**Capitulo 4 – La Transformación de modelo a texto (M2T) para el diagrama de contenido de moweba**

**4.1 Introducción**

En capítulo anterior se ha visto el metamodelo de contenido (*Content*) para la representación de la sintaxis abstracta de MoWebA con la nueva clasificación de elementos de interfaz, en la cual aparecen a diferencia de MoWebA tradicional que contempla elementos para la web 1.0, elementos enriquecidos que son comunes en las aplicaciones RIAS. También junto al metamodelo de contenido, se presentó el metamodelo de estructura (*Layout*), que permite definir la posición de los diferentes elementos del metamodelo de contenido, dentro de una página en particular. Al finalizar la presentación de los metamodelos, se dio pié a los perfiles para el modelado de los *PIM* de la aplicación, extendiendo a UML con las características nuevas propias del DSL MoWebA, específicamente a lo concerniente a contenido y posición.

En este capítulo, en primera instancia se presentarán algunos conceptos básicos acerca de la generación de código y los beneficios que aportan al proceso de desarrollo basado en modelos. El proceso M2T puede tener varios objetivos finales, como la generación, a partir de los modelos de documentación, código fuente, o cualquier componente de software necesario en una aplicación. Aquí nos concentraremos en la generación de código fuente para la interfaz de usuario de una aplicación web representada por medio de modelos que han sido definidos en MoWebA. Una revisión de algunas de principales las herramientas M2T existentes en la actualidad se dará a conocer, para luego presentar a la herramienta M2T Acceleo que es la herramienta utilizada para la generación de código. Estas secciones están basadas en libro de Brambilla y otros[[].

Seguidamente se presentará a las plantillas (*templates*) para llevar a cabo la transformación de los PIM de entrada de MoWebA correspondiente al contenido y a las plantillas de transformación para establecer la configuración de las posiciones de cada uno de los elementos del PIM.

Finalmente se concluirá el capítulo con un ejemplo de transformación M2T para PIM modelados con MoWebA.

**4.2 Conceptos básicos de la generación de código a partir de los modelos**

Uno de los aportes de MDSE (*Model Driven Software Engineering*), es obtener sistemas a partir de los modelos. Las plataformas de ejecución actuales son a menudo basadas en código, con pocas excepciones que permiten una interpretación directa de los modelos. De esta forma, las transformaciones M2T en el área MDSE son a menudo relacionadas con la generación de código para alcanzar la transición a partir del nivel del modelo al nivel de código.

Mientras que en el contexto de los compiladores, la generación de código, es el proceso de transformar el código fuente en código máquina, en el mundo MDE, la generación de código es el proceso de transformar modelos en código fuente.

Dentro de las preguntas esenciales cuando uno tiene que desarrollar un generador de código basado en modelos se encuentran las siguientes:

**¿Qué tanto código va a generarse?**

La pregunta principal aquí es que partes del código puede ser automáticamente generada a partir de los modelos. ¿Es posible llevar a cabo una generación de código parcial o total?. La generación parcial de código puede implicar muchas cosas en este contexto. Primero, puede implicar que una capa (horizontal o vertical) de la aplicación sea completamente generada, mientras que otra capa podría ser desarrollada completamente de manera manual. Más aún, una capa puede ser generada parcialmente y otras partes no cubiertas tienen que ser completadas con código manual. La generación parcial de código, también puede referirse al nivel de modelado, utilizando solamente la generación de código para ciertas partes del modelo, mientras que otras partes no son manipuladas por el generador de código y tiene que ser implementadas manualmente.

En el enfoque tomado en este trabajo, la generación de código es total a partir de los modelos de los PIM de entrada, para los elementos de la capa de presentación de MoWebA.

**¿Qué código va a generarse?**

Implica que clase de código fuente va a generarse. Por supuesto, el código a ser generado debe ser lo más conciso posible y debe ser código que puede ser entendido por los desarrolladores. La idea es, generar la menor cantidad de código, capaz de representar un sistema de la mejor manera.

En el contexto de este trabajo de fin de carrera, a generarse a partir de los modelos será HTML, Javascript para la representación de los distintos elementos de interfaz de MoWebA como así también CSS para el posicionamiento de estos elementos en las páginas.

**¿Como a va generarse?**

Muchos lenguajes pueden ser empleados para generar código a partir de los modelos y pueden ser GPLs y DSL. Para este trabajo de tesis, la herramienta M2T basado en plantillas Acceleo, será la protagonista de llevar a cabo la transformación M2T de los modelos de entrada.

**4.3 Beneficios de los lenguajes de transformación (M2T)**

**Separación de código estático y dinámico**:

Los lenguajes de transformación M2T separan el código estático y dinámico, utilizando el enfoque de plantillas (*templates)* para implementar las transformaciones M2T. Una plantilla puede ser vista como un proyecto, que define elementos de texto estáticos compartidos por todos los artefactos, como así también, partes dinámicas que deben ser completadas con información específica para cada caso en particular. Por lo tanto, un *template* contiene fragmentos de texto simple para las partes estáticas y los llamados metamarcadores (*meta-markers)* para las partes dinámicas. Los metamarcadores son marcadores de posición y deben ser interpretadas por un motor de *templates* que procesa los *templates* y consulta fuentes de datos adicionales para producir las partes dinámicas. Las fuentes adicionales de datos son los modelos. En la figura 1, se presenta el esquema tradicional de transformación basado en *templates.*

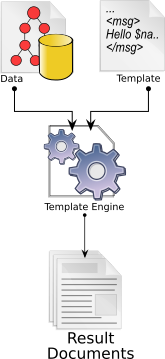


Figura 1 Plantilla, motor de plantillas y modelos de entrada para producir texto

Los *templates* permiten representar explícitamente la estructura del texto de salida dentro del *template*. Esto permite una especificación de la generación de código más entendible y leíble.

Dentro de los metamarcadores, el código es utilizado para acceder a la información almacenada dentro de los modelos.  El estándar OCL es la elección para llevar a cabo esta tarea en la mayoría de los lenguajes de transformación basados en *templates*. De esta forma, los lenguajes de transformación M2T también permiten el uso de OCL (o dialecto de OCL) para especificar a los metamarcadores.

Los lenguajes de transformación M2T actuales, vienen con soporte de herramienta, lo cual permite leer directamente los modelos y serializar texto en archivos, definiendo solamente archivos de configuración.

**4.4 UNA VISTA DE LOS LENGUAJES DE TRANSFORMACIÓN BASADOS EN TEMPLATES**

Diferentes lenguajes basados en *templates* existen en la actualidad,  los cuales pueden ser empleados para generar texto a partir de los modelos.

**XSLT[[1]](#footnote-1)**

La serialización XMI de los módulos, pueden ser procesados con XSLT, que es el estándar W3C para transformar documentos XML en documentos arbitrarios de texto. Sin embargo, en este caso, los  scripts de generación de código tienen que ser implementados basados en la serializaición XMI, que requiere ciertos conocimientos adicionales de como los modelos son actualmente codificados como archivos XML. Así, el enfoque opera directamente a nivel de modelo.

**JET Java Emitter Template[[2]](#footnote-2)**

Fue uno de los primeros enfoques para de desarrollo del EMF para la generación de código a partir de modelos. Pero JET no está limitada a modelos basados en EMF. En general, con JET, todo objeto basado en Java es transformable a texto. JET provee una sintaxis similar a JSP adaptada a la estructura *template* para transformación M2T. Expresiones Java arbitrarias pueden ser introducidas en los *templates*JET. Los *template* de JET son transformados a código Java puro para propósitos de ejecución. Sin embargo, no tiene un lenguaje de consulta dedicado para los modelos disponibles en JET.

**Xpand[[3]](#footnote-3)**

Este lenguaje de transformación provee un lenguaje dedicado para consultar modelos siendo este una combinación de Java y OCL (muchos iteradores basados en OCL están disponibles). La continuación a este proyecto se llama Xtend, que está basado en Java,pero ofrece muchas características adicionales propias del lenguaje. Por ejemplo, es posible incrustar *templates* de generación de código (para tener una sintaxis similar  al template Xpand) dentro del código Xtend.

**MOFScript[[4]](#footnote-4)**

Este proyecto provee otro lenguaje de transformación M2T proveyendo características similares tales como *Xpand*. *MOFScript* ha sido desarrollado como una propuesta de estandarización para la OMG y se encuentra disponible como un *plug-in*para el *Eclipse* y soporta modelos del tipo EMF.

**Acceleo[[5]](#footnote-5)**

*Acceleo* es una herramienta de transformación M2T basada en los estándares propuestos por la OMG y que actualmente forma parte de la *Eclipse Foundation*.  *Acceleo* es el resultado de varios años de investigación y desarrollo en el área de los lenguajes de transformación de modelos (MTL). Permite la des-serialización de modelos basados en UML del EMF como así también modelos basados en el metamodelo Ecore. Acceleo posee una herramienta de desarrollo bastante madura como así también una comunidad activa que la sostiene. Muchos proyectos en la industria han probado su eficacia en varios contextos.

**4.5  LA HERRAMIENTA DE TRANSFORMACIÓN M2T ACCELEO**

*Acceleo* posee varias características que la hacen interesante para la generación de código a partir de los modelos de entrada. *Acceleo* es un generador de códigos de uso abierto (*open source*). Como tal es posible utilizarlo, bifurcarlo y contribuir con la evolución del proyecto. Cuenta con una gran comunidad (*Eclipse Foundation*) que la mantiene.Está integrado con el IDE del *Eclipse*, un editor robusto, con corrector de sintaxis, detección de errores en tiempo real, soluciones rápidas, refactorización y mucho más. También contiene vistas dedicadas que ayudan a navegar amigablemente por el generador de código**.**

Por lo general, con el generador de código, es fácil perderse en el código generado. De manera a manejar este inconveniente, Acceleo contiene un motor de trazabilidad que permite encontrar fácilmente, que elementos del modelo y que parte del generador (plantilla de transformación) han sido utilizados para generar la pieza de código. Generadores de código son a menudo limitados a un conjunto de tecnologías. Con el enfoque basado en *templates*,  *Acceleo* puede generar código para cualquier tipo de lenguaje. Si es posible escribir la plantilla de transformación, *Acceleo* puede generar el código correspondiente.

En algún momento podría considerarse adecuado modificar  manualmente el código generado por el *template* de transformación  y mantener las modificaciones manuales realizadas, en caso que se desea regenerar el código de la aplicación. Acceleo provee de tal flexibilidad, permitiendo llevar a cabo generaciones incrementales.

**4.6 METAMARCADORES DE ACCELEO**

Para obtener los datos de los modelos UML de entrada, Acceleo hace uso de los metamarcadores (*tags*). Dentro de los metamarcadores más importantes pueden citarse los siguientes a continuación:

**Archivos:**Para generar código, los archivos deben ser abiertos, rellenados y posteriormente cerrados. En Acceleo, existe un *tag* de archivo (*file*) dedicado, el cual es utilizado para imprimir contenido creado entre el comienzo y el final del *tag file*. La ruta y el nombre de archivo son definidos por un atributo del *tag*.

**Estructura de control**: Existen *tags* para definir estructuras de control tales como *loops* (*for*) para iterar entre colecciones de elementos y en ramas condicionales (if tag).

**Consultas:** Las consultas OCL pueden ser definidas por medio del *tag query*. Los queries pueden ser llamados a través de todo el *template* y pueden ser utilizados para factorizar código que es recurrente.

**Expresiones:** Existen expresiones generales para incluir los valores de las expresiones computadas en el texto generado, de manera a producir las partes dinámicas del texto de salida. Las expresiones pueden también ser utilizadas para llamar a otros *templates* y de esa forma, incluir el código generado por el *template* llamado en el código producido por *template* llamante. La invocación a otros *templates* puede ser comparado a los métodos en Java.

**Áreas protegidas:** Un concepto especial llamado *protected area* ha probado ser útil y es soportado por el Acceleo utilizando el *tag protected*. Los *protected areas*, se emplean para marcar secciones en el código generado que no deben ser sobrescritos de nuevo, luego de una nueva ejecución del generador de código. Esta sección típicamente contiene código manualmente escrito.

**4.7 Transformacion a codigo de los pim de moweba con Acceleo.**

Acceleo propone un ambiente ameno de trabajo basado en el IDE del Eclipse. Uno puede seleccionar la vista propia del Acceleo en el IDE y obtendrá un ambiente personalizado de trabajo con todas las características anteriormente citadas, en donde se podrá ver el editor de plantillas de transformación, la grilla de propiedades, la grilla de errores y la barra exploradora en donde es posible navegar sobre un proyecto el formato de árbol de expansión. En él se encuentran las plantillas de transformación, los modelos de entrada en formato XMI y los módulos de servicio de Java para complementar a las plantillas de transformación.

Para poder llevar a cabo las transformaciones sobre los modelos de MoWebA, se tuvieron en cuenta las siguientes herramientas para el proceso de desarrollo con el Acceleo:

IDE Eclipse Kepler Service release 2

Acceleo Versión 3.4

UML Designer for Eclipse Kepler version 3.0

**4.7.1 Transformación de los modelos de MoWebA de MOF a EMF UML2 (v2.x) XMI.**

Teniendo en cuenta que Acceleo solamente puede des-serializar modelos de entrada UML en el formato EMF UML 2, es necesario primeramente exportar el proyecto con los modelos PIM y perfiles UML desde la herramienta Magic Draw 16.0 en la cual fueron modelados en primera instancia. Una vez llevado a cabo este paso, el proyecto es importado al Acceleo y de esta forma se tienen los modelos PIM junto a los perfiles UML en la versión UML2 que son los elementos de entrada a la herramienta de transformación, que serán posteriormente des-serializados por medio de las plantillas.

El enfoque tomado para llevar a cabo las transformaciones se basa principalmente en dos plantillas de transformación. La primera de ellas, la plantilla de contenido, se encarga de transformar a los distintos elementos de interfaz que han sido definidos por medio del perfil de contenido de MoWeba. Dependiendo de sí el elemento modelado, es un elemento de interfaz RIA o no, se generará el archivo HTML correspondiente a la página, con la sección *Javascript*, encerrada con las etiquetas *script* o no. Solamente los elementos que forman parte de la extensión propuesta a MoWebA presentan código Javascript para la librería jQueryUI y jQuery Form Validate. Por supuesto, dependiendo del elemento RIA definido, el código Javascript generado, presentará características propias del elemento y comportamientos que fueron definidos en el modelo PIM de contenido.

Por otro lado se encuentra la plantilla de estructura, que transforma las posiciones definidas en pixeles, a cada uno de los *compositeUIElement* en un archivo .css con las coordenadas de posicionamiento correspondiente a cada uno de ellos. A continuación se presentaran las plantillas de contenido y estructura respectivamente.

**4.7.2 Plantilla de transformación para los elementos del perfil de contenido.**

Esta plantilla tiene la responsabilidad de llevar a cabo la transformación de los distintos elementos de interfaz definidos en el perfil de contenido. Dentro de los elementos definidos en el perfil de contenido, tenemos a los elementos que no tienen propiedades enriquecidas y que no tienen características interactivas. Estos elementos son los correspondientes a los de la web 1.0 y son representados por medio de etiquetas y atributos HTML en el cuerpo *(body).*

Por otro lado se encuentran los elementos con propiedades enriquecidas (RIAS) como los *richToolTip*, *richAccrodion*, *richTabs*, *richDatePicker*, *richAutoSuggest* y los *richFieldLiveValidation* que son parte de la extensión llevada a cabo a MoWebA para este trabajo de fin de carrera. Estos elementos a la par de contar con la sección body del HTML para representar el elemento, también cuentan con una sección *Javascript* (encerradas en el *tag script*) para representar la parte dinámica del elemento. La sección correspondiente al tag script contiene el código jQuery correspondiente al elemento definido. Cabe destacar el punto de que el identificador (*id*) de todos los elementos de interfaz, se establecen por medio del nombre del elemento, sin espacios. La identificación de cada uno de los elementos por medio del *id*, resulta importante, principalmente para los elementos de interfaz RIAS, debido a que permiten machear el código *Javascript* generado para *jQuery* en la sección del *tag script* (correspondiente a la parte dinámica) con el código HTML generado en el *tag body* para el elemento (correspondiente a la parte estática).

Primeramente la plantilla inicia verificando la clase principal del PIM de contenido, esta clase es la clase con el estereotipo *PresentationPage*, que indica el nombre que va a tener la página. Por ende, abre un archivo HTML de salida con tal nombre, en donde todos los elementos de interfaz definidos en el resto de las clases del modelo de clases, estarán contenidos dentro de este archivo. El nombre de la página, junto a las dependencias CSS (correspondientes al posicionamiento de los elementos, obtenidos a partir de la plantilla de posicionamiento y los correspondientes a *jQueryUI* y *jQuery form validation*) y *Javascript* (correspondientes a *jQueryUI* y *jQuery form validation*)*,* están definidos dentro de la plantilla, encerradas dentro del *tag head*.

Seguidamente se definen los componentes correspondientes a los tags *script* (en caso de elementos enriquecidos solamente) y *body* respectivamente del archivo abierto HTML. En la Figura 2, se presenta un ejemplo del proceso de transformación para el elemento *richDatePicker*. En primera instancias se muestra el modelo PIM de entrada, seguido de la plantilla de transformación para la sección *Javascript* y por último la plantilla de transformación donde se define al elemento en sí mismo. Como puede apreciarse en el modelo PIM de entrada, el *richDatePicker* (marcado en celeste), puede estar definido junto a varios otros elementos de interfaz, dentro de la clase que la contiene. Cada uno de los elementos es definido por medio de atributos estereotipados y valores etiquetados específicos.

El estereotipo *richDatePicker* indica que el atributo *fecha de nacimiento*, es un calendario y los valores etiquetados del atributo, definen las características del *datePicker*. Dentro de los valores etiquetados definidos para el atributo *fecha de nacimiento*, tenemos a *changeMonth*, *dateFormat* y *yearRange;* que indican respectivamente que una lista de los meses se agregará al *datePicker*, que el formato de fecha con el cual el cuadro de texto será completado, luego de la selección de una fecha dada en el calendario desplegado, será del tipo *ISO 8601 –yy-mm—dd* y que el rango 1960:2015 será desplegado en una lista.

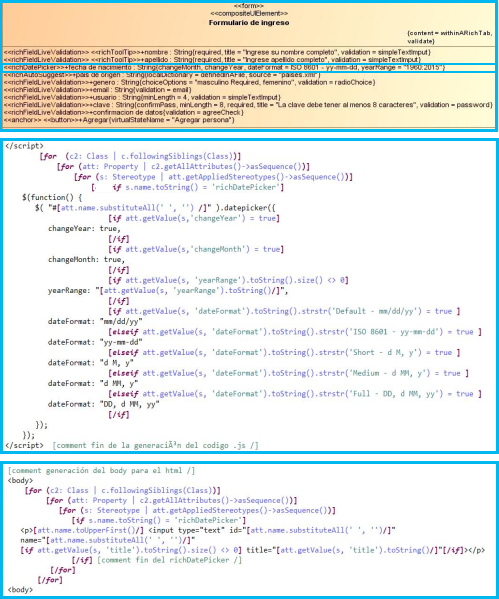


Figura 2 PIM de contenido de MoWebA y templates de trasnformación de contenido

La plantilla de contenido lleva a cabo dos iteraciones completas sobre las clases y sus atributos respectivos definidos en el PIM de presentación. La primera pasada, es para la generación del código *Javascript* correspondiente a los elementos enriquecidos. La otra pasada es para la definición del cuerpo del elemento que corresponde al *tag body*. En cada una de las iteraciones sobre las clases, se verifican sus atributos estereotipados para que en caso de estar definido en el modelo, un elemento en particular, se escriba el código correspondiente al elemento. El nombre del atributo estereotipado de un elemento enriquecido de la clase, se interpreta en la plantilla como el identificador (id) de elemento, y sirve para machear el comportamiento dinámico del elemento con la definición del mismo. Cada uno de los metamarcadores, iteradores y sentencias condicionales, permiten obtener los valores del modelo de entrada, iterar sobre los distintos elementos y preguntar sobre los distintos elementos de interfaz. En la Figura 3 se observa el código HTML generado por la plantilla de transformación para el elemento *richdatePicker* definido en el ejemplo de la Figura 2.

****

Figura 3 Código fuente HTML generado para el *richDatePicker* generado a partir de las plantillas

**4.7.3 Plantilla de transformación para el posicionamiento de los elementos de contenido.**

Con la plantilla de transformación para el posicionamiento de los elementos de interfaz dentro de las páginas, es posible generar el código correspondiente a los *cascading style sheets* (css) a partir de los modelos PIM de estructura (*Layout*). Primeramente al igual que en la plantilla de contenido presentada anteriormente, es necesario importar los servicios Java para poder utilizar dentro de la plantilla, expresiones que no son OCL estándar, como por ejemplo el método *hasStereotype* que permite saber si un elemento UML posee cierto estereotipo para llevar a cabo decisiones. Seguidamente se decide el nombre y la extensión del archivo de salida por medio del *tag file* y dentro de este *tag* comienza el proceso de recorrido dentro los elementos del tipo *package*, en donde se buscan los valores etiquetados del tipo *cUIElement*. Para cada uno de los valores etiquetados *cUIElement* encontrados dentro de unpaquete estereotipado con *Layout*, se agregan los valores correspondientes a las posiciones definidos en el modelo PIM. Las posiciones a definirse corresponden a los valores en pixeles del *height*, *width*, *xPosition(left)* y *yPosition(top)*. En la figura 4 se presenta el template de transformación de estructura.

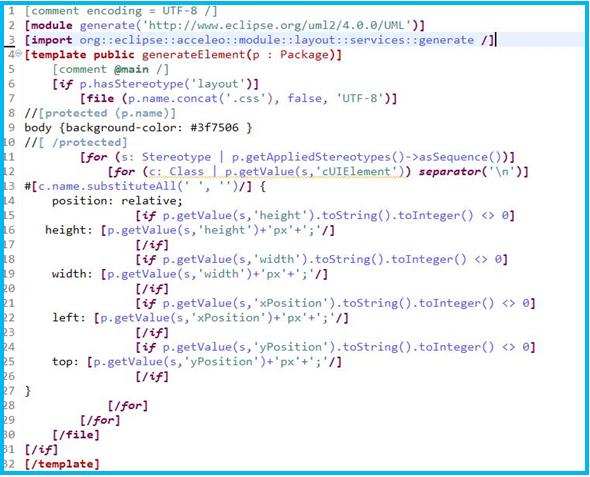


Figura 4 Plantilla de transformación para el posicionamiento de elementos

**4.8 Conclusiones**

En este capítulo se presentó una introducción a los conceptos básicos de las transformaciones M2T. Se dio una breve presentación de los elementos existentes en las plantillas de transformación como así también a algunas de las herramientas existentes en la actualidad para llevar a cabo la transformación de modelo a texto. Entre las herramienta existentes encontramos a XSLT, JET, XPAND, MOFScript y Acceleo. Para este trabajo de fin de carrera, se utilizó a Acceleo para llevar a cabo las transformaciones sobre los modelos UML de entrada, debido a su herramienta de trabajo que actualmente se encuentra bastante evolucionada y debido al soporte que ofrece su comunidad abierta que es parte de la *Eclipse Fundation*. Luego se dio pié a las principales características de Accelo junto a una descripción de los metamarcadores más importantes que pueden utilizarse dentro de las plantillas y que facilitan la des-serialización de los modelos de entrada, que corresponden al código que será generado dinámicamente.

Por último, se describieron a las plantillas de transformación de contenido y posicionamiento de MoWebA, que permiten generar el código (HTML, *Javascript para jQueryUI y jQuery Form Validate)* correspondiente a los distintos elementos que pueden ser definidos en el PIM de presentación, y como estos elementos una vez definidos pueden ser posicionados dentro de las páginas.

1. [www.w3.org/TR/xslt20/](http://www.w3.org/TR/xslt20/) 2015 [↑](#footnote-ref-1)
2. <https://projects.eclipse.org/projects/modeling.m2t.jet> 2015 [↑](#footnote-ref-2)
3. <https://eclipse.org/modeling/m2t/?project=xpand> 2015 [↑](#footnote-ref-3)
4. <https://eclipse.org/gmt/mofscript/> 2015 [↑](#footnote-ref-4)
5. [www.acceleo.org/](http://www.acceleo.org/) 2015 [↑](#footnote-ref-5)