dealing with Abstract Interaction Modelling in an MDE Development Process: a Pattern-based Approach, p. (to appear), Springer, ISBN: 978-1-84882-351-8

[Valverde et al., 2007] Valverde, F., Panach, J. I., Aquino, N., and Óscar Pastor: 2007, Hacia un modelo de interacción abstracto para la definición de interfaces multiplataforma, in J. A. M. Iglesias, A. G. i Saltiveri, and P. L. Andrés (eds.), *Proc. of VIII Congreso Internacional de Interacción Persona-Ordenador Interacción'07 (Zaragoza, September 11-14, 2007)*, pp 251–260, International Thomson Editores Spain, Madrid, España, ISBN: 978-84-9732-596-7

[Vanderdonckt, 2005] Vanderdonckt, J.: 2005, A mda-compliant environment for developing user interfaces of information systems, in O. Pastor and J. F. e Cunha (eds.), *CAiSE*, Vol. 3520 of *Lecture Notes in Computer Science*, pp 16–31, Springer

[Vanderdonckt, 2008] Vanderdonckt, J.: 2008, Model-driven engineering of user interfaces: Promises, successes, failures, and challenges, in *5th Romanian Conference on Human-Computer Interaction (RoCHI 2008)*

[Vanderdonckt and Bodart, 1993] Vanderdonckt, J. and Bodart, F.: 1993, Encapsulating knowledge for intelligent automatic interaction objects selection, in [Ashlund et al., 1993], pp 424–429

[Vanderdonckt et al., 2004] Vanderdonckt, J., Nunes, N. J., and Rich, C. (eds.): 2004, *Proceedings of the 2004 International Conference on Intelligent User Interfaces, January 13-16, 2004, Funchal, Madeira, Portugal*, ACM

A

Catálogo de Tipos de Parámetros para el Modelo de Presentación de OO-Method

Contenido

1 Introduction

This document is intended to present a catalog of parameters that could be used to guide the transformation process from an abstract user interface representation, the OO-Method Presentation Model, to code.

Each parameter is described with the following information:

- The name of the parameter
- A textual description
- The set of elements of the OO-Method Presentation Model which are affected by the parameter
    - Conditions: each affected element can has a set of conditions that must be satisfied in order the parameter to affect the element. A condition is a Boolean expression related to elements of the OO-Method Presentation Model or other parameters.
- The data type or set of possible values for the parameter, with a description and/or graphic that illustrates the effect of the value of the parameter
- A default value
- The set of elements of the OO-Method Presentation Model where the parameter can be applied
- A list of parameters that are overwritten by the parameter that is being defined
- Observations

The following template has been used in this document to describe the parameters according to the presented structure.

Name		
Description		
OBS		
Overwrites		
Affects the presentation of		
Element		**Conditions**
Possible values		
Value	**I**	**Effect description – Graphic**
Default value		
Applicable to		

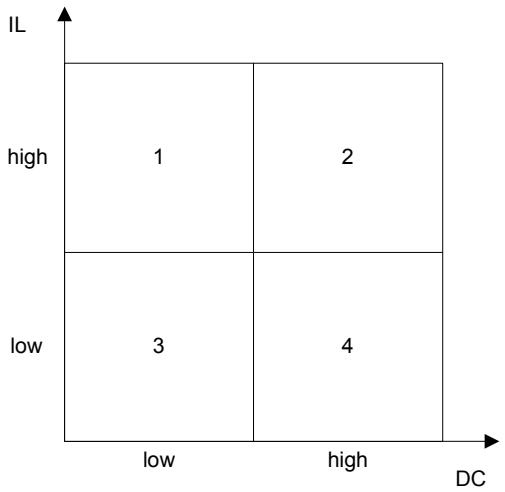
The I column (I is an abbreviation for Implemented) which is next to the Value column in the section of Possible Values of a Parameter is intended to specify if the current value is already implemented in OlivaNova (in this case the value is “Y”). Most of the parameters have a value that corresponds to the current way in which OlivaNova generates the code for a part of the User Interface. The other possible values of the parameters are alternatives for generating a part of the User Interface (in these cases the value is “N”).

Each parameter can be applied with the following scopes:

- Global: the parameter will be applied in the whole application
- Type: the parameter will be applied in a specific type of User Interface Pattern of the OO-Method Presentation Model (all SIUs, all Defined Selections, etc.)
- Instance: the parameter will be applied in a specific instance of a User Interface Pattern of the OO-Method Presentation Model (a specific SIU or Defined Selection, etc.)

The parameters that are specified with Global or Type scopes can be reused in similar projects, or projects for the same client.

An extended specification of the parameters adds estimations of the Importance Level and Development Cost of the parameters, as well as Usability Guidelines to select the value of the parameter. The Importance Level (IL), Development Cost (DC) and Usability Guidelines are added in the Posibble Values section of the Template. The IL and DC will be estimated in a scale that goes from 1 (low) to 5 (high). The estimation of IL and DC is intended to provide a proposal for the order of implementation of the parameters. The following picture illustrates the proposed order of implementation which begins with parameters of high IL and low DC and finishes with parameters of low IL and high DC.



In this section, two examples with the extended specification are presented. Then, Section 2 presents a Catalog of Parameters.

Two extended specified examples

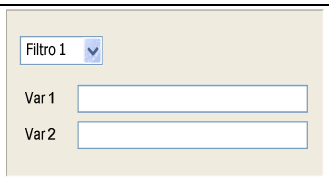
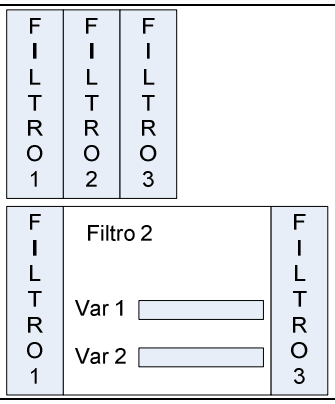
Name	definedSelectionWidget
Description	Allows to select the widget to be used for displaying a Defined Selection.
Overwrites	dateInputWidget
	timeInputWidget
	dateTimeInputWidget
	boolInputWidget
	numericOutputWidget
	stringOutputWidget
	dateOutputWidget
	timeOutputWidget
	dateTimeOutputWidget
	boolOutputWidget
Affects the presentation of	
Element	**Conditions**
Attributes	Attribute has a Defined Selection pattern associated
	The attribute is presented in a display set of a IIU
	The value of the parameter displaySetLayout which affects the current IIU is independentFields
Input argument	Input argument has a Defined Selection pattern associated
Filter variables	Filter Variable has a Defined Selection pattern associated

Possible values					
Value	**I**	**IL**	**DC**	**Effect description – Graphic**	**Usability Guidelines**
radioButton	N	4	2	◉ Opción 1 ○ Opción 2 ○ Opción 3	The number of options is not big (2 to 8) and adequate screen space is available.
comboBox	Y	-	-	Opción 1 Opción 2 Opción 3	The number of options is big and the space in the screen is small.
listBox	N	4	2	Opción 1 Opción 2 Opción 3	The number of options is big and adequate screen space is available.

monthSelector (see the ***Observation*** under this table)	N	2	5		The visual appearance is important and there is adequate available space in the screen.

Default value	comboBox
Applicable to	
Defined Selection	
Display Set	
Filter	
IIU	
PIU	
SIU	

Observation: The monthSelector is a widget that can be used for selecting a month. It is a selector widget with an associated semantic. Other selectors with semantic can be defined for Defined Selections, for example selectors for choosing a day of the week or gender.

Name	filterWidgetLayout				
Description	Allows to select the widget to be used to present the posible Filters of a PIU				
Affects the presentation of					
Element	**Conditions**				
Filter					
Possible values					
Value	**I**	**IL**	**IC**	**Effect description – Graphic**	**Usability Guidelines**
tabbedDialogBox	Y	-	-		The number of filters is not too big, less than 8
comboBox	N	3	2		The number of filters is high
accordion	N	2	4		The number of panels should be small, less than 8
Default value	tabbedDialogBox				
Applicable to					
Filter					
PIU					

2 Parameters Catalog

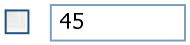
The Catalog presented in this section lists a set of proposed parameters. This list could be extended.

2.1 Parameters related to Attributes, Input Arguments and Filter Variables

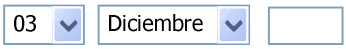
2.1.1 Parameters related to the presentation of data valued Input Arguments and Filter Variables

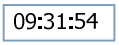
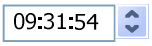
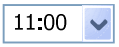
This section presents parameters related to the presentation of autonumber, text, date, time, dateTime and bool Input Arguments and Filter Variables. Each parameter gives options for selecting the widget that will be used to present the Input Arguments and Filter Variables based on their data types.
There are no parameters related to int, nat, real or string Input Arguments or Filter Variables because they can always be presented with inputTexts. There is neither a parameter related to blob Input Arguments because they can always be presented with a fileChooser.

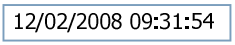
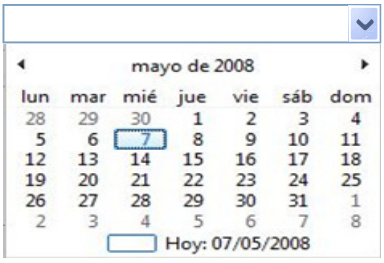
Name		autonumberInputWidgetCreate
Description		Defines the presentation of input arguments of autonumber data type, in services, transactions or operations of creation type
Affects the presentation of		
Element		**Conditions**
Input Argument		Input argument data type is autonumber
		The input argument belongs to a service, transaction or operation of creation type
Possible values		
Value	**I**	**Effect description – Graphic**
notVisible	N	The input argument is not visible in the user interface
label	N	A label with text “Auto” appears
editable	Y	A label with text “Auto” appears with a checked checkbox ☑ Auto If the checkbox is unchecked the label is replaced by an empty inputText for entering a value ☐
Default value		editable
Applicable to		
Input Argument – Argument Group Item		
SIU		
IIU		
PIU		

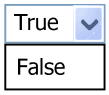
Name		autonumberInputWidgetEdit
Description		Defines the presentation of input arguments of autonumber data type, in services, transactions or operations which are not of creation type
Affects the presentation of		
Element		**Conditions**
Input Argument		Input argument data type is autonumber
		The input argument does not belong to a service, transaction or operation of creation type
Possible values		
Value	**I**	**Effect description – Graphic**
editable	Y	An inputText holding the current autonumber value and an unchecked checkbox appear  If the checkbox is unchecked the inputText is replaced by a label with text “Auto” 
notEditable	N	A readonly inputText appears showing the current autonumber value 
Default value		editable
Applicable to		
Input argument – Argument Group Item		
SIU		
IIU		
PIU		

Name		textInputWidget
Description		Defines the presentation of input arguments of text data type, in services, transactions and operations.
Affects the presentation of		
Element		**Conditions**
Input Argument		Input argument data type is text
Possible values		
Value	**I**	**Effect description – Graphic**
textArea	N	A textArea appears for entering the text 
textAreaEnlarge	Y	A textArea appears for entering the text. Also an Enlarge button appears. When clicked the button opens a popup window with a big textArea for entering the text. Both textAreas display the same information all the time. 
Default value		textAreaEnlarge
Applicable to		
Input argument – Argument Group Item		
SIU		
IIU		
PIU		

Name	dateInputWidget	
Description	Defines the presentation of input arguments of date data type, in services, transactions and operations; and the presentation of variables of date data type in filters.	
Affects the presentation of		
Element	**Conditions**	
Input argument	Input argument data type is date	
Filter variable	Filter variable data type is date	
Possible values		
Value	**I**	**Effect description – Graphic**
inputText	N	An inputText appears for entering a date 
datePickerCalendar	Y	A calendar appears for selecting a date 
datePickerCombo	N	Two comboBoxes and an inputText appear. The first comboBox is for selecting the day (among 1 and 31). The second comboBox is for selecting the months (January, February, ..., December). The inputText is for entering the year. When selecting a month or typing a year the first comboBox is updated to be limited to the correspondent final day of the selected month and year (28, 29, 30 or 31) 
Default value	datePickerCalendar	
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
IIU		
PIU		
SIU		

Name		timeInputWidget
Description		Defines the presentation of input arguments of time data type, in services, transactions and operations; and the presentation of variables of time data type in filters.
Affects the presentation of		
Element		**Conditions**
Input argument		Input argument data type is time
Filter variable		Filter variable data type is time
Possible values		
Value	**I**	**Effect description – Graphic**
inputText	Y	An inputText appears for entering a time in the format hh:mm:ss tt 
spin	N	A spin appears showing an initial time (current time or 00:00:00) in the format hh:mm:ss. Each of hh, mm and ss can be selected independently and augmented or decremented with the spin 
comboBox	N	A comboBox appears for selecting a time. The options in the comboBox are 00:00, 01:00, 02:00, ..., 23:00 
Default value		spin
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
IIU		
PIU		
SIU		

Name		dateTimeInputWidget
Description		Defines the presentation of input arguments of dateTime data type, in services, transactions and operations; and the presentation of variables of dateTime data type in filters.
Affects the presentation of		
Element		**Conditions**
Input argument		Input argument data type is dateTime
Filter variable		Filter variable data type is dateTime
Possible values		
Value	**I**	**Effect description – Graphic**
inputText	N	An inputText appears for entering a dateTime value 
dateTimePickerCalendar	Y	A calendar appears for selecting a date and time 
dateTimePickerCombo	N	Two comboBoxes, an inputText and a spin appear. The first comboBox is for selecting the day (among 1 and 31). The second comboBox is for selecting the months (January, February, ..., December). The inputText is for entering the year. When selecting a month or typing a year the first comboBox is updated to be limited to the correspondent final day of the selected month and year (28, 29, 30 or 31). The spin shows an initial time (current time or 00:00:00) in the format hh:mm:ss. Each of hh, mm and ss can be selected independently and augmented or decremented with the spin. 
Default value		dateTimePickerCalendar
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
IIU		
PIU		
SIU		

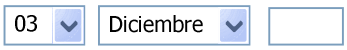
Name	boolInputWidget		
Description	Defines the presentation of input arguments of bool data type, in services, transactions and operations; and the presentation of variables of bool data type in filters.		
Affects the presentation of			
Element	**Conditions**		
Input argument	Input argument data type is bool		
Filter variable	Filter variable data type is bool		
Possible values			
Value	**I**	**Effect description – Graphic**	
checkbox	N	A checkbox appears, when checked it means True 	
radioButton	N	A radioButton appears with two mutually exclusive options, True and False 	
comboBox	Y	A comboBox with two options, True and False, appears 	
Default value	checkbox		
Applicable to			
Input argument – Argument Group Item			
Filter Variable			
Filter			
IIU			
PIU			
SIU			

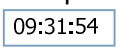
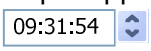
2.1.2 Parameters related to the presentation of data valued Attributes in Display Sets of IIUs which are specified to display their elements in independent fields

Name	numericOutputWidget	
Description	Defines de presentation of attributes of autonumber, int, nat or real data types in display sets of IIUs which are specified to display their elements in independent fields (not on tables)	
Affects the presentation of		
Element	**Conditions**	
Attributes	Attribute data type is autonumber, int, nat or real	
	The attribute is presented in a display set of a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Possible values		
Value	**I**	**Effect description – Graphic**
label	N	A label appears displaying the numeric value
inputText	Y	An inputText appears displaying the numeric value 45
Default value	inputText	
Applicable to		
Attribute – Display Set Item		
Display Set		
IIU		

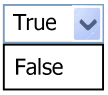
Name	stringOutputWidget	
Description	Defines de presentation of attributes of string data type in display sets of IIUs which are specified to display their elements in independent fields (not on tables)	
Affects the presentation of		
Element	**Conditions**	
Attributes	Attribute data type is string	
	The attribute is presented in a display set of a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Possible values		
Value	**I**	**Effect description – Graphic**
label	N	A label appears displaying the string value
inputText	Y	An inputText appears displaying the string value
Default value	inputText	
Applicable to		
Attribute - Display Set Item		
Display Set		
IIU		

Name	textOutputWidget	
Description	Defines de presentation of attributes of text data type in display sets of IIUs which are specified to display their elements in independent fields (not on tables)	
Affects the presentation of		
Element	**Conditions**	
Attributes	Attribute data type is text	
	The attribute is presented in a display set of a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Possible values		
Value	**I**	**Effect description – Graphic**
label	N	A label appears displaying the text value
textArea	Y	A textArea appears for displaying the text Presentation basic elements are the building blocks from which abstract user interfaces can be constructed. The represent specific aspects of the interaction between a human and a system, and can be composed into interaction units.
Default value	textArea	
Applicable to		
Attribute - Display Set Item		
Display Set		
IIU		

Name	dateOutputWidget	
Description	Defines de presentation of attributes of date data type in display sets of IIUs which are specified to display their elements in independent fields (not on tables)	
Affects the presentation of		
Element	**Conditions**	
Attributes	Attribute data type is date	
	The attribute is presented in a display set of a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Possible values		
Value	**I**	**Effect description – Graphic**
label	N	A label appears displaying the date value
inputText	Y	An inputText appears for displaying the date 
datePickerCombo	N	Two comboBoxes and an inputText appear. The first comboBox is for displaying the day (among 1 and 31). The second comboBox is for displaying the month (January, February, ..., December). The inputText is for displaying the year. 
Default value	inputText	
Applicable to		
Attribute - Display Set Item		
Display Set		
IIU		

Name	timeOutputWidget		
Description	Defines de presentation of attributes of time data type in display sets of IIUs which are specified to display their elements in independent fields (not on tables)		
Affects the presentation of			
Element	**Conditions**		
Attributes	Attribute data type is time		
	The attribute is presented in a display set of a IIU		
	The value of the parameter displaySetLayout which affects the current IIU is independentFields		
Possible values			
Value	**I**	**Effect description – Graphic**	
label	N	A label appears displaying the dateTime value	
inputText	Y	An inputText appears for displaying the time in the format hh:mm:ss 	
spin	N	A spin appears displaying the time in the format hh:mm:ss 	
comboBox	N	A comboBox appears for displaying the time. The options in the comboBox are 00:00, 01:00, 02:00, ..., 23:00. 	
Default value	inputText		
Applicable to			
Attribute - Display Set Item			
Display Set			
IIU			

Name	dateTimeOutputWidget	
Description	Defines de presentation of attributes of dateTime data type in display sets of IIUs which are specified to display their elements in independent fields (not on tables)	
Affects the presentation of		
Element	**Conditions**	
Attributes	Attribute data type is dateTime	
	The attribute is presented in a display set of a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Possible values		
Value	**I**	**Effect description – Graphic**
label	N	A label appears displaying the dateTime value
inputText	Y	An inputText appears displaying the dateTime value 12/02/2008 09:31:54
dateTimePickerCombo	N	Two comboBoxes, an inputText and a spin appear. The first comboBox is for displaying the day (among 1 and 31). The second comboBox is for displaying the month (January, February, ..., December). The inputText is for displaying the year. The spin displays the time. 03 Diciembre 09:31:54
Default value	inputText	
Applicable to		
Attribute - Display Set Item		
Display Set		
IIU		

Name	boolOutputWidget	
Description	Defines de presentation of attributes of bool data type in display sets of IIUs which are specified to display their elements in independent fields (not on tables)	
Affects the presentation of		
Element	**Conditions**	
Attributes	Attribute data type is bool	
	The attribute is presented in a display set of a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Possible values		
Value	**I**	**Effect description – Graphic**
label	N	A label appears displaying the bool value
checkbox	Y	A checkbox appears, when checked it means True 
radioButton	N	A radioButton appears with two mutually exclusive options, True and False. The current value is selected. 
comboBox	N	A comboBox with two options, True and False, appears. The current value is selected. 
Default value	checkBox	
Applicable to		
Attribute - Display Set Item		
Display Set		
IIU		

Name		blobOutputWidget
Description		Defines de presentation of attributes of blob data type in display sets of IIUs which are specified to display their elements in independent fields (not on tables)
Affects the presentation of		
Element		**Conditions**
Attributes		Attribute data type is blob
		The attribute is presented in a display set of a IIU
		The value of the parameter displaySetLayout which affects the current IIU is independentFields
Possible values		
Value	**I**	**Effect description – Graphic**
label	N	A label appears displaying the blob data
textArea	N	A textArea appears displaying the blob data Presentation basic elements are the building blocks from which abstract user interfaces can be constructed. The represent specific aspects of the interaction between a human and a system, and can be composed into interaction units.
imageComponent	N	An image appears 
videoComponent	N	a component for displaying a video appears 
link	N	A link that opens an external application appears (the external application is intended to show a document, image, sound or video) Acceso al documento
Default value		link
Applicable to		
Attribute - Display Set Item		
Display Set		
IIU		

2.1.3 Parameters related to the presentation of data valued Attributes in Display Sets of IIUs which are specified to display their elements in tables, and in Display Sets of PIUs

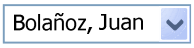
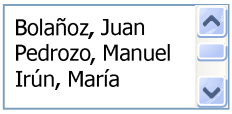
This section presents parameters related to the presentation of bool and blob Attributes in Display Sets of IIUs which are specified to display their elements in tables, and in Display Sets of PIUs.
There are no parameters related to autonumber, int, nat, real, string, text, date, time and dateTime Attributes because they can always be presented with labels.

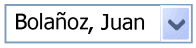
Name		boolTableWidget
Description		Defines de presentation of attributes of bool data type in display sets of PIUs and in display sets of IIUs which are specified to display their elements in tables.
Affects the presentation of		
Element		**Conditions**
Attributes		Attribute data type is bool
		The attribute is presented in a display set of a PIU or a IIU
		In case the attribute is presented in a displayset of a IIU, the value of the parameter displaySetLayout which affects the current IIU is table
Possible values		
Value	**I**	**Effect description – Graphic**
label	Y	A label appears displaying the value
Image	N	An image that represents the value appears ☑
Default value		label
Applicable to		
Attribute – Display Set Item		
Display Set		
IIU		
PIU		

Name		blobTableWidget
Description		Defines de presentation of attributes of blob data type in display sets of PIUs and in display sets of IIUs which are specified to display their elements in tables.
Affects the presentation of		
Element		**Conditions**
Attributes		Attribute data type is blob
		The attribute is presented in a display set of a PIU or a IIU
		In case the attribute is presented in a displayset of a IIU, the value of the parameter displaySetLayout which affects the current IIU is table
Possible values		
Value	**I**	**Effect description – Graphic**
label	Y	A label appears displaying the blob data
imageComponent	N	An image appears 
link	N	A link that opens an external application appears (the external application is intended to show a document, image, sound or video) Acceso al documento
Default value		link
Applicable to		
Attribute – Display Set Item		
Display Set		
IIU		
PIU		

2.1.4 Parameters related to the presentation of object valued Input Arguments and Filter Variables

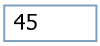
Name		objectWidget
Description		Defines the presentation of object valued input arguments in services, transactions and operations, and the presentation of object valued filter variables in filters of populations when they are not indicated to be preloaded.
Affects the presentation of		
Element		**Conditions**
Input argument		Input argument is object valued
		Preload = false
Filter variables		Filter variable is object valued
		Preload = false
Possible values		
Value	**I**	**Effect description – Graphic**
searchEdit	Y	An inputText and a magnifying glass icon appear. When the icon is pressed a PIU is opened for selecting an object. The code of the object (identifier - id) can be directly entered in the inputText when the user knows the code. 
search	N	A magnifying glass icon appears. When the icon is pressed a PIU is opened for selecting an object. 
Default value		searchEdit
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
IIU		
PIU		
SIU		

Name	preloadWidget	
Description	Defines the presentation of object valued input arguments in services, transactions and operations, and the presentation of object valued filter variables in filters of populations when they are indicated to be preloaded.	
Affects the presentation of		
Element	**Conditions**	
Input argument	Input argument is object valued	
	Preload = true	
Filter variables	Filter variable is object valued	
	Preload = true	
Possible values		
Value	**I**	**Effect description – Graphic**
comboBox	Y	A comboBox appears for selecting the object 
listBox	N	A listBox appears for selecting the object 
Default value	comboBox	
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
IIU		
PIU		
SIU		

Name		preloadAndSearch
Description		Defines the presentation of object valued input arguments in services, transactions and operations, and the presentation of object valued filter variables in filters of populations when they are indicated to be preloaded. Parameter preloadWidget allows to select the widget to be used and the current parameter, preloadAndSearch, allows to specify if the "Search" capability is also going to be offered in the user interface.
OBS		If this parameter is going to be implemented, then the Preload and PIU selection should not be mutually exclusive in the definition of Filter Variables and Input Arguments of Services, Transactions and Operations.
Affects the presentation of		
Element		**Conditions**
Input argument		Input argument is object valued
		Preload = true
Filter variables		Filter variable is object valued
		Preload = true
Possible values		
Value	**I**	**Effect description – Graphic**
Yes	N	
No	Y	
Default value		No
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
IIU		
PIU		
SIU		

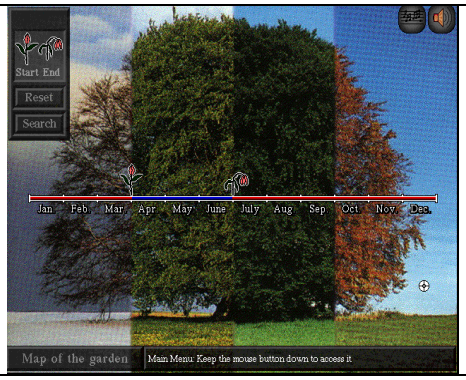
2.2 Parameters related to Auxiliaries or Elementary Patterns

2.2.1 Parameters related to Introduction

Name		introductionIntWidget
Description		Defines the presentation of the Introduction Pattern when it is related to the int data type.
OBS		This parameter needs a dataType attribute to be included in the Introduction pattern in OlivaNova. The dataType attribute in Introduction exists in Molina´s thesis and the “Model-Driven Architecture in Practice” book, but not in OlivaNova.
Overwrites		numericOutputWidget
Affects the presentation of		
Element		**Conditions**
Attributes		Attribute has an Introduction pattern associated
		The Introduction data type is nat or int
		The introduction has not null lower and upper ranges
		The attribute is presented in a display set of a IIU
		The value of the parameter displaySetLayout which affects the current IIU is independentFields
Input argument		Input argument has an Introduction pattern associated
		The Introduction data type is nat or int
		The introduction has not null lower and upper ranges
Filter variables		Filter Variable has an Introduction pattern associated
		The Introduction data type is nat or int
		The introduction has not null lower and upper ranges
Possible values		
Value	**I**	**Effect description – Graphic**
inputText	Y	
slider	N	
spin	N	
Default value		inputText
Applicable to		
Introduction		
Display Set		
Filter		
IIU		
PIU		
SIU		

2.2.2 Parameters related to Defined Selection

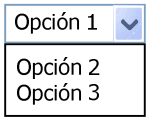
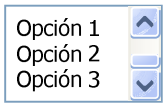
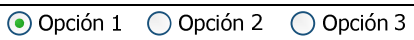
Name	definedSelectionWidget				
Description	Allows to select the widget to be used for displaying a Defined Selection.				
Overwrites	dateInputWidget				
	timeInputWidget				
	dateTimeInputWidget				
	boolInputWidget				
	numericOutputWidget				
	stringOutputWidget				
	dateOutputWidget				
	timeOutputWidget				
	dateTimeOutputWidget				
	boolOutputWidget				
Affects the presentation of					
Element	**Conditions**				
Attributes	Attribute has a Defined Selection pattern associated				
	The attribute is presented in a display set of a IIU				
	The value of the parameter displaySetLayout which affects the current IIU is independentFields				
Input argument	Input argument has a Defined Selection pattern associated				
Filter variables	Filter Variable has a Defined Selection pattern associated				
Possible values					
Value	**I**	**IL**	**IC**	**Effect description – Graphic**	**Usability Guidelines**
radioButton	N	4	2	◉ Opción 1 ○ Opción 2 ○ Opción 3	The number of options is not big (2 to 8) and adequate screen space is available.
comboBox	Y	-	-	Opción 1 Opción 2 Opción 3	The number of options is big and the space in the screen is small.
listBox	N	4	2	Opción 1 Opción 2 Opción 3	The number of options is big and adequate screen space is available.

monthSelector (see the ***Observation*** under this table)	N	2	5		The visual appearance is important and there is adequate available space in the screen.

Default value	comboBox
Applicable to	
Defined Selection	
Display Set	
Filter	
IIU	
PIU	
SIU	

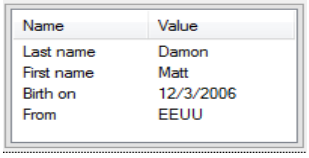
Observation: The monthSelector is a widget that can be used for selecting a month. It is a selector widget with an associated semantic. Other selectors with semantic can be defined for Defined Selections, for example selectors for choosing a day of the week or gender.

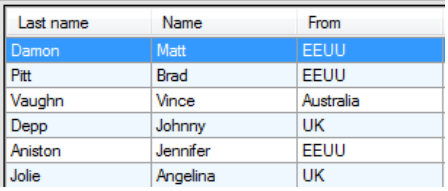
2.2.3 Parameters related to Order Criteria

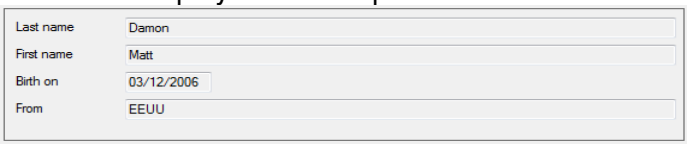
Name		orderCriteriaWidget
Description		Allows to select the widget to be used to present the posible Order Criterias of a PIU
Affects the presentation of		
Element		**Conditions**
Order Criteria		
Possible values		
Value	**I**	**Effect description – Graphic**
comboBox	Y	
listBox	N	
radioButton	N	
toogleButtonSet	N	
Default value		comboBox
Applicable to		
Order Criteria		
PIU		

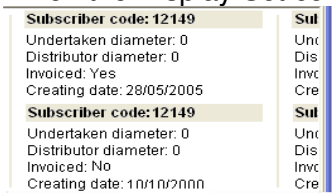
2.2.4 Parameters related to Display Set

<table>
<tr><td colspan="2">Name</td><td>displaySetLayout</td></tr>
<tr><td colspan="2">Description</td><td>Allows to specify if a display set is going to be displayed with a table or with a set of independent fileds.</td></tr>
<tr><td colspan="3">Affects the presentation of</td></tr>
<tr><td colspan="2">Element</td><td>Conditions</td></tr>
<tr><td colspan="2">Display Set</td><td></td></tr>
<tr><td colspan="3">Possible values</td></tr>
<tr><td>Value</td><td>I</td><td>Effect description – Graphic</td></tr>
<tr><td>table</td><td>Y

Y</td><td>When the Display Set corresponds to a IIU:
 

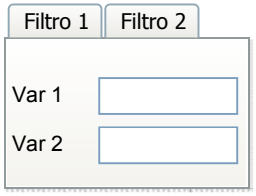
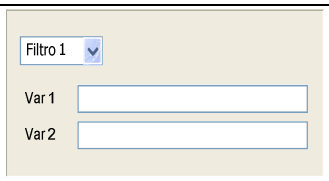
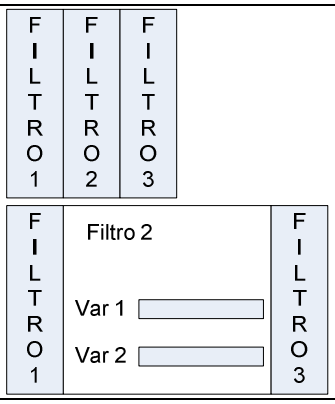
When the Display Set corresponds to a PIU:
  </td></tr>
<tr><td>independentFields</td><td>Y

N</td><td>When the Display Set corresponds to a IIU:
 

When the Display Set corresponds to a PIU:
  </td></tr>
<tr><td colspan="2">Default value</td><td>table</td></tr>
<tr><td colspan="3">Applicable to</td></tr>
<tr><td colspan="3">Display Set</td></tr>
<tr><td colspan="3">IIU</td></tr>
<tr><td colspan="3">PIU</td></tr>
</table>

Name	outputWidgetStyle	
Description	Allows to specify if the widgets that are present in a IIU (with a Display Set with independent fields) are going to be displayed in a readOnly or disabled fashion.	
Affects the presentation of		
Element	**Conditions**	
DisplaySet	The display set is presented in a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Possible values		
Value	**I**	**Effect description – Graphic**
readOnly	Y	The widgets appear as readonly Model-Driven Architecture in Practice
disabled	N	The widgets appear disabled Model-Driven Architecture in Practice
Default value	readOnly	
Applicable to		
Display Set		
IIU		

2.2.5 Parameters related to Filter

Name	filterWidgetLayout				
Description	Allows to select the widget to be used to present the posible Filters of a PIU				
Affects the presentation of					
Element	**Conditions**				
Filter					
Possible values					
Value	**I**	**IL**	**IC**	**Effect description – Graphic**	**Usability Guidelines**
tabbedDialogBox	Y	-	-		The number of filters is not too big, less than 8
comboBox	N	3	2		The number of filters is high
accordion	N	2	4		The number of panels should be small, less than 8
Default value	tabbedDialogBox				
Applicable to					
Filter					
PIU					

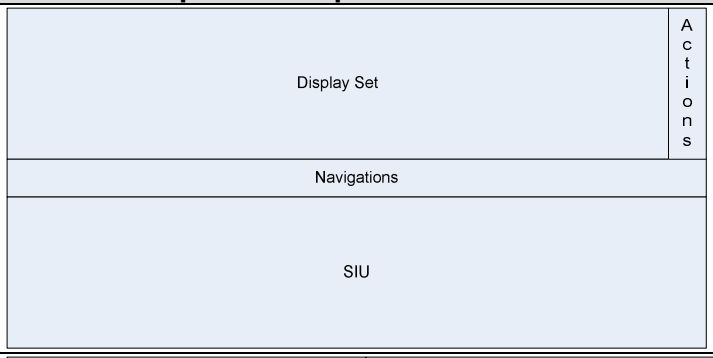
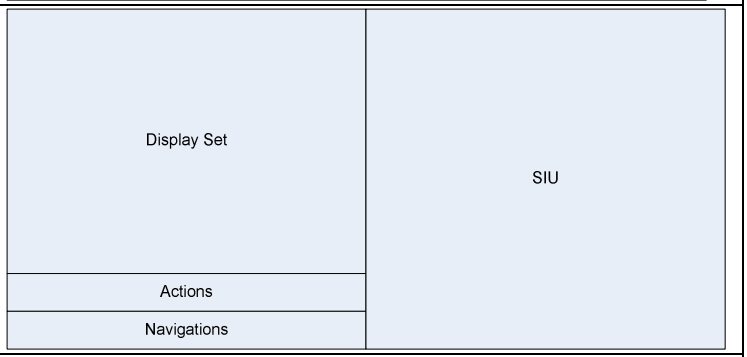
2.2.6 Parameters related to Actions and Navigations

Name	actionImageAlias	
Description	Specifies if an action item is going to be presented with its alias, its image (see parameters anNewImage, anDestroyImage, anSIUImage, anIIUImage, anPIUImage, anMDIUImage), or both.	
Affects the presentation of		
Element	**Conditions**	
Action Item		
Possible values		
Value	**I**	**Effect description – Graphic**
alias	N	The action item is represented just with its alias 
image	Y	The action item is represented just with its associated image 
aliasImage	N	The action item is represented with its alias and image 
Default value	image	
Applicable to		
Action Item		
Actions		
IIU		
PIU		

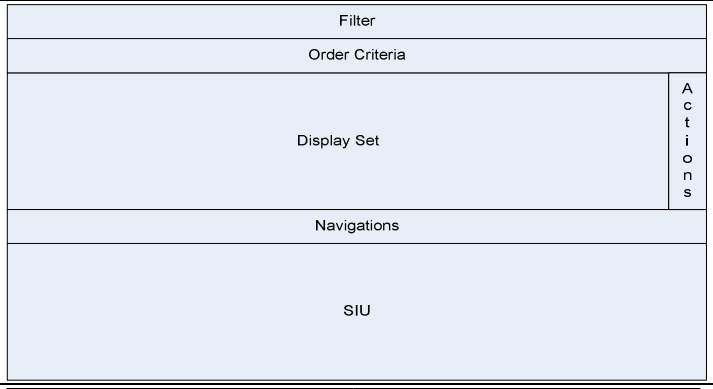
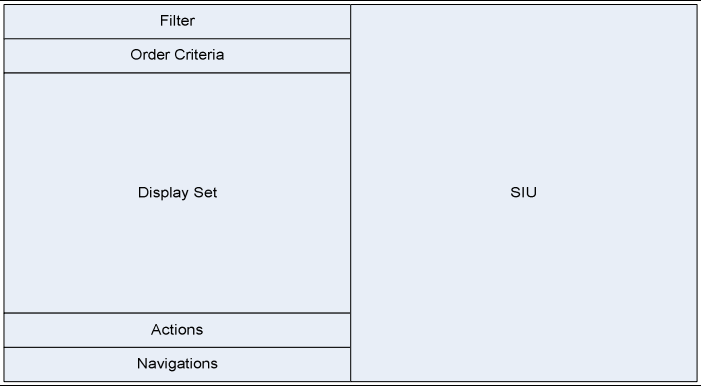
Name		navigationImageAlias
Description		Specifies if a navigation item is going to be presented with its alias, its image (see parameters anIIUImage, anPIUImage, anMDIUImage), or both.
Affects the presentation of		
Element		**Conditions**
Navigation Item		
Possible values		
Value	**I**	**Effect description – Graphic**
alias	Y	The action item is represented just with its alias 
aliasImage	N	The navigation ítem is represented with its alias and associated image, in case it has been specified in the parameters anIIUImage, anPIUImage or anMDIUImage 
Default value		alias
Applicable to		
Navigation Item		
Navigations		
IIU		
PIU		

2.3 Parameters related to Interaction Units

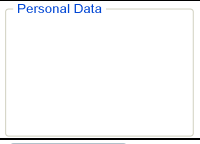
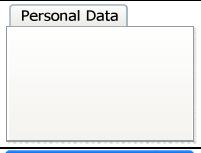
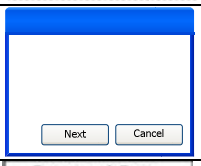
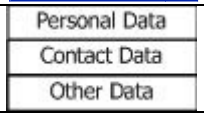
2.3.1 Parameters related to IIU

Name		layoutIIU
Description		Provides options for the structure of the IIU
Affects the presentation of		
Element		**Conditions**
IIU		
Possible values		
Value	**I**	**Effect description – Graphic**
layoutIIUopt1	Y	
layoutIIUopt2	N	
Default value		layoutIIUopt1
Applicable to		
IIU		

2.3.2 Parameters related to PIU

Name		layoutPIU
Description		Provides options for the structure of the PIU
Affects the presentation of		
Element		**Conditions**
PIU		
Possible values		
Value	**I**	**Effect description – Graphic**
layoutPIUopt1	Y	
layoutPIUopt2	N	
Default value		layoutPIUopt1
Applicable to		
PIU		

2.3.3 Parameters related to SIU

Name		layoutGroups
Description		Provides options for the structure of Argument Groups of a SIU
Affects the presentation of		
Element		**Conditions**
SIU		
Possible values		
Value	**I**	**Effect description – Graphic**
dialogBox	Y (when there are not argument groups)	
groupBox	Y (when there are arguments groups and less than 15 input arguments)	
tabbedDialogBox	Y (when there are argument groups and more than 15 input arguments)	
wizard	N	
accordion	N	
Default value		dialogBox
Applicable to		
SIU		
IIU		
PIU		

Name		notNullIndicator
Description		Allows to specify the way in which the not null input arguments are going to be distinguished from the nullable input arguments
Affects the presentation of		
Element		**Conditions**
Input Argument		Input argument is not null
Possible values		
Value	**I**	**Effect description – Graphic**
bgColor	Y	A background color is used in all widgets that are intended to enter a not null input argument
star	N	A star is placed to the right of the widgets that are intended to enter a not null input argument *
Default value		bgColor
Applicable to		
SIU		
IIU		
PIU		

Name			notNullColor
Description			Specifies the background color to be used in the widgets intended to enter a not null input argument
Affects the presentation of			
Element			**Conditions**
Input Argument			Input argument is not null
			The value of the parameter notNullIndicator that affects the current Input Argument is bgColor
Possible values			
Value	**I**	**Effect description – Graphic**	
A colour code	N		
Default value			A selected colour code
Applicable to			
SIU			
IIU			
PIU			

2.4 Other Parameters

2.4.1 Parameters related to Icons/Images

Name	fileChooserIcon	
Description	Specifies the icon to be used in the button that opens a fileChooser. The fileChooser is used in services, transactions or operations for blob input arguments.	
Affects the presentation of		
Element	**Conditions**	
Input argument	Input argument data type is blob	
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
Input argument – Argument Group Item		
SIU		
IIU		
PIU		

Name	enlargeIcon	
Description	Specifies the icon to be used in the button that opens a new window with a textArea for entering a text value. This is used in services, transactions or operations for text input arguments.	
Affects the presentation of		
Element	**Conditions**	
Input Argument	Input argument data type is text	
	The value of parameter textInputWidget is textAreaEnlarge	
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
Input argument – Argument Group Item		
SIU		
IIU		
PIU		

Name	searchIcon		
Description	Specifies the icon to be used in the button that opens a PIU for selecting an object, and in the button that executes a filter in a PIU.		
Affects the presentation of			
Element	**Conditions**		
Input Argument	Input argument is object valued		
Filter Variable	Filter variable is object valued		
Filter in a PIU			
Possible values			
Value	**I**	**Effect description – Graphic**	
The path to the image to show	N		
Default value	The path of a default image to show		
Applicable to			
Input argument – Argument Group Item			
Filter Variable			
Filter			
SIU			
IIU			
PIU			

Name	OkIcon	
Description	Specifies the icon to be used in OK buttons of SIUs	
Affects the presentation of		
Element	**Conditions**	
SIU		
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
SIU		
IIU		
PIU		

Name		cancelIcon
Description		Specifies the icon to be used in Cancel buttons of SIUs
Affects the presentation of		
Element		**Conditions**
SIU		
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value		The path of a default image to show
Applicable to		
SIU		
IIU		
PIU		

Name	nextIcon	
Description	Specifies the icon to be used in Next buttons of SIUs (if their Arguments Groups are presented like a wizard)	
Affects the presentation of		
Element	**Conditions**	
SIU		
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
SIU		
IIU		
PIU		

Name	closeIcon		
Description	Specifies the icon to be used in Close buttons of IIUs and PIUs		
Affects the presentation of			
Element	**Conditions**		
IIU			
PIU			
Possible values			
Value	**I**	**Effect description – Graphic**	
The path to the image to show	N		
Default value	The path of a default image to show		
Applicable to			
IIU			
PIU			

Name	boolTrueImage		
Description	Specifies the icon to be used for representing attributes of bool data type, with true value, in display sets of PIUs and in display sets of IIUs which are specified to display their elements in tables.		
Affects the presentation of			
Element	**Conditions**		
Attributes	Attribute data type is bool, and its value is true		
	The attribute is presented in a display set of a PIU or a IIU		
	In case the attribute is presented in a displayset of a IIU, the value of the parameter displaySetLayout which affects the current IIU is independentFields		
	The value of the parameter boolTableWidget is image		
Possible values			
Value	**I**	**Effect description – Graphic**	
The path to the image to show	N		
Default value	The path of a default image to show		
Applicable to			
Attribute – Display Set Item			
Display Set			
IIU			
PIU			

Name	boolFalseImage	
Description	Specifies the icon to be used for representing attributes of bool data type, with false value, in display sets of PIUs and in display sets of IIUs which are specified to display their elements in tables.	
Affects the presentation of		
Element	**Conditions**	
Attributes	Attribute data type is bool, and its value is false	
	The attribute is presented in a display set of a PIU or a IIU	
	In case the attribute is presented in a displayset of a IIU, the value of the parameter displaySetLayout which affects the current IIU is independentFields	
	The value of the parameter boolTableWidget is image	
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
Attribute – Display Set Item		
Display Set		
IIU		
PIU		

Name	anNewImage	
Description	Specifies the icon to be used for representing action items related to services, transactions or operations of creation type	
Affects the presentation of		
Element	**Conditions**	
Action Item	Action Item is related to a creation service, transaction or operation	
	The value of parameter actionImageAlias is image or aliasImage	
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
Action Item		
Actions		
IIU		
PIU		

Name	anDestroyImage	
Description	Specifies the icon to be used for representing action items related to services, transactions or operations of destroy type	
Affects the presentation of		
Element	**Conditions**	
Action Item	Action Item is related to a destroy service, transaction or operation	
	The value of parameter actionImageAlias is image or aliasImage	
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
Action Item		
Actions		
IIU		
PIU		

Name	anSIUImage	
Description	Specifies the icon to be used for representing action items related to SIUs of services, transactions or operations which are not of creation or destroy type	
Affects the presentation of		
Element	**Conditions**	
Action Item	Action Item is related to a SIU of a service, transaction or operation which is not of creation or destroy type	
	The value of parameter actionImageAlias is image or aliasImage	
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value	The path of a default image to show	
Applicable to		
Action Item		
Actions		
IIU		
PIU		

Name		anIIUImage
Description		Specifies the icon to be used for representing action or navigation items related to IIU
Affects the presentation of		
Element		**Conditions**
Action Item		The value of parameter actionImageAlias is image or aliasImage
		Action Item is related to an IIU
Navigation Item		The value of parameter navigationImageAlias is aliasImage
		Navigation Item is related to an IIU
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value		The path of a default image to show
Applicable to		
Action Item		
Navigation Item		
Actions		
Navigations		
IIU		
PIU		

Name		anPIUImage
Description		Specifies the icon to be used for representing action or navigation items related to PIU
Affects the presentation of		
Element		**Conditions**
Action Item		The value of parameter actionImageAlias is image or aliasImage
		Action Item is related to an PIU
Navigation Item		The value of parameter navigationImageAlias is aliasImage
		Navigation Item is related to an PIU
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value		The path of a default image to show
Applicable to		
Action Item		
Navigation Item		
Actions		
Navigations		
IIU		
PIU		

Name		anMDIUImage
Description		Specifies the icon to be used for representing action or navigation items related to MDIU
Affects the presentation of		
Element		**Conditions**
Action Item		The value of parameter actionImageAlias is image or aliasImage
		Action Item is related to an MDIU
Navigation Item		The value of parameter navigationImageAlias is aliasImage
		Navigation Item is related to an MDIU
Possible values		
Value	**I**	**Effect description – Graphic**
The path to the image to show	N	
Default value		The path of a default image to show
Applicable to		
Action Item		
Navigation Item		
Actions		
Navigations		
IIU		
PIU		

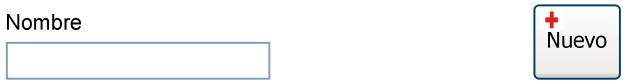
2.4.2 Parameters related to Masks

Name		intMask
Description		Specifies a mask to be applied in all autonumber, int or nat data presented in SIUs, Filters and Display Sets of PIUs and IIUs.
Affects the presentation of		
Element		**Conditions**
Input Arguments		Input Argument data type is autonumber, int or nat
Filter Variables		Filter Variable data type is autonumber, int or nat
Attributes		Attribute data type is autonumber, int or nat
Possible values		
Value	**I**	**Effect description – Graphic**
###.###	N	(el punto es el separador de miles)
###,###	N	(la coma es la separadora de miles)
Default value		###.###
Applicable to		
Input Argument – Argument Group Item		
Filter Variable		
Attribute – Display Set Item		
Filter		
Display Set		
SIU		
IIU		
PIU		

Name		realMask
Description		Specifies a mask to be applied in all real data presented in SIUs, Filters and Display Sets of PIUs and IIUs.
Affects the presentation of		
Element		**Conditions**
Input Arguments		Input Argument data type is real
Filter Variables		Filter Variable data type is real
Attributes		Attribute data type is real
Possible values		
Value	**I**	**Effect description – Graphic**
###.###,###	N	(el punto es el separador de miles y la coma es la separadora de decimales)
###,###.###	N	(la coma es la separadora de miles y el punto es el separador de decimales)
Default value		###.### ,###
Applicable to		
Input Argument – Argument Group Item		
Filter Variable		
Attribute – Display Set Item		
Filter		
Display Set		
SIU		
IIU		
PIU		

Name		dateMask
Description		Specifies a mask to be applied in all date and dateTime data presented in SIUs, Filters and Display Sets of PIUs and IIUs.
Affects the presentation of		
Element		**Conditions**
Input Arguments		Input Argument data type is date or dateTime
Filter Variables		Filter Variable data type is date or dateTime
Attributes		Attribute data type is date or dateTime
Possible values		
Value	**I**	**Effect description – Graphic**
dd/mm/yyyy	N	
mm/dd/yyyy	N	
yyyy/mm/dd	N	
Default value		dd/mm/yyyy
Applicable to		
Input Argument – Argument Group Item		
Filter Variable		
Attribute – Display Set Item		
Filter		
Display Set		
SIU		
IIU		
PIU		

2.4.3 Parameters related to alignment

Name	alignmentLabelWidgetImage	
Description	Specifies the alignment relationship between a label and a widget (for attributes in display sets, input arguments and filter variables); or between an image and the text in a button (for actions and navigations)	
Affects the presentation of		
Element	**Conditions**	
Attribute	The attribute is presented in a display set of a IIU	
	The value of the parameter displaySetLayout which affects the current IIU is independentFields	
Input argument		
Filter variable		
ActionItem	The value of parameter actionImageAlias is aliasImage	
NavigationItem	The value of parameter navigationImageAlias is aliasImage	
Possible values		
Value	**I**	**Effect description – Graphic**
upLeft	N	
upCenter	N	
upright	N	
leftTop	Y (the label/widget case)	
leftMiddle	N	
leftBottom	N	
Default value	leftTop	
Applicable to		
Attribute – Display Set Item		
Input argument – Argument Group Item		
Filter Variable		
Action Item		
Navigation Item		
Display Set		
Filter		
Actions		
Navigations		

IIU
PIU
SIU

Name		colNumber
Description		The number of columns for structuring the content (input arguments, filter variables and display set items) of Argument Groups, Filters and Display Sets of PIUs (independent fields)
Affects the presentation of		
Element		**Conditions**
Argument Group		
Filter		
Display Set		The Display Set is presented in a IIU in which the value of the parameter displaySetLayout is independentFields
Possible values		
Value	**I**	**Effect description – Graphic**
An integer number	N	
Default value		2
Applicable to		
Argument Group		
Filter		
Display Set		

Name		rowNumber
Description		The number of rows for structuring the content (input arguments, filter variables and display set items) of Argument Groups, Filters and Display Sets of PIUs (independent fields)
Affects the presentation of		
Element		**Conditions**
Argument Group		
Filter		
Display Set		The Display Set is presented in a IIU in which the value of the parameter displaySetLayout is independentFields
Possible values		
Value	**I**	**Effect description – Graphic**
An integer number	N	
Default value		
Applicable to		
Argument Group		
Filter		
Display Set		

Name		col
Description		The column in which the Argument Group Item, Filter Variable or Display Set Item is going to be placed.
Affects the presentation of		
Element		**Conditions**
Argument Group Item		
Filter Variable		
Display Set Item		The item is presented in a display set of a IIU
		The value of the parameter displaySetLayout which affects the current IIU is independentFields
Possible values		
Value	**I**	**Effect description – Graphic**
An integer number	N	
Default value		
Applicable to		
Argument Group Item		
Filter Variable		
Display Set Item		

Name		row
Description		The row in which the Argument Group Item, Filter Variable or Display Set Item is going to be placed.
Affects the presentation of		
Element		**Conditions**
Argument Group Item		
Filter Variable		
Display Set Item		The item is presented in a display set of a IIU
		The value of the parameter displaySetLayout which affects the current IIU is independentFields
Possible values		
Value	**I**	**Effect description – Graphic**
An integer number	N	
Default value		
Applicable to		
Argument Group Item		
Filter Variable		
Display Set Item		

Name		colSpan
Description		Defines the number of columns that the Argument Group Item, Filter Variable or Display Set Item is going to span.
Affects the presentation of		
Element		**Conditions**
Argument Group Item		
Filter Variable		
Display Set Item		The item is presented in a display set of a IIU
		The value of the parameter displaySetLayout which affects the current IIU is independentFields
Possible values		
Value	**I**	**Effect description – Graphic**
An integer number	N	
Default value		1
Applicable to		
Argument Group Item		
Filter Variable		
Display Set Item		

Name		rowSpan
Description		Defines the number of rows that the Argument Group Item, Filter Variable or Display Set Item is going to span.
Affects the presentation of		
Element		**Conditions**
Argument Group Item		
Filter Variable		
Display Set Item		The item is presented in a display set of a IIU
		The value of the parameter displaySetLayout which affects the current IIU is independentFields
Possible values		
Value	**I**	**Effect description – Graphic**
An integer number	N	
Default value		1
Applicable to		
Argument Group Item		
Filter Variable		
Display Set Item		

Name		numericAlignment
Description		Specifies the alignment in relation to the container (an inputText, a cell in a table, etc.) of all numeric data presented in SIUs, Filters and Display Sets of PIUs and IIUs.
Affects the presentation of		
Element		**Conditions**
Input Arguments		Input Argument data type is autonumber, nat, int or real
Filter Variables		Filter Variable data type is autonumber, nat, int or real
Attributes		Attribute data type is autonumber, nat, int or real
Possible values		
Value	**I**	**Effect description – Graphic**
left	Y	10.000
center	N	10.000
right	N	10.000
Default value		right
Applicable to		
Input Argument – Argument Group Item		
Filter Variable		
Attribute – Display Set Item		
Filter		
Display Set		
SIU		
IIU		
PIU		

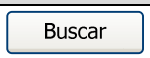
Name		textAlignment
Description		Specifies the alignment in relation to the container (an inputText, a cell in a table, etc.) of all string and text data presented in SIUs, Filters and Display Sets of PIUs and IIUs.
Affects the presentation of		
Element		**Conditions**
Input Arguments		Input Argument data type is string or text
Filter Variables		Filter Variable data type is string or text
Attributes		Attribute data type is string or text
Possible values		
Value	**I**	**Effect description – Graphic**
left	Y	El trueno entre las hojas
center	N	El trueno entre las hojas
right	N	El trueno entre las hojas
Default value		left
Applicable to		
Input Argument – Argument Group Item		
Filter Variable		
Attribute – Display Set Item		
Filter		
Display Set		
SIU		
IIU		
PIU		

Name		dateTimeAlignment
Description		Specifies the alignment in relation to the container (an inputText, a cell in a table, etc.) of all date, time and dateTime data presented in SIUs, Filters and Display Sets of PIUs and IIUs.
Affects the presentation of		
Element		**Conditions**
Input Arguments		Input Argument data type is date, time or dateTime
Filter Variables		Filter Variable data type is date, time or dateTime
Attributes		Attribute data type is date, time or dateTime
Possible values		
Value	**I**	**Effect description – Graphic**
left	Y	03/12/2007
center	N	03/12/2007
right	N	03/12/2007
Default value		center
Applicable to		
Input Argument – Argument Group Item		
Filter Variable		
Attribute – Display Set Item		
Filter		
Display Set		
SIU		
IIU		
PIU		

2.4.4 Others

Name		boolTrueLabel
Description		The string that will be used to represent a True Boolean value in radioButtons, comboBoxes, and labels that are used in SIUs, Filters and Display Sets for entering or displaying Boolean values.
Affects the presentation of		
Element		**Conditions**
Input argument		Input argument data type is bool and its value is true
		The value of parameter boolInputWidget is radioButton or comboBox
Filter variable		Filter variable data type is bool and its value is true
		The value of parameter boolInputWidget is radioButton or comboBox
Attributes		Attribute data type is bool and its value is true
		The value of parameter boolOutputWidget that affects the current attribute is label, radioButton or comboBox; or the value of parameter boolTableWidget that affects the current attribute is label.
Possible values		
Value	**I**	**Effect description – Graphic**
A string that represents the True Boolean value	N	
Default value		"True"
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
Attribute – Display Set Item		
Display Set		
IIU		
PIU		
SIU		

Name		boolFalseLabel
Description		The string that will be used to represent a False Boolean value in radioButtons, comboBoxes, and labels that are used in SIUs, Filters and Display Sets for entering or displaying Boolean values.
Affects the presentation of		
Element		**Conditions**
Input argument		Input argument data type is bool and its value is false
		The value of parameter boolInputWidget is radioButton or comboBox
Filter variable		Filter variable data type is bool and its value is false
		The value of parameter boolInputWidget is radioButton or comboBox
Attributes		Attribute data type is bool and its value is false
		The value of parameter boolOutputWidget that affects the current attribute is label, radioButton or comboBox; or the value of parameter boolTableWidget that affects the current attribute is label.
Possible values		
Value	**I**	**Effect description – Graphic**
A string that represents the False Boolean value	N	
Default value		“False”
Applicable to		
Input argument – Argument Group Item		
Filter Variable		
Filter		
Attribute – Display Set Item		
Display Set		
IIU		
PIU		
SIU		

Name		gralControlsImageAlias
Description		Specifies if a control button (search, enlarge, Ok, Cancel, Next, Close) is going to be presented with an alias, its associated image (see parameters fileChooserIcon, enlargeIcon, searchIcon, OkIcon, cancelIcon, nextIcon, closeIcon) or both
Possible values		
Value	**I**	**Effect description – Graphic**
alias	Y	
image	N	
aliasImage	N	
Default value		alias
Applicable to		
SIU		
IIU		
PIU		

2.5 More Parameters to define

This section lists other parameters that could be defined.

- Foreground color
- Background color
- Transparency rate
- Width and height of containers
- Background image
- Position of windows
- Type and style of fonts
- Properties of video components (alternate image, autoplay, isLoop, builtInControl)
- Visited/Active link color
- Properties of sliders (step and orientation)
- Properties of spin (orientation)
- Properties of comboBoxes and listBoxes (maxLineVisible)
- With of columns in Display Sets of PIUs and IIUs (table)
- …