Lenguajes para el MDUID: un análisis de propuestas existentes

MSc. Juan Carlos Mejias Cruz Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 1⁄2, Reparto Torrens, La Lisa, La Habana, Cuba. C.P.: 19370
ancarlosmejiascruz@gmail.com

MSc. Sandy Suárez Jiménez Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 1⁄2, Reparto Torrens, La Lisa, La Habana, Cuba. C.P.: 19370

Dr.C. Nemury Silega Martínez Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 1⁄2, Reparto Torrens, La Lisa, La Habana, Cuba. C.P.: 19370

Fecha de recibido: 2 de julio de 2019

Fecha de aprobado: 4 de julio de 2019

Abstract— La creación de una interfaz de usuario (IU) con la calidad idónea para satisfacer las necesidades de los usuarios finales, es una tarea compleja y costosa. Por ello, se afirma que el desarrollo de IU precisa de un proceso formal y bien definido que asegure la calidad del mismo; mediante la constitución de los métodos de desarrollo de software, basados en MDA. Este nuevo enfoque tiene el objetivo de proveer un ambiente donde puedan diseñarse e implementarse IU de manera profesional y sistemática. Para ello, estas se describen utilizando modelos, donde juegan un papel determinante los Lenguajes de Descripción de IU (UIDL). Luego dichos modelos son transformados sucesivamente gracias al uso de Lenguajes de Transformación de IU (UITL) hasta llegar a la obtención del código fuente de la IU final. En este trabajo se realiza un análisis de los lenguajes para el desarrollo de IU siguiendo el Desarrollo Dirigido por Modelos de Interfaces de Usuario (MDUID); se enfatiza en los UIDL declarativos y en los UITL con el propósito de ayudar a los diseñadores de IU a elegir la tecnología más adecuada en el contexto específico del desarrollo.

Keywords— Interfaz de usuario, Desarrollo dirigido por modelos, transformación, UIDL, UITL.

Abstract— Creating a quality user interface (UI) to meet the needs of end

users is complex and expensive. Therefore, it is stated that UI development requires a formal and well-defined process to ensure its quality; through the constitution of software development methods, based on MDA. This new approach aims to provide an environment where UI can be designed and implemented in a professional and systematic way. To do this, they are described using models, where UI Description Languages (UIDL) play a decisive role. These models are then successively transformed thanks to the use of UI Transformation Languages (UITL) until the source code of the final UI is obtained. In this work an analysis of the languages for the development of UI is carried out following the Directed Development by User Interface Models (MDUID); it is emphasized in declarative UIDLs and UITLs to help UI designers choose the most appropriate technology in the specific development context.

Keywords— User Interface, Model Driven Development, Transformation, UIDL, UITL.

I. INTRODUCCIÓN

La creación de una interfaz de usuario con calidad suficiente para satisfacer las necesidades de los usuarios, es una tarea compleja y costosa en la cual se suele aplicar un proceso iterativo, para conseguir el resultado final a través de una serie de refinamientos. Una interfaz de usuario (IU) permite establecer el vínculo entre el usuario y el programa que se ejecuta. Su desarrollo ha sido tradicionalmente, una labor artesanal llevada a cabo por los mismos desarrolladores de software, dependiendo la calidad de la habilidad y experticia.

Por una parte, contar con un IU de calidad 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 diseñarlas e implementarlas. Debido al impacto de la creación de interfaces en el proceso de desarrollo, es necesario contar con un proceso formal y bien definido que asegure la calidad de las mismas y que sea automatizable y re-aplicable.

Los métodos de desarrollo de software basados en una Arquitectura Dirigida por Modelos (MDA, por las siglas en inglés de Model-Driven Architecture) [1] incluyen definiciones que pueden aprovecharse. Este enfoque provee un ambiente que facilita el diseño e implementación de interfaces de usuario de manera profesional y sistemática. En el contexto MDA, las IU se describen utilizando modelos a diferentes niveles de abstracción.

Da Silva [2] distingue tres ventajas derivadas del uso de modelos de interfaces de usuario: 1. Pueden proveer una descripción más abstracta de una IU que las descripciones de interfaces de usuario provistas por las herramientas tradicionales de desarrollo de IU.

2. Facilitan la creación de métodos para diseñar e implementar una IU de manera sistemática y a que proveen capacidades para:

- Modelar IU a distintos niveles de abstracción.
- Refinar los modelos de manera incremental.
- Reutilizar especificaciones de IU.

3. Proveen la infraestructura requerida para automatizar las tareas relacionadas al diseño de la IU y a los procesos de implementación.

Diversas soluciones se han propuesto para el Desarrollo Dirigido por Modelos de Interfaces de Usuario (MDUID, por las siglas en inglés de Model-Driven User Interface Development), pero actualmente no se tiene una aproximación ampliamente aceptada y difundida. Cameleon [3] es un Marco de Referencia que cuenta con el apoyo de la comunidad, lo que se evidencia en la variedad de trabajos que adoptan sus definiciones. Cameleon descompone el contexto de uso de una IU en tres aspectos:

1. Los usuarios finales.
2. La plataforma de computación hardware y software.
3. El ambiente físico.

Asimismo, estructura el ciclo de desarrollo en cuatro niveles de abstracción:

1. Tareas y Conceptos (Dominio)
2. Interfaz de Usuario Abstracta
3. Interfaz de Usuario Concreta
4. Interfaz de Usuario Final

Estos niveles se encuentran desde los más abstractos a los más concretos, y contemplan 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 a nivel de los conceptos y tareas que se deben llevar a cabo en el dominio (Ver Figura 1).

Figura 1: Modelo de IU
Fuente: Elaboración Propia

A partir de ese modelo es posible derivar otros modelos de IU 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 IU. Las transformaciones se realizan, idealmente, de forma automática. Es clave el uso de Lenguajes de Descripción de Interfaces de Usuario (UIDL, por las siglas en inglés de User Interface Description Language) y de Lenguajes de Transformación de Interfaces de Usuario (UITL, por las siglas de User Interface Transformación Language) entre cada uno de los modelos definidos previamente.

En el presente trabajo se realiza un análisis de los lenguajes existentes para el desarrollo de IU según el paradigma MDUID. Las motivaciones principales para su realización pueden resumirse de la siguiente forma:

I. Ofrecer una visión general de los lenguajes que han sido más utilizados y difundidos en la

literatura relacionada con enfoques MDUID, que facilite su selección en el contexto específico del desarrollo.

II. Para los autores, constituye el punto de partida en la creación de una herramienta que de soporte al proceso de desarrollo MDUID con todos los beneficios que este paradigma proporciona a las organizaciones desarrolladoras de software.

II. MATERIALES Y MÉTODOS

II.I Lenguajes de Descripción de IU (UIDL)

Los lenguajes de descripción de IU están estrechamente relacionados con estilos de programación y se pueden dividir en dos categorías - declarativos e imperativos. Los lenguajes declarativos indican qué mostrar en la IU, mientras que los lenguajes imperativos se encargan de cómo hacer para mostrar los componentes de la IU. La principal ventaja de la programación declarativa es que decidiendo lo que se quiere mostrar, se hace uso de una herramienta automática para encontrar la manera de producirlo. Eso contrasta con la programación imperativa convencional, donde el programador tiene que indicar, paso a paso, cómo alcanzar el estado deseado.

Los lenguajes declarativos actuales comúnmente toman formas de especificaciones de marcado como HTML, XML y otros lenguajes relacionados. Los imperativos incluyen lenguajes de programación convencionales, tales como los lenguajes orientados a objetos, secuencias de comandos de procedimientos, etc. El uso de un determinado UIDL está dado por el propósito y el dominio de aplicación del software, y por las preferencias individuales del desarrollador de IU. Los entornos webs, por lo general imponen el uso de lenguajes declarativos. Estos suponen la existencia de un algoritmo automático integrado en cada navegador web, que construye la interfaz de usuario desde un lenguaje de especificación particular. Los entornos de

escritorio utilizan principalmente los lenguajes imperativos para la construcción de la interfaz de usuario.

La última tendencia en el desarrollo de UIDL es una solución híbrida, donde elementos de presentación de la interfaz se escriben utilizando construcciones declarativas; mientras que el comportamiento es definido en bloques imperativos. De esta manera, es posible lograr una visión consistente y un mismo comportamiento en diferentes plataformas.

Varios UIDL se han desarrollado en las últimas dos décadas. La mayoría de ellos están basados en XML, pues como se señala en [4], los principales motivos de aparición de estos lenguajes son:

- La captura de la exigencia de la IU como una definición abstracta que permanece estable a través de diversas plataformas. La estabilidad se refiere a la semántica de interacción.
- Diseño de una única IU para múltiples dispositivos y plataformas.
- Mejora de la reutilización de la interfaz de usuario.
- Apoyo a la evolución, extensibilidad y capacidad de adaptación de la interfaz de usuario.
- Generación automática de un código de interfaz de usuario.

Los lenguajes basados en XML se están perfilando como serios candidatos a soportar las especificaciones gracias a la versatilidad de mantenimiento, extensión y capacidades de refinamiento que proporcionan [5].

La tecnología XML, como estándar de representación común, permite la especificación del modelo de interfaz abstracto, la descripción de las características específicas de los diferentes dispositivos, así como la especificación del proceso de transformación de los objetos de interacción abstractos en objetos de interacción concretos [6].

II.I.I AAIML (Alternate Abstract Interface Markup Language)

AAIML, es un lenguaje basado en XML que se enmarca dentro de un proyecto más amplio, para el desarrollo de un protocolo estándar que facilite el acceso a interfaces alternativas. El objetivo de este proyecto es ampliar el mercado, haciendo más accesibles las interfaces y los dispositivos, por ejemplo, a personas que sufren alguna discapacidad.

El hecho de que los dispositivos electrónicos estén fabricados por diferentes compañías, requiere la definición de un estándar que permita utilizar una consola "alternativa" para controlar los dispositivos específicos. La especificación del concepto de Consola Remota Universal (URC) es esencial en cuanto al desarrollo de la definición de dicho estándar. El concepto de URC permite a las personas con, o sin discapacidades, controlar remotamente cualquier dispositivo electrónico desde su dispositivo personal de control remoto, que puede encontrarse situado en cualquier lugar [7]. El lenguaje AAIML debe especificar la definición de una interfaz de usuario abstracta para un determinado servicio o dispositivo. Dicha interfaz sería transmitida desde el dispositivo a la URC [7].

II.I.II AUIML (Abstract User Interface Markup Language)

El lenguaje AUIML es un lenguaje basado en XML desarrollado por IBM y diseñado para permitir la definición de la semántica de la interacción con el usuario. Está centrado, por tanto, en la descripción de aspectos de interacción más que en aspectos de presentación. Toda la información de la interacción se codifica una sola vez y se traduce dependiendo del dispositivo final. Está diseñado para ser independiente de la plataforma, del lenguaje de programación final y de la tecnología de implementación [8].

Consta de dos principales conjuntos de elementos, los que se representan a través del modelo de datos, que definen la estructura de la información necesaria para soportar la interacción

con el usuario, y los que se representan a través del modelo de presentación, que a su vez definen el estilo de la presentación. AUIML estaría englobado dentro de los niveles 2 y 3 del proceso de modelado propuesto por Cameleon.

II.I.III UIML (User Interface Markup Language)

El lenguaje UIML es un sencillo lenguaje basado en XML que permite realizar una descripción declarativa de la interfaz de usuario de un modo independiente del dispositivo. Uno de los objetivos de UIML es reducir el tiempo que los desarrolladores invierten en describir interfaces para múltiples familias de dispositivos [9].

Para describir una IU en UIML se debe realizar, por un lado la definición de la interfaz genérica, y por otro un documento UIML que representa el estilo de presentación apropiado para el dispositivo en el cuál la IU se va a ejecutar. [10].

Se puede decir que UIML se podría utilizar parcialmente en el nivel 1 y completamente en los niveles 2 y 3 del esquema de modelado presentado. La definición de la interfaz se enmarcaría en los niveles 1 y 2 y la definición del estilo en el nivel 3. Estas tareas incluso podrían ser llevadas a cabo por equipos de desarrollo diferentes. La flexibilidad para la selección de dispositivos finales es limitada, ya que aunque la parte correspondiente a la interfaz genérica puede mantenerse independientemente de que aumente el número de dispositivos finales, no sucede lo mismo con la parte que mapea a los dispositivos específicos, que crece cuando este número aumenta, aumentando también el coste de mantenimiento [11]. Este lenguaje permite la traducción automática al lenguaje utilizado por el dispositivo final. El proceso de traducción se realiza en el propio dispositivo, o bien en el servidor de la interfaz dependiendo del dispositivo del que se trate [10].

II.I.IV XIML (eXtensible Interface Markup Language)

El lenguaje XIML es un lenguaje de especificación basado en XML. Se

propone como lenguaje de especificación común e infraestructura de desarrollo para profesionales de la interfaz de usuario.

XIML se presenta como una propuesta de representación común para datos interactivos, ya que contempla los principales requisitos que debe cumplir un lenguaje de este tipo:

a. Soporta funcionalidad a lo largo del ciclo de vida completo del desarrollo de interfaces de usuario.
b. Es capaz de relacionar los elementos abstractos y concretos de una interfaz.
c. Permite a los sistemas basados en conocimiento tratar los datos capturados [12].

En su primera versión los componentes son tareas, dominio, usuario, diálogo y presentación, extraídos del estudio de los modelos declarativos del enfoque basado en modelos. Los tres primeros componentes se podrían considerar abstractos y los dos últimos específicos [12]. Por tanto, se puede decir que XIML podría ser utilizado en los niveles 1, 2 y 3 del proceso de modelado de interfaces de usuario presentado. La independencia de la plataforma de uso se hace posible gracias a la estricta separación entre la definición de la interfaz y la traducción de la misma. Existe un Foro de XIML en el cuál se incluyen representantes del mundo académico y de la industria, cuyo objetivo es promover la investigación, divulgación, adopción y estandarización de XIML. Hasta ahora, se han presentado resultados de la evaluación de sus requisitos y en este momento se encuentra en fase de expansión y divulgación.

II.I.V XUL (XML-based User-Interface Language)

XUL es un lenguaje de descripción de interfaces basado en XML, específicamente diseñado para aplicaciones en red como navegadores, programas de correo. Está integrado dentro de la arquitectura del navegador web Mozilla Firefox para el desarrollo de interfaces web multiplataforma, dentro de la cual se hace uso de tecnologías

W3C ya existentes. La arquitectura se basa en el uso de paquetes que pueden ser abordados desde una perspectiva abstracta o concreta. Los paquetes se componen de contenido, apariencia, comportamiento, localización, plataforma. En cada uno de ellos se hace uso de diferentes tecnologías [13].

Su ejecución deberá realizarse bajo el entorno de Mozilla y en los sistemas operativos en los cuáles Mozilla se ejecute. Tiene la capacidad de separar la interfaz de la lógica de la aplicación, lo cual facilita el mantenimiento de la interfaz sin necesidad de alterar la lógica de la aplicación. Se podría utilizar XUL para realizar la definición de la especificación de la interfaz concreta, nivel 3. Contempla parcialmente el nivel 2 a través del resto de las tecnologías de las que hace uso.

II.I.VI XForms

XForms es una propuesta del consorcio W3C para la especificación de formularios para la Web que puedan ser usados en una amplia variedad de plataformas. Su versión 1.0 ha llegado a ser una recomendación de W3C. El desarrollo de XForms pretende cubrir las limitaciones de los formularios HTML tradicionales que no disponen de una separación entre el propósito y la presentación de un formulario. XForms se compone de secciones separadas que describen lo que el formulario hace y cómo se presenta [14].

La especificación de XForms está compuesta de dos módulos, el Modelo XForms y el soporte de la interfaz de usuario. El modelo XForms representa las diferentes partes del formulario desde el punto de vista de los datos genéricos. La IU XForms define la parte del formulario que representa los elementos de la presentación, proporcionando flexibilidad y control sobre la misma a través de las diferentes presentaciones que pueden estar relacionadas con un mismo modelo XForms, dependiendo de la plataforma final.

XForms se podría englobar fundamentalmente en el nivel 3 de abstracción (parte del nivel 2), ya que es un lenguaje orientado a la implementación de formularios Web en diferentes dispositivos, pero no a la descripción de la interfaz en un nivel alto de abstracción.

II.II Lenguajes de Transformación de IU (UITL)

La Ingeniería dirigida por modelos asume que varios modelos describen diferentes aspectos de la IU. Las relaciones entre estos modelos se establecen a través de transformaciones. De esta manera, el desarrollo de IU puede ser visto como la cadena de transformación que comienza con modelos de alto nivel de abstracción y termina con versiones ejecutables de IU para una tecnología concreta. Una extensa taxonomía de los enfoques de transformación de modelo se ha propuesto en [15].

La variabilidad de la semántica entre diferentes modelos, sus formatos y herramientas causó diversas aproximaciones de transformación en el contexto del MDUID. Algunos de los UITL operan directamente sobre modelos, mientras que otros trabajan con sus formatos derivados. Unos están integrados en los modelos y otros se aplican externamente a los mismos. Es válido mencionar además que algunos UITL son editables y modificables, mientras que otros están integrados en herramientas y no se pueden modificar.

A continuación, se analizarán los UITL más populares, resaltando sus características más importantes.

II.II.I QVT (Query/View/Transformation)

QVT es un estándar para la transformación de modelos definido por el OMG, basado en MOF (por las siglas en inglés de Meta Object Facility) para lenguajes de transformación en MDA [15].

Este estándar es un híbrido de naturaleza declarativa e imperativa, donde la parte declarativa se divide en una arquitectura de dos NIVELES (FIGURA 2) constituida por un metamodelo y un lenguaje llamado Relations, el cual soporta

concordancia de patrones de objetos complejos y plantillas de creación de objetos; y un metamodelo y un lenguaje llamado Core, definido usando mínimas extensiones de EMOF (Essential MOF) y OCL (por las siglas en inglés de Object Constraint Language).

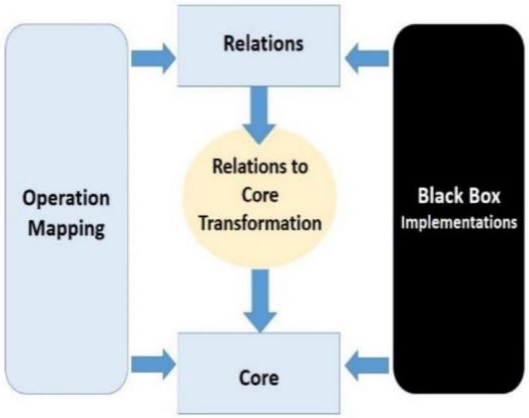

Figura 2: Arquitectura de QVT
Fuente: Elaboración Propia

Existen dos mecanismos para invocar implementaciones imperativas de transformaciones desde Relations y Core: un lenguaje estándar llamado Operational Mappings y una implementación no estándar llamada Black- box MOF Operation.

Operational Mappings constituye una manera estándar de proveer una implementación imperativa, la cual predomina en los mismos modelos y registros que en Relations. Proporciona extensiones de OCL que permiten utilizar un estilo más procedimental y una sintaxis concreta que resulta más familiar a los programadores de los lenguajes imperativos más habituales (C++, Java, etc.).

Black-box MOF Operation aprovecha que Relations hace posible ensamblar cualquier implementación a una operación MOF si se mantiene la signatura de dicha operación. Esta característica resulta muy conveniente, pues permite la codificación de algoritmos complejos en cualquier lenguaje de programación que tenga un puente a MOF (o bien que pueda ser ejecutado dentro de un lenguaje que lo tenga). Sin embargo, esta implementación no está carente de riesgos, en cuanto tiene acceso a referencia de objetos en los modelos, con ello podría realizar acciones con estos objetos las cuáles potencialmente podrían romper la encapsulación de los mismos.

II.II.II ATL (Atlas Transformation Language)

ATL es un lenguaje de transformación de modelos híbrido que permite, en su definición de transformaciones, especificar construcciones declarativas e imperativas. La propuesta es del ATLAS Group del INRIA & LINA, de la Universidad de Nantes y fue desarrollada como parte de la plataforma AMMA (ATLAS Model Management Architecture) [16].

ATL es compatible con los estándares de la OMG: es posible describir transformaciones modelo a modelo, modelos que deben conformar meta-modelos definidos con MOF. Además, el lenguaje se basa en QVT (en [17] se discute el nivel de compatibilidad de ATL y QVT) y permite definir pre y postcondiciones en OCL, resultando sumamente expresivo y de fácil escritura para sus usuarios.

En cuanto a las reglas de transformación, ATL define, tal y como se ha sugerido, un dominio (meta- modelo) para el origen (source) y otro dominio para el destino (target), siendo ambas instancias de MOF y con direcciones de entrada (in) y salida (out) respectivamente. Los meta-modelos source y target pueden tener iguales o diferentes dominios, pero, aun siendo iguales, ambos deben ser claramente identificados. Si bien las transformaciones son unidireccionales, ATL permite la definición de transformaciones bidireccionales mediante la implementación de dos transformaciones, una para cada dirección.

Como características particulares del lenguaje, se puede mencionar las estructuras que define este lenguaje. En primer lugar, las definiciones de transformaciones forman módulos (modules) que contienen las declaraciones iniciales y un número de helpers y reglas de transformación. Los helpers son una estructura intermedia dentro de las transformaciones que permiten definir operaciones y tuplas OCL, componentes que facilitan la navegación, la modularización y la reutilización.

La aplicación de las reglas se realiza de forma no determinista, por equiparación de patrones, no habiéndose provisto ninguna construcción o cláusula que permita aplicar en forma condicional las reglas. Además, provee mecanismos de reutilización, ya que las reglas se pueden heredar y sus módulos pueden incluir a otros módulos. En cuanto a la trazabilidad, este lenguaje crea un enlace de trazabilidad (traceability link) con la ejecución de cada regla, que se guarda en el motor de la transformación. Este enlace relaciona tres elementos: la regla, los elementos originales (source) y los elementos creados (target).

II.II.III MT Model Transformation Language

MT es un lenguaje de transformaciones de modelos implementado como un DSL embebido [18] en el lenguaje de programación Converge. Fue desarrollado por Laurence Tratt del King's College London. Este lenguaje en parte es una derivación del enfoque QVT- Partners [19]. Este enfoque se basa principalmente en el uso de patrones (en concreto, patrones equivalentes a las expresiones regulares textuales, pero definidos sobre modelos). Sin embargo, dichos patrones son pobres en expresividad, puede sólo equipararse contra un número fijo de elementos, y contienen defectos en los alcances de las reglas. Así mismo, QVT-Partners utiliza un lenguaje imperativo complejo para el cuerpo de las reglas [20].

MT se integra de manera natural con Converge, usando patrones declarativos para realizar las correspondencias entre los elementos del modelo de una manera concisa pero poderosa, y permitiendo a la vez embeber dentro de las reglas, código normal imperativo escrito en Converge. El hecho de que MT sea un DSL embebido, ha permitido que su diseño se haya mejorado rápidamente, haciendo posible una rápida experimentación y retroalimentación, de tal manera que se ha logrado un mayor grado de comprensión de las transformaciones de modelos y a su vez se ha conseguido un lenguaje de patrones

más sofisticado. En cuanto a la modularización del lenguaje, todas las reglas se encuentran en un solo espacio de nombres, no existiendo la notación para el manejo de módulos. Sin embargo, en futuras versiones el autor tiene previsto implementar esta característica.

En cuanto a la trazabilidad, MT mantiene registros que se crean automáticamente conforme se ejecutan las reglas de transformación. Sin embargo, por omisión, sólo se registran aquellos elementos equiparados por patrones de elementos no anidados. Esto significa que los elementos fuente que son almacenados en la información de traza no constituyen necesariamente el universo entero de elementos pasados como parámetros a las transformaciones.

II.II.IV MOFScript

MOFScript es un lenguaje de transformación de modelo a texto presentado actualmente como candidato en el proceso OMG RFP [21]. Ha sido desarrollado por la comunidad de desarrollo SINTEF, soportado y probado por el proyecto europeo de integración MODELWARE. Este lenguaje presta particular atención a la manipulación de texto y al control e impresión de salida de archivos. Es especialmente útil para realizar implementaciones dependientes de plataforma, como:

a) la creación de código fuente en lenguajes como Java o C# a partir de modelos UML.
b) la generación automática del código de creación de una base de datos mediante SQL a partir de un Modelo de base de datos relacional.
c) la creación de documentación o transformación de modelos a lenguajes textuales como XML o HTML.

MOFScript se basa en otros estándares de la OMG: es compatible con QVT y MOF para el modelo de entrada (el modelo objetivo –target- siempre es texto). Para las reglas de transformación se define un metamodelo de entrada (source) sobre el cual operarán las reglas.

Las transformaciones son siempre unidireccionales, y no es posible definir pre y post condiciones para ellas. La separación sintáctica resulta clara por la misma definición de reglas, lo que hace al lenguaje legible. No provee estructuras intermedias, pero sí la parametrización necesaria para la invocación de reglas. En cuanto a la aplicación de reglas, éstas se aplican en forma determinista y en orden secuencial. Se provee aplicación condicional e iteración de reglas.

MOFScript no organiza las reglas en módulos propiamente dichos. Ahora bien, en la definición de una regla se pueden invocar a otras reglas, utilizando incluso parámetros, con lo cual las reglas en sí pueden entenderse como mecanismos básicos de modularización. También es posible definir jerarquías de transformaciones. Finalmente, para trazabilidad, MOFScript define un conjunto de conceptos para relacionar elementos fuente con sus ubicaciones en los archivos de texto generados en el target. En cuanto a la composición de transformaciones (combinación de reglas para obtener una nueva regla), no se contempla expresamente.

II.II.V UMLX

UMLX es un lenguaje gráfico de transformaciones entre modelos (M2M) y que se basó en extensiones mínimas a UML. Es una propuesta de E. Willink, del GMT Consortium. Su implementación se ha realizado sobre Eclipse [22].

La definición de este lenguaje se basó en QVT. Asimismo, los modelos participantes en las transformaciones deben ser instancias de MOF. Cada uno de estos juega un rol diferente (uno de entrada in y otro de salida out), aunque se trate de los mismos modelos. En cuanto a las reglas de transformación, UMLX utiliza diferentes íconos gráficos para las transformaciones, reglas y relaciones de creación, preservación y eliminación de elementos. Cabe destacar que la semántica de los íconos del lenguaje no se ha definido formalmente, y aunque es gráficamente intuitivo, para algunas relaciones, no queda claro cuál es su significado.

III. RESULTADOS Y DISCUSIÓN

El principal resultado de este trabajo es el análisis de lenguajes para el desarrollo formalizado de IU siguiendo el MDUID. Para una mejor comprensión, será presentado en forma de comparación.

3.1 Comparación de Lenguajes UIDL

En la Tabla 1 se muestra una comparación de aspectos relativos a los lenguajes declarativos analizados en cuanto a:

a) Niveles de abstracción del proceso de modelado en los que se puede situar cada lenguaje, (los niveles que aparecen entre paréntesis son cumplidos parcialmente).
b) Si es capaz o no de abarcar dispositivos diversos.
c) Cuál es su ámbito de aplicación.
d) Si el tipo de interfaz generada es exclusivamente una interfaz gráfica de usuario (GUI) o puede contemplar otros tipos de interfaces (IU).
e) Qué versiones de cada lenguaje se encuentran desarrolladas o especificadas.
f) Si el lenguaje se encuentra o no en fase de mejora y expansión. Con respecto a los modelos y niveles de abstracción del proceso de modelado de la IU, cabe destacar que el único lenguaje de lo revisados que contempla el ciclo de desarrollo completo es XIML, cubriendo todos los niveles de abstracción. Posiblemente, esto se derive de su evolución a partir de la investigación en el enfoque basado en modelos. El resto de los lenguajes presentados no cubre completamente el ciclo de desarrollo de un enfoque basado en modelos.

AUIML, XUL y XForms contemplan los niveles 2 y 3 exclusivamente del Modelo de IU. Su ámbito de aplicación es más limitado. En el caso de UIML el objetivo es el desarrollo rápido de interfaces facilitando el prototipado de las mismas. Se centra en la definición de la interfaz abstracta en lo relativo a los aspectos independientes del

dispositivo, y en su mapeado a la interfaz de usuario específica para cada dispositivo. Por tanto, su ámbito es más concreto y también más limitado en principio que el de XIML, pero al igual que éste se presenta como un formato de intercambio universal que pretende ser un estándar y que orienta su desarrollo futuro hacia un lenguaje abstracto para la descripción de interfaces en un nivel alto de abstracción.

Algunas de estas propuestas pueden resultar complementarias, como es el caso de la generación de formularios de XForms a partir de una especificación UIML.

Lenguaje	Niveles Modelo de IU	Diferentes Dispositivos	Ámbito de Aplicación	Tipo de IU	Estado de Desarrollo
AAIML		Si	Global	IU	Proyecto
UIML	(1),2,3	Si	Disp. de red	IU	1.0, 2.0, 3.0 En desarrollo
Xforms	(2), 3	Si	Web	GUI	1.0 En desarrollo
AUIML	(2), 3	(Si)	Global	GUI	En desarrollo
XUL	(2), 3	No	Aplic. de red	GUI	1.0 En desarrollo
XIML	1, 2, 3	Si	Global	IU	1.0 En desarrollo

Tabla 1: Resumen Comparación de los UIDL

III.II Comparación de Lenguajes UITL

En la Tabla 2 se puede encontrar una comparación de las principales características de los lenguajes de transformación analizados.

Como se ha podido apreciar en el epígrafe anterior, existen varios lenguajes de transformación, donde cada uno posee características deseables al igual que inconvenientes: los lenguajes basados en QVT como ATL tienen la ventaja de ceñirse a un estándar, lo cual los hace más compatibles con otras tecnologías y comprensibles, dada la formalidad de su especificación, pero pueden carecer de expresividad y verse limitados por su naturaleza cerrada. En cuanto a los lenguajes implementados como DSLs embebidos, tal es el caso de MT, ofrecen la ventaja de aprovechar todo el potencial del lenguaje host. No obstante, esta característica también los hace poco compatible con otras herramientas que se basen en los estándares del OMG (QVT, OCL, MOF, etc.). Por otra parte, los lenguajes gráficos como UMLX son sumamente intuitivos, pero a la vez limitados por la misma conformación de la simbología del lenguaje.

Características	ATL	MOFScript	MT	UMLX
Tipo	Textual/M2M	Textual/M2M	Textual/M2M	Gráfico/M2M
Estilo	Declarativo y operacional	Declarativo y operacional	Declarativo y operacional	Declarativo y operacional
Pre y Postcondición	Si (OCL)	No	Si (Converge)	No
Compatible con MOF/QVT	Si	Si	Parcialmente	Si
Modularización	Si	No	No	Si (Capas)
Trazabilidad	Si	Si	Si	No
Reusabilidad de reglas	Si	Si	Si	Si
Soporte en herramientas	Si (IDE Eclipse)	Si (IDE Eclipse)	Si	Si (IDE Eclipse)

Tabla 2: Resumen Comparación de los UITL

IV. CONCLUSIONES

En el trabajo se realizó un análisis de lenguajes para el MDUID, haciendo énfasis en los UIDL declarativos y en los UITL. En cuanto a los UIDL, se aprecia una tendencia a la búsqueda de la estandarización, de un lenguaje de representación común para los datos interactivos, y al mismo tiempo una evolución y continuo desarrollo de propuestas tanto desde el ámbito académico como de la industria.

Se pudo constatar que las metodologías y entornos de desarrollo de IU basados en modelos, tienden a contemplar el uso de alguno de los lenguajes analizados en las fases del proceso de modelado. Es importante resaltar, que de todos los lenguajes analizados el único que contempla el ciclo de desarrollo completo es XIML, cubriendo todos los niveles de abstracción.

Por otra parte, teniendo en cuenta que la definición de reglas de transformación entre modelos constituye una tarea fundamental a realizar dentro del paradigma de desarrollo en cuestión, se analizaron los UITL que más se utilizan en propuestas existentes en la literatura, constatándose que muchas de estas propuestas siguen aún en evolución. Como aspecto interesante y que resaltan su valía se puede mencionar el uso de estándares como el del OMG, que los hace compatibles con diferentes tecnologías.

Como principal resultado del trabajo se tiene una comparación de cada uno de los lenguajes para ayudar a los diseñadores de IU a elegir la tecnología más adecuada en el contexto específico del desarrollo.

En el caso de los autores, posibilitó seleccionar el entorno tecnológico que dará soporte a la implementación de una guía

metodológica definida para el desarrollo de IU siguiendo el MDUID.

V. REFERENCIAS

1. OMG. Model Driven Architecture. Document number ormsc/2001-07-01. s.l.: Miller, J., Mukerji, J., 2001.
2. da Silva, P. P.: 2000, User Interface Declarative Models and Development Environments: A Survey, in DSV-IS, pp. 207- 226
3. 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, 289-308
4. Mõttus, M., & Lamas, D. Aesthetics of Interaction Design: A Literature Review. In Proceedings of the Mulitimedia, Interaction, Design and Innnovation (p. 1). ACM, 2015.
5. Pineda Alvarez, D. F. Comparativa entre herramientas MDD enfocada en la versatilidad del lenguaje con respecto a la implementación de requerimientos. Caso práctico Integranova WebRatio, 2017.
6. Boucher, Q., Perrouin, G., Davril, J. M., & Heymans, P. Engineering configuration graphical user interfaces from variability models. In Human Centered Software Product Lines (pp. 1-46). Springer, Cham, 2017.
7. Ballesteros, F. J., Guardiola, G., & Soriano- Salvador, E. O/live: Transparent Distribution, Persistence, and Partial Replication for Ubiquitous User Interfaces. International Journal of Human-Computer Interaction, 30(10), 755-770, 2014.
8. Johnsson, B. A. Inverted GUI Development for IoT with Applications in E-Health (Doctoral dissertation, Lund University, Sweden), 2017.
9. Wessels, P. D. Leveraging behavioural domain models in Model-Driven User Interface Development with GLUI (Master's thesis, University of Twente), 2018.
10. White Paper:The UIML Vision. Harmonia Inc. Blacksburg, Virginia USA. February, 2000

11. Rehman, S., Ullah, R. M. K., Tanvir, S., & Azam, F. Development of User Interface for Multi-platform Applications Using the Model Driven Software Engineering T echniques. In 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) (pp. 1152-1158). IEEE, 2018.
12. Boucher, Q., Perrouin, G., Davril, J. M., & Heymans,P. Engineering configuration|graphical user interfaces from variability models. In Human Centered Software Product Lines (pp. 1-46). Springer, Cham, 2017.
13. Lozano, M.D. Entorno Metodológico Orientado a Objetos para la Especificación y Desarrollo de Interfaces de Usuario. Tesis doctoral. Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia. Octubre, 2001.
14. Villegas, M. L., Collazos, C. A., Giraldo, W. J., & González, J. M. Evaluación de elementos de modelado en el desarrollo de sistemas interactivos. In 2016 IEEE 11th Colombian Computing Conference (CCC) (pp. 1-7). IEEE, 2016.
15. Object Management Group, Inc. (OMG). (2007, Diciembre) MOF 2.0/XMI Mapping, Version 2.1.1. [Online]. http://www.omg.org/cgi-bin/doc?formal/2007- 12-02
16. ATL Project. [Online]. http://www.eclipse.org/m2m/atl/
17. Frédéric Jouault and Ivan Kurtev. On the Architectural Alignment of ATL and QVT. [Online]. http://www.sciences.univ-nantes.fr/lina/ atl/www/papers/ATLandQVT-PRELIMINARY%20VERSION.pdf
18. Tata Consultancy Services. (2003, Marzo) Initial submission for MOF 2.0 Query / Views / Transformations RFP: QVT Partners. [Online]. http://www.tratt.net/laurie/research /publications/papers/qvtpartners1.0.pdf
19. Fowler and Martin. MF Bliki: Domain Specific Language. [Online].

http://www.martinfowler.com/bliki/DomainSpe cificLanguage.html

20. Laurence Tratt, “The QVT-Partners model transformations approach - Issues with the approach,” in The MT model transformation language. Department of Computer Science, King’s College London, 2005, ch. 3.4, pp. 10- 11.
21. MOFScript Home page. [Online]. http://www.eclipse.org/gmt/mofscript/
22. Edward D. Willink. UMLX: A Graphical Transformation Language. [Online]. http://dev.eclipse.org/viewcvs/indextech.cgi/g mt- home/subprojects/UMLX/doc/MDAFA2003-4/MDAFA2003-4.pdf